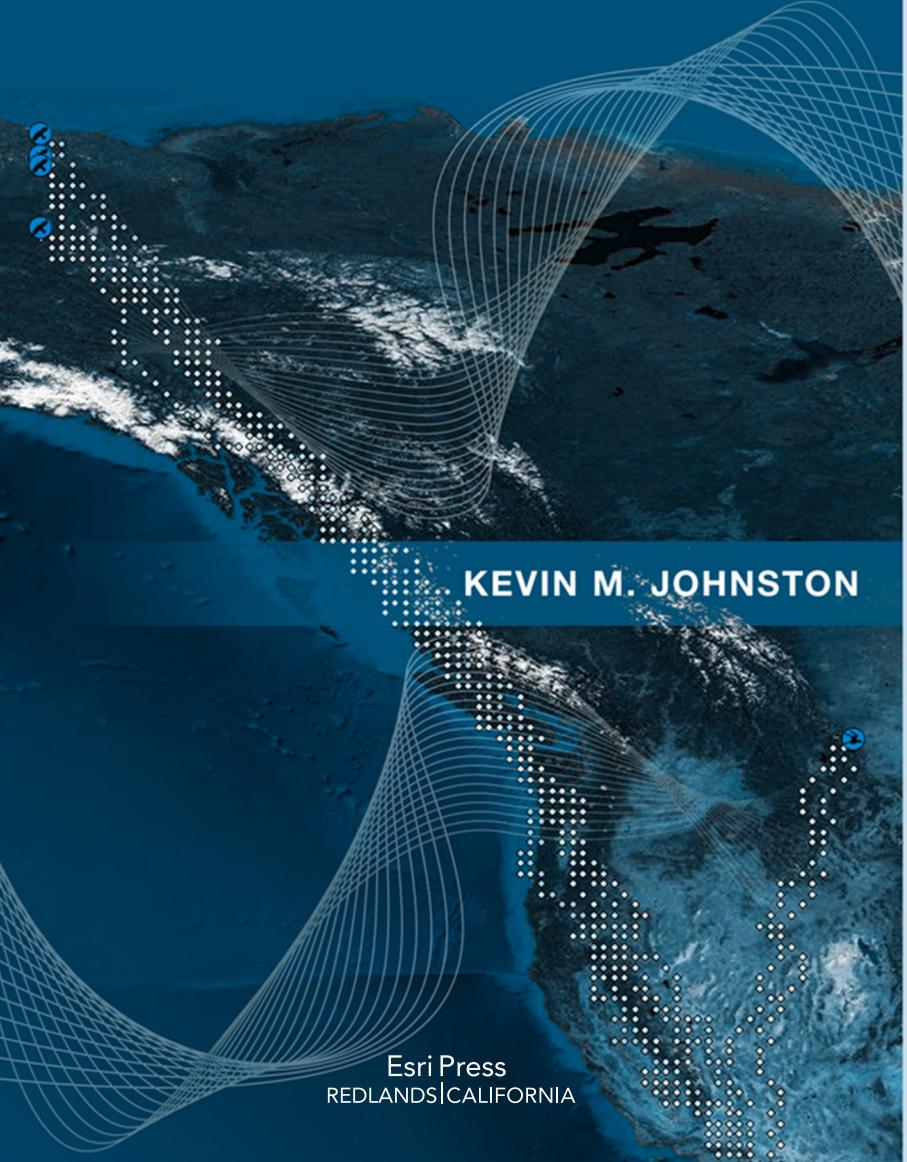


AGENT ANALYST

Agent-Based Modeling in ArcGIS®



KEVIN M. JOHNSTON

Esri Press
REDLANDS | CALIFORNIA

On the cover: Imagery from NASA's Earth Observatory and bird data from Nathan Strout.

Esri Press, 380 New York Street, Redlands, California 92373-8100

Copyright © 2013 Esri

All rights reserved. First edition 2013

Printed in the United States of America

17 16 15 14 13 1 2 3 4 5 6 7 8 9 10

ISBN: 9781589483231

The information contained in this document is the exclusive property of Esri unless otherwise noted. This work is protected under United States copyright law and the copyright laws of the given countries of origin and applicable international laws, treaties, and/or conventions. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying or recording, or by any information storage or retrieval system, except as expressly permitted in writing by Esri. All requests should be sent to Attention: Contracts and Legal Services Manager, Esri, 380 New York Street, Redlands, California 92373-8100, USA.

The information contained in this document is subject to change without notice.

US Government Restricted/Limited Rights: Any software, documentation, and/or data delivered hereunder is subject to the terms of the License Agreement. The commercial license rights in the License Agreement strictly govern Licensee's use, reproduction, or disclosure of the software, data, and documentation. In no event shall the US Government acquire greater than RESTRICTED/LIMITED RIGHTS. At a minimum, use, duplication, or disclosure by the US Government is subject to restrictions as set forth in FAR §52.227-14 Alternates I, II, and III (DEC 2007); FAR §52.227-19(b) (DEC 2007) and/or FAR §12.211/12.212 (Commercial Technical Data/Computer Software); and DFARS §252.227-7015 (DEC 2011) (Technical Data – Commercial Items) and/or DFARS §227.7202 (Commercial Computer Software and Commercial Computer Software Documentation), as applicable. Contractor/Manufacturer is Esri, 380 New York Street, Redlands, CA 92373-8100, USA.

@esri.com, 3D Analyst, ACORN, Address Coder, ADF, AML, ArcAtlas, ArcCAD, ArcCatalog, ArcCOGO, ArcData, ArcDoc, ArcEdit, ArcEditor, ArcEurope, ArcExplorer, ArcExpress, ArcGIS, arcgis.com, ArcGlobe, ArcGrid, ArcIMS, ARC/INFO, ArcInfo, ArcInfo Librarian, ArcLessons, ArcLocation, ArcLogistics, ArcMap, ArcNetwork, *ArcNews*, ArcObjects, ArcOpen, ArcPad, ArcPlot, ArcPress, ArcPy, ArcReader, ArcScan, ArcScene, ArcSchool, ArcScripts, ArcSDE, ArcSdl, ArcSketch, ArcStorm, ArcSurvey, ArcTIN, ArcToolbox, ArcTools, ArcUSA, *ArcUser*, ArcView, ArcVoyager, *ArcWatch*, ArcWeb, ArcWorld, ArcXML, Atlas GIS, AtlasWare, Avenue, BAO, Business Analyst, Business Analyst Online, BusinessMAP, CityEngine, CommunityInfo, Database Integrator, DBI Kit, EDN, Esri, esri.com, Esri—*The GIS Company*, Esri—The GIS People, Esri—The GIS Software Leader, FormEdit, GeoCollector, Geographic Design System, Geography Matters, Geography Network, geographynetwork.com, Geoloqi, Geotrigger, GIS by Esri, gis.com, GISData Server, GIS Day, gisday.com, GIS for Everyone, JTX, MapIt, Maplex, MapObjects, MapStudio, ModelBuilder, MOLE, MPS—Atlas, PLTS, Rent-a-Tech, SDE, SML, Sourcebook•America, SpatiaLABS, Spatial Database Engine, StreetMap, Tapestry, the ARC/INFO logo, the ArcGIS Explorer logo, the ArcGIS logo, the ArcPad logo, the Esri globe logo, the Esri Press logo, The Geographic Advantage, The Geographic Approach, the GIS Day logo, the MapIt logo, The World's Leading Desktop GIS, *Water Writes*, and Your Personal Geographic Information System are trademarks, service marks, or registered marks of Esri in the United States, the European Community, or certain other jurisdictions. CityEngine is a registered trademark of Procedural AG and is distributed under license by Esri. Other companies and products or services mentioned herein may be trademarks, service marks, or registered marks of their respective mark owners.

Ask for Esri Press titles at your local bookstore or order by calling 800-447-9778, or shop online at esri.com/esripress. Outside the United States, contact your local Esri distributor or shop online at eurospanbookstore.com/esri.

Esri Press titles are distributed to the trade by the following:

In North America:

Ingram Publisher Services

Toll-free telephone: 800-648-3104

Toll-free fax: 800-838-1149

E-mail: customerservice@ingrampublisherservices.com

In the United Kingdom, Europe, Middle East and Africa, Asia, and Australia:

Eurospan Group

3 Henrietta Street

London WC2E 8LU

United Kingdom

Telephone: 44(0) 1767 604972

Fax: 44(0) 1767 601640

E-mail: eurospan@turpin-distribution.com

Contents

Acknowledgments vii

Introduction ix

Section 1 An introduction to agent-based modeling

| | | |
|-----------|---|---|
| Chapter 1 | Introducing agent-based modeling in the GIS environment | 1 |
|-----------|---|---|

Section 2 The basics of points, polygons, and rasters in agent-based modeling

| | | |
|-----------|--|----|
| Chapter 2 | Creating an agent-based model using point agents | 31 |
|-----------|--|----|

Exercise 2a Installing Agent Analyst and exercise data 34

Exercise 2b Exploring the capabilities of Agent Analyst 36

Exercise 2c Importing, exploring, and running an existing agent-based model 41

Exercise 2d Creating a new agent-based model: defining agents, properties, and actions, and then scheduling and running the model 47

Exercise 2e Initializing agents with a starting state 52

Exercise 2f Updating point agent locations in Agent Analyst and watching them dynamically move in ArcMap 54

Exercise 2g Creating additional fields or properties and moving the agents with greater sophistication 59

Exercise 2h Accessing raster data and working with graphs 63

Exercise 2i Tracking the path of the moving point agent 68

Exercise 2j Building complex agents from multiple agents 71

Exercise 2k Developing interaction between agents 75

| | | |
|-----------|---|----|
| Chapter 3 | Changing attribute values of polygon agents | 81 |
|-----------|---|----|

Exercise 3a Investigating the completed polygon agent model 85

Exercise 3b Initializing polygon agents 89

Exercise 3c Initializing generic agents 93

Exercise 3d Creating neighborhoods for polygon agents 97

Exercise 3e Formulating the agent decision making 102

Exercise 3f Developing agent move rules 105

Exercise 3g Using multiple decision rules 107

| | | |
|--------------------|--|------------|
| Chapter 4 | Querying and changing raster data with Agent Analyst | 113 |
| <i>Exercise 4a</i> | Running the model | 117 |
| <i>Exercise 4b</i> | Loading raster data into Agent Analyst | 121 |
| <i>Exercise 4c</i> | Accessing raster data and raster properties with Agent Analyst | 125 |
| <i>Exercise 4d</i> | Selecting a random location in a raster | 129 |
| <i>Exercise 4e</i> | Reading raster pixel values | 133 |
| <i>Exercise 4f</i> | Building the SOME model | 136 |
| <i>Exercise 4g</i> | Copying raster data | 141 |
| <i>Exercise 4h</i> | Running computational experiments with the SOME model | 144 |

Section 3 Advanced techniques for points, polygons, and rasters

| | | |
|--------------------|---|------------|
| Chapter 5 | Moving point agents based on multiple-criteria decision making | 153 |
| <i>Exercise 5a</i> | An overview of the model | 160 |
| <i>Exercise 5b</i> | Querying rasters for attributes of locations that will determine an agent's actions—evaluating the level of security for the eight surrounding cells for the security criterion | 165 |
| <i>Exercise 5c</i> | Using probability and random numbers to determine stochastic events—making a kill and weighting the eight surrounding cells for the remaining with the kill criterion | 172 |
| <i>Exercise 5d</i> | Using raster attributes and field observation data to determine the characteristics of an event—identifying the type of kill and how long it will take the cougar agent to consume it | 181 |
| <i>Exercise 5e</i> | Detecting and tracking other agents using attractors—weighting the eight surrounding cells for the pursuing a female cougar agent criterion | 187 |

| | | |
|----------------------|---|------------|
| Chapter 6 | Moving agents on representative networks | 203 |
| <i>Exercise 6a</i> | Introduction to a basic vector movement model | 206 |
| <i>Exercise 6b-1</i> | Deriving a representative network from a street file | 208 |
| <i>Exercise 6b-2</i> | Creating a list of neighboring nodes for routing | 212 |
| <i>Exercise 6c</i> | Recognizing the representative network in Agent Analyst | 215 |
| <i>Exercise 6d</i> | Using random number generators to simulate random movement | 223 |
| <i>Exercise 6e</i> | Using the cartographic capabilities of the ArcMap interface to represent agent movement | 230 |

| | | |
|--------------------|--|------------|
| Chapter 7 | Adding complexity to polygon agents using an urban growth model | 239 |
| <i>Exercise 7a</i> | Exploring the complete urban growth model | 244 |
| <i>Exercise 7b</i> | Using GIS data to add the landscape to the model | 253 |
| <i>Exercise 7c</i> | Making decisions at each step based on multiple criteria | 260 |
| <i>Exercise 7d</i> | Building probability and randomness into the agent's behavior | 271 |
| <i>Exercise 7e</i> | Initializing and keeping track of parcel agents' permanent and dynamic conditions | 279 |
| <i>Exercise 7f</i> | Tracking changes at the model level due to the parcel agents' actions | 289 |
| <i>Exercise 7g</i> | Building scenarios using different weights on suitability criteria | 295 |
| Chapter 8 | Adding complexity to moving discrete point agents over continuous surfaces | 309 |
| <i>Exercise 8a</i> | Geometric deterrents that influence decision making | 315 |
| <i>Exercise 8b</i> | Adding memory to develop movement with intent | 321 |
| <i>Exercise 8c</i> | Monitoring the status of each agent and using that status for agent assessment | 326 |
| <i>Exercise 8d</i> | Creating probability distributions from the neighboring locations for the multiple criteria to create the decision space for the agent | 331 |
| <i>Exercise 8e</i> | Moving the point agents | 340 |
| <i>Exercise 8f</i> | Overriding the multicriteria decision-making process with momentary events | 343 |
| Chapter 9 | Adding complexity to agent movement on representative networks | 359 |
| <i>Exercise 9a</i> | Building an advanced vector movement model | 363 |
| <i>Exercise 9b</i> | Creating routes consisting of street nodes | 367 |
| <i>Exercise 9c</i> | Defining and using activity spaces in Agent Analyst | 385 |
| <i>Exercise 9d</i> | Moving agents around predefined activity spaces | 393 |
| <i>Exercise 9e</i> | Using the cartographic capabilities of the ArcMap interface to represent agent movement | 397 |
| <i>Exercise 9f</i> | Incorporating variety into activity spaces | 400 |

Section 4 Agent-based modeling accentuated with GIS

Chapter 10 Building synergy between GIS and agent-based modeling 411

| | | |
|---------------------|---|-----|
| <i>Case study 1</i> | Using GIS analysis tools for data generation and organization in the model process | 415 |
| <i>Case study 2</i> | Using ArcGIS spatial modeling capability to call ArcObjects from Agent Analyst—developing a fire model | 420 |
| <i>Case study 3</i> | Adding temporal related viewing capability to the ABM output using ArcGIS Tracking Analyst and the bird migration model | 423 |
| <i>Case study 4</i> | Integrating third-party movie-making software with agent-based models to facilitate dynamic visualization | 428 |
| <i>Exercise 10a</i> | An overview of the bird migration model | 435 |
| <i>Exercise 10b</i> | Creating a tracking log using ArcObjects to save agents at each tick | 439 |
| <i>Exercise 10c</i> | Demonstrating the 2D ArcMap animation tools using the bird migration model | 446 |
| <i>Exercise 10d</i> | Demonstrating the 3D ArcGlobe animation tools using the bird migration model | 453 |

Section 5 Advanced techniques for points, polygons, and rasters

Chapter 11 Patterns of movement in agent-based modeling 463

| | | |
|---------------------|--|-----|
| <i>Exercise 11a</i> | Performing simple movements for a point agent | 466 |
| <i>Exercise 11b</i> | Moving a point agent toward a stationary point agent | 475 |
| <i>Exercise 11c</i> | Moving a point agent toward a moving point target | 480 |
| <i>Exercise 11d</i> | Having a point agent evade a moving point agent target | 483 |
| <i>Exercise 11e</i> | Moving a polygon agent toward a fixed polygon agent | 487 |
| <i>Exercise 11f</i> | Having a polygon agent follow a moving polygon agent | 496 |
| <i>Exercise 11g</i> | Having a polygon agent evade a moving polygon agent | 500 |
| <i>Exercise 11h</i> | Moving a point agent relative to a raster surface | 504 |

Appendix: Not Quite Python 525

About the contributors 543

Data source credits 546

Esri Data License Agreement 547

Acknowledgments

The contributors thank the following people and groups for contributing data, images, videos, and expertise to this book.

For reviewing the manuscript:

- Brian Frizzelle, Spatial Analysis Services, Carolina Population Center, University of North Carolina at Chapel Hill.
- Jim Kuiper, GIS Project Developer/Analyst, Environmental Science Division, Argonne National Laboratory, Lemont, Illinois.
- James Rineer, PE, Geospatial Science and Technology Program, RTI International, Research Triangle Park, North Carolina.
- Tim Clark, United States Army, Geospatial Center, Alexandria, Virginia.
- Jerry D. Davis, PhD, Chair, Department of Geography and Human Environmental Studies, and Director, Institute for Geographic Information Science, San Francisco State University.

United States Census Bureau for the census block information for Wenatchee, Washington, for chapter 3 and for Chicago for chapter 11.

Michigan Center for Geographic Information for the base data used in chapter 4, and the United States National Science Foundation for support of the work represented in chapter 4, under grant #GEO-0814542.

David J. Mattson, PhD, and Terry Arundel of the United States Geological Survey, Southwest Biological Science Center in Flagstaff, Arizona, for the expertise in the development of the cougar model rules and decision making and the cougar spatial and field data for chapters 5 and 8.

City of Seattle for the street center line data used in chapters 6 and 9.

City of Redlands, California, for the base data for chapter 7.

The United States Army Research Office, contract number W911NF-07-1-0392, for their support in the model development in chapter 7.

Vermont Center for Geographic Information for the data provided for case study 1 in chapter 10.

Tony Turner, GIS consultant, owner of Yucaipa Media Lab, for his input to case study 4 of chapter 10 and the initial development of chapter 11.

Paul M. Torrens, PhD, of the Geosimulation Research Laboratory, School of Geographical Sciences and Urban Planning, Arizona State University, for his input, examples, and videos for case study 4 in chapter 10.

Samad Paydar and Ali Ghazinejad, Center for Research on Mediated Interaction, Indiana University, Bloomington, for development of the code for the examples in chapter 11.

Michelle Gudorf, consultant, not only for her contributions to chapters 8 and 10, but for her significant efforts in editing the book.

Matthew Lisk, a graduate student at Prescott University, Prescott, Arizona; and Sarah Lamb, an undergraduate at Johnson State College, Johnson, Vermont, for their assistance in testing the exercises.

Introduction

Agent Analyst: Agent-Based Modeling in ArcGIS teaches agent-based modeling (ABM) techniques and how to integrate those techniques with spatial data. ABM explores the causality of phenomena. From the aggregation of individual decision making, the agents produce the resulting perceived patterns.

This book is particularly useful for those using ArcGIS software or those wishing to integrate spatial components into their agent-based models. This workbook was written with an assumption that you have some experience with a geographic information system (GIS), in particular with ArcGIS software. If you do not have this introductory knowledge, we encourage you to first work through *Getting to Know ArcGIS Desktop* (Esri Press 2010). A basic understanding of scripting, in particular with Python or Java, is useful but not essential.

The step-by-step exercises in this book teach ABM principles and use Agent Analyst (AA) software, which is an extension designed specifically for developing and running agent-based models in a spatial environment. Agent Analyst is a free, open-source extension compatible with ArcGIS software, voluntarily created by several of the authors (Collier, Johnston, and North). In the spirit of open-source software, Agent Analyst is owned by the ABM community: it is your software.

Agent Analyst is built on the Not Quite Python (NQPy) language. NQPy is a powerful language similar to Python. It translates functions from NQPy to Java. Almost all Java functions are available through this easier-to-use, more Python-like syntax (with Python being the preferred scripting language for ArcGIS software). NQPy will meet the majority of the needs of ArcGIS users; however, advanced users can export their models to Java and continue to perform very sophisticated techniques. A complete discussion of the NQPy language is presented in the appendix.

You will need ArcGIS Desktop 10 to complete the exercises in *Agent Analyst: Agent-Based Modeling in ArcGIS*. Any license level of ArcGIS Desktop 10 software will work. This book comes with the Agent Analyst extension and data and models required to complete the exercises. You need to install both the extension and the data to complete the exercises in the book. You can download the Agent Analyst extension and the data and models from <http://resources.arcgis.com/en/help/agent-analyst/index.html>.

The exercises in this workbook walk you through a series of modeling applications that range from cougar movement to segregation modeling, land-use change, and crime analysis. The applications vary in complexity and are as realistic as possible, but more importantly, they are a means for presenting ABM techniques. Occasionally, the rules are simplified for demonstration purposes. If you question a specific rule based on your own experience, you can easily change the model rules to capture your insight.

Chapter 1 provides an overview of ABM. Chapters 2, 3, and 4 provide basic techniques of ABM for point, polygon, and raster data that correspond to agent types, respectively. Chapters 5 and 8 expand on the ABM techniques for points, and chapter 7 covers more advanced techniques for polygons. Chapter 6 introduces you to networks, and chapter 9 builds on these basic network techniques. The discussions throughout chapters 1 through 9 address how ABM accentuates the GIS, leading to chapter 10, which describes how a GIS can accentuate an agent-based model. Chapter 11 synthesizes the movement principles presented in chapters 1 through 10 and generalizes movement in code snippets so that you can use the code when building your own agent-based models.

Each chapter is designed to stand alone. Each chapter covers a single model application. However, several applications are continued in multiple chapters. Chapters have five to eight exercises each. The exercises are designed to stand alone within each chapter. Each exercise demonstrates a different ABM technique relative to the agent type covered in the chapter. Almost every exercise is associated with an ArcMap document (.mxd), an ArcGIS toolbox, and an Agent Analyst model. Because each exercise can stand alone, if you make a mistake in an exercise, you can continue with the subsequent exercises. Each exercise requires 20 to 40 minutes to complete.

We encourage you to work through all the exercises, but it is recommended that all readers complete chapters 1, 2, 10, and 11. Based on your model and data types, you might focus on specific chapters. If your problem involves points, we suggest that you also work through chapters 5 and 8. If your model involves polygons, work through chapters 3 and 7; when using rasters, work through chapter 4; and if your model involves networks, work through chapters 6 and 9.

Welcome to the ABM community. It is now time to explore your phenomenon from a different perspective—from an ABM perspective.

Section 1: An introduction to agent-based
modeling

Chapter 1

Introducing agent-based modeling in the GIS environment

by Kevin M. Johnston, Michael J. North,
and Daniel G. Brown

If you are reading this book, it's likely you want to hone your skills in agent-based modeling (ABM) or want to build models using Agent Analyst software. Perhaps you want to add ABM capabilities to your ArcGIS analysis, or you are just curious what ABM can do for you. No matter what your reason, this book will address your interest and questions. This workbook teaches you basic as well as advanced ABM techniques, and you will implement these concepts using the Agent Analyst open-source software that is compatible with ArcGIS software. In addition, it will demonstrate how to implement ABM concepts through detailed discussions of actual code. The techniques are implemented through various realistic models providing you with examples that you can adapt for your specific problem.

We are under no illusions; it is not easy to create an agent-based model. It is difficult to define the problem, difficult to understand the intricacies of the phenomenon that may be necessary for your agent-based models, and difficult to define the actions and rule sets for the model because coding is required. We provide guidance on how to define the ABM problem. By working through the models presented in each chapter, you will be exposed to the process of representing your phenomenon in a model. This workbook will walk the novice as well as the experienced coder through developing the actions and rule sets.

Developing a model represents a process. This is especially true for ABM. By building a model, you explore what you know as well as what you do not know about that process. Let's get started so you can begin to understand your phenomenon from the perspective of dynamics and causality using agent-based modeling.

In this chapter you will be given an overview of agent-based modeling, its integration within a geographic information system (GIS), a quick tour of the Agent Analyst software, and some considerations for creating an agent-based model. This chapter is not intended to be a comprehensive theoretical background to the field. We provide you with a series of references to texts and articles at the end of the chapter that will enhance your understanding of the subject.

An introduction

Conceptually, in ABM you give instructions to virtual agents that allow the agents to interact. Agents can be animals, tanks, parcels, delivery trucks, or any discrete object. From the resulting decisions and actions of the agents, patterns are created in space and time. Unlike many other modeling techniques that quantify and then re-create the patterns, agent-based models explore the causes of the patterns; the patterns are emergent properties from the individual decisions of the agents.

Agent-based modeling, combined with spatial data, allows you to address a wide array of problems such as the following:

- Developing corridor connectivity networks for wildlife movement
- Anticipating potential terrorist attacks
- Analyzing traffic congestion or producing evacuation strategies
- Planning for the potential spread of disease such as bird or swine flu
- Understanding land-use change
- Optimizing timber tract cutting
- Exploring energy flow on electrical networks
- Performing crime analysis to deter future impact

Many phenomena or agents exist and make decisions in, and relative to, space. The location of an agent and its surrounding environment will influence the agent's decision making. The agent can influence or change the landscape it interacts with. A GIS is a spatial modeling tool that stores, displays, and analyzes data on spatial relationships. A natural synergy exists between ABM and a GIS. Agent Analyst is free, open-source software developed to integrate an ABM development platform—the Recursive Porous Agent Simulation Toolkit (Repast)—within a GIS (ArcGIS). Agent Analyst is a mid-level integration that takes advantage of both modeling environments.

The intent of this book is to provide guiding principles for defining agents and interaction rules. The following chapters provide detailed descriptions via examples, sample code, and models of the mechanics of developing an agent-based model in Agent Analyst.

Agent Analyst: Agent-Based Modeling in ArcGIS is presented in a step-by-step tutorial style. Each exercise is designed to expose you to different ABM techniques on agents that are associated with different types of spatial features. The book focuses on agents that can be represented as points, polygons, or raster cells and agents on networks, and its structure is based on these agent types. The exercises build upon previous chapters and progressively get more sophisticated. They are placed in the context of application models that include animal movement, housing segregation, land-use change, and criminal activity; however, this book is not about creating specific application models. The example models selected for this book are used only to establish context for the specific ABM techniques being presented. For those of you who are domain experts in these application areas, the rules used are only for example, and you can change them to meet your needs.

Overview of the chapters

Chapter 1 The philosophy and history of ABM, GIS, and their integration. A quick tour of Agent Analyst is provided.

Chapter 2 How to get started with Agent Analyst. An introduction to the mechanics of Agent Analyst is presented through modeling a cougar, a point agent. Since point agents are the easiest to conceptually understand, these are the best agents to start with when learning ABM techniques.

Chapter 3 An introduction to polygon agents presented through a district segregation model.

Chapter 4 An introduction to querying and changing raster data (loosely referred to as *raster agents*) presented through a land-use change model.

Chapter 5 Additional ABM techniques on point agents is explored by building on the cougar model from chapter 2.

Chapter 6 An introduction to network agents using a crime model.

Chapter 7 Additional ABM techniques on polygon agents using a land-use change parcel model.

Chapter 8 Advanced techniques on point agents using the cougar model.

Chapter 9 Additional ABM techniques on network agents is presented by further developing the crime model from chapter 6.

Chapter 10 Techniques on how ArcGIS accentuates an agent-based model.

Chapter 11 Generic movement techniques for points, polygons, and raster agents, presented in stand-alone code snippets.

The philosophy of agent-based modeling

At its heart, agent-based modeling can be thought of as a way to determine the possible system-level consequences of the behaviors of groups of individuals (North and Macal 2007). ABM allows modelers to specify the behavioral rules of individuals or groups, identify the situations or contexts in which the individuals reside, and then execute these rules many times in their appropriate contexts to determine possible system outcomes. ABM can support both the creation of deterministic hierarchical models, where results at higher levels are simply the unavoidable product of accumulated lower-level actions, and holistic systems, where stochastic rules

on multiple levels work together in irreducible ways to produce overall system-level results. For example, ABM can represent feedback in multilevel systems where the higher and lower levels simultaneously influence and limit one another (Van Huyssteen 2003).

Agents usually represent the decision makers within agent-based models. As such, agents tend to display both adaptation and variability within each situation. Agents generally are individually identifiable components that are in some sense discrete. However, agents need not be completely separate from one another. The boundaries surrounding agents can be diffuse, and agents can share substantial amounts of information with one another.

ABM agents generally exhibit Herbert Simon's (1957) concept of bounded rationality. Bounded rationality means that agents can use only a finite amount of information and think for a limited period of time for each decision. This is in stark contrast to the assumptions made in more traditional paradigms such as neoclassical economics, where decision makers can use unbounded informational and computation resources before making a choice (Simon 1957).

In addition to being boundedly rational, agents also can use only local information (North and Macal 2007). In this case, local does not mean physically contiguous, but rather that each agent has a limited and potentially changing neighborhood from which the agent can obtain information. Agent neighborhoods may include both geographically proximate information sources and remote sources such as the telephone system, instant messaging, e-mail, and the World Wide Web. The common factors for locality are that each agent has a finite number of information sources and no one agent instantly knows everything all of the time.

Agents generally adapt or learn from their experiences over time. This learning can be deep, in that agents can develop completely new behavioral strategies, or shallow, in that agents adjust simple parameters that determine their next action. In the words of John Casti (1998), adaptation requires "rules and rules to change the rules." The sophistication of both the "rules and rules to change the rules" (Casti 1998) depends on the situation being modeled. Sometimes models are structured in an agent-based way, but the decision-making agents themselves lack the critical spark of adaptation or learning. In this case, the simulation is described as a "proto-agent model" and the decision makers as "proto-agents" (North and Macal 2007).

The environment is also an important contributor to agent-based models. Unlike agents, the environment does not make decisions. However, it is nonetheless interactive in that it can evolve and change over time. It can also provide a form of memory or information storage for the consequences of past agent behaviors that can influence future agent decision making.

As will be discussed in more detail later in the section on Repast, object-oriented programming (OOP) is a powerful foundation for building agent-based models. Objects within OOP systems are discrete software components that fuse data with functions in special ways. The agent data can be continuous or discrete attributes that can affect actions or interactions between agents. Actions of agents can be scheduled to take place synchronously (i.e., every agent performs actions at each discrete time step) or asynchronously (i.e., agent actions are scheduled

with reference to a clock or to the actions of other agents). The actions of agents can vary from completely reactive (i.e., agents perform actions only when triggered to do so by some external stimulus, such as actions of another agent) to goal-directed (e.g., seeking a particular goal). Research in artificial intelligence provides both frameworks for agent learning, in which goal-directed agents can improve their ability to reach goals through feedback from previous actions, and an understanding of how the collective actions of multiple agents acting in a self-interested way can produce aggregate patterns that either enhance or diminish the collective well-being of all agents.

From an object-oriented point of view, we can say that agents are self-directed objects in that they have data, action functions to operate on that data, self-direction functions to determine what functions to invoke next, and adaptation functions that use the stored data to make different choices in the future. Of course, it is possible for individual functions to mix roles and perform combinations of action, self-direction, and adaptation activities. The functions themselves are made up of sets of instructions or rules. The complexity of individual functions can range from simple activities composed of one instruction to complicated learning algorithms. This gives ABM built-in flexibility.

The inherent flexibility of ABM allows it to represent a vast array of important situations. What modeling questions are suitable for ABM? According to North and Macal (2007), some of the major factors to look for include situations in which

- the referent (i.e., the real-world situation to be represented) is clearly made up of individual components, especially when these components learn over time;
- the referent has a structure that changes, particularly when the change is due to internal forces;
- the referent develops higher-level structures that emerge from lower levels; and
- the referent includes a geographic component.

If one or more of these factors are present, then the modeling question is likely to be a good candidate for ABM. More details on the philosophy of agent-based modeling can be found in Gilbert and Troitzsch (2005), Grimm and Railsback (2005), and North and Macal (2007).

The history of agent-based modeling

Many researchers trace the history of computational ABM back more than 40 years to Stanislaw Ulam's cellular automata (CA) (Wolfram 2002). CAs are simple time-stepped models in which a large set of identical machines are spread out on a uniform rectangular grid, with one machine on each square. Each machine can independently take on one of two colors at each time step. The machines use simple programmable rules that are a function of their immediate

neighbors' colors. The goal was to determine whether simple worlds such as this could produce self-replicating machines. The short answer is "Yes, they can." From this humble beginning more complex models grew. Improvements included increasing the complexity of the rules, making the grids nonuniform so that different squares could have machines with different rules, and freeing the machines from squares entirely. Things evolved so far that some researchers today do not even consider CAs advanced enough to be agent-based models.

In terms of using ABM for social science, the initial pioneer is generally regarded as Thomas Schelling (1971). Schelling introduced a model of housing segregation that was intended to explore how arguably neutral people may unintentionally produce segregated neighborhoods. According to early reports, this model was not executed on computers but was manually calculated by students. As can be seen later in this book, ABM tools have come a long way since then.

Building on these early successes, researchers such as Robert Axelrod (1984) have looked at simple strategic interactions, while Joshua Epstein and Robert Axtell (1996) have begun to simulate whole societies. More recently, there has been an explosion in the number of successful ABM applications. North and Macal (2009) provide a survey of examples in the area of artificial life. More details on the history of agent-based modeling can be found in North and Macal (2007).

Why space can be integral to agent-based modeling

In some agent-based models, spatial relationships can become an integral part of the decision making of the agents for the following reasons:

- **One or more behaviors of an agent may involve movement** The agent may choose to run, walk, flow, or disperse in a time step. For example, a chemical spill may flow downhill, over the less-porous surfaces, following the steepest slope.
- **Agents often make decisions in space** An agent's location and its environment influence its decision making. Locations of other agents in relation to the processing agent can also influence the decision making of an agent. For example, in a model of human dispersal in a crowded market when shots go off, people (agents) may react by moving away from the origins of the shots or look for an exit or place to hide, and the agents might be making these decisions based on what other agents are doing.
- **Agents can change the spatial arrangement of features on the landscape** In a grazing model for an ungulate, the availability of browse will be reduced if a herd of ungulate agents feeds upon it. The resulting vegetation quantity map may dictate where the ungulate agents should move to continue browsing.

- **The agent's decision making may change with the changing landscape** A model for allocating resources for fighting a fire will change with the current fire location, the fire's intensity, what locations have been burned, the wind direction, and the changing moisture content of the landscape.

To define spatial relationships, an agent's behavior may be based on information derived from complex spatial analysis. For example, how far an agent is from the closest fire or modeling the location of a flood for any particular time step can be determined in a spatial modeling software package specifically designed for these types of applications. The resulting spatial surface may influence the decision making of the agent in that time step.

By integrating a GIS and ABM, you can utilize the strengths of both by using the dynamic object-oriented modeling capabilities of an agent-based model and the spatial modeling capabilities of a GIS to more accurately capture the nature of the phenomenon you are modeling.

How agent-based modeling capabilities accentuate those of a GIS

Most spatial models describe and explain the observed spatial patterns. They aim to describe and quantify existing patterns but do not provide a means of describing causality, or the dynamic process by which a pattern is formed. For example, predictive habitat and species distribution models in ecology associate the presence or abundance of a species with landscape characteristics like land cover, terrain, and hydrology. They use this information as indicators of where a species is likely to be found, but usually without using specific descriptions of mechanisms like dispersal, reproduction, or mortality (Guisan and Zimmermann 2000).

Similarly, models of land-use change are often evaluated by the degree to which they match observed patterns of land use, rather than the degree to which the processes that form those patterns are reasonable (Pontius et al. 2004). Agent-based modeling is intended to explore the relationship between observed spatial patterns and the processes that create them (Parker et al. 2003).

How GIS capabilities accentuate those of agent-based models

A GIS is generally a good analysis tool for spatial data and also offers a wide-ranging suite of tools that can model the effects of change from external forces onto the simulated landscape. These include both natural changes and changes brought about by the decision making and action of the agents in the model simulation.

When modeling decisions, a GIS might accentuate an agent-based model for the following reasons:

- There is a great deal of GIS data readily available.
- A GIS has a rich set of tools to manage, create, and edit spatial data.
- A GIS has sophisticated tools to display spatial data.
- There are many tools in a GIS to create complex spatial analysis and spatial models.
- There are many third-party applications built on a GIS that can be instrumental to ABM.
- GIS is a flexible, open platform on which to integrate many technologies in a single environment.

Integrating agent-based modeling with a GIS

The dynamic nature of agent-based models has made them difficult to integrate with a GIS, where the focus is on spatial representation, often at the expense of temporal representations (Peuquet 2002). Developments over the past two decades in representing time in spatial databases and dynamics in spatial processes have improved this situation (e.g., Miller 2005). Most of the early efforts in representing dynamics have been focused on raster GIS and the change of raster values exhibiting time (Karsenberg 2002). However, recent developments in graphical modeling (for example, in IDRISI software and in ArcGIS ModelBuilder) make dynamic models much more accessible to the GIS user.

The focus of early dynamic models on raster GIS has limited the types of models that can be developed. Because agents have the ability to move within a continuous space, agent-based models are fundamentally different from many of these models. For example, while dynamics in a cellular automaton model are usually defined by a set of fixed interactions (i.e., the neighborhood a cell interacts with is fixed by the cellular geometry), interactions in the agent-based model can be dynamically changed as the model runs because they are defined at the agent level rather than in terms of the partitioning of space. The dynamic nature of spatial interactions in ABM relates more favorably to the vector data model in GIS and to the time geography ideas developed originally by Hägerstrand (1967) and furthered by Miller (2005) and others. Although many applications of ABM require a combination of dynamics on a raster and dynamics of vector objects, traditional cellular models are limited to only one of these types of dynamics.

The recent availability of an object-oriented approach to composing GIS software, which parallels the structure of ABM programming tools and uses common architectures like the Component Object Model (COM), has facilitated the integration of ABM and GIS tools. What this means for the user or modeler is that the features in a GIS, represented as database objects,

can be manipulated by software agents that describe the dynamic actions of agents that can be thought of as the same entity as the spatial feature, or as a separate entity that can take action on those spatial features (Brown et al. 2005).

Although raster cells are not represented as objects in a GIS, in the same manner as vector features, the raster cell values also can be manipulated by software agents in an agent-based model. By combining these capabilities, interacting GIS and agent-based models can be used to represent a full range of dynamic spatial models, from cellular automata to agent-based models with reactive agents to richer agent-based models with goal-directed agents that can learn and update their behaviors.

There are several levels of integration for a GIS within ABM software (Brown et al. 2005). Some agent-based models will take data such as slope, aspect, and distance to specified features extracted from a GIS and incorporate it into the agent model. Often this is a one-time transfer; however, the results of the agent-based model can be ported back to the GIS for display. This transfer of data from and to a GIS—and to and from an agent-based model—is a loose coupling between the two software technologies. This may not be adequate if there is a need to update GIS data in order for changes in agents or their environment to have effects on subsequent agent actions or model dynamics. This more dynamic interaction between GIS and ABM generally requires a tighter integration of the two environments. Tightly integrated approaches can be agent-based-centric, GIS-centric, or a middleware approach that can link an existing GIS with agent-based software. Agent Analyst takes the middleware approach to create an integrated GIS/ABM environment.

The history of integrating GIS and agent-based modeling

Significant efforts to make explicit conceptual and software linkages between GIS and ABM began in the 1990s (with several examples outlined in Gimblett 2002). Aimed at providing a general-purpose modeling environment for developing agent-based models, development of the Swarm programming library, written in the Objective-C language, began in the mid-1990s at the Santa Fe Institute (Langton et al. 1995). Though Swarm did not have explicit linkages to GIS software, it was able to read and incorporate GIS data into models in a variety of ways. Concurrently, development efforts were under way that recognized the value of incorporating agent-type behaviors into the GIS toolkit. Perhaps most notable was the work on integrating mobile objects (or agents) with the open-source GRASS GIS (Westervelt and Hopkins 1999) to model wildlife populations, and with MapInfo to model recreation behaviors (Gimblett 1997). A hallmark of many of these earlier efforts was the significant amount of work that went into both creating the computer code to integrate the spatial data within the context of the agent behaviors (and vice versa) and structuring the data for passage between the ABM environment and a GIS visualization and analysis environment.

Despite the software challenges, a number of successful early applications in the fields of archaeology, ecology, and resource management clearly demonstrated the value of integrating agent-based models with spatial data and GIS functionality. In the field of archaeology, a long-running set of projects to study human adaptation to environment and social evolution was organized and carried out in the American Southwest by the Santa Fe Institute. The project demonstrated the value of agent-based models in explaining the spatial and temporal dynamics of the Anasazi civilization and the role of environmental change in that dynamic (Kohler and Gumerman 2000). In ecology, models of habitat use by animals, for example, the desert tortoise (Westervelt and Hopkins 1999), demonstrated the value of coupling landscape dynamics with mobility in animal populations to better understand habitat utilization. Models of resource utilization by recreationalists illustrated how models of dynamic human behavior can represent the feedbacks that affect resource utilization when, for example, decision making about where to go is affected by how many people are already at a location (Deadman and Gimblett 1994). These early successes led to a wide recognition of the power of combining spatial information with dynamic agent-based models in such areas as land use (Parker et al. 2003), epidemiology (Bian 2004), and criminology (Groff and Mazerolle 2008), as well as further development of these applications.

The availability of tools for agent-based modeling and the range of problems to which they have been applied have expanded rapidly over the past two decades. Given the wide range of actions that agents can take, some programming is still required to describe those actions. Agent-based models use object-oriented programming languages, such as C++, Java, or Objective-C, and there are several tools available to assist in building agent-based models separate from a GIS (Gilbert and Banks 2002). Building on the success of Swarm, a number of new software libraries that provide predefined routines have been designed specifically for use in agent-based modeling. These include Repast, Ascape, and MASON, all written with Java. The object-oriented programming paradigm of agent-based models allows for the incorporation of a range of other software libraries, including those that provide GIS functionality.

Some efforts have been made to reduce the prior programming knowledge required to build a functional agent-based model. NetLogo was explicitly designed to reduce the programming requirements for model building and is commonly used in teaching settings (Wilensky and Rand, *in press*). It provides a graphic visualization environment together with a restricted set of higher-level functions that can be used to easily construct agent behaviors in the context of a Euclidean space. Like Repast, NetLogo provides functions that can be used to read GIS data and make it available to the agents by passing files back and forth from a GIS database. There is a pervasive trade-off between ease of use and flexibility that affects the functionality of ABM tools.

Agent Analyst provides an interface that can integrate the functionality of the Repast ABM software libraries with the geoprocessing environment in ArcGIS. As always, some programming is necessary, but a graphical user interface provides a menu environment within which Python-like code can be structured. The coding environment is sensitive to the context and structure of ArcGIS and allows for easier and more direct access to a GIS database and visualization environment.

Additional information on agent-based modeling

The intent of this chapter is to provide an introduction to the history and philosophy of ABM and how the spatial component has been integrated with agent-based models. See the following references for additional discussions on the background of ABM.

For agent-based modeling in ecology:

- *Individual-Based Models and Approaches in Ecology: Populations, Communities, and Ecosystems* by D. L. DeAngelis and L. J. Gross (1992).
- *Individual-Based Modeling and Ecology* by V. Grimm and S. F. Railsback (2005).
- “Individual-Based Modeling of Ecological and Evolutionary Processes” by D. L. DeAngelis and W. M. Mooij (2005).
- “A Standard Protocol for Describing Individual-Based and Agent-Based Models” by V. Grimm et al. (2006).

For agent-based modeling for land-use and land-cover change:

- “Multi-Agent Systems for the Simulation of Land-Use and Land-Cover Change: A Review” by D. C. Parker et al. (2002).

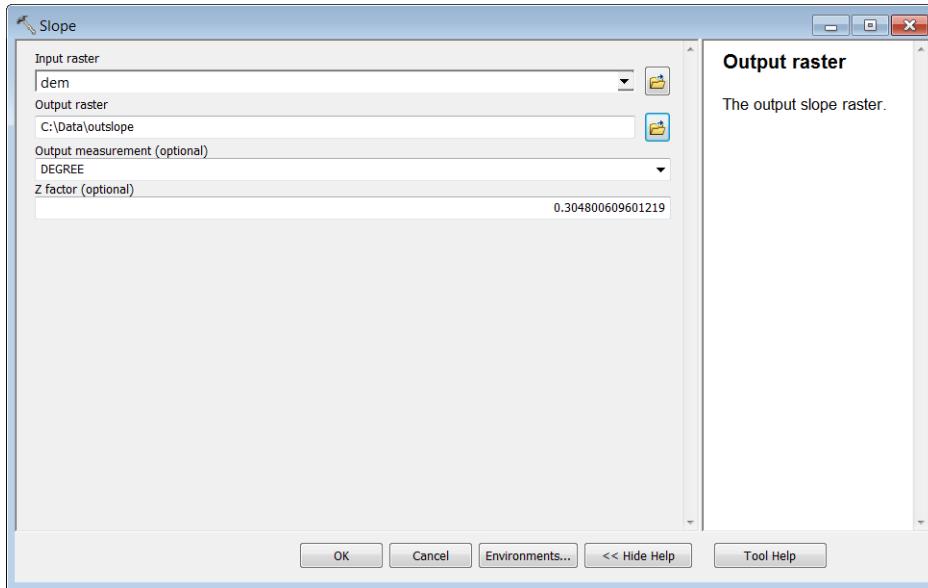
For general agent-based modeling principles:

- *Managing Business Complexity: Discovering Strategic Solutions with Agent-Based Modeling and Simulation* by M. J. North and C. M. Macal (2007).

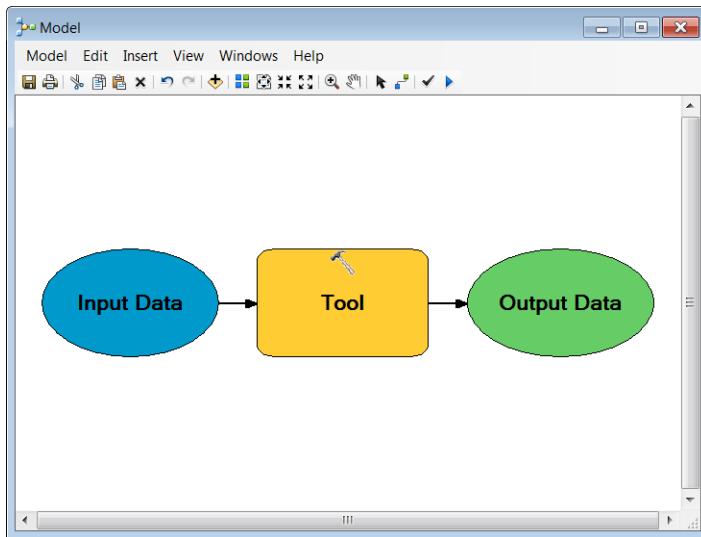
Additional references are listed at the end of this chapter.

What the ArcGIS geoprocessing environment is

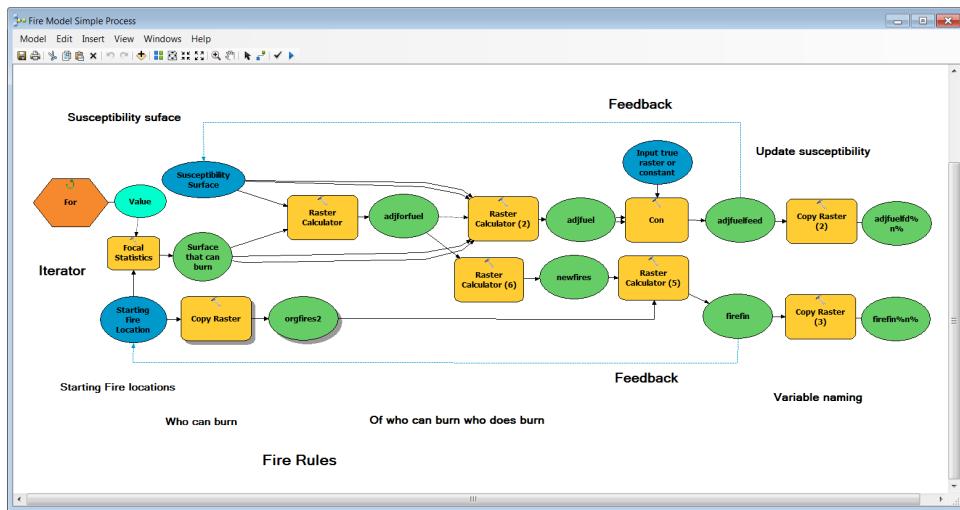
The ArcGIS geoprocessing environment is used for performing spatial analysis, modeling, and automating work flows. The geoprocessing environment is composed of a full suite of tools organized into toolboxes. A tool generally performs an operation or ArcGIS analysis and outputs a new geodataset. A tool can be accessed by dialog box (such as the Slope tool shown in the following figure), by ModelBuilder, or through scripting.



ModelBuilder is a graphical user interface used to create models, providing you another way to implement geoprocessing tools. In ModelBuilder, a process is composed of input data, the tool to apply to the data, and the resulting output data.



The output of one tool can be entered as the input to another, allowing for sequencing of the tools to model spatial relationships. A ModelBuilder model can range from simple to very complex. The following figure shows an example of a simple fire-movement model.



Through the use of iterators, feedback loops, and branching, ModelBuilder allows you to apply a series of tools to base input data—in this case, the current locations of the burning fires and the susceptibility surface defining the potential for each location to burn based on fuel load, slope, and various other factors—and reapply the tools on the resulting data from the previous model run or time step. As a result, you are able to capture and model dynamic processes.

What is Repast?

The Recursive Porous Agent Simulation Toolkit (Repast) is a family of free and open-source toolkits that were developed by Sallach, Collier, North, Howe, Vos, and others (North, Collier, and Vos 2006; North et al. 2007; ROAD 2009). Repast was originally developed with a focus on modeling social behavior, but it is not limited to social simulation. Shortly after its inception, Repast's wide applicability to questions outside the social sciences was recognized, and the toolkit was generalized. Fortunately, the underlying features needed to support social science, such as networks, have proven highly valuable in other application areas.

Users have applied Repast to a wide variety of applications, which range from ecology to evolutionary systems, social systems, and market modeling. For examples from the area of artificial life, see the survey by North and Macal (2009).

Repast was originally created at the University of Chicago in 2000 and was then expanded by Argonne National Laboratory (Samuelson and Macal 2006). Subsequently, responsibility for the ongoing development of Repast was assumed by the Repast Organization for Architecture and Design (ROAD) (ROAD 2009).

There are several separate versions of Repast in the Repast family of toolkits (North, Collier, and Vos 2006; North et al. 2007; ROAD 2009). The Repast 3 group provides a full suite of agent-based modeling capabilities in both the Java and Microsoft .NET environments. Repast 3 for Java is known as Repast J, and Repast for the Microsoft .NET Framework is known as Repast .NET. Repast for Python Scripting (Repast Py) provides an easy-to-use graphical interface for creating Repast J models. Agent Analyst is a specialization of Repast Py that makes the power of Repast 3 available as an Esri ArcGIS model tool. Repast Symphony (North et al. 2007) is the newest member of the Repast family. It offers advanced point-and-click simulation functionality.

Repast 3 is organized as a set of user-callable modules that provide specific simulation services. These modules include a user interface system, a batch run system, a simulation engine, an adaptive behaviors module, and a special domain support module (North, Collier, and Vos 2006). The interface system allows users to monitor and control the progress of simulation runs, create and observe charts, and create log files. The Agent Analyst user interface is synchronized with ArcGIS so that changing agent states within a given model will be correctly shown by the corresponding ArcGIS maps and displays. The batch run system provides users with an automated mechanism to execute multiple model runs. The simulation engine performs the core work of the model, including executing the agent behavior rules and tracking the states of the agents. The adaptive behaviors module provides users with the option to add adaptive tools to their agents, such as neural networks and genetic algorithms. The special domain support module includes features such as Repast's network tools. Agent Analyst makes these modules conveniently available to ArcGIS users. Agent Analyst users may add additional ArcGIS functions or third-party Java libraries to their models. More information on the architecture of Repast is provided in North, Collier, and Vos (2006).

What is Agent Analyst?

Agent Analyst integrates two development platforms through a middleware approach (Brown et al. 2005) between the Repast ABM software and Esri ArcGIS software (in particular, the geoprocessing environment). The extension is free and open source, created by the authors of this book (Collier, Johnston, and North). Repast is used for the creation of the agent rules, object support, and scheduling. ArcGIS is used for data creation, GIS analysis, and display of the simulations. Agent Analyst is implemented as a new model-tool type supported in the ArcGIS geoprocessing environment. As a result, an Agent Analyst model can be used as any model tool in the geoprocessing environment (for example, it can be added to ModelBuilder, parameterized, and documented like any ArcGIS model). With Agent Analyst, the model developer can utilize ArcGIS Java objects, allowing the developer to access the full suite of GIS functions and incorporate them into the agent model.

Agent Analyst takes care of some of the complexity of creating an agent-based model, especially models based on spatial relationships. It automatically creates agents from GIS layers, helps define properties, schedules agents, and displays the results of each time step in the ArcGIS display. There is communication to and from the agent-based modeling software (Repast) and ArcGIS. Agents can query a changing landscape modeled in the GIS and base their decisions on the landscape.

How does an agent-based model work?

In its simplest form, an agent-based model works in the following manner: identify the agents and define their action.

First, the agents need to be identified. Agents can be animals, terrorists, land parcels, delivery trucks, weather patterns, or anything that “makes a decision,” performs an action, or changes state. “Making a decision” is being used in the most general meaning: the agent does “this” as opposed to “that.” A discussion of whether the decision is conscious is beyond the scope of this chapter.

Next, the agents act; they perform an action or no action. An animal agent might run, walk, eat, or sleep. A terrorist might attack. A land parcel might change from agricultural to residential.

An agent’s action or behavior is based on the following:

- Its state, which can be physical or mental or any other measure influencing its decision making.
- Its interactions with other agents.
- Its interactions with the external world.

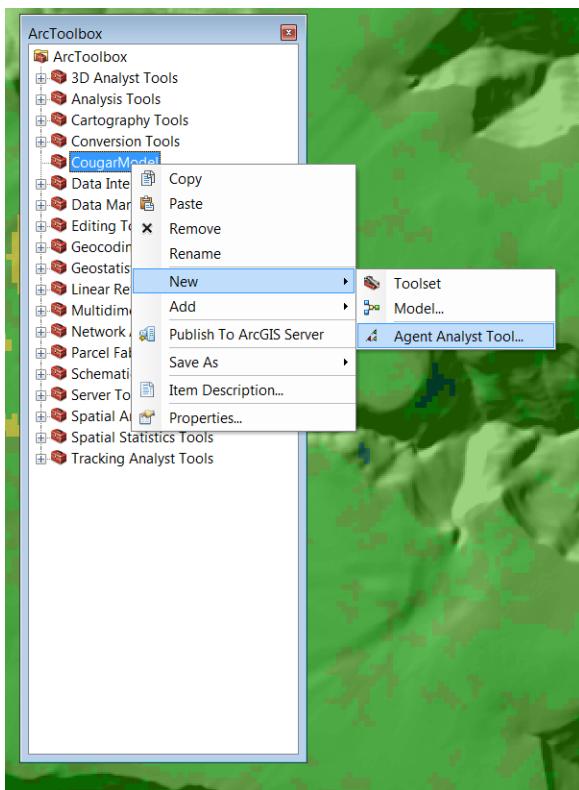
External interactions stem from global factors, prompting a reaction or action by the agent. For example, the prevailing winds may influence the movement of pollution emanating from a smoke stack. There are also environmental landscape factors—for example, a military vehicle slowing down as it ascends a steep slope or crosses swampy terrain. Finally, there are external societal factors such as determining whether a land parcel should change from agriculture to residential based on the economic value of one use relative to the other.

Time is explicit in the simulation of an agent-based model. Each decision is made in some specified time step. The length of each time step is controlled through a scheduler. The duration for the time step should be determined based on the decision making and characteristics of the agents being modeled. For example, if the model is exploring delivery truck movement, the time step will be much smaller (e.g., every hour), as opposed to possibly using a much longer time interval when modeling the change in land-use types based on an economic model.

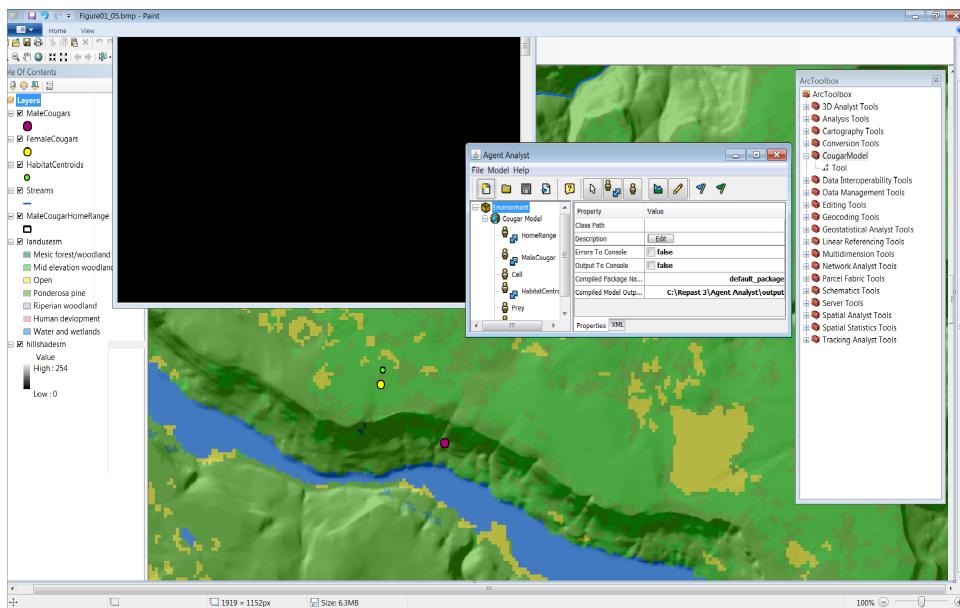
A quick tour of Agent Analyst

The following is a quick tour of how an agent-based model is created in ArcGIS using Agent Analyst. The model presented in this tour is a cougar movement model that is explored in greater depth in chapters 2, 5, and 8. The intent of the tour is to provide you with an introduction to Agent Analyst so that you can learn how to access the extension, see what it looks like, and explore what it can do.

Agent Analyst is an ABM extension that can be used within ArcGIS. An agent-based model is added as a new tool in the ArcGIS geoprocessing environment as shown in the following screen shot.

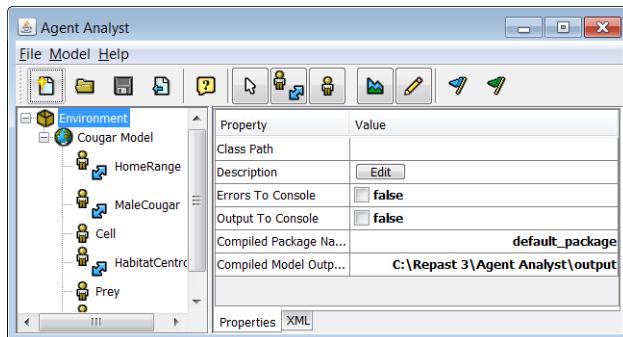


When the Agent Analyst tool is initialized, the Repast/Agent Analyst user interface appears.

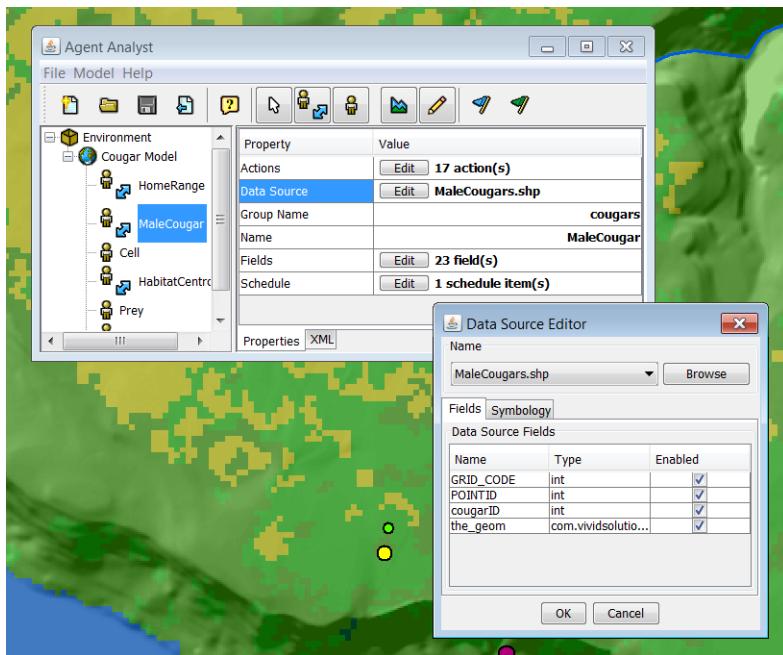


First, the agents must be identified when you create an agent-based model. Defining the agents is the most important step of the modeling process because the agents will dictate all the remaining aspects of the model. The agents need to represent the phenomenon being modeled and contribute uniquely to the interactions within the phenomenon.

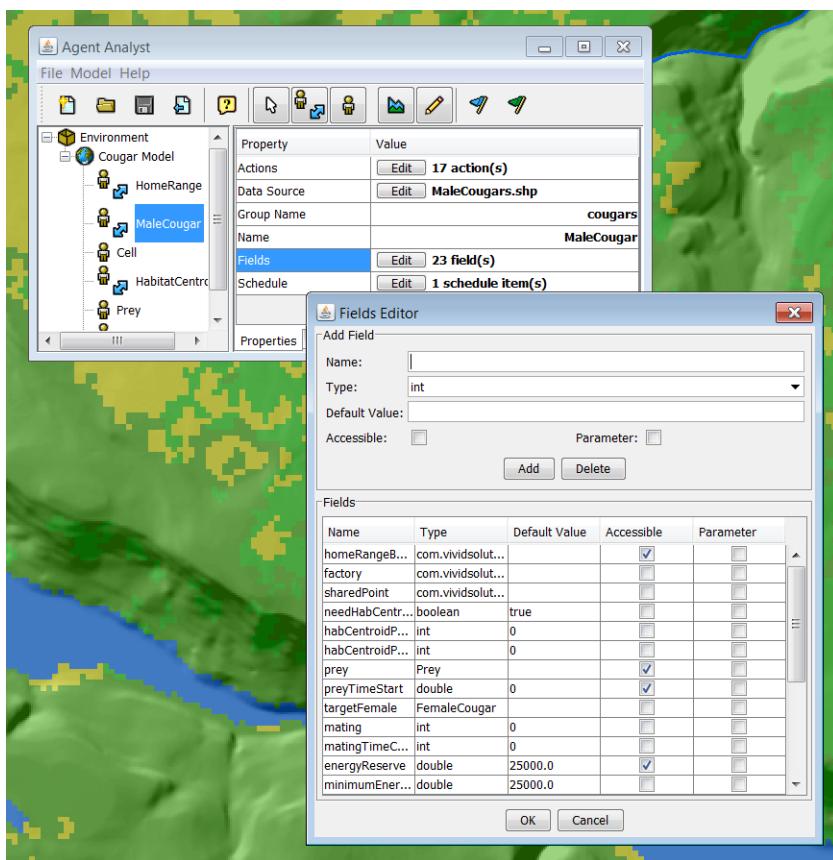
There are two primary agents defined in the cougar model: male cougars and female cougars. Four lesser agents—HomeRange, Cells, HabitatCentroid, and Prey—aid in the decision making of the cougar agents. In the following image, six agents exist in the cougar model: HomeRange, MaleCougar, Cell, HabitatCentroid, Prey, and FemaleCougar. These agents are listed in the Environment panel of the Agent Analyst window. The creation of these agents and their actions will be demonstrated in chapters 2, 5, and 8.



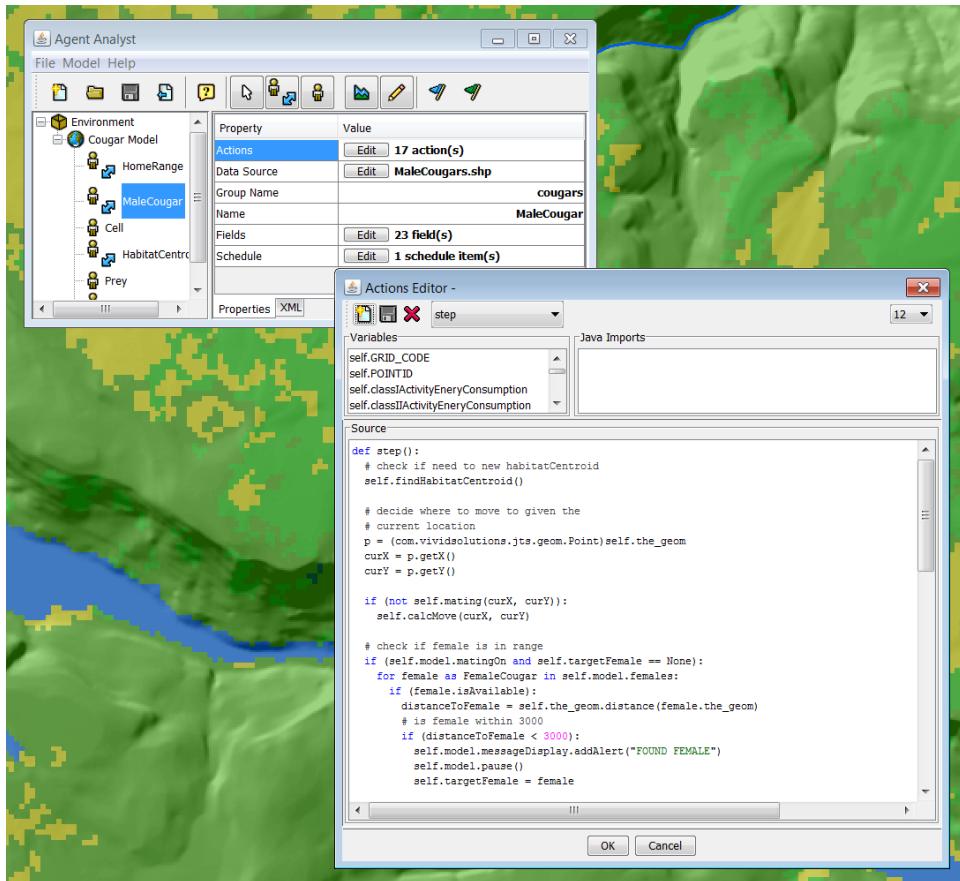
The cougar agents are derived from ArcGIS map layers from which the existing locations of the cougars and the accompanying attributes are automatically associated with the agents using Agent Analyst. The next image shows the Data Source Editor for the male cougar agents. It reflects a cougar GIS shapefile as the data source for male cougar agents, and the fields (the properties) for the agents were derived from the shapefile.



Once the agents are identified, the fields (properties) of the agent are defined. More fields can be created for additional characteristics for those agents derived from a GIS shapefile. The fields or properties are the characteristics that describe the agent. If the agent represents a human, their height, hair color, and gender might be properties. In the case of the cougar model, the home range boundary, the closest x,y habitat point, whether a cougar agent has made a kill, the prey type of the kill, when a cougar started consuming a kill, and the agent's fitness at the current time step are fields (properties) that aid the cougar agent in making the decision of where to move in a time step. The Fields Editor for specifying the fields (properties) for the cougar agents is displayed in the following screen shot.

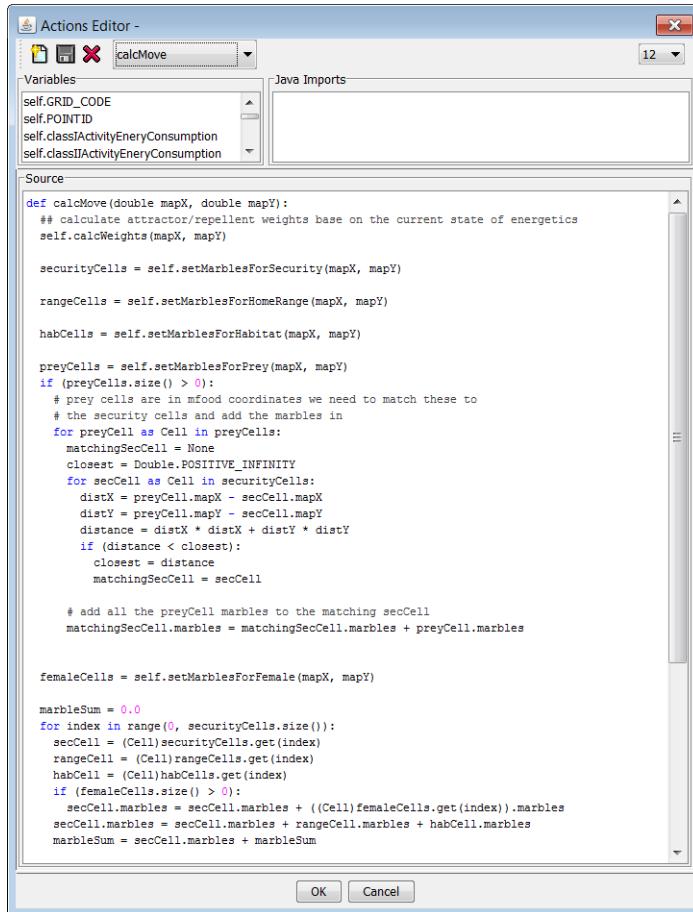


Agents need to perform actions during each time step. Actions must be defined. Clearly, not all possible actions an agent can perform can be realized or coded. Therefore, a subset of all possible actions to address the question being asked needs to be identified. In the next image, the *step* action for the male cougar agent is identified in the Source panel of the Actions Editor. The Actions Editor helps you define the actions for the agent. In each time step of the simulation, the activities defined in the *step* action are performed. As you can see, an action is defined using a Python-like (or Java) code. This coding can be complex, and a good understanding of object-oriented coding is helpful. The exercises in this book will help you develop actions.



The step action for the male cougar agent is called by the model each time step. The action controls how the male cougar agent decides which of the eight surrounding cells it should move into while considering six criteria. The criteria that the male cougar trades off each time step are security, remaining with the kill, pursuing a female cougar, staying within the home range, moving with intent toward good habitat/hunting, and mating. The current energy level of the cougar agent for the time step will dictate how much to weight each criterion relative to one another.

Seventeen actions are defined for the male cougar agent. In the following image, the calcMove action defines how the cougar agent will decide how to move at each time step.



The screenshot shows the Actions Editor window with the title bar "Actions Editor - calcMove". The window has three main sections: Variables, Java Imports, and Source. The Source section contains the following Python code:

```

Variables:
self.GRID_CODE
self.POINTID
self.classActivityEnergyConsumption
self.classIIActivityEnergyConsumption

Java Imports:
None

Source:
def calcMove(double mapX, double mapY):
    ## calculate attractor/repellent weights base on the current state of energetics
    self.calcWeights(mapX, mapY)

    securityCells = self.setMarblesForSecurity(mapX, mapY)

    rangeCells = self.setMarblesForHomeRange(mapX, mapY)

    habCells = self.setMarblesForHabitat(mapX, mapY)

    preyCells = self.setMarblesForPrey(mapX, mapY)
    if (preyCells.size() > 0):
        # prey cells are in mfood coordinates we need to match these to
        # the security cells and add the marbles in
        for preyCell as Cell in preyCells:
            matchingSecCell = None
            closest = Double.POSITIVE_INFINITY
            for secCell as Cell in securityCells:
                distX = preyCell.mapX - secCell.mapX
                distY = preyCell.mapY - secCell.mapY
                distance = distX * distX + distY * distY
                if (distance < closest):
                    closest = distance
                    matchingSecCell = secCell

            # add all the preyCell marbles to the matching secCell
            matchingSecCell.marbles = matchingSecCell.marbles + preyCell.marbles

    femaleCells = self.setMarblesForFemale(mapX, mapY)

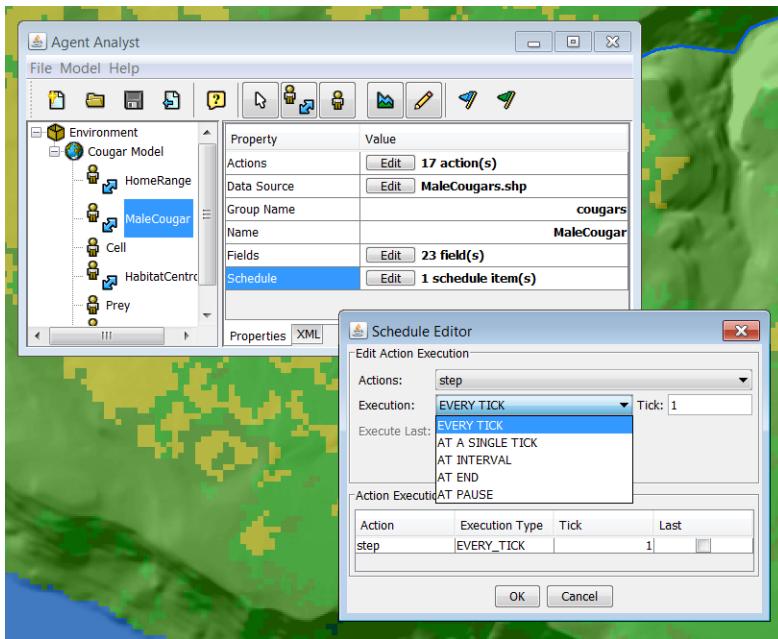
    marbleSum = 0.0
    for index in range(0, securityCells.size()):
        secCell = (Cell)securityCells.get(index)
        rangeCell = (Cell)rangeCells.get(index)
        habCell = (Cell)habCells.get(index)
        if (femaleCells.size() > 0):
            secCell.marbles = secCell.marbles + ((Cell)femaleCells.get(index)).marbles
            secCell.marbles = secCell.marbles + rangeCell.marbles + habCell.marbles
    marbleSum = secCell.marbles + marbleSum

```

After the cougar agent weights each of its eight surrounding cells based on the six criteria mentioned previously and adjusts the weighting based on the importance of each criterion, one of the surrounding cells is selected through the model by the cougar agent from a distribution created by the weighting of the eight cells. The selected cell is the location the cougar agent should move into.

The actions need to be scheduled through the Schedule Editor. The cougar will perform an action (actually a series of actions), and the cougar's fields (properties) will be updated at every

tick or time step. In this model, the step action is the controlling action that calls other actions and is scheduled every tick or time step.



When the model is run, each agent is processed at each time step, and the agent performs the appropriate actions and updates its fields (properties) based on how often the action is to be performed as defined in the Schedule Editor. The model runs for the specified number of time steps or until criteria are met. During ABM simulations, agents can die, new ones are born, and agents can learn, all resulting in changes in actions.

Chapter 2 will introduce you to the basic mechanics of how to create the cougar model presented in this quick tour. You will learn how to define point agents, create fields or properties for the agents, establish actions, schedule the actions, and run the model. In chapters 5 and 8 you will complete the development of the cougar model.

Considerations when creating an agent-based model

Creating an agent-based model can be difficult. This type of model requires an in-depth understanding of the phenomena and generally a great deal of detailed data. The following is a suggested outline defining the series of steps and questions you may ask when formulating your agent-based model. The examples used for each point are relative to the cougar model that you just toured, but the considerations apply to any agent-based model.

1. Define the goal.
 - a. What is the fundamental question the model is intended to answer (e.g., understanding cougar behavior and movement so that human/cougar interaction can be minimized)?
 - b. How will you define a successful model (e.g., do the virtual cougars move in a similar manner to actual cougars)?
 - c. What quantifiable methods will you use to measure the success of the model (e.g., the virtual male cougars mate with a virtual female cougar or eat prey approximately the same percentage of time as actual male cougars)?
2. Define the agents.
 - a. How will you represent each of the agents?
 - i. Points
 - ii. Polygons
 - iii. Raster layers
 - iv. Networks
 - v. Nonspatially
 - b. Will you be using multiple types of agents in the model (e.g., cougars are point agents, while a fire in the model might be implemented using raster layers)?
 - c. Identify the individual agents (e.g., adult male, adult female, and young cougar agents).
 - d. Do you model all the phenomena as agents or in conjunction with other modeling techniques (e.g., the cougars will be represented as agents, but the prey movement will be identified by seasonal distribution maps not through interacting prey agents)?

3. Define the fields or properties for each agent.
 - a. What fields does an individual agent need to monitor in order to influence the agent's decision making (e.g., energetics determines how hungry a cougar is)?
 - b. What fields might influence how an agent interacts with other agents (e.g., overall strength or whether a cougar agent is injured)?
4. Define the actions for the agents.
 - a. What are the actions for each agent (e.g., walking, running, sleeping, or eating)? Note that no action is still an action.
 - b. Will movement be an action in your model?
 - c. How does the action affect the fields (e.g., running reduces total energetics at a faster rate than walking)?
 - d. How does the action affect the environment (e.g., grazing reduces the quantity of grass and nutrition for a location)?
5. Define how the agents will make decisions, either consciously or without conscious cognition. The following may affect the agent's decision making:
 - a. The agent's state, which can be physical or mental or any other characteristic (e.g., how hungry the animal is).
 - b. The agent's interaction with other agents (e.g., an animal may avoid a predator).
 - c. The agent's interaction with the external world:
 - i. The regional environmental factors the agent encounters (e.g., many animals seek shelter due to a storm).
 - ii. The specific environmental landscape factors like elevation, water bodies, or vegetation (e.g., an animal will slow down as it ascends a steep slope).
 - iii. The external societal factors (e.g., an animal might make a suboptimal personal decision to remain accepted by its group).
 - d. Do agents change their decision making by
 - i. seasons (e.g., in the hot summer months the cougar may reduce its activity level);
 - ii. time of day (e.g., the cougar may prefer secure locations in the daylight);
 - iii. under specific conditions (e.g., when the cougar is hungry, the cougar opts to hunt)?
 - e. Does the agent learn (e.g., a young cougar can make mistakes such as coming too close to a road and then learn not to go near it again)?

6. Define where the information will come from for the agents to make a decision.
 - a. Base layers (e.g., what the security level is for each of the eight cells surrounding an agent can be identified from a GIS raster layer).
 - b. Derived layers (e.g., for interaction with other agents, a layer may be created storing their current locations).
 - c. Regional characteristics (e.g., the warmer average daytime temperatures may influence decisions).
 - d. Other models (e.g., a fire movement model can be used each time step to define the current location of a fire for the cougar agents to avoid).
7. Define the time interval for each time step if the model is time stepped. The duration of a time step may be dependent on
 - a. your understanding of the phenomena, which dictates the specified rules (e.g., is your understanding of the phenomena and the resulting rules specified by hour, by day, by week, by month, or by year);
 - b. the decision-making process of the agents (e.g., the cougar may move every minute, but if you are modeling the evolutionary change of the genetics of the cougar, the model will require a longer time step);
 - c. characteristics of the modeled agents (e.g., how far the animal can move in the specified time step);
 - d. the resolution of the data (e.g., if you are using raster data and the cell size is 8 kilometers, modeling cougar movement every 15 minutes would not be possible).
8. Determine the duration of the model run (e.g., a day, a week, a year). The number of iterations depends on
 - a. the phenomena you are modeling;
 - b. what you know of the phenomena;
 - c. what the model is attempting to capture;
 - d. the duration of each time step.
9. Determine how you will verify the model (see North and Macal [2007] for more information on model verification).

10. Determine how you will validate the model (note that validation is never generic and is only relative to a well-defined question) (see North and Macal [2007] for more information on model validation):
 - a. Using measured objectives defined in point 1.
 - b. Using observational data.
 - c. Using knowledge of the phenomena.

Model creation is an iterative process. The goals of the model are defined, the agents created, the fields and actions established, and the model is run. For example, when you are validating the model, if the movement of the virtual agents does not resemble the actual observed movement, you may want to question whether the agents have been correctly defined and whether the right fields and actions have been specified. You may choose to update as necessary and rerun the model. You then need to return to verification and validation.

Modeling, in particular ABM, is a process. The process identifies what you know and do not know about the phenomena.

References

- Axelrod, R. 1984. *The Evolution of Cooperation*. New York: Basic Books.
- Berryman, S. 2004. "Democritus." In *Stanford Encyclopedia of Philosophy*. <http://plato.stanford.edu/entries/democritus/>.
- Bian, L. 2004. "A Conceptual Framework for an Individual-Based Spatially Explicit Epidemiological Model." *Environment and Planning B* 31 (3): 381–95.
- Brown, D. G., R. L. Riolo, D. Robinson, M. North, and W. Rand. 2005. "Spatial Process and Data Models: Toward Integration of Agent-Based Models and GIS." *Journal of Geographical Systems* 7 (1): 1–23.
- Casti, J. L. 1998. *Would-Be Worlds: How Simulation Is Changing the Frontiers of Science*. New York: Wiley.
- Deadman, P., and H. R. Gimblett. 1994. "The Role of Goal-Oriented Autonomous Agents in Modeling People-Environment Interactions in Forest Recreation." *Mathematical and Computer Modelling* 20 (8): 121–33.
- DeAngelis, D. L., and L. J. Gross. 1992. *Individual-Based Models and Approaches in Ecology: Populations, Communities, and Ecosystems*. New York: Chapman and Hall.

- DeAngelis, D. L., and W. M. Mooij. 2005. "Individual-Based Modeling of Ecological and Evolutionary Processes." *Annual Review of Ecology, Evolution, and Systematics* 36: 147–68.
- Epstein J., and R. Axtell. 1996. *Growing Artificial Societies: Social Science from the Bottom Up*. Cambridge, MA: MIT Press.
- Gilbert, N., and S. Bankes. 2002. "Platforms and Methods for Agent-Based Modeling." *Proceedings of the National Academy of Sciences of the United States of America* 99 (Suppl. 3): 7197–98.
- Gilbert, N., and K. G. Troitzsch. 2005. *Simulation for the Social Scientist*. Berkshire, England: Open University Press.
- Gimblett, H. R. 1997. "Simulating Recreation Behaviour in Complex Wilderness Landscapes Using Spatially-Explicit Autonomous Agents." Unpublished dissertation, University of Melbourne.
- , ed. 2002. *Integrating Geographic Information Systems and Agent-Based Modeling Techniques for Simulating Social and Ecological Processes*. New York: Oxford University Press.
- Grimm, V., U. Berger, F. Bastiansen, S. Eliassen, V. Ginot, J. Giske, J. Goss-Custard, T. Grand, S. K. Heinz, G. Huse, A. Huth, J. U. Jepsen, C. Jørgensen, W. M. Mooij, B. Müller, G. Pe'er, C. Piou, S. F. Railsback, A.M. Robbins, M. M. Robbins, E. Rossmanith, N. Rüger, E. Strand, S. Souissi, R. A. Stillman, R. Vabø, U. Visser, D. L. DeAngelis. 2006. "A Standard Protocol for Describing Individual-Based and Agent-Based Models." *Ecological Modelling* 198: 115–26.
- Grimm, V., and S. F. Railsback. 2005. *Individual-Based Modeling and Ecology*. Princeton, NJ: Princeton University Press.
- Groff, E., and L. Mazerolle. 2008. "Simulated Experiments and Their Potential Role in Criminology and Criminal Justice." *Journal of Experimental Criminology* 4: 187–93.
- Guisan, A., and N. E. Zimmermann. 2000. "Predictive Habitat Distribution Models in Ecology." *Ecological Modelling* 135 (2–3): 147–86.
- Hägerstrand, T. 1967. *Innovation Diffusion as a Spatial Process*. Chicago: University of Chicago Press.
- Karssenberg, D. 2002. "Building Dynamic Spatial Environmental Models." Doctor's dissertation, Utrecht University.
- Kohler, T. A., and G. J. Gumerman, eds. 2000. *Dynamics in Human and Primate Societies: Agent-Based Modeling of Social and Spatial Processes*. New York: Oxford University Press.

- Langton, C., N. Minar, and R. Burkhardt. 1995. *The Swarm Simulation System: A Tool for Studying Complex Systems*. Santa Fe, NM: Santa Fe Institute.
- Miller, H. J. 2005. "A Measurement Theory for Time Geography." *Geographical Analysis* 37: 17–45.
- North, M. J., N. T. Collier, and R. J. Vos. 2006. "Experiences Creating Three Implementations of the Repast Agent Modeling Toolkit." *ACM Transactions on Modeling and Computer Simulation* 16 (1): 1–25.
- North, M. J., and C. M. Macal. 2007. *Managing Business Complexity: Discovering Strategic Solutions with Agent-Based Modeling and Simulation*. New York: Oxford University Press.
- . 2009. "Foundations of and Recent Advances in Artificial Life Modeling with Repast 3 and Repast Symphony." In *Artificial Life Models in Software*, 2nd ed., edited by A. Adamatzky and M. Komosinski. Heidelberg, FRG: Springer.
- North, M. J., E. Tatara, N. T. Collier, and J. Ozik. 2007. "Visual Agent-Based Model Development with Repast Symphony." *Proceedings of the Agent 2007 Conference on Complex Interaction and Social Emergence*. Argonne, IL: Argonne National Laboratory.
- Parker, D. C., T. Berger, and S. M. Manson. 2002. *Agent-Based Models of Land-Use and Land-Cover Change*. LUCC Focus 1 Office, Indiana University.
- Parker, D. C., S. M. Manson, M. A. Janssen, M. J. Hoffmann, and P. Deadman. 2003. "Multi-Agent Systems for the Simulation of Land-Use and Land-Cover Change: A Review." *Annals of the Association of American Geographers* 93 (2): 314–37.
- Peuquet, Donna J. 2002. *Representations of Space and Time*. New York: Guilford.
- Pontius Jr., R. G., D. Huffaker, and K. Denman. 2004. "Useful Techniques of Validation for Spatially Explicit Land-Change Models." *Ecological Modelling* 179 (4): 445–61.
- Repast Organization for Architecture and Design (ROAD). 2009. *Repast Home Page*. <http://repast.sourceforge.net/>.
- Samuelson, D. A., and C. M. Macal. 2006. "Agent-Based Modeling Comes of Age." *OR/MS Today* (August).
- Schelling, T. 1971. "Dynamic Models of Segregation." *The Journal of Mathematical Sociology* 1 (2): 143–86.
- Schumpeter, J. 1909. "On the Concept of Social Value." *Quarterly Journal of Economics* 23: 213–32.

- Simon, H. 1957. "A Behavioral Model of Rational Choice." In *Models of Man, Social and Rational: Mathematical Essays on Rational Human Behavior in a Social Setting*. New York: Wiley.
- Van Huyssteen, W. 2003. "Downward Causation." In *Encyclopedia of Science and Religion*. New York: MacMillan.
- Westervelt, J. D., and L. B. Hopkins. 1999. "Modeling Mobile Individuals in Dynamic Landscapes." *International Journal of Geographic Information Systems* 13 (3): 191–208.
- Wilensky, U., and W. Rand. In Press. *An Introduction to Agent-Based Modeling: Modeling Natural, Social, and Engineered Complex Systems with NetLogo*. Cambridge, MA: MIT Press.
- Wolfram, S. 2002. *A New Kind of Science*. Champaign, IL: Wolfram Media.

Section 2: The basics of points, polygons, and rasters in agent-based modeling

Chapter 2

Creating an agent-based model using point agents

by Nicholson Collier, Kevin M. Johnston,
and Arika Ligmann-Zielinska

- ◆ Installing Agent Analyst and exercise data
- ◆ Exploring the capabilities of Agent Analyst
- ◆ Importing, exploring, and running an existing agent-based model
- ◆ Creating a new agent-based model: defining agents, properties, and actions, and then scheduling and running the model
- ◆ Initializing agents with a starting state
- ◆ Updating point agent locations in Agent Analyst and watching them dynamically move in ArcMap
- ◆ Creating additional fields or properties and moving the agents with greater sophistication
- ◆ Accessing raster data and working with graphs
- ◆ Tracking the path of the moving point agent
- ◆ Building complex agents from multiple agents
- ◆ Developing interaction between agents

This chapter introduces you to point agents and the basics of agent-based modeling (ABM) in Agent Analyst. Point agents are the easiest to understand in the ABM environment, and therefore they are the best way to introduce you to Agent Analyst.

Examples of models using points as agents include the following:

- Understanding wildlife movement to establish corridors
- Predicting possible terrorist activity
- Estimating potential customers for a store
- Tracking cells in the body to understand cellular interactions
- Altering parcels' land-use type with economic changes
- Tracking the spread of an infectious disease
- Monitoring airplanes for air traffic control
- Following migrating neotropical birds
- Predicting pirates' actions

For this chapter, a simple animal movement model of cougars was selected to introduce you to the basics of building an agent-based model. The quick tour presented in chapter 1 provides the overall context for the exercises in this chapter.

Three chapters in this book are devoted to creating the cougar model. This chapter develops the fundamentals of creating an agent-based model with point agents. Chapter 5 will demonstrate how the cougar balances the trade-offs in multicriteria decision making. Finally, chapter 8 completes the model development by integrating an energetics model that is used to weigh the criteria. With the development of each chapter, more cougar biology is captured.

The intent of this chapter is to introduce you to ABM principles rather than to create a cougar model. If the rules are not satisfactory, they can be changed.

The modeling scenario

Flagstaff, Arizona, is experiencing problems with human population growth and the resulting reduction and fragmentation of cougar habitat. There are increasing encounters between humans and cougars. The objective of the model is to understand the cougars' movements in order to reduce human/cougar conflict. ABM is particularly appropriate here; it will explore what motivates individual cougars to behave the way they do. With this information, a landscape manager can reduce human access (e.g., limit the number of trails) in locations the cougars need for their survival. Through this causal understanding, the manager can explore different management scenarios to determine which scenario will potentially result in fewer encounters.

2

3

4

The model simulates animal behavior on a landscape represented in a geographic information system (GIS). Specifically, the model focuses on cougar movement and uses surrogates for human activity (e.g., locations of buildings and roads).

The cougar movement rules are derived from data gathered from collared cougars and through consultation with domain experts. The model integrates empirical data from the collared cougars, field observations, and expert opinion with spatial data. This data provides input for the decision rules to produce a spatially explicit agent-based model.

Installing Agent Analyst and exercise data

The following exercises will provide you with the basic principles of how to create an agent-based model using the Agent Analyst environment. You will learn how to install, start, and run Agent Analyst. You will also learn the basic rules of creating a simple agent-based model, which can later serve as a foundation for building more complex models.

Exercise 2a

Agent Analyst is free software that works in conjunction with ArcGIS Desktop software. To complete the exercises in this book you need to have ArcGIS Desktop software installed on your computer.

Download and install Agent Analyst software

- 1 Download Agent Analyst from <http://resources.arcgis.com/en/help/agent-analyst/index.html>.

The download is an executable file that runs the Agent Analyst setup wizard.

- 2 Start the executable and, when prompted, choose an installation location.
- 3 Click Install, and then click Finish on the final page of the wizard.

Download and install the data and the models

- 4 Download the data and models from <http://resources.arcgis.com/en/help/agent-analyst/index.html>.

Each chapter's data and models are contained in an executable file. You can install just the one for the current chapter, or you can install them all now.

- 5 Find the data file for chapter 2 and run it.

The chapter install wizard starts and presents the welcome screen.

- 6 Click Next, and then accept the license agreement.

The Destination folder page appears.

- 7 Accept the suggested folder, the top level of your C drive, and then click Next.

Note: The data, models, toolboxes, and ArcMap documents (.mxd files) must be installed at the top level on the C drive (C:). The explicit path is critical within the Agent Analyst models. As a result of installing these items at the top level on the C drive, an ESRIPress directory is created, and within an AgentAnalyst subdirectory, the data, models, toolboxes, and ArcMap documents will be stored in separate directories based on the chapter number (for example, C:/ESRIPress/AgentAnalyst/Chapter02).

- 8 Close the wizard when installation is complete.

Note: The data files can be large and might take a few minutes to install.

2

3

4

Exploring the capabilities of Agent Analyst

Now that you have installed Agent Analyst, you will access it.

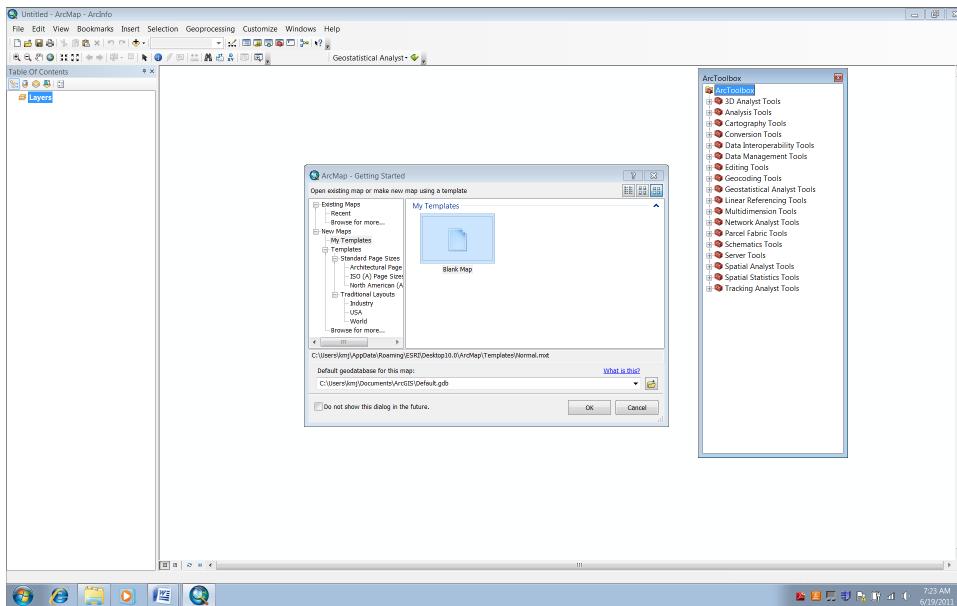
Exercise 2b

In this exercise, you will explore a variety of the Agent Analyst menu options and command buttons to gain familiarity with the software.

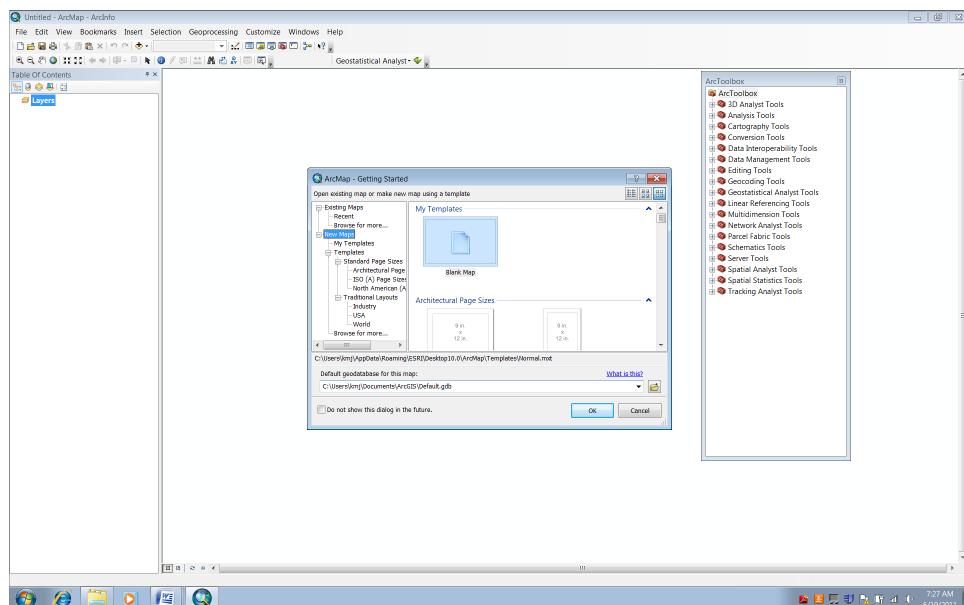
Since Agent Analyst is integrated with the geoprocessing environment, you must first start ArcMap and display ArcToolbox to access the geoprocessing environment.

Start ArcMap and explore Agent Analyst

- 1 Start ArcMap by clicking the Start button on the Windows taskbar, pointing to All Programs, clicking ArcGIS, and clicking ArcMap 10. The ArcMap—Getting Started dialog box appears.



- 2 In the ArcMap—Getting Started dialog box, click New Maps, click Blank Map, and then click OK.



- 3** If the ArcToolbox window is not displayed, on the Standard toolbar, click the ArcToolbox Window button. The ArcToolbox window opens between the Table Of Contents and the map display, or wherever it was last displayed.

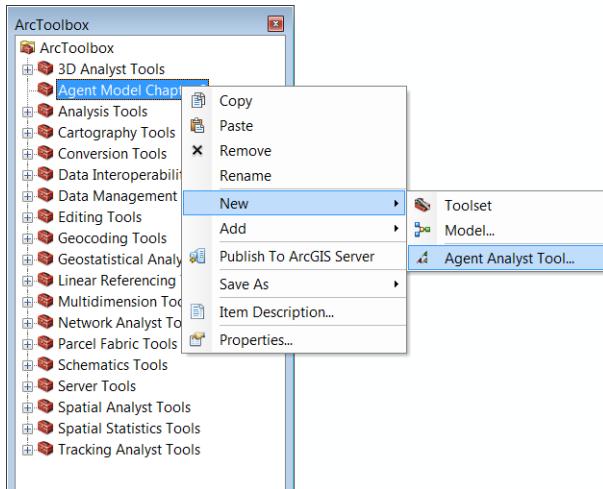
An Agent Analyst model is a new model type in the geoprocessing environment, accessed through ArcToolbox. To create a new model you first define a new toolbox and then create a new Agent Analyst model within the new toolbox.

- 4** To create a new toolbox, right-click ArcToolbox and click Add Toolbox. In the Add Toolbox dialog box, navigate to C:\ESRIPress\AgentAnalyst\Chapter02\MyData and click the New Toolbox icon.
- 5** Rename the toolbox **Agent Model Chapter 2.tbx**. Deselect the toolbox and reselect it. Click Open.
- 6** Right-click the Agent Model Chapter 2 toolbox, point to New, and then click Agent Analyst Tool.

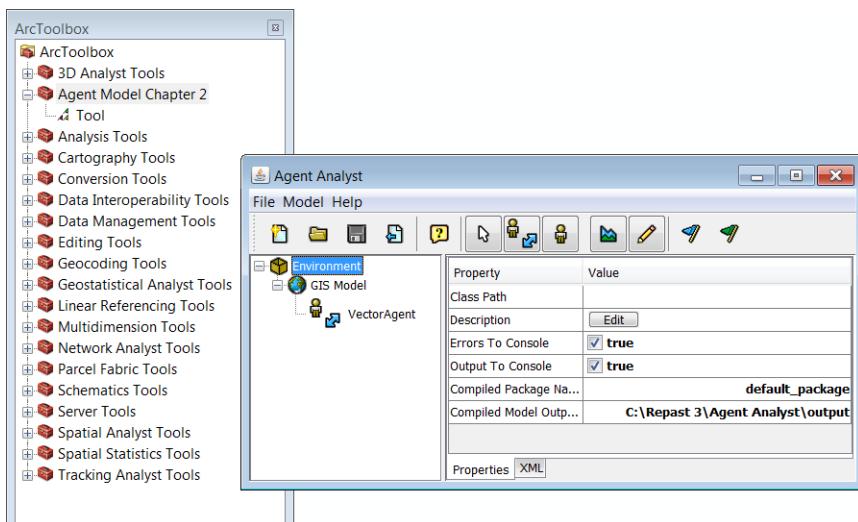
2

3

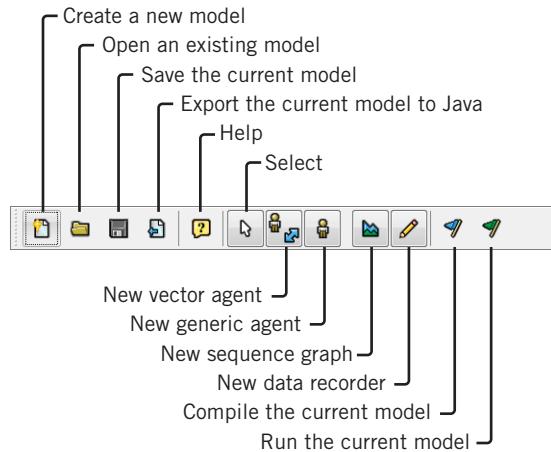
4



The Agent Analyst user interface appears.



Agent Analyst has three menus: File, Model, and Help. It also has a toolbar (shown in the following figure) with buttons for standard operations (such as New Model, Open Model, Save, or Help), different model components (like Vector Agent and Generic Agent), and buttons for compiling and running models.



Two panels are included beneath the toolbar. The left panel, called the Environment panel, displays all components of the current model, such as the GIS model, the vector agent, and the generic agent. The right panel, called the Property panel, displays various properties of the selected component.

In the following steps, you will explore some of the components found in the Agent Analyst Environment and Property panels.

- 7** In the Environment panel, click Environment.

When you click Environment, note that the Property panel updates to reflect the environment settings of the current project. For example, you may add a description of the model by clicking the Edit button to the right of the Description property. The Description Editor window appears, in which you can add the text.

- 8** Click GIS Model in the Environment panel.

The general model properties are displayed. This component comprises all the elements that pertain to the model as a whole. Apart from a series of properties, such as Model Name, Display Name, and the GIS package used for display, there are special-purpose properties that contribute to the model behavior, especially Fields, Schedule, and Actions.

- 9 Click the Generic Agent button on the toolbar. Now click GIS Model in the Environment panel.

You just added a generic agent template to your model. Examine the vector agent and the generic agent properties in the Property panel by clicking their respective buttons in the Environment panel.

Explore the remaining components in the Environment and Property panels by clicking the appropriate buttons. You will not be saving the results.

- 10 Close Agent Analyst by clicking the Close button (the X) in the top right of the Agent Analyst window. When prompted to save the model, click No.
- 11 Close ArcMap. If prompted to save changes, click No.

Importing, exploring, and running an existing agent-based model

2

3

4

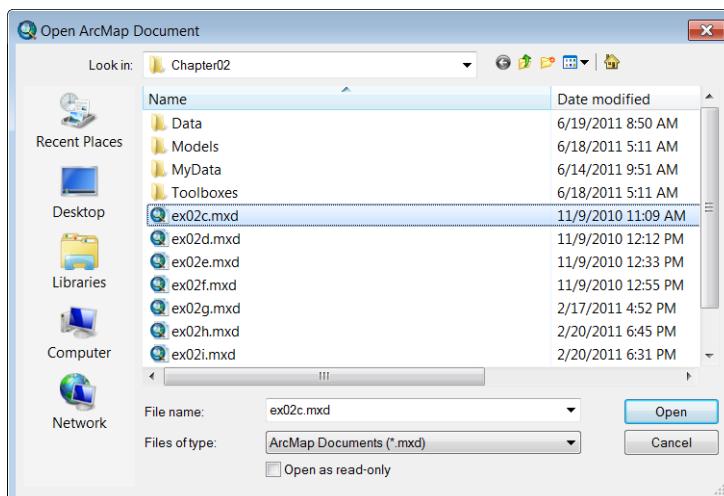
Thus far you have installed the Agent Analyst extension and you have become familiar with the modeling environment. Now it is time to work with an actual model.

Exercise 2c

In this exercise, you will import an existing model from the data you downloaded for this book. You will first examine the model and then run it.

Load the ArcMap document and the Agent Analyst model

- 1 Start ArcMap by clicking the Start button on the Windows taskbar, pointing to All Programs, clicking ArcGIS, and clicking ArcMap 10.
- 2 In the ArcMap—Getting Started dialog box, under the Existing Maps section, click “Browse for more...”. If ArcMap is already running and you don’t see the Getting Started dialog box, click the File menu and then click Open.
- 3 In the Open ArcMap Document dialog box, navigate to C:\ESRIPress\AgentAnalyst\Chapter02. Select ex02c.mxd, and then click Open.



The map document opens. In the ArcMap display, you will see purple dots representing cougar locations. In ArcToolbox, the Chapter02Ex02c toolbox has been created for you.

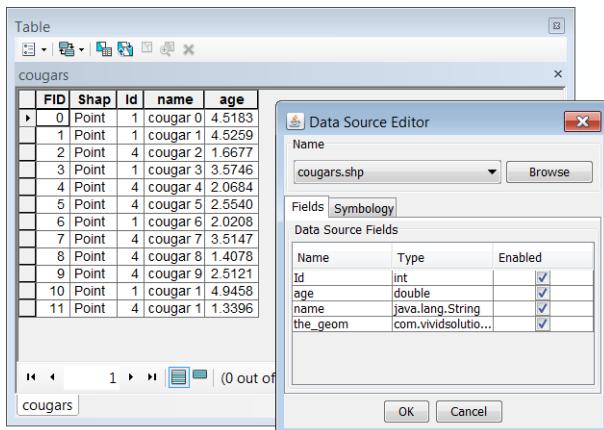
- 4 Add a new Agent Analyst tool to the Chapter02Ex02c toolbox. (Reminder: in ArcToolbox right-click the Chapter02Ex02c toolbox, point to New, and then click Agent Analyst Tool.)

You have now added a new model to the Chapter02Ex02c toolbox, and the Agent Analyst window is open. For the purposes of this exercise, instead of adding to the new model, you will open an existing model.

- 5 In the Agent Analyst window, click File and select Import. Do not save the changes when you're queried.
- 6 Navigate to C:\ESRIPress\AgentAnalyst\Chapter02\Models, select ex02c.sbp, and click Open.

The agents for this model were defined from an existing shapefile called cougars.shp (currently displayed in the ArcMap window), developed in ArcGIS. Cougars.shp is a point shapefile, with each point representing the location of a cougar. There are 12 individual cougars in the shapefile. The name of the cougar and its age are stored as attributes for each point in the shapefile. Using Agent Analyst tools, each point in the shapefile was transformed into a cougar agent. The name, age, and cougar location (“the_geom”) were carried as fields or properties for each cougar agents. Let’s examine the cougar agents.

- 7 In the Table Of Contents pane of ArcMap, right-click cougars and then click Open Attribute Table. In the Agent Analyst Environment panel, click Cougar. In the Property panel, click the Edit button to the right of the Data Source property. The attributes of the shapefile become the properties of the vector agents, as seen in the Data Source Editor.



Notice that the cougar agents were created from the cougars.shp file located in the chapter 2 data folder (C:\ESRIPress\AgentAnalyst\Chapter02\Data). Each of the fields in the shapefile table becomes a field or property of the cougar agents.

8 Close the editor.

Define actions

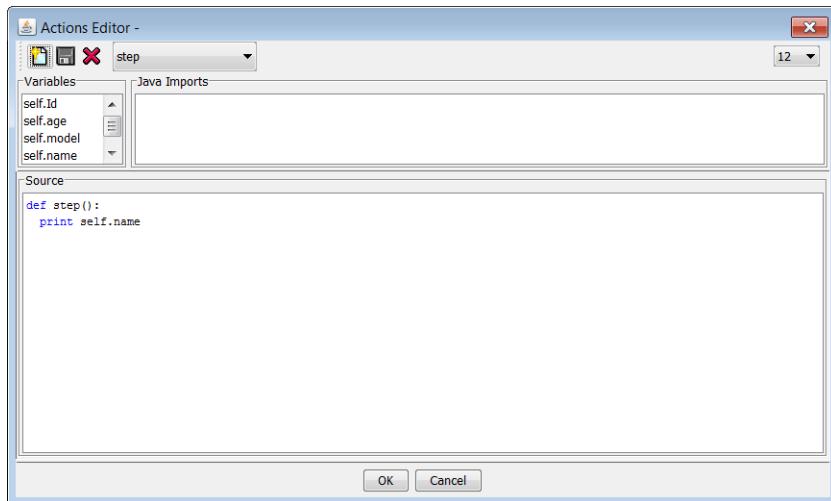
2

Actions are what the agent can perform during each time step. To view and create actions for the corresponding agent, select the agent in the Environment panel. The actions for that agent are identified in the Actions component of the Property panel. You will now explore some of the actions that have already been created for each cougar agent.

3

- 9** In the Environment panel, click Cougar. In the Property panel, click the Edit button to the right of the Actions property. The Actions Editor window appears.

4



Notice that there is an action defined as “step” that prints the name of the cougar agent.

The actions you create in this chapter are simplified examples to introduce you to the basic concepts of agent-based modeling. More realistic actions will be added in chapters 5 and 8.

The step action is applied to each of the 12 cougar agents in the model (processed in random order). The cougar that is being processed is referred to as *self*. To access the variable or property of the agent being processed, use *self* followed by a period and the property name (*self.propertyname*). The statement `print self.name` will print the name of the cougar agent being processed to the RePast Output window. Since each cougar agent is processed each time step, this model will print the names of each cougar each time the step action is called.

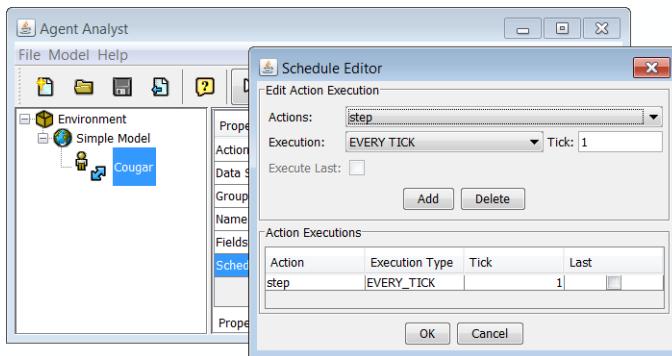
Note that the variables (properties) available for the cougar agent are located in the Variables list in the top left of the Actions Editor. These variables were imported from the fields in the cougar shapefile as discussed previously.

- 10** Close the Actions Editor.

Schedule actions

The Scheduler controls which actions will be performed and when. Each agent type and its actions can be scheduled for different time steps. You will examine the actions scheduled for the cougar agents in the following step.

- 11** In the Environment panel, click Cougar. In the Property panel, click the Edit button to the right of the Schedule property. The Schedule Editor window appears.



Notice that only the step action is scheduled to be performed at EVERY TICK. A tick equates to a time step.

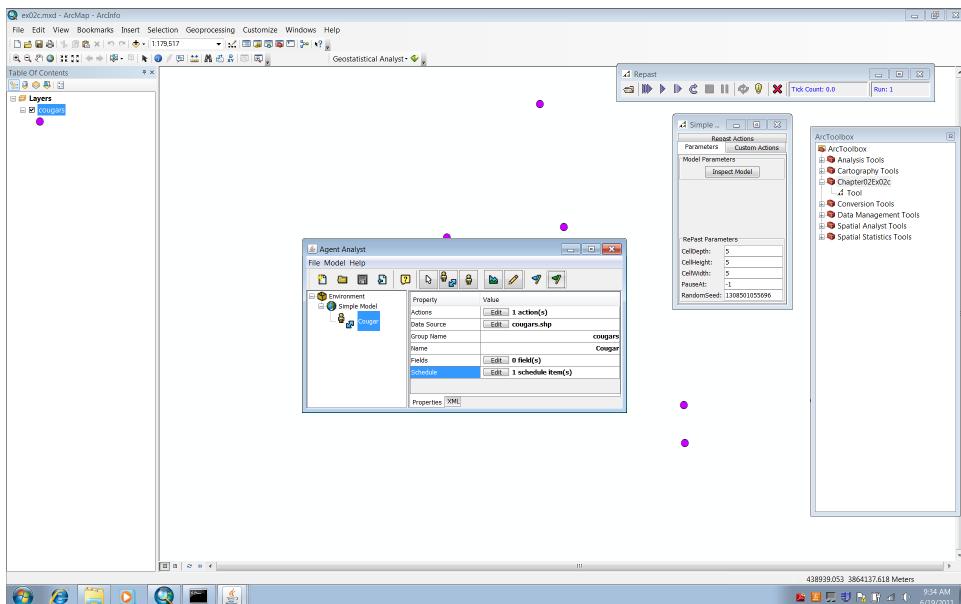
- 12** Click OK to close the window.

Because the step action is scheduled for every tick, each cougar name will be printed each time step.

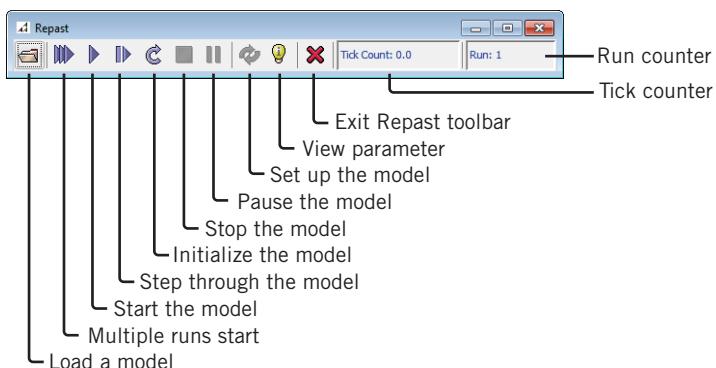
Run the model

- 13** To start the model, on the Agent Analyst toolbar, click the Run button.

The Repast toolbar appears with the model parameters window, labeled Simple Model Settings.



The Repast toolbar (shown in the following figure) allows you to control the running of the model defined in the Agent Analyst main menu. You can start it, step through it, pause it, stop it, and so forth.



Running the model in Agent Analyst is a two-step process. First, you click the Run button on the Agent Analyst toolbar to initiate the model preparation windows (the Repast toolbar and the Simple Model Settings window). Second, to run the model defined by these preparation windows, you click the Start button on the Repast toolbar.

The Parameters tab of the Simple Model Settings window displays the input values of variables exposed by the model developer. The user of the model can change these parameters by entering new values. It is useful to expose variables to make it easy to run different scenarios of the model. For example, when water is plentiful for a specific year, you may want to lower the weight applied to the need for water for the cougar for that year. No parameters were set for this model as seen on the Parameters tab of the Simple Model Settings window. Repast-specific parameters are identified.

Now you will run the model.

- 14 On the Repast toolbar, click the Start button.

As expected, from the step action, `print self.name`, you will see the cougars' names listed in a pop-up window with their numeric identifier.

- 15 Stop the model by clicking the Stop button. Close the Repast toolbar by clicking Exit.
- 16 Close Agent Analyst by clicking File and then Exit without saving changes.

Creating a new agent-based model: defining agents, properties, and actions, and then scheduling and running the model

2

3

4

So far you have loaded and run an existing model. In this exercise, you will build your own model. Using a simple model structure, you will create agents, define fields, develop actions, schedule the actions, and run the model.

In each of the remaining exercises in this chapter, you will build on this single model. Each exercise adds a new component to the agent-based model. You can either continue adding to the model in subsequent exercises, or you can import the specified model for that exercise located in the Models folder (C:\ESRIPress\AgentAnalyst\Chapter02\Models\), with the model's name being identified at the beginning of each exercise.

Exercise 2d

In this exercise, you will first open an existing ArcGIS document. As you did in previous exercises, you will add a new Agent Analyst model to an existing toolbox (e.g., the Chapter02Ex02d toolbox). Then you will create an empty new model.

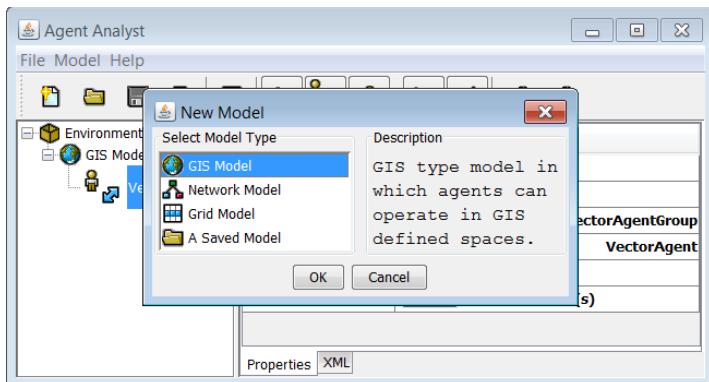
Load the ArcMap document

- 1 Start ArcMap. In the Getting Started dialog box, under the Existing Maps section, click “Browse for more...”. If ArcMap is already running, click the File menu and then click Open. Navigate to C:\ESRIPress\AgentAnalyst\Chapter02. Select ex02d.mxd, and then click Open.

In the ArcMap display, you will see purple dots representing cougar locations. In ArcToolbox, the Chapter02Ex02d toolbox has been created for you.

Create a new Agent Analyst model

- 2 Add a new Agent Analyst tool to the Chapter02Ex02d toolbox. (Reminder: in ArcToolbox right-click the Chapter02Ex02d toolbox, point to New, and click Agent Analyst Tool). If Agent Analyst does not open automatically, click the new tool and select Edit, and Agent Analyst will appear.
- 3 In Agent Analyst, create a new model by clicking File and then clicking New. Do not export the current model when prompted. Select the GIS Model type. Click OK.



The new model has no agents. In the following steps, you will create cougar agents. Cougars are point agents, and they will be created as vector agents. You will name these cougar agents and rename the entire model.

- 4 On the Agent Analyst toolbar, click Vector Agent to create vector agents. In the Environment panel, click GIS Model.

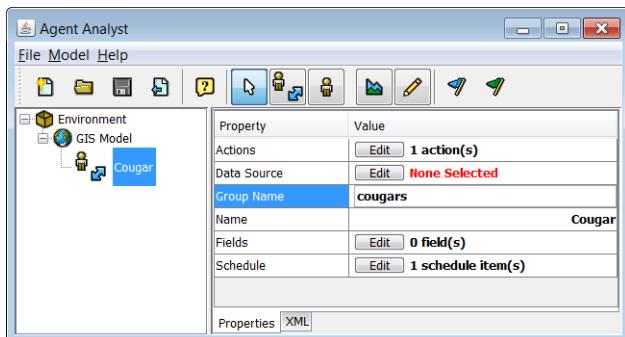
Rename the agents

- 5 In the Environment panel, the new vector agent is selected. In the Property panel, click the Value box to the right of the Name property. Type **Cougar**, and then press Enter.

Define the agent group for the collection of cougars

- 6 In the Property panel, click the Value box to the right of the Group Name property, and rename vectorAgentGroup to **cougars**.

An agent group is a collection of agents that all share the same fields and follow the same rules through actions. The properties, actions, fields, and schedule for each agent group are set in the Property panel.



Rename the model

- 7 In the Environment panel, click GIS Model and change the value of the Display Name property to **Simple Model**. Also, make sure that the value of the GIS Package property is set to ArcGIS.

You have established the definition for cougar agents. In the next step you will create the agents. You will derive the cougar agents from an existing ArcGIS shapefile and associate them with the newly created cougar agents in Agent Analyst. To establish this relationship, you need to identify the data source as the shapefile for your cougar agents.

- 8 In the Environment panel, click Cougar. In the Property panel, click the Edit button to the right of the Data Source property. Click Browse, navigate to C:\ESRIPress\AgentAnalyst\Chapter02\Data, and select cougars.shp. Click Open.

Notice that the Data Source fields are automatically populated with the attributes from the cougar shapefile. These fields can be used as properties for the cougar agents.

- 9 Click OK.

You have created a collection of cougar agents, but they do nothing. In the following steps you will create an action for the cougar agents.

Create an action for the cougar agents

- 10 In the Environment panel, click Cougar. In the Property panel, click the Edit button to the right of the Actions property. The Actions Editor appears. Note that the step action is automatically created.
- 11 Click the step action in the Actions Editor, and in the Source panel type the following code:

```
print self.name
```

- 12 Click OK.

You must click OK to save what you typed in the Source panel.

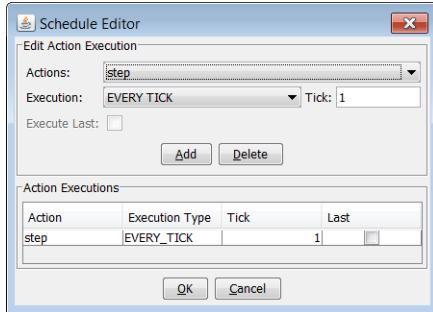
You have just programmed each cougar to print its name to the RePast Output window. But you have not specified when each cougar agent should do so. You will use the Scheduler to establish the timing of the action.

- 13 In the Environment panel, click Cougar. In the Property panel, click the Edit button to the right of the Schedule property. In the Schedule Editor window, notice that the step action is automatically set to EVERY TICK. Click OK.

2

3

4



Run your new model.

- 14 On the main Agent Analyst toolbar, click Run. The Repast toolbar appears. Click Start on the Repast toolbar.

As before, you will see the cougars' names listed with their numeric identifiers.

Note: When you run this particular model on certain operating systems, you might see the Simple Model Display window appear. You can ignore the window or minimize it. With these operating systems it is generally not recommended to create an agent model using the New option from the Agent Analyst menu. Instead, simply alter the default model that is created from the toolbox.

- 15 Stop the model by clicking the Stop button on the Repast toolbar. Close the Repast toolbar by clicking Exit.
- 16 Close Agent Analyst by clicking File, Exit. Save the model in the C:\ESRIPress\AgentAnalyst\Chapter02\MyData folder as Chapter02MyModel4.sbp.

What is NQPy?

NQPy stands for Not Quite Python, which is a small subset of the Python programming language used in Agent Analyst. It is the primary language for scripting agent behavior. NQPy borrows Python's syntax in order to simplify the programming of agent actions. This NQPy code is then compiled as Java.

NQPy uses the basics of the Python syntax. Comments, for and while loops, the range function, the definition of variables, dot notation, and if-elif-else are all represented in NQPy with the Python syntax. Also, the Python indentation rule used for blocks of code is followed in NQPy.

However, more advanced Python functionality is not available—you must use Java coding instead. For example, to define a list you need to use the Java ArrayList class:

```
newList = ArrayList()
newList.add("first")
newList.add("second")
newList.add("third")
print newList[1:]
```

The last line will print “second” and “third” to the RePast Output window. Notice that Python slicing (retrieving part of the list) is present in NQPy.

Although NQPy provides a relatively easy way to specify agent behavior, there are several points to consider when writing NQPy code. These considerations include the following:

- Users cannot define stand-alone classes or functions with NQPy. Agent Analyst models may be exported as a RepastJ model and then further developed with pure Java code using a Java integrated development environment (IDE) such as Eclipse for an advanced model development.
- Python lists, dictionaries, and tuples are not supported.
- Python module imports are not supported. Because NQPy is compiled as Java, Java import statements may be used by entering them in the Java Imports box in the Actions Editor. This allows for the import of ArcObjects Java classes, which will be discussed more in later chapters.

For more information on NQPy, please see the Agent Analyst help document by clicking Help on the Agent Analyst toolbar and then scrolling to Help Topics. The appendix provides further details on the specifics of NQPy. For more information on Python, see <http://www.python.org/>.

2

3

4

Initializing agents with a starting state

In the previous exercises, you ran your model with the properties for the cougar agents being specified prior to the model run. However, you may want to initialize your agents with some starting property values that are randomly (e.g., randomly assigning the age) or deterministically (e.g., assigning names from a table) generated. Or you may want to perform other preparations before each model run (e.g., positioning the agents at specific starting locations).

Exercise 2e

In this exercise, you will learn how to initialize agents with a defined starting status.

Load the ArcMap document

- 1 Navigate to C:\ESRIPress\AgentAnalyst\Chapter02, and then double-click ex02e.mxd.

In the ArcMap display you will see purple dots representing cougar locations. In the ArcToolbox window, the Chapter02Ex02e toolbox has been created for you.

Note: You can continue with the model from the previous exercise (skip steps 2 through 4).

Load the Agent Analyst model

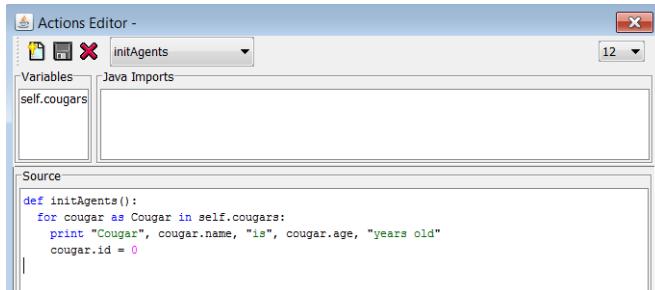
- 2 Right-click the Chapter02Ex02e toolbox, point to New, and click Agent Analyst Tool.
- 3 Once the Agent Analyst window appears, click File and select Import. Do not save the changes when prompted.
- 4 Navigate to C:\ESRIPress\AgentAnalyst\Chapter02\Models and open ex02e.sbp. The Agent Analyst window appears.

Initialize the agents

- 5 In the Environment panel, click Simple Model. In the Property panel, click the Edit button to the right of the Actions property.
- 6 In the Actions Editor, in the drop-down list located above Java Imports, select initAgents. Enter the following code in the Source panel:

```
def initAgents():
    for cougar as Cougar in self.cougars:
        print "Cougar", cougar.name, "is", cougar.age, "years old"
        cougar.id = 0
```

The Source panel code should look like the code in the following image.



The Actions Editor window displays the `initAgents` action. The code prints each cougar's name and age, and sets its ID to 0. The code is:

```
def initAgents():
    for cougar as Cougar in self.cougars:
        print "Cougar", cougar.name, "is", cougar.age, "years old"
        cougar.id = 0
```

2

3

4

The `initAgents` action can initialize the agents with any specified status or can establish other characteristics, producing the starting state from which the model runs. The `initAgents` action in this exercise for each cougar agent simply prints the starting status of the agent (`print "Cougar", cougar.name, "is", cougar.age, "years old"`) and sets the Agent's ID to a default value of 0 (`cougar.id = 0`).

- 7 Click OK to save the action and close the Actions Editor.

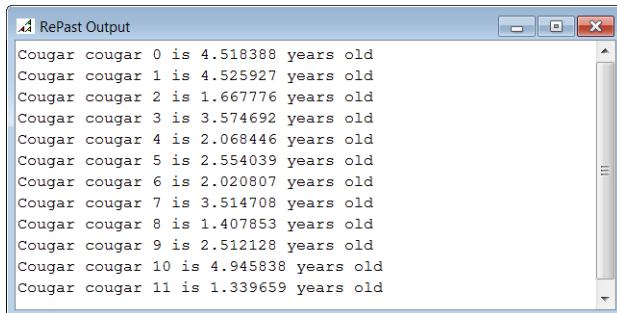
Initialize the model

- 8 Open the Repast toolbar by clicking the Run button in the Agent Analyst window.

The Repast toolbar appears along with the Simple Model Settings window.

- 9 On the Repast toolbar, click the Initialize button.

You see the cougars' names and their ages printed to the RePast Output window during the initialization process of the agents.



The RePast Output window shows the following log output:

```
Cougar cougar 0 is 4.518388 years old
Cougar cougar 1 is 4.525927 years old
Cougar cougar 2 is 1.667776 years old
Cougar cougar 3 is 3.574692 years old
Cougar cougar 4 is 2.068446 years old
Cougar cougar 5 is 2.554039 years old
Cougar cougar 6 is 2.020807 years old
Cougar cougar 7 is 3.514708 years old
Cougar cougar 8 is 1.407853 years old
Cougar cougar 9 is 2.512128 years old
Cougar cougar 10 is 4.945838 years old
Cougar cougar 11 is 1.339659 years old
```

- 10 Close the Repast toolbar. You can close Agent Analyst by clicking File, Exit. Save the model as Chapter2MyModel5.sbp in the C:\ESRIPress\AgentAnalyst\Chapter02\MyData folder.

Updating point agent locations in Agent Analyst and watching them dynamically move in ArcMap

Up to this point you have not used the ArcGIS visualization environment with Agent Analyst. In this exercise, you will create agents that move across the landscape in the ArcMap display.

Exercise 2f

Load the ArcMap document

- 1 Start ArcMap. Navigate to C:\ESRIPress\AgentAnalyst\Chapter02 and open ex02f.mxd.

In the ArcMap display, you will see purple dots representing cougar locations. In the ArcToolbox window, the Chapter02Ex02f toolbox has been created for you.

Note: You can continue with the model from the previous exercise (skip steps 2 through 4).

Load the Agent Analyst model

- 2 Right-click the Chapter02Ex02f toolbox, point to New, and click Agent Analyst Tool.
- 3 Once the Agent Analyst window appears, click File and select Import. Do not save the changes when prompted.
- 4 Navigate to C:\ESRIPress\AgentAnalyst\Chapter02\Models and open ex02f.sbp. The Agent Analyst window appears.

Now you will create a new action. The ArcGIS display will allow you to monitor the actions of the cougar agents.

- 5 In the Environment panel, click Cougar. In the Property panel, click the Edit button to the right of the Actions property.
- 6 Add a new action, called **move**, by clicking the New Action button in the Actions Editor. Type the name, and click OK.



- 7 Type the following code in the Source panel of the Actions Editor:

```
def move():
    # change the location of a Point
    # by retrieving its geometry coordinate attribute and changing →
    ➔ its x,y coordinates
    coordinate = self.the_geom.coordinate
    coordinate.x = coordinate.x + 200
    coordinate.y = coordinate.y + 300
```

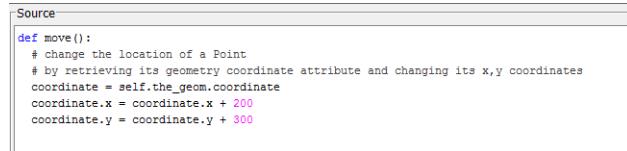
2

3

4

Note: This book uses arrow symbols as shown in the preceding code to indicate where a long line of code was broken into multiple lines to fit the book dimensions.

The Source panel code should look like the code in the image below.



```
Source
def move():
    # change the location of a Point
    # by retrieving its geometry coordinate attribute and changing its x,y coordinates
    coordinate = self.the_geom.coordinate
    coordinate.x = coordinate.x + 200
    coordinate.y = coordinate.y + 300
```

The code states that for every cougar agent, the move function reads its current location (`coordinate = self.the_geom.coordinate`), and then the agent changes its location by 200 units to the east (`coordinate.x = coordinate.x + 200`) and 300 units to the north (`coordinate.y = coordinate.y + 300`).

One way to move agents is to simply change the current coordinate values. This gives you the maximum amount of control when moving an agent in geographic space, but it may also require additional programming to best calculate target coordinates. In this example we are moving a constant direction and distance, but the coordinate changes might represent speed and direction among other things.

Use the step action to activate the move action

- 8 Select the step function from the drop-down list in the Actions Editor. Modify it as follows:

```
def step():
    self.move()
```

- 9 Click OK to save the code.

To see the agents moving in ArcMap, the cougars' shapefile must be updated with each move of the agents.

- 10** In the Environment panel, click Simple Model. In the Property panel, click the Edit button to the right of the Actions property. Select the writeAgents action from the Actions Editor drop-down list.

The writeAgents action template is automatically created by Agent Analyst when the model is created because it is a basic action for all models. You will modify the action.

- 11** In the Source panel of the Actions Editor, enter the following code in the writeAgents action:

```
def writeAgents():
    self.writeAgents(self.cougars, "C:\ESRIPress\AgentAnalyst\Chapter02\Data\cougars.shp")
```

The Source panel code should look like the code in the following image.



```
Source
def writeAgents():
    self.writeAgents(self.cougars, "C:\ESRIPress\AgentAnalyst\Chapter02\Data\cougars.shp")
```

This write action code reads as follows:

```
self.writeAgents(your_agentGroup_here, your_shapefile_here)
```

In this code you have instructed that the new locations of the cougars be written to the shapefile that contains their current locations. If you want to preserve the starting locations of the cougars (they may be empirically observed locations), make a copy of the shapefile and write to the copy. This step is not necessary for this exercise because you are provided with the initial conditions in the C:\ESRIPress\AgentAnalyst\Chapter02\Data\Data_copy folder.

- 12** Click OK to close the Actions Editor.

Like any action, the writeAgents action needs to be scheduled.

- 13** In the Environment panel, click Simple Model. In the Property panel, click the Edit button to the right of the Schedule property.
- 14** From the Actions list, select writeAgents and set Execution to EVERY TICK. Click Add.

The writeAgents action updates only the shapefile and does not automatically update the display in ArcMap. If you want to display the movement, you also need to schedule the updateDisplay action (see the sidebar “The updateDisplay() action”).

The updateDisplay() action

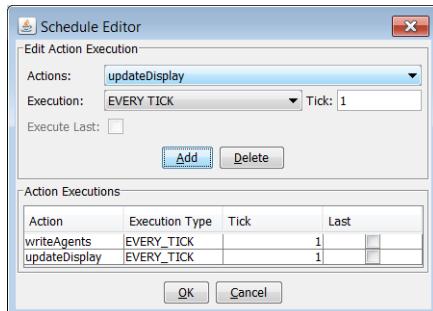
The updateDisplay action is a model-level action that refreshes the ArcMap display. This is accomplished by calling an executable named Refresh.exe. The default directory for this file is C:\Program Files\Repast 3\Agent Analyst\Refresh.

Depending on the number and complexity of the layers within the ArcMap document, a refresh can often be slow, because when ArcMap refreshes, it redraws every layer, even though the vector agent layer may be the only one that has changed. This inevitably results in a time lag between your model run and the display in ArcMap.

There are a few solutions to this problem:

- Turn off all unnecessary layers.
- Reduce the complexity of the layers. If possible, resample rasters and simplify the vector layers.
- Zoom in to your area of interest instead of using the full extent.
- Set the updateDisplay action to execute less frequently through the model scheduler. This is not optimal because not all of the model results will be displayed.

- 15** In the Schedule Editor window, from the Actions list, select updateDisplay. Make sure that Execution is set to EVERY TICK. Click Add (see the following figure).



- 16** Click OK.

Run the model

- 17** Click the Run button in Agent Analyst, and then click the Start button when the Repast toolbar appears.

Observe the agents' movement in ArcMap, and close and save the model

- 18** Exit the model.

- 19** Close Agent Analyst by clicking Exit. Save the model as Chapter2MyModel6.sbp in the C:\ESRIPress\AgentAnalyst\Chapter02\MyData folder.

Creating additional fields or properties and moving the agents with greater sophistication

2

3

4

Exercise 2g

Load the ArcMap document

- 1 Start ArcMap. Navigate to C:\ESRIPress\AgentAnalyst\Chapter02 and open ex02g.mxd.

In the ArcMap display you will see purple dots representing cougar locations. In ArcToolbox, the Chapter2Ex02g toolbox has been created for you.

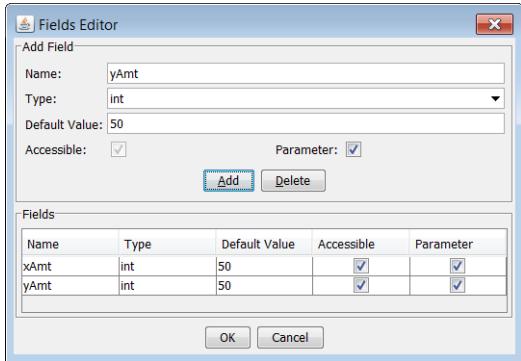
Note: You can continue with the model from the previous exercise (skip steps 2 through 4).

Load the Agent Analyst model

- 2 Right-click the Chapter02Ex02g toolbox, point to New, and click Agent Analyst Tool.
- 3 Once the Agent Analyst window appears, click File and select Import. Do not save the changes when prompted.
- 4 Navigate to C:\ESRIPress\AgentAnalyst\Chapter02\Models and open ex02g.sbp. The Agent Analyst window appears.

To create a more realistic move action, begin by creating two new fields for your cougar agents. These fields are new variables or properties to store x and y movement amounts. The distance to move in the x direction will be defined by the variable xAmt, and the distance to move in the y direction will be defined by the variable yAmt.

- 5 In the Environment panel, click Simple Model.
- 6 In the Property panel, click the Edit button to the right of the Fields property.
- 7 To create a new field, type **xAmt** in the Name field. Keep int as the Type. Set the default value to 50, and select both Accessible and Parameter. Click Add.
- 8 Create another field and name it **yAmt**. Use the same settings specified in the previous step.



If a variable is defined outside the current component you work on, you must set it as Accessible in the Fields Editor to make it available to other components. By setting the fields to Parameter, they are accessible to the user when the model is run and can be set to any desired value. You have seen these user-accessible parameters in the parameters window (earlier labeled Simple Model Settings) when running models in earlier exercises.

9 Click OK.

You will now modify the move action for the cougar agents by adding randomness to the movement so they make more realistic movements.

- 10** In the Environment panel, click Cougar. In the Property panel, click the Edit button to the right of the Actions property.
- 11** In the Actions Editor, select the move action, and in the Source panel modify the code as follows. Note the use of xAmt and yAmt:

```
def move():
    # get the amount to move the model
    xAmt = self.model.xAmt
    yAmt = self.model.yAmt

    # add some randomness to the movement
    # for example: 50% chance to move left/right or up/down
    xRnd = Random.uniform.nextIntFromTo(0,1)
    yRnd = Random.uniform.nextIntFromTo(0,1)
    if (xRnd == 0):
        xAmt = -xAmt
    if (yRnd == 0):
        yAmt = -yAmt

    # set the coordinates
    coordinate = self.the_geom.coordinate
    coordinate.x = coordinate.x + xAmt
    coordinate.y = coordinate.y + yAmt
```

The Source panel code should look like the code in the image below.

```

Source
def move():
    # get the amount to move the model
    xAmt = self.model.xAmt
    yAmt = self.model.yAmt

    # add some randomness to the movement
    # for example: 50% chance to move left/right or up/down
    xRnd = Random.uniform.nextIntFromTo(1,0)
    yRnd = Random.uniform.nextIntFromTo(1,0)
    if (xRnd == 0):
        xAmt = -xAmt
    if (yRnd == 0):
        yAmt = -yAmt

    # set the coordinates
    coordinate = self.the_geom.coordinate
    coordinate.x = coordinate.x + xAmt
    coordinate.y = coordinate.y + yAmt

```

2

3

4

This code specifies that the amounts xAmt and yAmt to move a cougar agent will be retrieved and stored in variables with the same names, xAmt (`xAmt = self.model.xAmt`) and yAmt (`yAmt = self.model.yAmt`). The xAmt and yAmt for each cougar agent to move are derived from the fields defined using the Fields Editor in steps 7 and 8 earlier. These fields are defined as parameters, and therefore they can be changed by the user. The default value was set to 50.

To dictate the direction of the movement, you call the Random function within Agent Analyst to select a random integer equal to 0 or 1 from a uniform distribution. The random numbers are assigned to the xRnd (`xRnd = Random.uniform.nextIntFromTo(0, 1)`) and yRnd (`yRnd = Random.uniform.nextIntFromTo(0, 1)`) variables. If the random number for the x is 1, nothing happens in the `if` construct. Therefore, xAmt will remain unchanged, and a positive (eastward) movement will be performed (in this case, 50 map units, the specified default value). If the random number is equal to 0 (`if (xRnd == 0) :`), then the x movement will be 50 map units in the negative direction, to the west (`xAmt = -xAmt`). The same test is evaluated for the random number selected for the y coordinate (`if (yRnd == 0) :`) with similar results. If the random number is 1, the move will be 50 map units to the north, and if it is equal to 0, the agent will move 50 map units to the south (`yAmt = -yAmt`).

The location of the cougar is identified (`coordinate = self.the_geom.coordinate`), the xAmt is added to the x position (`coordinate.x = coordinate.x + xAmt`), and the yAmt to the y position (`coordinate.y = coordinate.y + yAmt`). In this model, the cougar will move in the x or y direction based on random decision making. The random numbers are used only for demonstration purposes to add variety in the evaluation of the decision making of the agents. Real cougars have intelligence and move with intent. In chapters 5 and 8, you will learn how to move the cougars with code based on observed movements.

- 12** Click OK to save the action and close the Actions Editor.

Run the model

- 13** On the Agent Analyst toolbar, click the Run button.

The Repast toolbar appears with the Simple Model Settings window. Notice that the xAmt and yAmt attributes appear on the Parameters tab and are assigned 50 by default.

- 14** On the Repast toolbar, click Start.

While executing the model, observe the difference between the previous unidirectional movement and the new randomized movement.

- 15** Close and save the model.

Exit the model

- 16** Close Agent Analyst by clicking Exit. Save the model as Chapter2MyModel7.sbp in the C:\ESRIPress\AgentAnalyst\Chapter02\MyData\ folder.

Accessing raster data and working with graphs

In this exercise, you will add a field to the cougar agent to track the amount of food available to each cougar during each time step. The amount of food that is gathered each time step will be determined from a raster layer that stores “food” values. The total of the average food amount the cougar agents gain will be graphed.

In this model the cougar moves each time step to a new location and obtains the amount of food at that location. The amount of food that can be retrieved is stored in a raster layer. The model randomly determines which of the eight surrounding cells the cougar moves into. After the move, the cougar retrieves food stored as the cell value and adds the food amount to its total accumulated food property.

Exercise 2h

Load the ArcMap document

- 1 Start ArcMap. Navigate to C:\ESRIPress\AgentAnalyst\Chapter02 and open ex02h.mxd.

In the ArcMap display you will see purple dots representing cougar locations. In the ArcToolbox window, the Chapter02Ex02h toolbox has been created for you.

Note: You can continue with the model from the previous exercise (skip steps 2 through 4).

Load the Agent Analyst model

- 2 Right-click the Chapter02Ex02h toolbox, point to New, and click Agent Analyst Tool.
- 3 Once the Agent Analyst window appears, click File and select Import. Do not save the changes when prompted.
- 4 Navigate to C:\ESRIPress\AgentAnalyst\Chapter02\Models and open ex02h.sbp. The Agent Analyst window appears.

Create a field to document the total accumulated amount of food for each cougar

Since the accumulation of food is a property of a cougar, the field will be added to the cougar agent.

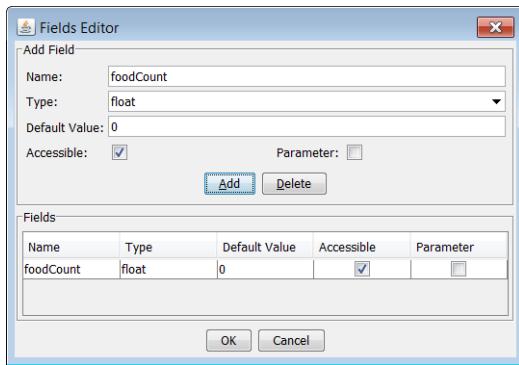
- 5 In the Environment panel, click Cougar. In the Property panel, click the Edit button to the right of the Fields property.

2

3

4

- 6 In the Name field, type **foodCount**. From the Type list, select float. Set the default value to 0, and select Accessible.



The field foodCount is controlled by the model, so it needs to be accessible to other actions, but it should not be available to the user since the value should not be changed.

With these settings the field will be accessible to any action or agent in the model, but it is not checked as a parameter and will therefore not be available to the user during model execution.

- 7 Click Add, and then click OK.

Create a new action to allow the cougars to capture or collect food, with the amount of food determined by the cell the cougar moves into.

- 8 In the Environment panel, click Cougar. In the Property panel, click the Edit button to the right of the Actions property.
- 9 Click the New Action button to create a new action and name it **captureFood**. Click OK. Type the following code in the action's Source panel:

```
def captureFood():
    # gets the location of the agent
    point = self.the_geom.coordinate

    # identifies the raster that determines the amount of food to obtain
    raster = self.model.getRaster("food")

    # identifies the location of where the cougar is
    # in relation to the cell location on the food raster
    # zero is the raster band number:
    foodAmt = raster.getValueAtMap(point.x, point.y, 0)

    # adds the amount of food obtained at the location
    # during the time step
```

```
# (foodAmt) to the total of food for the cougar (foodCount)
self.foodCount = self.foodCount + float(foodAmt)
```

This code first gets the location of the cougar (`point = self.the_geom.coordinate`). Second, it identifies the raster storing the amount of food gained at each cell location (`raster = self.model.getRaster("food")`). Third, it gets the food amount (the raster value) at the cougar's current location (`foodAmt = raster.getValueAtMap(point.x, point.y, 0)`). And fourth, it adds the food gained for the time step to the cougar's total accumulative count of gained food (`self.foodCount = self.foodCount + float(foodAmt)`).

2

3

4

- 10** Click OK to save the action and close the Actions Editor.

Schedule the captureFood action for cougars to iterate the action

- 11** In the Environment panel, click Cougar. In the Property panel, click the Edit button to the right of the Schedule property to open the Schedule Editor.
- 12** Schedule the captureFood action to be executed EVERY TICK. Click Add, and then click OK.

The captureFood action must become aware of what raster to query for the food amount. In the code for the captureFood action, you need to tell the model how and where to get the food raster dataset. You specified the name of the raster to use within the model, "food," but you now need to make the model aware of which ArcGIS raster dataset will be assigned to "food." You do so through the following action.

- 13** In the Environment panel, click Simple Model.
- 14** In the Property panel, click the Edit button to the right of the Actions property.
- 15** Create a new action, and name it **loadRaster**. Click OK. Type the following code in the Action Editor's Source panel:

```
def loadRaster():
    # creates a raster object that identifies the food raster
    # parameters: a path to the folder of the raster, raster name
    raster = ESRIRaster("C:/ESRIPress/AgentAnalyst/Chapter02/Data/ ➔
    ➔ rasters/", "food")

    # loads the raster and gives it a local label
    # i.e. the name used within the model
    self.addRaster("food", raster)
```

In the preceding code, you assigned the raster dataset food on disk to an object named `raster` (`raster = ESRIRaster("C:/ESRIPress/AgentAnalyst/Chapter02/Data/rasters/", "food")`). You then added the raster to the model with the name "food" (`self.addRaster("food", raster)`).

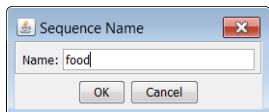
- 16 Click OK.

Schedule the loadRaster action

- 17 In the Environment panel, click Simple Model.
- 18 In the Property panel, click the Edit button to the right of the Schedule property.
- 19 In the Schedule Editor, select loadRaster from the Actions list. Since this action needs to happen only once per model run, select AT A SINGLE TICK from the Execution list, and in the Tick field enter **0.1** (which forces the model to execute this action just after the model starts). Click Add, and then click OK.

Since the cumulative amount of food is a property of a cougar agent and cannot be seen in the ArcGIS display, you will graph the average food intake of the cougar agents for each time step directly in Agent Analyst.

- 20 On the Agent Analyst toolbar, click the Sequence Graph button. In the Environment panel, click Simple Model to add the graph to the model.
- 21 To define the variable that you want to plot, in the Property panel, click the Edit button to the right of the Series property. The Series Editor opens.
- 22 Click New Action, and name the sequence **food**. Click OK.



- 23 Enter the following code in the Source panel:

```
# get the number of agents
count = self.cougars.size()

# calculate the sum of cougars' food
sum = 0.0
for cougar as Cougar in self.cougars:
    # sum is the running tabulation of the food gathered by ➔
    ➔ each cougar
    sum = sum + cougar.foodCount

    # return that average food for all agents
return sum/count
```

- 24 Click OK to close the Series Editor.

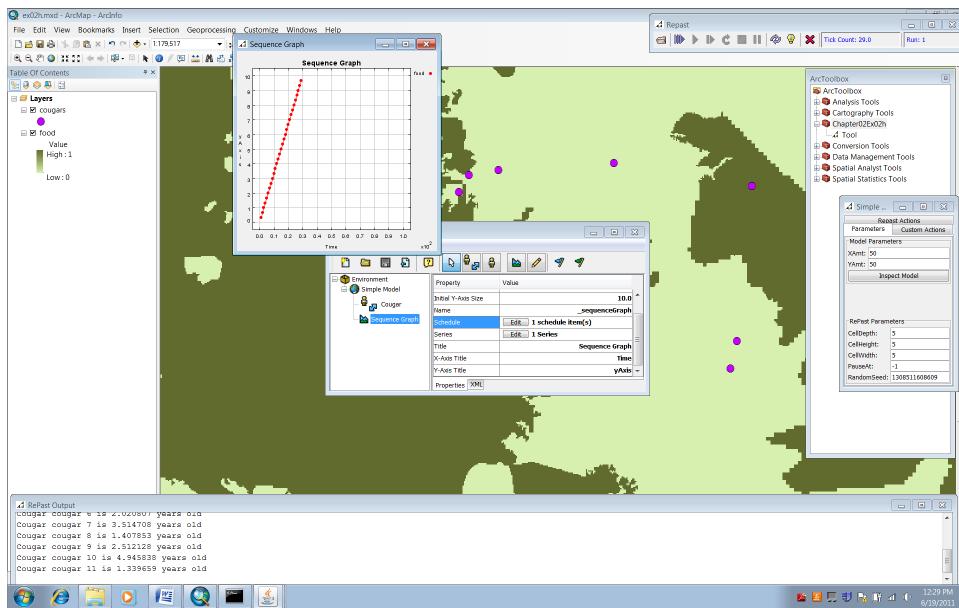
Schedule the action to be executed

- 25** In the Sequence Graph Property panel, click the Edit button to the right of the Schedule property. Select the update action to be executed EVERY TICK. Click Add, and then click OK.

For display purposes you will add the food raster to ArcMap view so that you can see the current locations of cougars in reference to the availability of food.

- 26** In ArcMap, click the Add Data button , and then navigate to the C:\ESRIPress\AgentAnalyst\Chapter02\Data\rasters\ folder and add the food raster (named food) to ArcMap. Change the symbology of the food raster to whatever you prefer.
- 27** Run the model.

You will see the cougar agents moving in the ArcGIS display and a dynamic graph tabulating the total average food intake for the cougar agents.



Close Agent Analyst

- 28** Click Exit. Save the model as Chapter2MyModel8.sbp in the C:\ESRIPress\AgentAnalyst\Chapter02\MyData folder.

2

3

4

Tracking the path of the moving point agent

In the previous exercises you learned how to move cougar agents and display their new locations. However, you could not see the path each agent took to get to the current location. In this exercise, you will track the paths of the moving agents.

Exercise 2i

Load the ArcMap document

- 1 Start ArcMap. In the ArcMap—Getting Started dialog box, under the Existing Maps section, click “Browse for more...”. (If ArcMap is already running, click the File menu and click Open.) Navigate to C:\ESRIPress\AgentAnalyst\Chapter02 and open ex02i.mxd.

In the ArcMap display, you will see purple dots representing cougar locations. In ArcToolbox, the Chapter02Ex02i toolbox has been created for you.

Note: If you continue with the model from the previous exercise, skip steps 2 through 4.

You will use a raster layer to trace the paths of moving agents. Initially, all cells in the raster are assigned the value 1. The paths are only one cell wide, so to see the paths before continuing, zoom in to a selected cluster of cougars.

Load the Agent Analyst model

- 2 Right-click the Chapter02Ex02i toolbox, point to New, and click Agent Analyst Tool.
- 3 Once the Agent Analyst window appears, click File and select Import. Do not save the changes when prompted.
- 4 Navigate to C:\ESRIPress\AgentAnalyst\Chapter02\Models and open ex02i.sbp. The Agent Analyst window appears.

Add the tracking raster to your loadRaster action

- 5 In the Environment panel, click Simple Model. In the Property panel, click the Edit button to the right of the Actions property.
- 6 Select loadRaster from the list. Add the following code to the end of the action in the Source panel:

```
# add the 'tracking' raster
raster = ESRIRaster("C:/ESRIPress/AgentAnalyst/Chapter02/Data/ ➔
▶ rasters/", "tracking")

self.addRaster("tracking", raster)
```

The Source panel code should look like the code in the following image:

```
Source
def loadRaster():
    # creates a raster object that identifies the food raster
    # parameters: a path to the folder of the raster, raster name
    raster = ESRIRaster("C:/ESRIPress/AgentAnalyst/Chapter02/Data/rasters/", "food")

    # loads the raster and gives it a local label
    # i.e. the name used within the model
    self.addRaster("food", raster)

    # add the 'tracking' raster
    raster = ESRIRaster("C:/ESRIPress/AgentAnalyst/Chapter02/Data/rasters/", "tracking")
    self.addRaster("tracking", raster)
```

You have identified the raster (`raster = ESRIRaster("C:/ESRIPress/AgentAnalyst/Chapter02/Data/rasters/", "tracking")`) and added it to the model (`self.addRaster("tracking", raster)`).

- 7 Click OK to close the Actions Editor.

Now you will create the action code to track the cougar paths as the cougars move.

- 8 In the Environment panel, click Simple Model.
- 9 In the Property panel, click the Edit button to the right of the Actions property.
- 10 Create a new action and name it **trackCougars**. Type the following code in the Actions Editor Source panel:

```
def trackCougars():
    raster = self.getRaster("tracking")
    for cougar as Cougar in self.cougars:
        point = cougar.the_geom.coordinate
        pixelPoint = raster.mapToPixel(point.x, point.y, 0)
        raster.writePixelValue(4, int(pixelPoint.x), int(pixelPoint.y), 0)
```

This code first calls the tracking raster (`raster = self.getRaster("tracking")`), and then for each cougar agent (`for cougar as Cougar in self.cougars:`) it retrieves the location of the cougar agent (`point = cougar.the_geom.coordinate`), finds the corresponding cell in the tracking raster (`pixelPoint = raster.mapToPixel(point.x, point.y, 0)`), and sets the cell's value to 4 (`raster.writePixelValue(4, int(pixelPoint.x), int(pixelPoint.y), 0)`).

2

3

4

- 11 Click OK to close the Actions Editor.

Schedule the action to activate the trackCougars action

- 12 In the Environment panel, click Simple Model. In the Property panel, click the Edit button to the right of the Schedule property. Select the trackCougars action and schedule it for EVERY TICK. Click Add, and then click OK.

Run the model

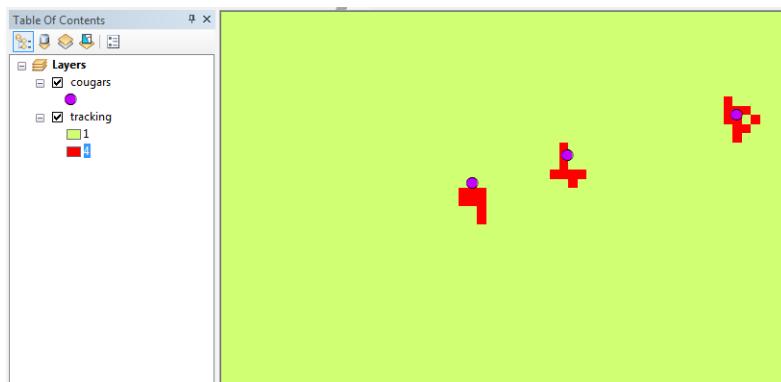
- 13 Click the Run button in Agent Analyst, and then click the Start button on the Repast toolbar.
14 After a number of model iterations, click the Stop button.

You will see the cougars moving in the ArcGIS display window. To see the results of the tracks, you must load the “tracking” raster.

Note: You may notice a series of messages (including “VATBuild: error in building VAT”) appear in the command window. These are internal messages, and they can be ignored. You can minimize the command window if you want.

- 15 On the ArcGIS toolbar, click the Add Data button. Navigate to C:\ESRIPress\AgentAnalyst\Chapter02\Data\rasters, click tracking, and then click Add.

On the tracking raster you will see the tracks the cougar agents took during the model run. Your result will look different from the following image.



Close Agent Analyst

- 16 Close the Repast toolbar, and click Exit in Agent Analyst. Save the model as Chapter2MyModel9.sbp in the C:\ESRIPress\AgentAnalyst\Chapter02\MyData\ folder.

Building complex agents from multiple agents

Thus far, the model has consisted of single agents with properties and actions. In this exercise, you will learn how to represent a more complex phenomenon by breaking down the model into multiple interacting agents. You will track the battery level in collars placed on each cougar agent. The collars will be agents worn by the cougar agents.

2

3

4

Exercise 2j

Load the ArcMap document

- 1 Start ArcMap. Navigate to C:\ESRIPress\AgentAnalyst\Chapter02 and open ex02j.mxd.

In the ArcMap display you will see purple dots representing cougar locations. In the ArcToolbox window, the Chapter02Ex02j toolbox has been created for you.

Note: You can continue with the model from the previous exercise (skip steps 2 through 4).

Load the Agent Analyst model

- 2 Right-click the Chapter02Ex02j toolbox, point to New, and click Agent Analyst Tool.
- 3 Once the Agent Analyst window appears, click File and select Import. Do not save the changes when prompted.
- 4 Navigate to C:\ESRIPress\AgentAnalyst\Chapter02\Models and open ex02j.sbp. The Agent Analyst window appears.

First, create the collar agents and the appropriate fields to manage the collars for the cougars. Because the spatial component is not central to the collars, and the type of representation for the collars is not significant to the agent (for example, whether it is a point or polygon), we will represent the collars through generic agents. The status and characteristics of the collars will be tracked through the generic agents' properties and actions.

- 5 On the Agent Analyst toolbar, click the Generic Agent icon. In the Environment panel, click Simple Model to add the agents to the agent-based model. Rename the generic agent to **Collar** and rename the group name from GenericAgentGroup to **collars**.

- 6 In the Environment panel, click Collar. In the Property panel, click the Edit button to the right of the Fields property. Create the following three necessary fields for the collar agent:
- Name: duration
Type: integer
Default: No default value
Accessible: Yes
Parameter: No
 - Name: time
Type: integer
Default: 0
Accessible: Yes
Parameter: No
 - Name: cougar
Type: Cougar
Default: No default value
Accessible: Yes
Parameter: No

Note that the last field is a Cougar class rather than a primitive data type. This field represents the cougar to which the collar belongs. To specify a type other than the default types in the list, type **Cougar** in the Type text box.

- 7 Click OK to close the Fields Editor.

The collars must be associated with each cougar, and the duration of the collars must be established. This will be done in the initialization action of the model.

- 8 In the Environment panel, click Simple Model. In the Property panel, click the Edit button to the right of the Actions property.
- 9 Select the initAgents action from the list, and type the following code in the Source panel (the first five lines of code have already been entered for you):

```
def initAgents():
    for cougar as Cougar in self.cougars:
        # print "Cougar", cougar.name, "is", cougar.age, "years old"
        # cougar.id = 0

        # GIVE EACH COUGAR A COLLAR:
        # create a collar object and add it to collars
        collar = Collar()
        self.collars.add(collar)
        # set the duration of the collar (battery)
```

```

duration = Random.uniform.nextIntFromTo(5, 25)
collar.duration = duration
# associate a cougar with the collar
collar.cougar = cougar

```

In the preceding code, a collar agent is created for each Cougar in the cougars group (cougar as Cougar in self.cougars:) and assigned to the variable collar (collar = Collar()). The collar is added to the collars group (self.collars.add(collar)). The variable duration is set to a random value (duration = Random.uniform.nextIntFromTo(5, 25)) defining how long the battery will last, and that value is assigned to the duration property for the collar (collar.duration = duration). (The random variable is only used for demonstration. In a real situation, the actual battery duration would be set to the duration property for each collar.) Finally, the code assigns the current cougar agent that is being processed to the newly created collar (collar.cougar = cougar).

Define the collars step action. The action will reduce the battery duration by the appropriate amount each time step.

- 10 In the Environment panel, click Collar. In the Property panel, click the Edit button to the right of the Actions property.
- 11 Select the step action from the list, and type the following code in the Source panel:

```

def step():
    # implementing the collar battery run down
    if self.time < self.duration:
        self.cougar.id = 4
    else:
        self.cougar.id = 1
    self.time = self.time + 1
    print "current time:",self.time,"duration:",self.duration,"id:", ➔
    ➔ self.cougar.id

```

If current model time (the number of time steps that have transpired since the start of the model) is less than the duration of the battery (if self.time < self.duration:), the cougar agent's ID (a property of the cougar agent) is set to 4, indicating that the collar is still active (self.cougar.id = 4). When the battery runs out (self.time >= self.duration), the cougar agent's ID is set to 1 (self.cougar.id = 1). The time field is incremented by one (self.time = self.time + 1).

2

3

4

Schedule the collar step action

- 12** In the Environment panel, click Collar. In the Property panel, click the Edit button to the right of the Schedule property. The step action for the collar agents has been automatically scheduled, as is the step action for all agents. You do not need to do anything. Click OK.
- 13** Run the model.

You will see the cougar agents moving within the ArcGIS display, a graph of the average sum of their food intake, and the life of the batteries for each cougar agent.

Close Agent Analyst

- 14** Click Exit. Save the model as Chapter2MyModel10.sbp in the C:\ESRIPress\AgentAnalyst\Chapter02\MyData folder.

Developing interaction between agents

In the final exercise of this chapter, you will learn how to code the agents to interact with each other every time step. The locations of other agents and their actions will influence the behavior of a specific agent. The agents will be able to look for other agents within a specified distance.

2

3

4

Exercise 2k

Load the ArcMap document

- 1 Start ArcMap. Navigate to C:\ESRIPress\AgentAnalyst\Chapter02 and open ex02k.mxd.

In the ArcMap display you will see purple dots representing cougar locations. In the ArcToolbox window, the Chapter02Ex02k toolbox has been created for you.

Note: You can continue with the model from the previous exercise (skip steps 2 through 4).

Load the Agent Analyst model

- 2 Right-click the Chapter02Ex02k toolbox, point to New, and click Agent Analyst Tool.
- 3 Once the Agent Analyst window appears, click File and select Import. Do not save the changes when prompted.
- 4 Navigate to C:\ESRIPress\AgentAnalyst\Chapter02 and open ex02k.sbp. The Agent Analyst window appears.

The interaction action rule states that if a cougar agent is within 7,000 meters of another cougar agent, the cougar agent will move half the distance toward that cougar agent (for example, possibly searching for a mate). Add the interaction code between agents to the move action for each cougar agent.

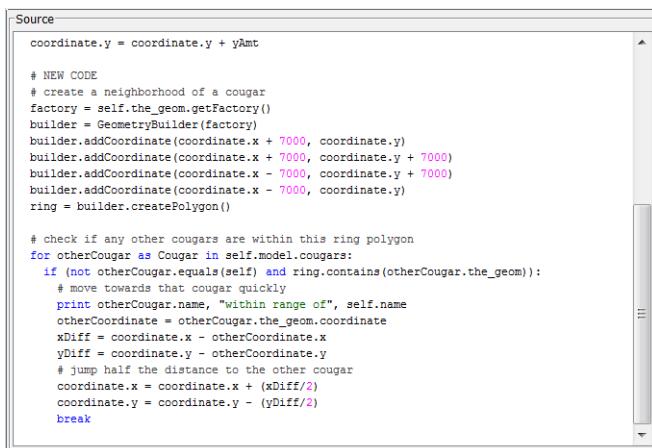
- 5 In the Environment panel, click Cougar. In the Property panel, click the Edit button to the right of the Actions property.

- 6 From the list, select the move action. In the Source panel, type the following code at the end of the move action:

```
# NEW CODE
# create a neighborhood of a cougar
factory = self.the_geom.getFactory()
builder = GeometryBuilder(factory)
builder.addCoordinate(coordinate.x + 7000, coordinate.y)
builder.addCoordinate(coordinate.x + 7000, coordinate.y + 7000)
builder.addCoordinate(coordinate.x - 7000, coordinate.y + 7000)
builder.addCoordinate(coordinate.x - 7000, coordinate.y)
ring = builder.createPolygon()

# check if any other cougars are within this ring polygon
for otherCougar as Cougar in self.model.cougars:
    if (not otherCougar.equals(self) and ➔
        ➔ ring.contains(otherCougar.the_geom)):
        # move towards that cougar quickly
        print otherCougar.name, "within range of", self.name
        otherCoordinate = otherCougar.the_geom.coordinate
        xDiff = coordinate.x - otherCoordinate.x
        yDiff = coordinate.y - otherCoordinate.y
        # jump half the distance to the other cougar
        coordinate.x = coordinate.x + (xDiff/2)
        coordinate.y = coordinate.y - (yDiff/2)
        break
```

The action causes a cougar agent to jump toward a nearby cougar agent. The Source panel code should look like the code in the following image.



The screenshot shows the Source panel with the following Python code:

```
Source
coordinate.y = coordinate.y + yAmt

# NEW CODE
# create a neighborhood of a cougar
factory = self.the_geom.getFactory()
builder = GeometryBuilder(factory)
builder.addCoordinate(coordinate.x + 7000, coordinate.y)
builder.addCoordinate(coordinate.x + 7000, coordinate.y + 7000)
builder.addCoordinate(coordinate.x - 7000, coordinate.y + 7000)
builder.addCoordinate(coordinate.x - 7000, coordinate.y)
ring = builder.createPolygon()

# check if any other cougars are within this ring polygon
for otherCougar as Cougar in self.model.cougars:
    if (not otherCougar.equals(self) and ring.contains(otherCougar.the_geom)):
        # move towards that cougar quickly
        print otherCougar.name, "within range of", self.name
        otherCoordinate = otherCougar.the_geom.coordinate
        xDiff = coordinate.x - otherCoordinate.x
        yDiff = coordinate.y - otherCoordinate.y
        # jump half the distance to the other cougar
        coordinate.x = coordinate.x + (xDiff/2)
        coordinate.y = coordinate.y - (yDiff/2)
        break
```

In the preceding code, a GeometryBuilder object is first created (`builder = GeometryBuilder(factory)`). Any coordinates that are within 7,000 meters of the existing location of the cougar being processed are identified (`builder.addCoordinate(coordinate.x + 7000, coordinate.y)`) and so on. The coordinates are made into a polygon, which serves as a neighborhood around the processing cougar (`ring = builder.createPolygon()`). For each of the other cougars (for `otherCougar` as `Cougar` in `self.model.cougars:`) the model determines whether the other cougar is within the specified neighborhood of the cougar being processed (if `(not otherCougar.equals(self) and ring.contains(otherCougar.the_geom))`). The coordinates of the first cougar that is detected within the neighborhood (`otherCoordinate = otherCougar.the_geom.coordinate`) are determined. The processing cougar then moves half the distance toward the detected cougar (`coordinate.x = coordinate.x + (xDiff/2); coordinate.y = coordinate.y - (yDiff/2)`).

7 Run the model.

Notice that when the cougars come in close contact they jump toward one another.

Close Agent Analyst

- 8 Click Exit. Save the model as `Chapter2MyModel11.sbp` in the `C:\ESRIPress\AgentAnalyst\Chapter02\MyData` folder.

You will continue to develop the cougar model in chapters 5 and 8 by adding additional complexity to more accurately capture their biology.

GIS data, fields, and actions dictionaries

Table 2.1 Data dictionary of GIS datasets

| Dataset | Data type | Description |
|----------|-----------|---|
| cougars | shapefile | Points representing cougars |
| food | raster | Dataset storing food for cougars |
| tracking | raster | Dataset that captures cougars' movement |

Table 2.2 Fields dictionary for the cougar model

| Field | Data type | Description |
|----------------------------|---------------|--|
| Cougar simple model | | |
| xAmt | integer | Step of cougars' move in the east-west direction |
| yAmt | integer | Step of cougars' move in the north-south direction |
| Cougar | vector agent | Model decision makers |
| Collar | generic agent | Collars carried by cougars |
| Sequence graph | graph | Average amount of food gathered by cougars, plotted per tick (time series) |
| Cougar agent | | |
| foodCount | float | Cumulative amount of food captured (consumed) by a cougar |
| Id | integer | Unique cougar agent identifier |
| age | double | Cougar's age |
| name | string | Cougar's label |
| model | GIS Model | Simple Model |
| Collar agent | | |
| duration | integer | Collar battery life |
| time | integer | Time passed since the beginning of simulation |
| cougar | Cougar | The cougar agent assigned to this collar |
| model | GIS Model | Simple Model |
| Sequence graph | | |
| cougars | Cougar | A collection of cougar agents |

Table 2.3 Actions dictionary for the cougar model

| Declared actions | Description | |
|----------------------------|---|---|
| Cougar simple model | | |
| initAgents | Invoke agent initialization | 2 |
| updateDisplay | Update ArcMap display | 3 |
| writeAgents | Save current cougar data to the shapefile | 4 |
| loadRaster | Load food and tracking rasters | |
| Cougar agent | | |
| step | Invoke the move action | |
| move | Find a new location and move there | |
| captureFood | Collect food at current location | |
| Collar agent | | |
| step | Calculate current battery life | |

Section 2: The basics of points, polygons, and rasters in agent-based modeling

Chapter 3

Changing attribute values of polygon agents

by Arika Ligmann-Zielinska

- ◆ Investigating the completed polygon agent model
- ◆ Initializing polygon agents
- ◆ Initializing generic agents
- ◆ Creating neighborhoods for polygon agents
- ◆ Formulating the agent decision making
- ◆ Developing agent move rules
- ◆ Using multiple decision rules

In chapter 2 you were introduced to the basics of modeling point agents. This chapter is the first of two that introduce you to modeling using polygon agents. In this chapter, you will learn the fundamentals of modeling with polygon agents. Then, in chapter 7, you will build on these basic concepts by exploring more complex techniques demonstrated through a land-use change model of parcel agents.

In this chapter, census blocks in Wenatchee, Washington, are represented as polygon agents. However, any entity that can be represented by a polygon can be a polygon agent. Examples of possible polygon agents include the following:

- Town parcels
- Timber cutting tracts
- Voting districts
- Land-use types
- Combat zones
- Buildings
- Census blocks or ZIP Codes

In this chapter, you will explore a simple model of segregation (phenomenon clustering based on like characteristics) implemented in Agent Analyst. In this model, segregation indirectly results from the level of similarity between a given agent and its immediate surroundings. The model shows separation of agents based on discriminatory individual decisions and simple local rules of preference. The model illustrates the basic principles you can use to update polygon agent attributes based on the information retrieved from the agent's nearest environment. You will learn how to initialize polygon agents based on spatial and textual data. You will also explore code that updates polygons based on decisions made by dynamic, movable agents that reside in these polygons.

The intent of this chapter is to introduce you to agent-based modeling (ABM) principles, not to teach you how to create a segregation model. If the rules are not satisfactory, they can be changed.

The modeling scenario

Imagine a hypothetical town consisting of census blocks to be occupied by a number of residents, where a block can be inhabited by only one resident type at a time. The resident population is slightly less than the total number of blocks, so some blocks will always remain vacant. The residents are either red or green. Initially, you assign residents randomly to the blocks.

However, the agents are very specific in color preferences; they will move out if the number of differently colored neighbors is higher than a predefined tolerable value. And so they move.

Each time step, an agent scans its neighborhood and counts the number of neighbors who are a different color. If this number is not within a tolerable threshold, the agent looks for a new, unoccupied place and abandons its current location.

Your eyewitness task is to observe a byproduct of these individual local decisions—a pattern of segregation forming globally. Segregation is defined here as patches of one-color neighborhoods.

Background information

Segregation is a byproduct of the need to separate oneself from others who are different. In residential terms, it amounts to living close to people who are somehow similar. If the neighborhood becomes too discrepant, an agent moves out to get close to those it is attracted to and away from those it is not. Here, distance is defined by adjacency, not travel distance.

In 1969, Thomas Schelling proposed a simple model for household migration based on such neighborhood residential dissonance. In a general form the resettlement decision rule may be defined as follows:

Let R be a resident with identity i_r , and let t be the tolerance threshold, then

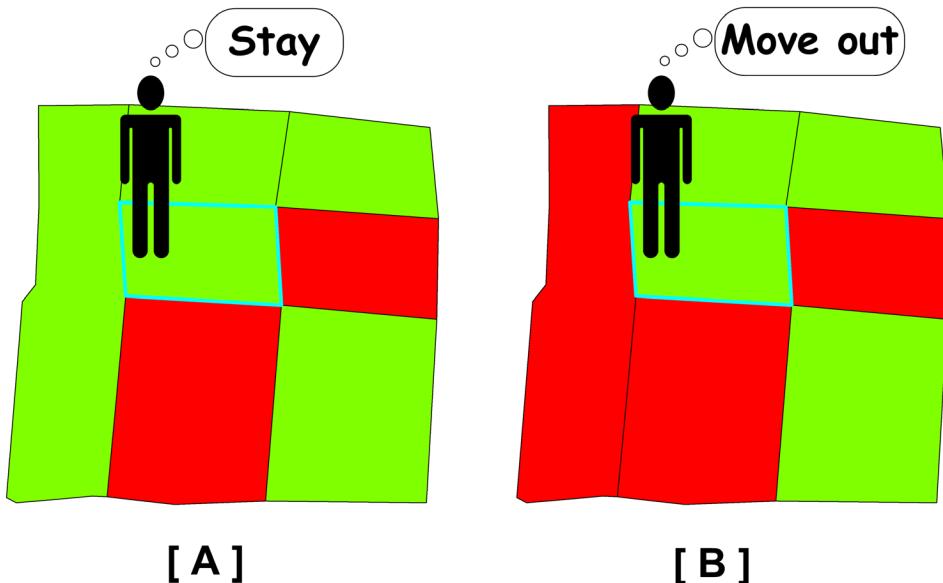
R scans its direct neighborhood and counts the number of neighbors who have an identity different from itself: $\sim i_r$

R establishes the dissimilarity d by calculating the ratio of $\sim i_r$ to the total number of neighbors T_n

$$d = \sim i_r / T_n$$

If $d \leq t$, then R is satisfied and stays at its location. Otherwise, R moves out to a new and potentially more favorable location.

As an example, suppose that the tolerance was set to 40 percent (i.e., $t = 0.4$), and the agent identity is green.



The agent stays or moves out, based on the similarity of its neighborhood.

In case *A*, the agent's dissimilarity is $d = 2/6 = 0.33$ (the location the agent is at is green, and two of his six neighbors are red), which is lower than t . Thus the agent stays at the current location. In case *B*, $d = 3/6 = 0.5$. Since $d > t$, the agent moves out.

The first Schelling segregation models, played manually on a chessboard, revealed an interesting regularity. It has been discovered that the magic neutrality threshold is 30 percent (Benenson and Torrens 2004). This threshold value results in nice pattern formation and leads to model equilibrium, where each and every agent is satisfied with its place. Higher tolerance values stabilize the system before any segregation emerges. For lower values, the process of movement never ends because agents are too selective to settle.

Investigating the completed polygon agent model

The following exercises focus on core Agent Analyst functionality pertaining to decision making based on analyzing neighborhood data. Through these exercises, you will learn how to define polygon-based neighborhoods, retrieve and update shapefile attribute values, define more complex decision rules, and use Java libraries in Not Quite Python (NQPy) code.

You will first explore the completed segregation model and then build the individual components in subsequent exercises.

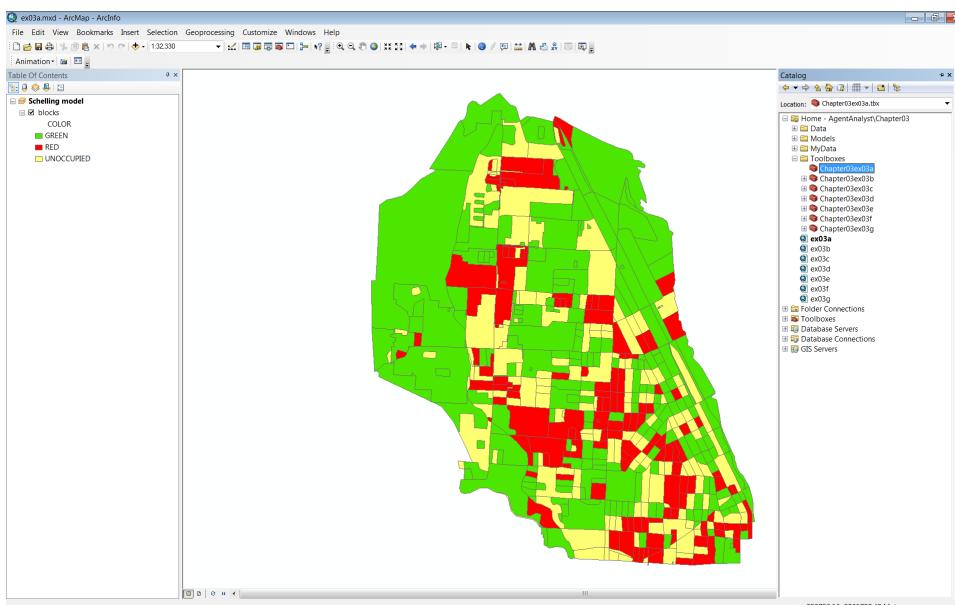
Exercise 3a

Begin by examining the completed Schelling segregation model, which will be developed in the consecutive exercises.

Load the ArcMap document

- 1 Start ArcMap. Navigate to C:\ESRIPress\AgentAnalyst\Chapter03. Select ex03a.mxd, and then click Open.

The map document for the Schelling segregation ArcGIS project opens. (See the following figure.) In the ArcMap display, you will see the blocks shapefile with the polygons symbolized as green, red, or unoccupied. In ArcToolbox, the Chapter03Ex03a toolbox has been created for you.



- 2 Right-click the blocks layer, and click Open Attribute Table to open the layer's attribute table.

Notice the COLOR field that has been used to symbolize census blocks. COLOR is the primary attribute used in the decision-making process of the model. Block colors are randomly reset with every model execution. Therefore, the colors of blocks on your map may look different from the map presented here.

- 3 Close the attribute table.

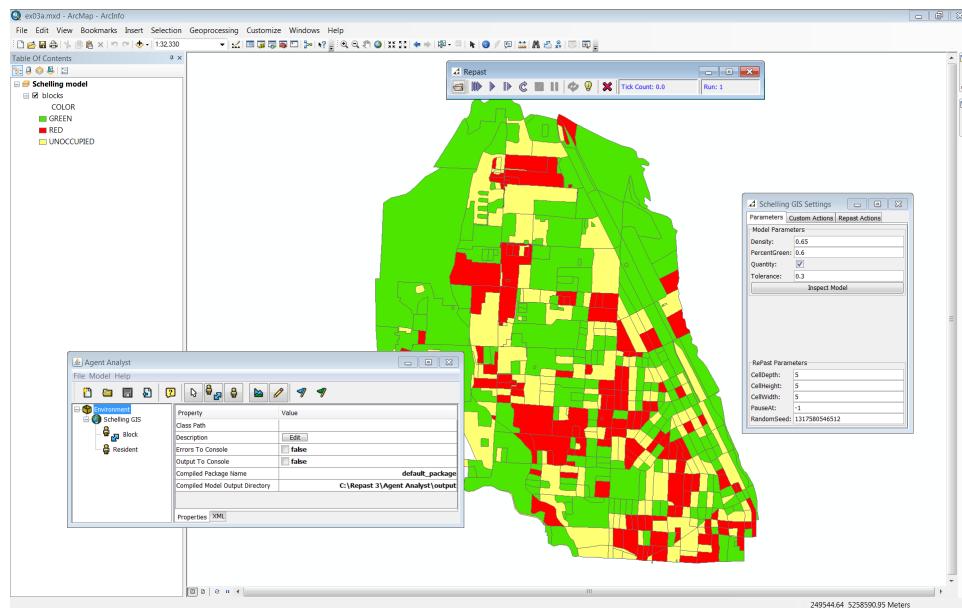
Open the Schelling model through the geoprocessing environment

- 4 Right-click the Chapter03Ex03a toolbox, point to New, and click Agent Analyst Tool.
- 5 Once the Agent Analyst window appears, click File and select Import. Do not save the changes when prompted.
- 6 Navigate to C:\ESRIPress\AgentAnalyst\Chapter03\Models and open ex03a.sbp. The Agent Analyst window appears.

The model presented in this chapter was implemented on a polygon feature layer rather than a raster. The model is composed of a spatial layer of census block polygons, which symbolize the locations of house parcels. This vector polygon agent layer is mutable—that is, its attributes may be changed during model execution. However, as GIS polygon agents, they are nonmoveable (i.e., they represent fixed geographic entities). In addition to vector polygon agents, the model contains movable generic agents representing simplified residents that make decisions such as whether to move to a new location.

View the model parameters

- 7 Click the Run button. The Repast toolbar and the Schelling GIS Settings window appear.



2

3

4

This model contains four parameters that can be changed during initialization: Density, PercentGreen, Quantity, and Tolerance. The density of populated blocks determines the vacancy rate in the area, which is symbolized by the UNOCCUPIED value of the COLOR attribute. PercentGreen defines the ratio of green agents to total agents, dividing the resident population into green and red groups. Tolerance is a threshold parameter defined as the fraction of an agent's neighboring agents whose color is different from the agent's color.

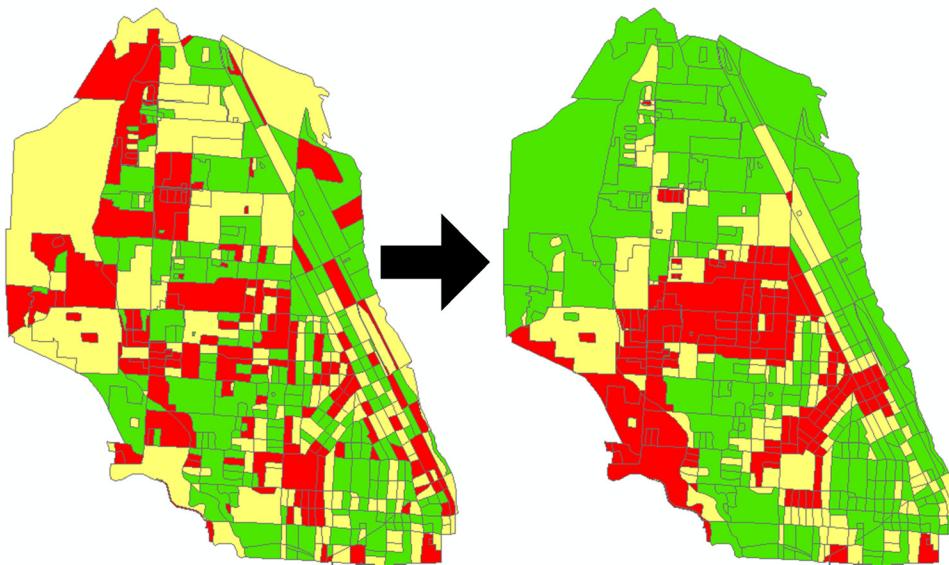
Explore the actions the resident agents can perform

- 8 In the Environment panel in the Agent Analyst window, click Resident. In the Property panel, click the Edit button to the right of the Actions property. Review the actions defined for residents in the following order: step, chooseQuantity, chooseArea, and move.

The “quantity” Boolean parameter influences the selection of the two alternative decision rules. The chooseQuantity rule is based on the algorithm described in the introduction to this chapter, which compares a count of differently colored neighbors to the total number of neighbors. The chooseArea rule uses an area of opposite color occupancy as the catalyst of migration.

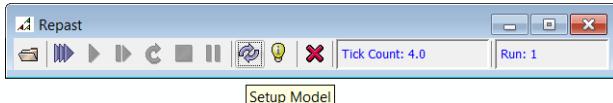
Run the model

- 9 Click the Start button on the Repast toolbar. Observe the emerging segregation clustering of red and green patches, as shown here:



You can execute the model multiple times with different parameter settings. Explore the different levels of segregation (patchiness) generated with various parameter settings.

- 10 To change the default values of the parameters without closing the Repast toolbar, click Stop, click the Setup Model button (see the following figure), type the parameter values you want to use in the Parameters tab of the Schelling GIS Settings window, and run the model as usual.



- 11 Close Agent Analyst by clicking File and then Exit. Do not save any changes when prompted.

Initializing polygon agents

2

3

4

As you know from the previous chapter, an agent-based model starts with a setup, into which your model loads data, initializes the components, and sets up the agents.

Exercise 3b

In this exercise, you will learn how to define actions to initialize polygon agents.

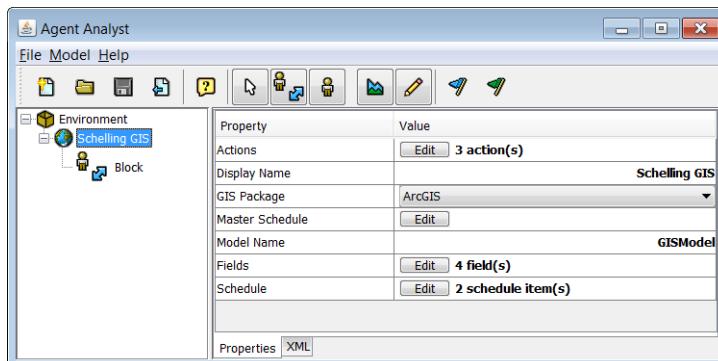
Load the ArcMap document

- 1 Start ArcMap. Navigate to C:\ESRIPress\AgentAnalyst\Chapter03 and open ex03b.mxd.

In the ArcMap display, you will see the blocks shapefile with the polygons symbolized as green, red, or unoccupied. In ArcToolbox, the Chapter03Ex03b toolbox has been created for you.

Load the Agent Analyst model

- 2 Right-click the Chapter03Ex03b toolbox, point to New, and click Agent Analyst Tool.
- 3 Once the Agent Analyst window appears, click File and select Import. Do not save the changes when prompted.
- 4 Navigate to C:\ESRIPress\AgentAnalyst\Chapter03\Models and open ex03b.sbp. The Agent Analyst window appears.



The model you have loaded consists of two components in the Environment panel: a Schelling GIS model and a Block vector agent that serves as a template for this exercise. The data source, component names, and basic actions have already been declared. You will use this template to build your own model.

Create the initialization action to assign residents to the polygon block agents

- 5 In the Environment panel, click the Schelling GIS model. In the Property panel, click the Edit button to the right of the Actions property to open the Actions Editor.
- 6 Select the initAgents action, and add the following code in the Source panel:

```
Random.createUniform()
```

This line of code creates a random uniform distribution. For demonstration purposes, you will randomly assign your residents to block polygons as a starting condition.

- 7 Click OK to save the code that you just typed and to close the Actions Editor.

You need to create actions to prepare the data for the model. Add a new action to your model.

- 8 In the Environment panel, click Schelling GIS. In the Property panel, click the Edit button to the right of the Actions property.
- 9 Click the New Action icon, and name the action **setupBlocks**. Click OK.

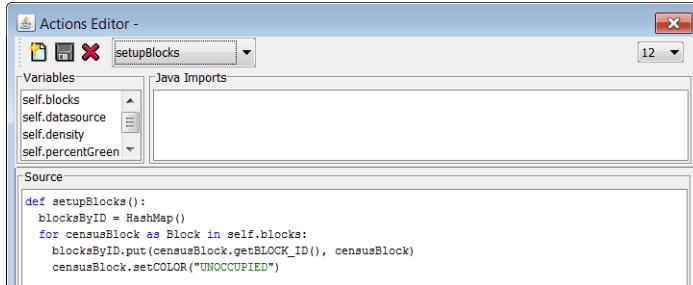


This action will be used to create a hash map table of your blocks and later to define their neighborhoods. This step is necessary so that your residents can scan their immediate neighborhood.

- 10 In the `setupBlocks()` action definition, type the following code in the Source panel:

```
def setupBlocks ():  
    blocksByID = HashMap()  
    for censusBlock in self.blocks:  
        blocksByID.put(censusBlock.getBLOCK_ID(), censusBlock)  
        censusBlock.setCOLOR("UNOCCUPIED")
```

The Source panel code should look like the code in the following figure.



This code first creates a hash map (`blocksByID = HashMap()`), which is a dictionary-like object. Then the block vector agents are put into the hash map (`for censusBlock as Block in self.blocks:`), where the key is the `BLOCK_ID` and the value is the census block object itself (`blocksByID.put(censusBlock .getBLOCK_ID(), censusBlock)`). It also sets the `COLOR` attribute of each block to "UNOCCUPIED" (`censusBlock.setCOLOR("UNOCCUPIED")`). Note that the `COLOR` attribute values from the previous model execution are reinitialized.

`COLOR` is one of the fields in the `blocks` shapefile attribute table. By default, every field in a shapefile becomes accessible in every Agent Analyst component. Accessible fields are equipped with special methods that start with "get" and "set" (getters and setters, respectively). These methods allow you to retrieve data from the agents (with get) or write data to the agents (with set). Since these methods are automatically built by Agent Analyst, you do not have to define them explicitly in the Actions Editor.

To finish block initialization, you need to invoke the `setupBlocks` action in agent initialization.

- 11** In the Actions Editor, select `initAgents` from the list and add the following code:

```
self.setupBlocks()
```

- 12** To test the initialization and setup actions, type the following line in the `setupBlocks` action:

```
#testing code....
print "number of vector agents in the hash map: ", blocksByID.size()
```

The Source panel code should look like the code in the following figure.

2

3

4



The screenshot shows the Actions Editor window with the title "Actions Editor -". It contains a toolbar with icons for file operations and a dropdown menu. Below the toolbar is a tab bar with "setupBlocks" selected. On the left, there's a "Variables" pane listing "self.blocks", "self.datasource", "self.density", and "self.percentGreen". To the right of the variables is a "Java Imports" pane which is currently empty. The main area is titled "Source" and contains the following Python code:

```

def setupBlocks():
    blocksByID = HashMap()
    for censusBlock as Block in self.blocks:
        blocksByID.put(censusBlock.getBLOCK_ID(), censusBlock)
        censusBlock.setCOLOR("UNOCCUPIED")

    #testing code...
    print "number of vector agents in the hash map: ", blocksByID.size()

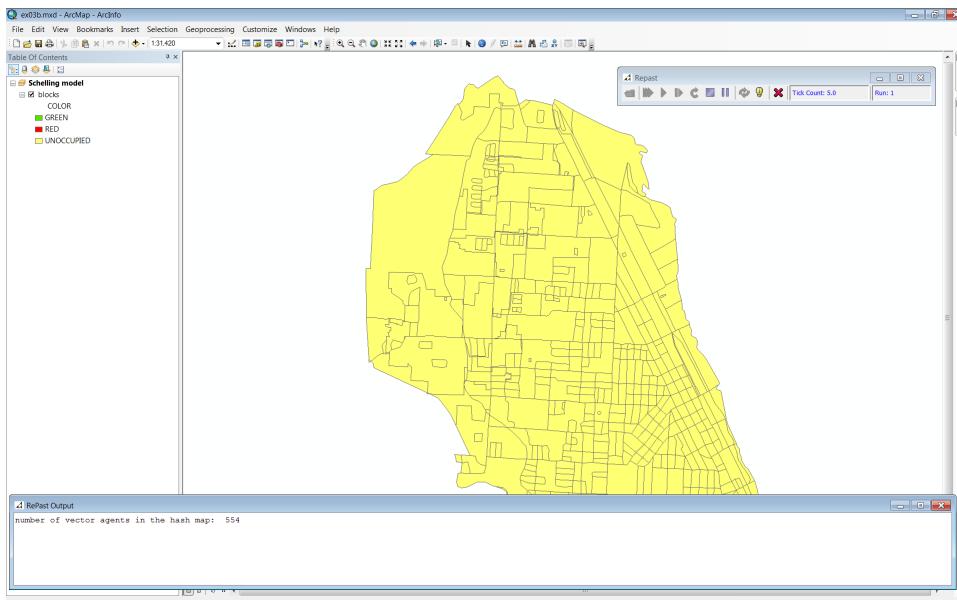
```

Size (in `blocksByID.size()`) is a property of the defined hash map.

13 Click OK.

14 Run the model.

Assuming your model's "Output to Console" property is set to true (if not, go to the Environment panel and set "Output to Console" to true), you should see the following output from the initialization and setup actions:



You should have 554 agents in the map. All the blocks should be assigned "UNOCCUPIED" (`censusBlock.setCOLOR ("UNOCCUPIED")`). You have not defined any process yet, so nothing else happens.

15 Close Agent Analyst by clicking File and then Exit. Save the model in C:\ESRIPress\AgentAnalyst\Chapter03\MyData as Chapter03Exercise2.sbp.

Initializing generic agents

Thus far in this chapter, you have worked only with polygon vector agents. However, the model needs the real decision makers—resident agents who change their location depending on their preferences.

2

3

4

Exercise 3c

In this exercise, you will learn how to define generic agents (an agent of no specific type) and assign them to polygons.

Load the ArcMap document

- 1 Start ArcMap. Navigate to C:\ESRIPress\AgentAnalyst\Chapter03 and open ex03c.mxd.

In the ArcMap display you will see the blocks shapefile. In ArcToolbox, the Chapter03Ex03c toolbox has been created for you.

Load the Agent Analyst model

- 2 Right-click the Chapter03Ex03c toolbox, point to New, and click Agent Analyst Tool.
- 3 Once the Agent Analyst window appears, click File and select Import. Do not save the changes when prompted.
- 4 Navigate to C:\ESRIPress\AgentAnalyst\Chapter03\Models and open ex03c.sbp. The Agent Analyst window appears.

Define the generic agent

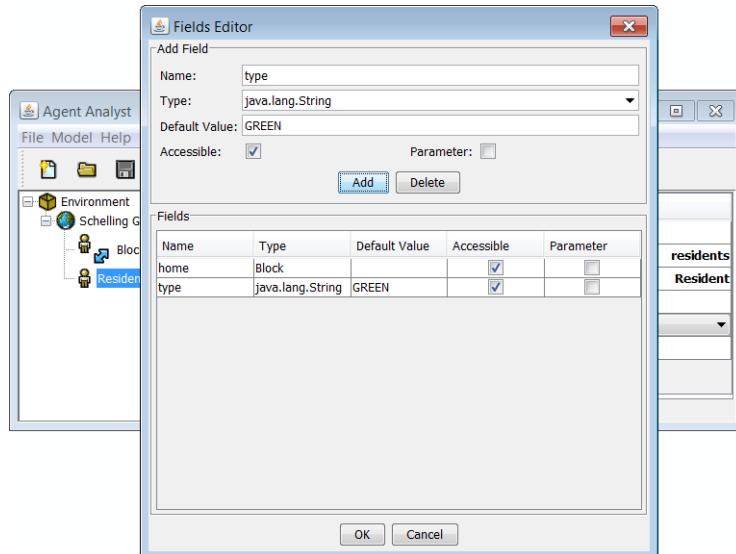
- 5 Click the Generic Agent icon on the Agent Analyst toolbar. In the Environment panel, click Schelling GIS to add the agent to the model. Rename the generic agent to **Resident** and its group to **residents**.

Your agents will have a field (or property) that will put them into one of two color groups: RED or GREEN. Create this field.

- 6 In the Environment panel, click Resident. In the Property panel, click the Edit button to the right of the Fields property to open the Fields Editor. Name the field (property) **type** and select java.lang.String as the data type. Type **GREEN** as its default value and select Accessible so that the field can be updated by other model components. Click Add to add the field.

Since your resident generic agents need a place to live, you also have to define their homes. Add a second property for the resident agent.

- 7 Define another field (property). Name it **home** of type Block. Do not set any default values. Notice that this field is the block vector agent. Make it accessible. Click Add, and then click OK to close the dialog box.



You have developed the structure for the resident agents. Now create the starting distribution for them.

- 8 In the Environment panel, click Schelling GIS. In the Property panel, click the Edit button next to the Actions property. Add a new action called **setupAgents**.

This action will execute code that generates resident agents, adds a color to them, and places them randomly in one of the blocks.

- 9 Type the following code in the Source panel to assign the generic agents (residents) to vector agents (polygons):

```
def setupAgents():
    for censusBlock as Block in self.blocks:
        if (Random.uniform.nextDoubleFromTo(0, 1) < self.density):
            resident = Resident()
            resident.setModel(self)
            resident.setHome(censusBlock)
```

```

# Randomly assign a COLOR of RED or GREEN
# to the resident of each Block
if (Random.uniform.nextDoubleFromTo(0, 1) < self.percentGreen):
    resident.setType("GREEN")
    censusBlock.setCOLOR("GREEN")
else:
    resident.setType("RED")
    censusBlock.setCOLOR("RED")

self.residents.add(resident)

self.writeAgents()

```

2

3

4

The Source panel code should look like the code in the following figure.

```

Source
def setupAgents():
    for censusBlock as Block in self.blocks:
        if (Random.uniform.nextDoubleFromTo(0, 1) < self.density):
            resident = Resident()
            resident.setModel(self)
            resident.setHome(censusBlock)

        # Randomly assign a COLOR of RED or GREEN
        # to the resident of each Block
        if (Random.uniform.nextDoubleFromTo(0, 1) < self.percentGreen):
            resident.setType("GREEN")
            censusBlock.setCOLOR("GREEN")
        else:
            resident.setType("RED")
            censusBlock.setCOLOR("RED")

        self.residents.add(resident)

    self.writeAgents()

```

The procedure starts by iterating over every block agent (`for censusBlock as Block in self.blocks:`). To ensure that a certain predefined vacancy rate is met (the `Density` parameter), a resident agent is assigned to a block only if the value randomly drawn from a range from 0 to 1 is below the threshold density rate (`if (Random.uniform.nextDoubleFromTo(0, 1) < self.density):`). If the condition is met, a resident agent is generated (`resident = Resident()`). Also, the block becomes the resident's home (`resident.setHome(censusBlock)`). Notice how the properties are being set using the `set` method followed by the property name.

Next, both the resident and its block are color-coded. Similar to the density rate, the percentage of residents that become green is estimated based on a value randomly drawn from a uniform distribution compared to the value of the `percentGreen` parameter (`if (Random.uniform.nextDoubleFromTo(0, 1) < self.percentGreen):`). Also, the color of the resident becomes the color of the block it occupies (`resident.setType("GREEN"); censusBlock.setCOLOR("GREEN")`). When a color is established, the resident is added to the residents collection (`self.residents.add(resident)`). Note that a resident and a block are in a one-to-one relationship: at most one resident can be assigned to a block, and a resident can "own" only one block.

Calling writeAgents at the end of the setupAgents action updates the value of the COLOR attribute in the blocks shapefile.

To implement the setupAgents action, you need to call it from the initAgents action.

- 10 Click OK. In the Environment panel, click Schelling GIS. In the Property panel, click the Edit button to the right of the Actions property. In the initAgents action Source panel, add the following line of code:

```
self.setupAgents()
```

- 11 To test the initialization actions you have defined, go back to the setupAgents action and add the following line of code. The Source panel should look like the figure that follows:

```
print "For density",self.density,"the number of residents is", →  
    self.residents.size()
```

```
self.writeAgents()  
print "For density",self.density,"the number of residents is",self.residents.size()
```

- 12 Click OK.

- 13 Run the model.

Observe that the blocks have now been randomly recolored. You should also have the information about the number of generated residents printed to the RePast Output window.

Close Agent Analyst

- 14 Click File and Exit. Save the model as C:\ESRIPress\AgentAnalyst\Chapter03\MyData\Chapter03Exercise3.sbp.

Creating neighborhoods for polygon agents

At this time, both blocks and residents are blind to their neighborhood. You need to add code that will designate the neighborhood of each census block. This exercise shows how to accomplish this task.

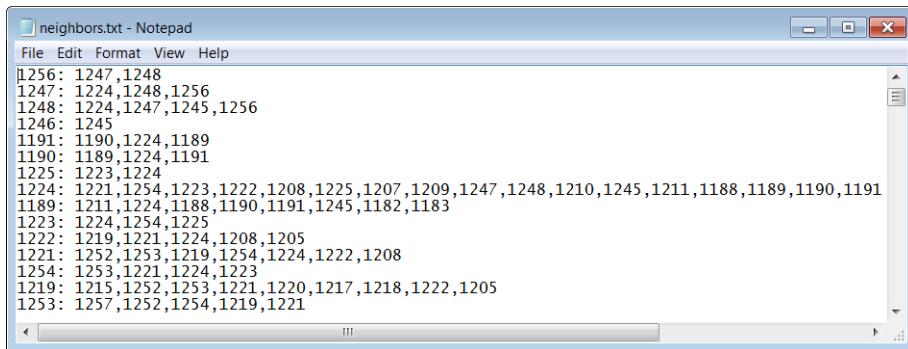
2

3

4

Exercise 3d

To generate neighborhoods, you will use a text file, neighbors.txt, located in the Data folder (C:\ESRIPress\AgentAnalyst\Chapter03\Data). The file provides a list of neighboring polygons for every block of interest and stores ID information about adjacent neighbors. This neighborhood file, shown in the following figure, will be imported later in this exercise.



Each line contains a selected block followed by a colon and then a list of its neighboring polygons represented using the BLOCK_ID identifiers. Neighbors are those polygons that share a boundary or a corner with a given block.

Polygon neighborhoods

Neighborhoods can be established in many different ways. The most common criterion uses topological adjacency, which assumes that neighbors are polygons that share a boundary with a target polygon. You can further define the neighborhood based on rook contiguity, where neighbors are polygons that share an edge with a given polygon, or queen contiguity, in which neighbors can share edges as well as vertices. The latter rule leads to a higher number of neighbors than the former one. Neighborhoods can also be defined based on distance from the target polygon.

As a guide, a sample custom Python arcpy script tool, Generate Neighbors (generate_neighbors.py), is provided in the Chapter03ex03d toolbox in the Chapter03\Toolboxes folder. (Some modifications may be necessary.) The file uses the queen contiguity to define the neighborhood. The input polygon shapefile must have an integer field with unique polygon IDs.

Load the ArcMap document

- 1 Start ArcMap. Navigate to C:\ESRIPress\AgentAnalyst\Chapter03. Open ex03d.mxd.

In the ArcMap display, you will see the blocks shapefile with the polygons symbolized as green, red, or unoccupied. In ArcToolbox, the Chapter03Ex03d toolbox has been created for you.

Load the Agent Analyst model

- 2 Right-click the Chapter03Ex03d toolbox, point to New, and click Agent Analyst Tool.
- 3 Once the Agent Analyst window appears, click File and select Import. Do not save the changes when prompted.
- 4 Navigate to C:\ESRIPress\AgentAnalyst\Chapter03\Models, select ex03d.sbp, and click Open. The Agent Analyst window appears.

You will start by defining a new field for the Block component to store the neighbors for each block agent.

Define a new field to store the neighbors for each block agent

- 5 In the Environment panel, click the Block vector agent. In the Property panel, click the Edit button to the right of the Fields property to open the Fields Editor.
- 6 Add a field, naming it **myNeighbors**, and type **java.util.ArrayList** as the data type. Make the field accessible.
- 7 Click Add, and then click OK to close the window.

Revisit the setupBlocks action discussed in exercise 3b

- 8 In the Environment panel, click Schelling GIS. In the Property panel, click the Edit button to the right of the Actions property to open the Schelling GIS model Actions Editor. Select the setupBlocks action from the Actions list. For convenience, comment out (by preceding each line of code with a #) or delete “testing code” from the current source code in the setupBlocks action definition (see the figure in step 10).

You need to invoke the myNeighbors variable for each block to initialize an empty container for the census block neighbors.

- 9 Add the following line of code to the censusBlock for loop discussed earlier:

```
censusBlock.setMyNeighbors (ArrayList())
```

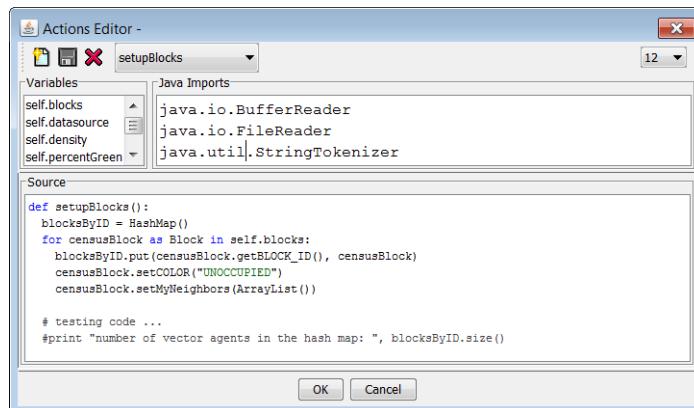
This initializes an empty container for the census block neighbors.

Set the neighbors for each block agent from the neighbors text file. To read data from neighbors.txt, you need to import three Java classes.

- 10** Type the following text in the Java Imports panel:

```
java.io.BufferedReader
java.io.FileReader
java.util.StringTokenizer
```

The Source panel code should look like the code in the following figure.



You need to import these Java classes to obtain specific functionality that is not contained within Agent Analyst. The first two classes are used to read data from the file using a line-by-line format. The third class splits each block line and extracts the identifiers.

- 11** Type the following code into the Actions Editor Source panel for the setupBlocks action:

```
# Read the neighbors file and set the neighbors of each block.
# The neighbors.txt file lists each block ID
# and the IDs of the neighboring blocks.
reader = BufferedReader(FileReader("C:/ESRIPress/AgentAnalyst/ ➔
➡ Chapter03/Data/neighbors.txt"))
line = reader.readLine()
while (line):
    tokenizer = StringTokenizer(line, ":")
    if (tokenizer.hasMoreTokens()):
        blockID = tokenizer.nextToken().trim()
        censusBlock = (Block)blocksByID.get(blockID)
        nghs = censusBlock.getMyNeighbors()
        nghIDs = tokenizer.nextToken().trim()
        nghTok = StringTokenizer(nghIDs, ",")
        while (nghTok.hasMoreTokens()):
```

```
ngh = blocksByID.get(nghTok.nextToken()) #from the HashMap above
nghs.add(ngh)
line = reader.readLine()
reader.close()
```

The complete setupBlocks source code defining the adjacent neighbors for a given polygon should look like the source code shown in the following figure.

```
Source
def setupBlocks():
    blocksByID = HashMap()
    for censusBlock as Block in self.blocks:
        blocksByID.put(censusBlock.getBLOCK_ID(), censusBlock)
        censusBlock.setCOLOR("UNOCCUPIED")
        censusBlock.setMyNeighbors(ArrayList())

    # Read the neighbors file and set the neighbors of each block.
    # The neighbors.txt file lists each block ID
    # and the IDs of the neighboring blocks.
    reader = BufferedReader(FileReader("C:/ESRIPress/AgentAnalyst/Chapter03/Data/neighbors.txt"))
    line = reader.readLine()
    while (line):
        tokenizer = StringTokenizer(line, ":")
        if (tokenizer.hasMoreTokens()):
            blockID = tokenizer.nextToken().trim()
            censusBlock = (Block)blocksByID.get(blockID)
            nghs = censusBlock.getMyNeighbors()
            nghIDs = tokenizer.nextToken().trim()
            nghTok = StringTokenizer(nghIDs, ",")
            while (nghTok.hasMoreTokens()):
                ngh = blocksByID.get(nghTok.nextToken()) # from the HashMap above
                nghs.add(ngh)
            line = reader.readLine()
    reader.close()
```

The code works as follows: the text file is opened (`reader = BufferedReader(FileReader ("C:/ESRIPress/AgentAnalyst/Chapter03/Data/neighbors.txt"))`), and the first line is read (`line = reader.readLine()`). Then each line of the `neighbors.txt` file is read (`while (line):`).

Each line from the `neighbors.txt` file is split at the colon, where the first token is the block ID (`blockID = tokenizer.nextToken().trim()`) and the second token is the list of neighbors (`nghIDs = tokenizer.nextToken().trim()`). Using the ID as the key, the block is found in the hash map table (`censusBlock = (Block)blocksByID.get(blockID)`)—remember the hash map is named `blocksByID`—and the block's `myNeighbors` variable is retrieved (`nghs = censusBlock.getMyNeighbors()`). The neighbors `BLOCK_ID` list is split using a comma as the separator (`nghTok = StringTokenizer(nghIDs, ",")`). Each of the neighboring blocks is then retrieved from the hash map and added to `myNeighbors`. Look at the following line from the code above:

```
censusBlock = (Block)blocksByID.get(blockID)
```

The `(classname)` object syntax is called casting and is specific to Java. It allows for converting an object of one data type to another. In this example, it informs the model that the returned hash map element is a block agent. To view the result, print the size of `myNeighbors` for selected blocks.

- 12** With the Actions Editor still open, add a new action called **printNeighbors**.

- 13** In the Source panel, type the following code to iterate over block agents and display the size of their myNeighbors attribute:

```
def printNeighbors():
    # for the first 10 blocks prints the number of neighbors to console
    i = 0
    for parcel as Block in self.blocks:
        id = parcel.getBlock_ID()
        count = parcel.getMyNeighbors().size()
        print "Block:", id, "has", count, "neighbors"
        i = i + 1
        if i > 10:
            break
```

- 14** Call this action from the setupBlocks action by adding this line at the end:

```
self.printNeighbors()
```

- 15** Click OK to close the Actions Editor.

- 16** Run the model, and click the Step button on the Repast toolbar.

For the first 11 blocks, the number of neighbors is printed to the RePast Output window, as shown in the following figure.



You have completed the Schelling GIS model initialization. It is time to put the decision-making process into play.

Close Agent Analyst

- 17** Click File and then Exit. Save the model as C:\ESRIPress\AgentAnalyst\Chapter03\MyData\Chapter03Exercise4.sbp.

Formulating the agent decision making

Agent decision making is the essence of agent-based modeling. In this exercise, you will extend the Schelling model with decision-making rules. You will build the basic code for decision making using the step action.

Exercise 3e

In the Schelling GIS model, only the resident generic agents make decisions. Block agents are just passive executors of residents' choices. Thus, the choice rules will be programmed in the Resident component.

Load the ArcMap document

- 1 Start ArcMap. Navigate to C:\ESRIPress\AgentAnalyst\Chapter03. Select ex03e.mxd, and then click Open.

In the ArcMap display, you will see the blocks shapefile with the polygons symbolized as green, red, or unoccupied. In ArcToolbox, the Chapter03Ex03e toolbox has been created for you.

Load the Agent Analyst model

- 2 Right-click the Chapter03Ex03e toolbox, point to New, and click Agent Analyst Tool.
- 3 Once the Agent Analyst window appears, click File and select Import. Do not save the changes when prompted.
- 4 Navigate to C:\ESRIPress\AgentAnalyst\Chapter03\Models and open ex03e.sbp.

Create the decision rules for the resident agents

- 5 In the Environment panel, click Resident. In the Property panel, click the Edit button to the right of the Actions property to open the Actions Editor for the resident agent.

- 6 Select the step action, and type the following code in the Source panel:

```
def step():
    # count the number of differently colored neighbors
    count = 0.0
    for nghBlock as Block in self.home.getMyNeighbors():
        state = nghBlock.getColor()
        if (not state.equals("UNOCCUPIED")):
            if (not state.equals(self.type)):
                count = count + 1

    # is the number of different neighbors tolerable?
    totalNeighbors = self.home.getMyNeighbors().size()
    if (count / totalNeighbors > self.model.getTolerance()):
        print "Block:",self.home.getBlock_ID(), "will move"
```

Note that the step action is explicitly devoted to agent decision making and has been automatically declared for you.

A resident agent's decision-making mechanism is very simple. The agent counts its occupied neighboring blocks that have a color different from itself. Then it calculates a ratio of the number of these differently colored neighbors to the total number of neighbors. If the ratio is higher than the tolerance parameter, the resident decides to move.

The actual code works as follows. For each block, it gets the list of the neighbors (`for nghBlock as Block in self.home.getMyNeighbors() :`). The color is determined (`state = nghBlock.getColor()`). If the color is not unoccupied (`if (not state.equals("UNOCCUPIED")) :`) or is not equal to its own type (`if (not state.equals(self.type)) :`), a counter is incremented (`count = count + 1`). The ratio of the number of different neighbors is calculated, and then it's determined whether it is unacceptable (`if (count / totalNeighbors > self.model.getTolerance()) :`).

- 7 Click OK to close the Actions Editor.
- 8 Make sure that the step action is scheduled for EVERY TICK in the resident's Schedule Editor.

9 Run the model.

In the RePast Output window, shown in the following figure, you will see which residents will move based on their tolerance and current neighborhood configuration.



Close Agent Analyst

- 10** Click File and Exit. Save the model as C:\ESRIPress\AgentAnalyst\Chapter03\MyData\Chapter03Exercise5.sbp.

Developing agent move rules

Your agents have made their movement decisions, but they do not yet know how to move. In this exercise, you will provide your residents with an additional migration action. You will also learn how to update agent field values.

2

3

4

Exercise 3f

Load the ArcMap document

- 1 Start ArcMap. Navigate to C:\ESRIPress\AgentAnalyst\Chapter03 and open ex03f.mxd.

In the ArcMap display you will see the blocks shapefile with the polygons symbolized as green, red, or unoccupied. In ArcToolbox, the Chapter03Ex03f toolbox has been created for you.

Load the Agent Analyst model

- 2 Right-click the Chapter03Ex03f toolbox, point to New, and click Agent Analyst Tool.
- 3 Once the Agent Analyst window appears, click File and select Import. Do not save the changes when prompted.
- 4 Navigate to C:\ESRIPress\AgentAnalyst\Chapter03\Models and open ex03f.sbp.

Create the move action

- 5 In the Environment panel, click Resident. In the Property panel, click the Edit button to the right of the Actions property to open the Actions Editor.
- 6 Add a new action, name it **move**, and then type the following code in the Source panel:

```
def move():
    # randomly look for a new home (block)
    blocks = self.model.getBlocks()
    newHome = (Block)blocks.getRandomItem()

    # is the selected home available?
    while (not newHome.getCOLOR().equals("UNOCCUPIED")):
        newHome = (Block)blocks.getRandomItem()

    self.home.setCOLOR("UNOCCUPIED") # moving out...
    self.home = newHome             # ...moving in
    self.home.setCOLOR(self.type)   # color-code the new home
```

The migration procedure starts from a random scanning of blocks. First, a block is randomly selected (`newHome = (Block)blocks.getRandomItem()`). It is determined whether the selected block is unoccupied (`while (not newHome.getColor().equals("UNOCCUPIED")) :`). If it is occupied, another block is randomly selected (`newHome = (Block)blocks.getRandomItem()`). When an unoccupied block is found, the resident abandons its current location (`self.home.setCOLOR("UNOCCUPIED")`), occupies the new location (`self.home = newHome`), and changes its color to the resident's color (`self.home.setCOLOR(self.type)`).

- 7 To initiate the action, type the following line of code in the step action:

```
self.move()
```

- 8 Additionally, comment out the print statement as shown in the following figure:

```
# is the number of different neighbors tolerable?  
totalNeighbors = self.home.getMyNeighbors().size()  
if (count / totalNeighbors > self.model.getTolerance()):  
    #print "Block:",self.home.getBLOCK_ID(), "will move"  
    self.move()
```

- 9 Click OK to close the Actions Editor.

- 10 Run the model, and observe the emerging segregation.

Close Agent Analyst

- 11 Click File and Exit. Save the model as C:\ESRIPress\AgentAnalyst\Chapter03\MyData\Chapter03Exercise6.sbp.

Using multiple decision rules

In the final exercise in this chapter, you will introduce an alternative decision rule that drives the residential behavior. You will also introduce a Boolean model parameter that allows the user to pick one of the two decision rules as the decision-making mechanism.

Exercise 3g

Load the ArcMap document

- 1 Start ArcMap. Navigate to C:\ESRIPress\AgentAnalyst\Chapter03 and open ex03g.mxd.

In the ArcMap display you will see the blocks shapefile with the polygons symbolized as green, red, or unoccupied. In ArcToolbox, the Chapter03Ex03g toolbox has been created for you.

Load the Agent Analyst model

- 2 Right-click the Chapter03Ex03g toolbox, point to New, and click Agent Analyst Tool.
- 3 Once the Agent Analyst window appears, click File and select Import. Do not save the changes when prompted.
- 4 Navigate to C:\ESRIPress\AgentAnalyst\Chapter03\Models and open ex03g.sbp.

Create the model parameter to determine which rule set to use

- 5 In the Environment panel, click Schelling GIS. In the Property panel, click the Edit button to the right of the Fields property.
- 6 Add a new field, name it **quantity**, and give it a boolean type with a default value set to **true**. Make it a parameter.
- 7 Click Add, and then click OK to close the window.

Create the alternative decision-making action, which is based on total area of the neighbors not the number of neighbors

2

3

4

- 8 Add a new action for the resident agent, name it **chooseArea**, and then type the following code:

```
def chooseArea():
    # this decision rule focuses on area of differently
    # colored neighbors rather than their count
    colorArea = 0.0      # area of differently colored neighbors
    totalArea = 0.0       # total neighborhood area
    for nghBlock as Block in self.home.getMyNeighbors():
        totalArea = totalArea + nghBlock.getAREA()
        state = nghBlock.getColor()
        if (not state.equals("UNOCCUPIED")):
            if (not state.equals(self.type)):
                colorArea = colorArea + nghBlock.getAREA()

    toleranceArea = self.model.getTolerance() * totalArea

    # is the area of different neighbors tolerable?
    if (colorArea > toleranceArea):
        self.move()
```

So far, the decision to move has been based on the ratio of the number of differently colored neighbors to the total number of agents. The new decision rule calculates the ratio of the area of differently colored polygons (`colorArea = colorArea + nghBlock.getAREA()`) to the total area of neighboring blocks (`totalArea = totalArea + nghBlock.getAREA()`). It then compares this ratio with a threshold tolerance area to decide whether to migrate (`if (colorArea > toleranceArea) :`).

Create a new action named `chooseQuantity` and copy the code from the step action (in which the decision rule is based on the number of different neighborhoods) and paste it into the new action.

- 9 With the Actions Editor still open, go to the step action and copy its content.

- 10 Add a new action, called **chooseQuantity**, and paste the copied code in the Source panel:

```
def chooseQuantity():
    # count the number of differently colored neighbors
    count = 0.0
    for nghBlock as Block in self.home.getMyNeighbors():
        state = nghBlock.getCOLOR()
        if (not state.equals("UNOCCUPIED")):
            if (not state.equals(self.type)):
                count = count + 1
    # is the number of different neighbors tolerable?
    totalNeighbors = self.home.getMyNeighbors().size()
    if (count / totalNeighbors > self.model.getTolerance()):
        #print "Block:",self.home.getBLOCK_ID(), "will move"
        self.move()
```

Next, create the code to call the appropriate decision-making action based on how the parameter is set. This code will be placed in the step action for the resident agents.

- 11 Go back to the step action, delete its current code, and type the following:

```
def step():
    # which decision rule?
    if (self.model.getQuantity() == true):
        print "Choice is based on quantity"
        self.chooseQuantity()
    else:
        print "Choice is based on area"
        self.chooseArea()
```

- 12 Click OK to close the Actions Editor.

- 13 Run the model.

Note that the Schelling GIS Settings window has been updated with the new Quantity parameter.

If you clear the check box for the Quantity parameter before running the model, you will use the chooseArea action instead of the default chooseQuantity action.

- 14 Exit Agent Analyst and save the final model as C:\ESRIPress\AgentAnalyst\Chapter03\MyData\Chapter03Exercise7.sbp.

GIS data, fields, and actions dictionaries

Table 3.1 Data dictionary of GIS datasets

| Dataset | Data type | Description |
|---------|-----------|---------------|
| blocks | shapefile | Census blocks |

Table 3.2 Fields dictionary for the Schelling model

| Field | Data type | Description |
|----------------------------|---------------|--|
| Schelling GIS model | | |
| datasource | string | Path to the blocks shapefile |
| Density | double | Percent of occupied (nonblank) blocks |
| tolerance | double | The maximum tolerable number of differently colored neighbors as a fraction of all neighbors |
| percentGreen | | The ratio of green agents to the total number of agents |
| quantity | boolean | The choice between a decision rule based on a count of different neighbors and a decision rule based on an area of different neighbors |
| Blocks | vector agent | Vector agents representing locations to live |
| residents | generic agent | Model decision makers |
| Block | | |
| BLOCK_ID | string | Unique block identifier |
| COLOR | string | Block color |
| myNeighbors | arraylist | A collection of neighboring blocks |
| Model | GIS Model | Schelling GIS model |
| Resident | | |
| Type | string | The color of the resident |
| Home | block | The block currently assigned to the resident |
| Model | GIS Model | Schelling GIS model |

Table 3.3 Actions dictionary for the Schelling model

| Declared actions | Description |
|----------------------------|---|
| Schelling GIS model | |
| initAgents | Invoke agent initialization |
| updateDisplay | Update ArcMap display |
| writeAgents | Save current block data to the shapefile |
| setupBlocks | Initialize blocks and their neighborhood |
| setupAgents | Initialize residents and assign a color to them |
| printNeighbors | Print block's neighbor count to console |
| Resident | |
| step | Select the decision rule based on the Quantity parameter |
| chooseQuantity | Check color dissimilarity within the neighborhood based on quantity |
| chooseArea | Check color dissimilarity within the neighborhood based on area |
| move | Find a new location and move there |

2

3

4

References

- Schelling, T. C. 1969. "Models of Segregation (in Strategic Theory and Its Applications)." *The American Economic Review* 59 (2). Papers and Proceedings of the Eighty-first Annual Meeting of the American Economic Association. (May): 488–93.
- Benenson, I., and P. M. Torrens. 2004. *Geosimulation: Automata-based Modeling of Urban Phenomena*. Chichester, West Sussex, England: Wiley.

Data source: http://www.esri.com/data/download/census2000_tigerline/index.html.

Section 2: The basics of points, polygons, and rasters in agent-based modeling

Chapter 4

Querying and changing raster data with Agent Analyst

by Derek T. Robinson and Daniel G. Brown

- ◆ Running the model
- ◆ Loading raster data into Agent Analyst
- ◆ Accessing raster data and raster properties with Agent Analyst
- ◆ Selecting a random location in a raster
- ◆ Reading raster pixel values
- ◆ Building the SOME model
- ◆ Copying raster data
- ◆ Running computational experiments with the SOME model

In the previous chapters you used point and polygon vector agents. Point and polygon data is primarily used to represent discrete objects or entities in space. As you saw in chapter 2, point agents are used to represent individuals, animals, and firms. As you saw in chapter 3, polygon agents are used to represent political zones, land-use types, and large buildings. In contrast to discrete objects represented by point and polygon data, raster data is typically used to represent continuous surfaces. Examples of processes represented by raster agents include the following:

- Urban growth
- Forest fire spread
- Seed dispersal and forest growth
- Deforestation
- Landslides

This chapter provides an introduction to creating an agent-based model in Agent Analyst using raster data. Using a simple model of urban growth you will learn the fundamentals of querying a raster dataset to get information from a single location and from the neighbors of that location. You will learn how to write values to a raster and save the changes to the raster dataset for further analyses. Lastly, you will use the ArcGIS Spatial Analyst extension to create new raster data and use that data in combination with Agent Analyst in ArcGIS.

The intent of the chapter is to introduce you to agent-based modeling (ABM) principles using raster data, not how to create an urban change model. If the rules are not satisfactory, they can be changed.

The modeling scenario

The areas around the city of Ann Arbor, Michigan, have been experiencing a high rate of residential development. The city of Ann Arbor and its adjoining townships are trying to understand the patterns of residential development and how household preferences may alter development patterns. In an effort to better understand the effects of household preferences for nearness to urban service centers versus environmental aesthetic quality on development patterns, the city of Ann Arbor and surrounding councils requested the development of an agent-based model. The model will allow these decision makers the ability to change the agent preference weights and visualize how the change alters the interaction between nearness to urban centers with preference for aesthetic quality, which affects the number of settlements beyond a preferred urban growth region. In addition to residential location, the model needs to incorporate the attraction that residential developments have on commercial and employment (i.e., urban) services and the reciprocal attraction of residential developments to these services.

The model is applied to landscape data from the Ann Arbor region. It will be developed in several phases outlined in the rest of this chapter.

Background information

Since models are simplified representations of reality, it is important to start with very simple models that can help us acquire intuition about the phenomenon being studied.

The process of modeling is typically iterative, since new intuitions about the studied phenomenon often guide model improvements and subsequent changes. Starting with a simple model provides a strong foundation for expansion and can help us to better understand our assumptions about the modeled system. In this chapter, we are interested in understanding the causes and effects of residential sprawl at the urban-rural fringe. Specifically we are interested in the effects of individual decision making on aggregate settlement patterns. Using an urban growth model, named SLUCE's Original Model for Experimentation (SOME), we consider the question of how different behavioral assumptions and representations of the landscape affect the extent of sprawl.

In this chapter, using Agent Analyst, you will investigate how behavioral and landscape attributes contribute to sprawl. The size of the landscape is 151 by 152 cells, and each cell has a 65-meter resolution. An initial service center has been placed in the center of this grid, represented by the red cell. Residents are displayed as black squares. In total, the model is run for 340 time steps. At each time step 10 new residential households settle in the landscape to create a total of 3,400 households at the end of the model run, with approximately 15 percent of landscape developed. Residents prefer locations that are close to service centers and have high aesthetic quality. The raster named “aesthetic” displays the level of aesthetic quality: the lighter the cell color, the higher the aesthetic appeal of that location.

The landscape

The aesthetic quality of each cell is set at the beginning of each model run using a parameter that specifies the name of the map. We provide two maps: “aesthetic,” which is a map of aesthetic quality of Scio Township derived and used by Robinson and Brown (2009) in a more complex model of urban growth, and a random assignment of aesthetic-quality values. You may notice that the aesthetic map has an irregular boundary. A map with an irregular boundary is utilized so that you will similarly be able to use irregular boundary data. Later in this chapter, you will be asked to create a homogeneous aesthetic-quality map by using the ArcGIS Spatial Analyst Raster Calculator. Feel free to create and use your own aesthetic-quality maps by changing the parameter `aestheticFName`. More details on how to do this will be provided in the exercises that follow.

Residential household agents

Residential household agents locate themselves in the cells that maximize their *utility*. However, they get to sample only some small number of cells before selecting one. The parameter `numTests` determines how many cells each resident gets to sample and, in this version of the model, we set the default to 15. By providing the resident agents incomplete information about the available cells in the landscape, the agents face a bounded choice on where to locate.

2

3

4

Resident agents evaluate utility by calculating a score for each sampled cell that is a function of that cell's distance to the nearest service center, that cell's aesthetic quality, and the importance the resident agent places on distance and aesthetic quality. In this case, all resident agents prefer cells that are closer to a service center (i.e., $\beta_d = 1$) and have higher aesthetic quality (i.e., $\beta_a = 1$). The utility function has the following form:

$$\text{utility of a cell to resident} = (1 - |\beta_a - \text{aesth}|^{\alpha_a}) * (1 - |\beta_d - \text{scdist}|^{\alpha_d})$$

The distance of each cell to the nearest service center (scdist) is calculated each time a new service center enters. The importance resident agents place on aesthetic quality (α_a) and distance to service centers (α_d) is set by the user as a parameter in the SOME model. The total of the two alpha values should equal one; however, if what you specify does not, the model will automatically scale the values to sum to one. If α_d is given a value of 1.0 and α_a is given a value of 0.0, then all emphasis is on distance, with no consideration of aesthetic quality. Conversely, a value of 0.0 for α_d forces the resident agents to consider only aesthetic quality, with no consideration of service center distance. All resident agents have the same utility function and parameter values in this version of the model.

Service-center agents

Each time a new set of 100 resident agents locate in the landscape, a new service center establishes itself by selecting a random cell from among the immediate neighbors of the last resident agent placed. If these cells are full, the service center looks at the next distant set of adjacent cells and so on until it acquires a location. This is a relatively crude approach to locating service centers, but it captures our intuition that services tend to locate near people, who can serve as both customers and employees. This process can reinforce the location decisions of early residents as a form of positive feedback.

Running the model

The following exercises will provide you with the basic skills and knowledge of how to create an agent-based model using raster data in the Agent Analyst environment. You will learn how to display, query, and change the values within a raster. You will also learn how to create a simple model of urban growth that can be expanded in many ways to represent additional complexity. Several researchers from the University of Michigan have done just that, and for those interested we recommend the following publications: Brown et al. 2004, Brown et al. 2005a, Brown et al. 2005b, Brown and Robinson 2006, Brown et al. 2008, Robinson and Brown 2009, and Robinson et al. in press.

2

3

4

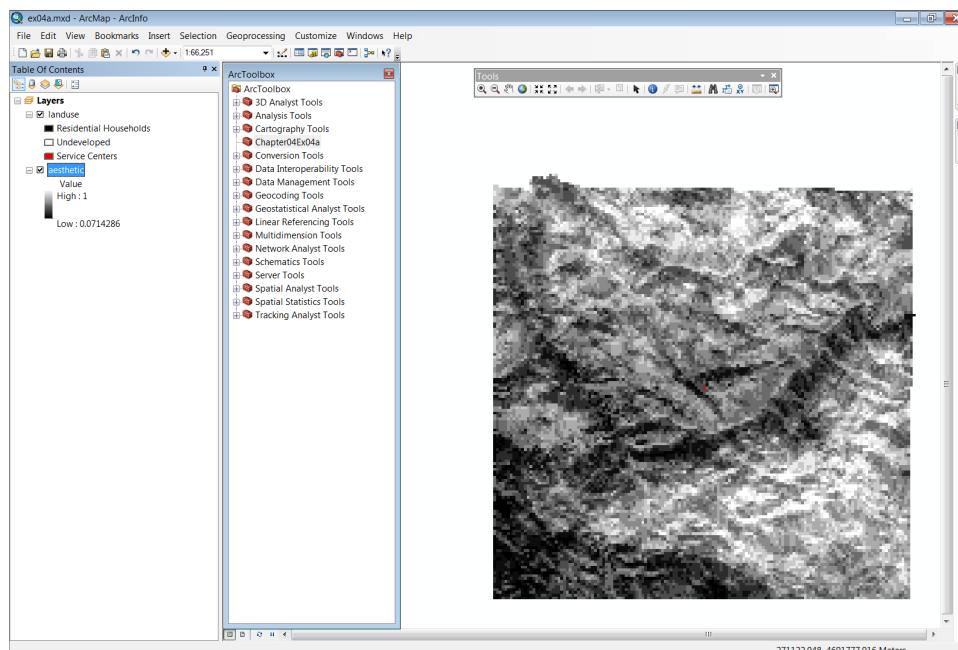
Exercise 4a

In chapter 2 you learned how to install Agent Analyst and how to load and start a model. As a first exercise here, you will view what the SOME model looks like when it is completed.

Load the ArcMap document

- 1 Start ArcMap. Using the “Browse for more...” option, navigate to C:\ESRIPress\AgentAnalyst\Chapter04. Select ex04a.mxd, and click Open.

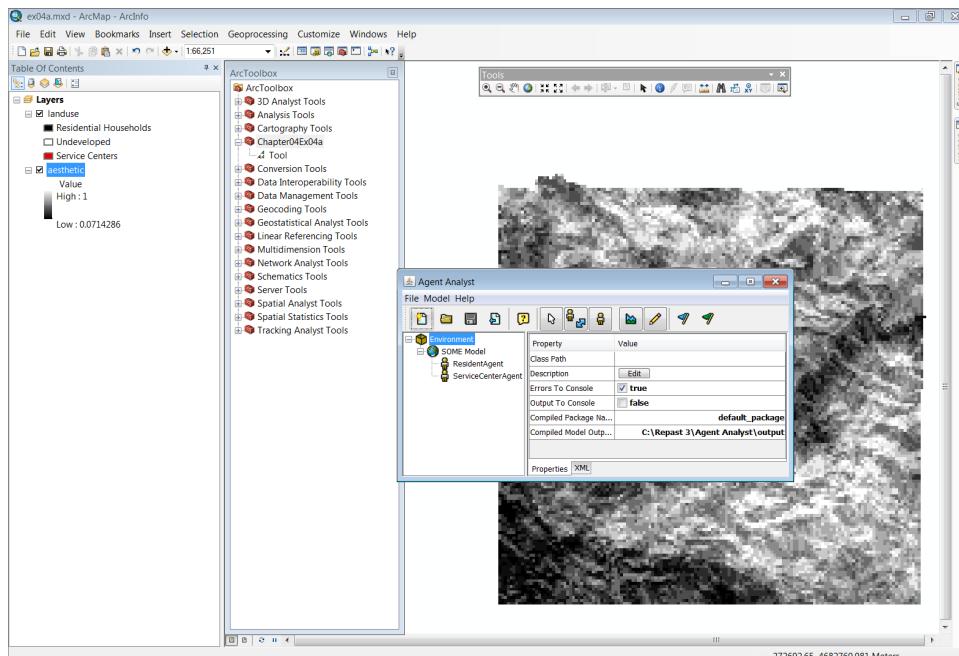
The map document opens. In the ArcMap display you will see in the Table Of Contents pane the land-use and aesthetic layers that will be used in the SOME model. In ArcToolbox, the Chapter04Ex04a toolbox has been created for you.



The ArcMap table of contents contains two layers. The first is the land-use layer (i.e., landuse), which is used to represent land that is developed or nondeveloped. The second is the aesthetic-quality layer (i.e., aesthetic), which represents the beauty and ecological appeal of locations within the landscape. If the aesthetic-quality layer is not rendered as shown in the preceding image, right-click the layer name (i.e., aesthetic), select Properties, select the Symbology tab, and then select the Stretched option in the list on the left. (If you receive the warning “Statistics do not exist. Do you want to compute Statistics?,” click Yes.) Next, move down to the Stretch frame in the same window, and from the Type box select Standard Deviations (it may already be selected). Click Apply. Your screen should now look like the preceding image. Click OK.

Open Agent Analyst and load the SOME model through the geoprocessing environment

- 2 Right-click the Chapter04Ex04a toolbox, point to New, and select Agent Analyst Tool.
- 3 Once the Agent Analyst window appears, click File and select Import. Do not save the changes when prompted.
- 4 Navigate to C:\ESRI\Press\AgentAnalyst\Chapter04\Models, select ex04a.sbp, and click Open. The Agent Analyst window appears.

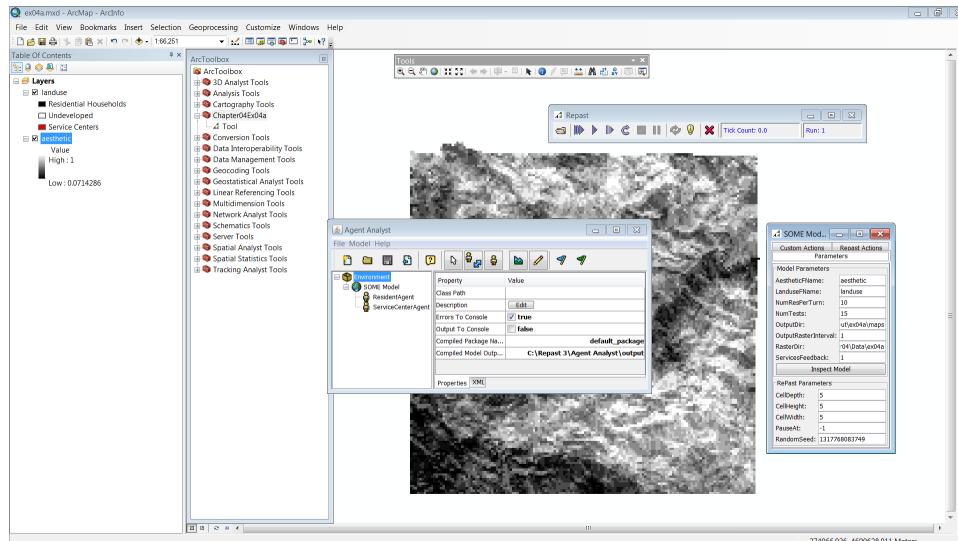


The Environment panel displays the SOME model as well as the two agent types used in the SOME model and described earlier in this chapter. The agents are generic agents and

are not tied to any specific location in either of the raster data layers. Unlike the polygon agents described in chapter 3, the agents described in this chapter will evaluate different locations within the raster landscape before altering the values within that landscape.

Observe the pattern of residential development created by the SOME model

- 5** Click the Run button on the Agent Analyst toolbar. The Repast toolbar and the SOME Model Settings window are displayed.



The Parameters tab of the SOME Model Settings window shows eight model parameters as described in the data dictionary at the end of this chapter. The default parameter values are those specified in the background information and model description at the beginning of this chapter.

- 6** Click the Initialize button on the Repast toolbar.

Note: If you receive an error, chances are the output rasters for the exercise already exist. Open ArcCatalog from the ArcMap Windows menu or by clicking the Catalog button, navigate to C:\ESRIPress\AgentAnalyst\Chapter04\Output\ex04a, and delete all the files in the maps subfolder (you may need to close ArcMap to delete the files). Run the model again.

Upon initialization, the model prints several statements to the RePast Output window that describe the actions being performed. These actions include loading the aesthetic-quality and landuse raster data, copying the original data to preserve the original data from manipulation, and creating a service center at the center of the landscape.

2

3

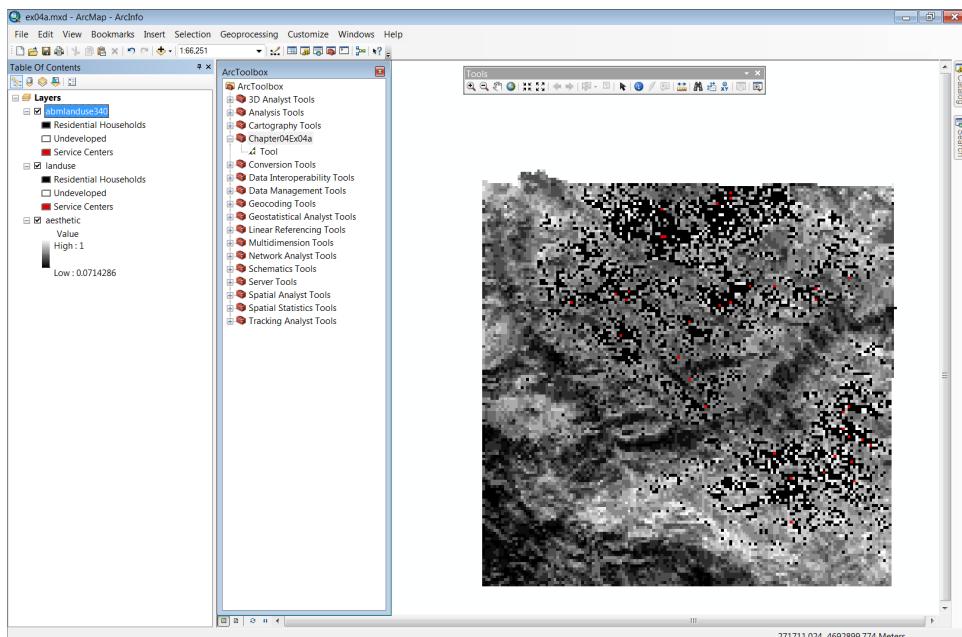
4

Run the SOME model

- 7 Click the Start button on the Repast toolbar.

Additional progress statements will print to the RePast Output window to verify the model's behavior. Errors are output to the command window (you can ignore the VATBuild errors that appear in the window; if you want, you can minimize the command window). The model will run for 340 time steps and then pause. You can monitor the tick count on the Repast toolbar. You may pause or stop the model at an earlier time step if you want to or press the Start or Step button to continue the model run. At 340 time steps, the development pattern produced by the SOME model may look something like the pattern shown in the following image; however, there are random processes within the SOME model, and it is highly unlikely that any two model runs will be identical. To view the output of your run, click the Add Data button and add one of the new land-use data layers the model created in your output folder, located in C:\ESRIPress\AgentAnalyst\Chapter04\Output\ex04a\maps. The output file name will be of the form abmlanduse##, where ## is replaced by the time step number at which the output was created. In the following image, the output from time step 340 is displayed by adding the data layer named abmlanduse340. This is a typical output of the SOME model showing residential development pattern and service-center locations displayed over the aesthetic-quality layer.

Note: If your output map obscures the aesthetic layer, right click the 0 (undeveloped) layer, and then click No Color.



- 8 Close Agent Analyst by clicking File and then Exit. Do not save any changes when prompted.

Loading raster data into Agent Analyst

The previous exercise provided you with a visualization of what the model looks like when complete. In the following exercises, you will build various components of the model so that when you have completed this chapter you will have the knowledge and experience to build a raster data model. The first step to using raster data in Agent Analyst is learning to load raster data into Agent Analyst.

2

3

4

Exercise 4b

In this exercise you will load (link to and register with) your raster data into Agent Analyst.

Load the ArcMap document

- 1 Start ArcMap if you closed it after the previous exercise. Navigate to C:\ESRIPress\AgentAnalyst\Chapter04. Open ex04b.mxd.

The landuse raster is displayed over the aesthetic raster. In ArcToolbox, the Agent Model Chapter04Ex04b toolbox has been created for you. If the aesthetic raster displays as all gray, change its symbology to Stretched (compute Statistics if asked) with Type set to Standard Deviations (see step 1 of exercise 4a).

Load the Agent Analyst model

- 2 Right-click the Chapter04Ex04b toolbox, point to New, and select Agent Analyst Tool.
- 3 Once the Agent Analyst window appears, click File and select Import. Do not save the changes when prompted.
- 4 Navigate to C:\ESRIPress\AgentAnalyst\Chapter04\Models and open ex04b.sbp.

In the SOME model, the loadRaster action was designed to isolate the process of linking and registering GIS raster data with your Agent Analyst model.

View the loadRaster action

- 5 In the Environment panel, click SOME Model. In the Property panel, click the Edit button to the right of the Actions property.
- 6 In the Actions Editor you will see five actions that have already been created for the SOME model.

You will now create the action to load the raster data into the model.

- 7 From the Actions drop-down list, select loadRaster. Type the following code in the Source panel (use the existing code in the action as a reference when adding the following code).

```
def loadRaster():
    print "Loading Rasters..."

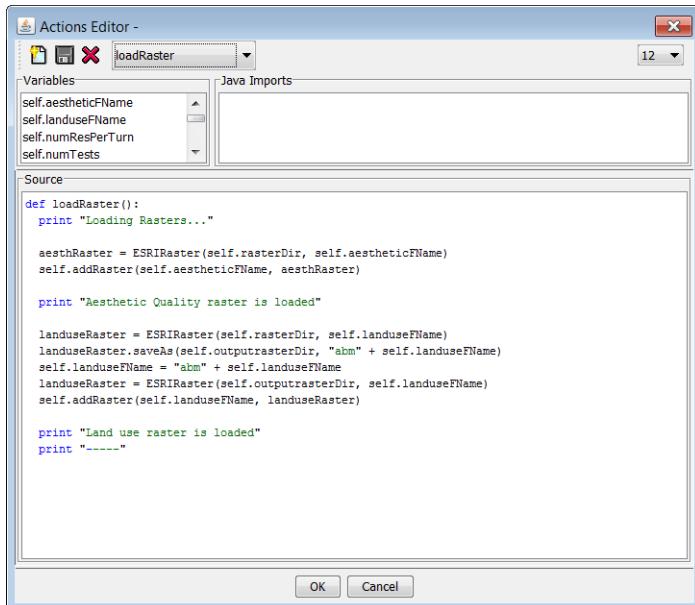
    aesthRaster = ESRIRaster(self.rasterDir, self.aestheticFName)
    self.addRaster(self.aestheticFName, aesthRaster)

    print "Aesthetic Quality raster is loaded"

    landuseRaster = ESRIRaster(self.rasterDir, self.landuseFName)
    landuseRaster.saveAs(self.outputDir, "abm" + self.landuseFName)
    self.landuseFName = "abm" + self.landuseFName
    landuseRaster = ESRIRaster(self.outputDir, self.landuseFName)
    self.addRaster(self.landuseFName, landuseRaster)

    print "Land use raster is loaded"
    print "----"
```

The code in the Source panel should match the code in the following image.



The action loads the aesthetic and landuse rasters into the model.

This code first prints progress information to the console so that you know that the model is loading the raster data (print "Loading Rasters . . ."). Next, a raster object is

created with reference to the aesthetic-quality raster dataset (`aesthRaster = ESRIRaster(self.rasterDir, self.aestheticFName)`). The format of the line of code is:

2
3
4

```
<raster variable name> = ESRIRaster(<raster directory location>,
<raster filename>)
```

Then you register the object with the SOME model (`self.addRaster(self.aestheticFName, aesthRaster)`). ESRIRaster is an Agent Analyst object that connects the simulation libraries to the actual raster, which you can see in the ArcMap table of contents or in ArcCatalog. The line of code tells the model to add this raster object to itself so that it can use the raster and agents within the model can access this raster. The format of this line of code is

```
self.addRaster(<raster file name>, <raster variable name>)
```

The second progress message indicates the aesthetic raster is loaded (`print "Aesthetic Quality raster is loaded"`).

Next, the reference for the landuse raster is created (`landuseRaster = ESRIRaster(self.rasterDir, self.landuseFName)`). A copy of the landuse raster is made because you will be making land-use changes and you do not want to corrupt the original data layer. To do this you use `landuseRaster.saveAs(self.outputDir, "abm" + self.landuseFName)`, which has the format `<raster object>. <raster method saveAs> (<raster directory location to store file>, <raster file name>)`. Then the model parameter that pointed to the old raster file name is updated to the new raster (`self.landuseFName = "abm" + self.landuseFName`). Next the `landuseRaster` variable is set to the new raster the same way you did earlier for the original land-use data (`landuseRaster = ESRIRaster(self.outputDir, self.landuseFName)`). This line of code works the same as it did earlier because you altered the parameter variable `self.landuseFName` to point to the new data layer. Then the raster is added to the model by using the same approach as you did for the aesthetic raster (`self.addRaster(self.landuseFName, landuseRaster)`). Additional progress messages are added at the end to indicate the rasters have been successfully loaded (`print "Land use raster is loaded"`) and a spacing blank line created (`print "-----"`).

The location of the rasters could have been explicitly provided as “C:\ESRIPress\AgentAnalyst\Chapter04\Data”. Similarly, the names of the individual raster files could have been explicitly listed as a string (e.g., “landuse” and “aesthetic”). However, we created parameter values at the model level to hold this information so that you can change this information at run time and reference those variables from elsewhere in the model and not need to know the names of the individual rasters.

Run the SOME model

- 8 Click the Compile button in Agent Analyst to ensure there are no errors in the SOME model. Then click the Initialize button.

Note: If you receive an error, chances are the output rasters for the exercise already exist. Open ArcCatalog and navigate to C:\ESRIPress\AgentAnalyst\Chapter04\Output\ex04b and delete all the files in the maps subfolder (you may need to close ArcMap to delete the files). Then run the model again.

- 9 Click the Start button on the Repast toolbar.

Because you have not yet changed any of the model parameters, the pattern of development should look similar to the pattern you obtained from exercise 4a.

- 10 Close Agent Analyst by clicking File and then Exit. Do not save any changes when prompted.

Accessing raster data and raster properties with Agent Analyst

2

3

4

In this exercise, you will learn how to initiate the loadRaster action you created in the previous exercise, access a raster dataset from the model, determine the number of rows and columns it has, and change a value in the raster. Your objective in performing these tasks is to create the initial conditions for the model to run. These consist of loading the raster data and creating a service-center agent at the center of the raster landscape.

Exercise 4c

Load the ArcMap document

- 1 Start ArcMap if it is not already running. Navigate to C:\ESRIPress\AgentAnalyst\Chapter04. Select ex04c.mxd, and then click Open.

The landuse raster is displayed over the aesthetic raster. In ArcToolbox, the Chapter04Ex04c toolbox has been created for you. If the aesthetic raster displays as all gray, change its symbology to Stretched (compute Statistics if asked) with Type set to Standard Deviations (see step 1 of exercise 4a).

Load the Agent Analyst model

- 2 Right-click the Chapter04Ex04c toolbox, point to New, and select Agent Analyst Tool.
- 3 Once the Agent Analyst window appears, click File and select Import. Do not save the changes when prompted.
- 4 Navigate to C:\ESRIPress\AgentAnalyst\Chapter04\Models and open ex04c.sbp.

An action within the SOME model called initAgents, which is short for initialize agents, has already been created for you. You will code the initAgents action to create a new service center, tell that service center to set its model reference to the SOME model, and then add that service-center agent to the group of service-center agents maintained by the SOME model.

- 5 In the Environment panel, click SOME Model. In the Property panel, click the Edit button to the right of the Actions property. From the Actions drop-down list, select initAgents.
- 6 Type the following code in the Source panel, starting with the blank line immediately below the `def initAgents()` statement (use the existing code in the action as a reference when adding the code that follows).

```

def initAgents():
    self.loadRaster()

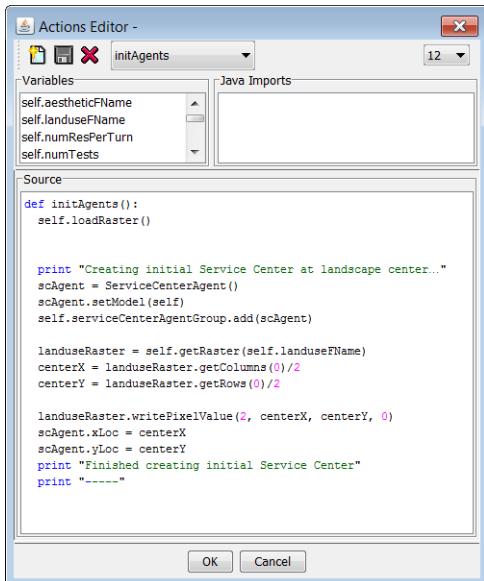
    print "Creating initial Service Center at landscape center..."
    scAgent = ServiceCenterAgent()
    scAgent.setModel(self)
    self.serviceCenterAgentGroup.add(scAgent)

    landuseRaster = self.getRaster(self.landuseFName)
    centerX = landuseRaster.getColumns(0)/2
    centerY = landuseRaster.getRows(0)/2

    landuseRaster.writePixelValue(2, centerX, centerY, 0)
    scAgent.xLoc = centerX
    scAgent.yLoc = centerY
    print "Finished creating initial Service Center"
    print "-----"

```

The code in the Source panel should match the code in the following image. This completes the initAgents action in the SOME model.



In the previous exercise, you created the loadRaster action. The code you just added calls the loadRaster action from within the initAgents action (`self.loadRaster()`). The `self.loadRaster()` action is defined as an action because of the parentheses that follow its name. You could translate this line of code as saying “SOME model perform the `loadRaster()` action now.” You could have placed the code for the `loadRaster` action directly in the `initAgents` action, but it is better practice, and easier to debug, when you logically separate the code into specific actions.

Next, a service-center agent is created (`scAgent = ServiceCenterAgent()`) and told that it is affiliated with the model associated with the current Actions Editor (`scAgent.setModel(self)`). The service-center agent is added to the group of all service centers that will exist over the duration of a model run using the code (`self.serviceCenterAgentGroup.add(scAgent)`).

2

3

4

The third block of code in the `initAgents` action gets a reference to the landuse raster to determine the size of the raster so that a service-center agent can be placed in the center of the landscape. Because you already referenced the landuse and aesthetic-quality rasters to the SOME model in exercise 4b, you can now obtain a direct reference to the landuse raster from the SOME model (`landuseRaster = self.getRaster(self.landuseFName)`). This code tells the model to access the raster using the code on the right side of the equal sign and store it in the variable on the left side of the equal sign. If you remove `self.landuseFName` from within the parentheses, you are calling an action just as you did when you called the `loadRaster` action. However, this action requires an input argument (i.e., the name of the raster to get) and returns a raster object, which will be stored in the variable `landuseRaster`. Because you are working in the actions associated at the model level, you can reference the model as “self.” The `landuseFName` parameter is a parameter of the model, and in exercise 4b you assigned the landuse raster to it.

If `landuseFName` was not a parameter of the model, you would need to directly call the raster name. Therefore, you would need to replace the previous line of code with one of two options. The first is

```
landuseRaster = self.getRaster("landuse")
```

The disadvantage of this option is that to run the model using a different or second landuse raster you have to rename the second raster to “landuse” or go into the code and change the text between the quotation marks to match the file name you are using.

The second available option is

```
landuseRaster = ESRIRaster(self.rasterDir, self.landuseFName)
```

However, this type of code requires more time and an extra variable reference.

The next two lines of code are used to get information on the dimensions of the raster so that you can determine the coordinates of the center of the raster and place a service-center agent at that location:

```
centerX = landuseRaster.getColumns(0)/2  
centerY = landuseRaster.getRows(0)/2
```

The ESRIRaster object has two actions that get information about the properties of a raster. The first action is `getColumns(<band>)`, which returns the number of columns in the raster, and the second is `getRows(<band>)`, which returns the number of rows in the raster. Because rasters can be composed of multiple bands (e.g., red, green, and blue are the three bands that make up true color rasters), actions that work on raster objects typically require reference to the specific band you want to work with. Because there are no additional bands on either the landuse or aesthetic-quality raster, zero is specified for the `<band>` argument. To identify the center cell in the landuse raster, the number of columns is divided by two to define the middle x-axis value, and the number of rows is divided by two to define the middle y-axis value. The center cell's value is changed from a zero to a two to locate a service center there (`landuseRaster.writePixelValue(2, centerX, centerY, 0)`). The line of code uses the `writePixelValue` action to write a value to the raster and commit or save the raster file immediately after the change. Now that the service center is located on the raster, the associated service-center agent must store this location (`scAgent.xLoc = centerX scAgent.yLoc = centerY`) in its `centerX` and `centerY` fields so that it knows its location in the raster. The result of your work in this exercise should produce code that looks like that shown in the figure on page 128. Make sure to save your changes.

Run the SOME model

- 7 Click the Initialize button on the Repast toolbar.

Note: If you receive an error, chances are the output rasters for the exercise already exist. Open ArcCatalog and navigate to C:\ESRIPress\AgentAnalyst\Chapter04\Output\ex04c and delete all the files in the maps subfolder (you may need to close ArcMap to delete the files). Then run the model again.

- 8 Click the Start button on the Repast toolbar.

You will see the tick count increment on the Repast toolbar.

- 9 Close Agent Analyst by clicking File and then Exit. Do not save any changes when prompted.

You have now identified the landuse and aesthetic rasters and have altered the landuse raster by placing a service center in the middle of it.

Selecting a random location in a raster

In the previous exercise, you changed the value of a cell at the center of a raster. As noted in the “Background information” section earlier, each residential-household agent evaluates the utility it would gain at 15 randomly located cells within the landscape.

2

3

4

Exercise 4d

In this exercise, you will select a random location within a raster.

You can select any location in the raster by specifying a valid x,y coordinate. The origin of the coordinate system is located in the upper-left corner. Therefore, the coordinate for the upper-left cell is (0,0) and the coordinate for the lower-right cell is (<raster>.getColumns(<band>), <raster>.getRows(<band>)).

Load the ArcMap document

- 1 Start ArcMap. Navigate to C:\ESRIPress\AgentAnalyst\Chapter04. Open ex04d.mxd.

The landuse raster is displayed over the aesthetic raster. In ArcToolbox, the Chapter04Ex04d toolbox has been created for you. If the aesthetic raster displays as all gray, change its symbology to Stretched (compute Statistics if asked) with Type set to Standard Deviations (see step 1 of exercise 4a).

Load the Agent Analyst model

- 2 Right-click the Chapter04Ex04d toolbox, point to New, and select Agent Analyst Tool.
- 3 Once the Agent Analyst window appears, click File and select Import. Do not save the changes when prompted.
- 4 Navigate to C:\ESRIPress\AgentAnalyst\Chapter04\Models and open ex04d.sbp.

In previous exercises, you viewed and changed actions created at the model level. In this exercise, you will look at the actions of the resident agent.

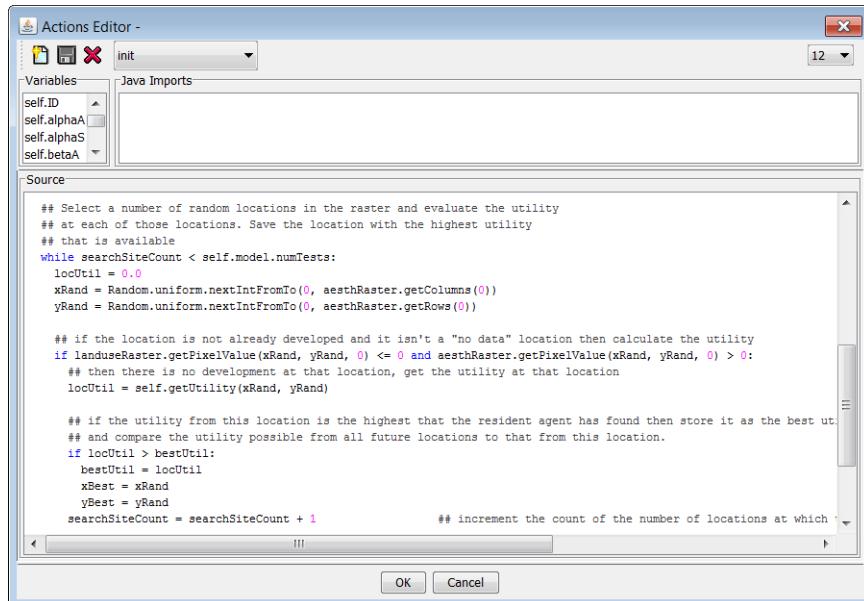
Examine some of the actions

- 5 In the Environment panel, click ResidentAgent. Then, in the Property panel, click the Edit button to the right of the Actions property. From the Actions drop-down list, select init.

There is a substantial amount of code in the init action. It has been thoroughly documented for you in the final exercise. Type the following code in the Source panel, starting with the blank line immediately below the `locUtil = 0.0` statement.

```
xRand = Random.uniform.nextIntFromTo(0, aesthRaster.getColumns(0))
yRand = Random.uniform.nextIntFromTo(0, aesthRaster.getRows(0))
```

The code in the Source panel should match the code in the following image. This is only a portion of the code for the init action for the resident agent.



This code will continue to search for a location for the resident agent to settle at while the number of searches is less than the number of searches specified as the numTests parameter (`while searchSiteCount < self.model.numTests:`).

As mentioned earlier, each resident searches the landscape randomly for a location to settle. The code shown in the preceding figure draws a random integer value between the minimum x-axis value and the maximum x-axis value (`xRand = Random.uniform.nextIntFrom(0, aesthRaster.getColumns(0))`). And it does the same for the y axis (`yRand = Random.uniform.nextIntFrom(0, aesthRaster.getRows(0))`).

`Random.uniform.nextIntFromTo(<min value in range>, <max value in range>)` is a Repast function that randomly selects an integer with equal probability (i.e., uniform) from within the specified range. Since you know the origin of the raster is (0,0), you set the `<min value in range>` argument to zero. In exercise 4c you learned

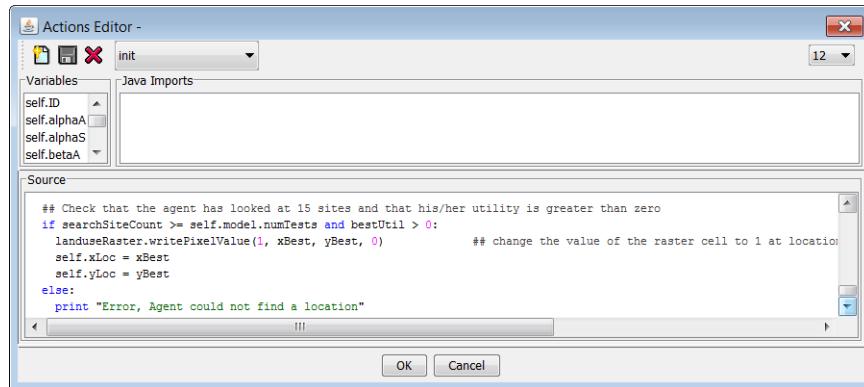
2
3
4

how to obtain the maximum width or x-axis value using the `<raster>.getColumns(<band>)` action and the maximum height or y-axis value using the `<raster>.getRows(<band>)` action. This provides the information needed for setting the `<max value in range>` argument. A random x and y location within the raster can now be selected.

After selecting a random x and y location, the code determines whether the location is already occupied by first making sure the value in the landuse raster is less than or equal to zero (a value of 1 represents a resident and 2 represents a service center). The model also makes sure the location is within the irregular boundary by testing whether the location has an aesthetic-quality value greater than zero—values of zero or less are beyond the study area boundary for the aesthetic-quality raster (`if landuseRaster.getPixelValue(xRand, yRand, 0) <= 0 and aesthRaster.getPixelValue(xRand, yRand, 0) > 0 :).`

Lastly, the utility for the randomly selected location is identified using the `getUtility(<x location>, <y location>)` action found in the resident agent (i.e., `self`, because we are working within the Actions Editor window of the resident agent). The newly calculated utility is compared with the previous utility, and if it is better, the x and y coordinates are saved and a counter of the number of locations that have been searched is increased (`searchSiteCount = searchSiteCount + 1`).

6 Scroll down until the code in the window matches the code in the following image.



Note that at the bottom of the init action for the resident agent, the code checks to ensure that the resident agent has queried the total number of sites and that a utility greater than zero was acquired (`if searchSiteCount >= self.model.numTests and bestUtil > 0 :`). If these conditions are met, the agent has a location to settle at and that location is changed to a 1 in the landuse raster (`landuseRaster.writePixelValue(1, xBest, yBest, 0)`). As mentioned earlier, a value of 1 represents a residential development and a value of 2 represents a service-center development.

Now run the model to make sure that the changes you have made work and the model runs as expected. Again the results should be similar to the previous exercises because you have not yet changed any of the parameter values.

Run the model

- 7 Click Run in Agent Analyst, and then click the Initialize button on the Repast toolbar.

Note: If you receive an error, chances are the output rasters for the exercise already exist. Open ArcCatalog and navigate to C:\ESRIPress\AgentAnalyst\Chapter04\Output\ex04d and delete all the files in the maps subfolder (you may need to close ArcMap to delete the files). Then run the model again.

- 8 Click the Start button on the Repast toolbar.

The tick count will begin to increment on the Repast toolbar.

- 9 Close Agent Analyst by clicking File and then Exit. Do not save any changes when prompted.

Reading raster pixel values

So far, you imported and ran the simple residential location model named SOME. Then you created an action, named loadRaster, to load raster data into the SOME model and initialized the model from the SOME model's initAgents action. You retrieved information on the dimensions of the raster and used that information to determine the location of the center cell in the raster and altered its pixel value. In the previous exercise, you selected a random location and changed the value of the cell at that location within the landuse raster. In this exercise, you will learn how to read values from a raster. You will read values from the aesthetic-quality raster and use those values in the calculation of utility for residential household agents.

2

3

4

Exercise 4e

Load the ArcMap document

- 1 Start ArcMap. Navigate to C:\ESRIPress\AgentAnalyst\Chapter04 and open ex04e.mxd.

The landuse raster is displayed over the aesthetic raster. In ArcToolbox, the Chapter04Ex04e toolbox has been created for you. If the aesthetic raster displays as all gray, change its symbology to Stretched (compute Statistics if asked) with Type set to Standard Deviations (see step 1 of exercise 4a).

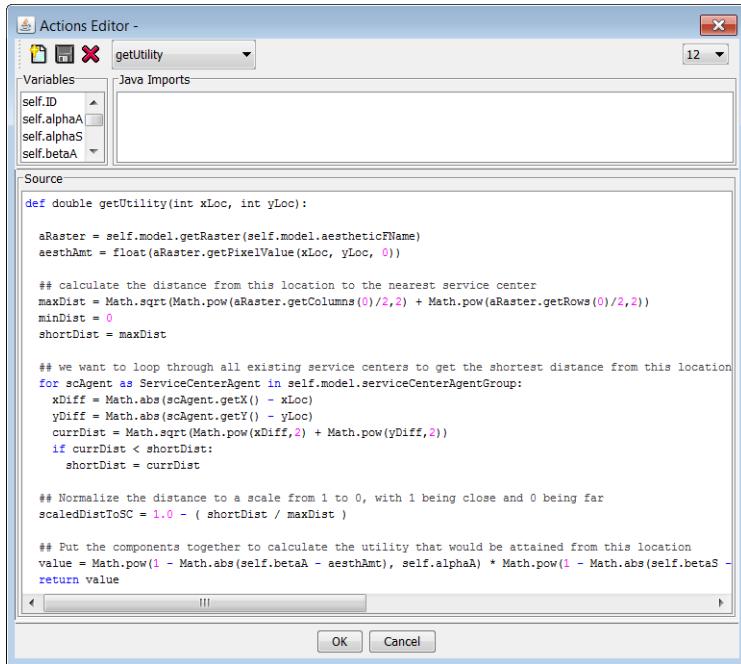
Load the Agent Analyst model

- 2 Right-click the Chapter04Ex04e toolbox, point to New, and select Agent Analyst Tool.
- 3 Once the Agent Analyst window appears, click File and select Import. Do not save the changes when prompted.
- 4 Navigate to C:\ESRIPress\AgentAnalyst\Chapter04\Models and double-click ex04e.sbp.

Create the resident agent's utility action

The calculation of a residential household agent's utility is done by the getUtility action.

- 5 In the Environment panel, click ResidentAgent. In the Property panel, click the Edit button to the right of the Actions property. From the Actions drop-down list in the Actions Editor, select getUtility. Uncomment the second and third lines of code so that the code matches the following image. This action calculates the utility an agent would acquire from the location (xLoc, yLoc) passed as an argument to this action from the resident agent's init action.



By now you should be familiar with the first line of code (`aRaster = self.model.getRaster(self.model.aestheticFName)`). It retrieves a reference to the raster with the name `aestheticFName`, which was saved as a parameter at the model level. In exercise 4b, you registered the aesthetic-quality raster with the model so that you could make calls to the model to obtain the raster without having to locate it using a full path name.

In the second line (`aesthAmt = float(aRaster.getPixelValue(xLoc, yLoc, 0))`), the right side of the equation reads a value from a raster and assigns that value to the `aesthAmt` variable. The values in `aRaster` are floating point decimal numbers, and `aesthAmt` should also be a floating-point value. Therefore, the value returned from the `getPixelValue` action is cast using the `float(<action>)` action.

Casting is one way to convert one type of number or object (e.g., `Integer`) to another number type or object (e.g., `Double`).

The `getPixelValue` action is similar to the `writePixelValue` action that you saw in exercise 4c. The format of the `getPixelValue` action is as follows:

```
<number variable> =
<raster variable>.getPixelValue (<x-coordinate>, <y-coordinate>,
<band>)
```

Notice that the variables corresponding to the x and y coordinates (i.e., `xLoc` and `yLoc`) are arguments that have been passed to the `getUtility` action from the `ResidentAgent` init action (`def double getUtility(int xLoc, int yLoc):`).

The remaining code calculates the distance of the location (`xLoc, yLoc`) to the nearest service center and then calculates a utility value for the location. To do this, the farthest possible distance from a service center is identified, which is the distance from the service center at the center of the raster to the corner of the raster (`maxDist = Math.sqrt(Math.pow(aRaster.getColumns(0)/2, 2) + Math.pow(aRaster.getRows(0)/2, 2))`). That distance is stored as the starting distance for determining the closest service center (`shortDist = maxDist`).

2
3
4

For each service center (for `scAgent` as `ServiceCenterAgent` in `self.model.serviceCenterAgentGroup:`), the difference in the coordinates from the resident agent to the service center is determined first in the x coordinate (`xDiff = Math.abs(scAgent.getX() - xLoc)`) and then in the y coordinate (`yDiff = Math.abs(scAgent.getY() - yLoc)`). Notice that the `abs` Python Math module function is used for the calculations. The actual Euclidean distance is then calculated using the Pythagorean theorem (`currDist = Math.sqrt(Math.pow(xDiff, 2) + Math.pow(yDiff, 2))`).

The distance from the resident agent and each service center is tested to see whether it is the shortest distance thus far (if `currDist < shortDist`). If it is, the distance is saved to a variable (`shortDist = currDist`). The distance is normalized to a 0 to 1 scale (`scaledDistToSC = 1.0 - (shortDist / maxDist)`), and the utility for the location of the resident agent is determined (`value = Math.pow(1 - Math.abs(self.betaA - aesthAmt), self.alphaA) * Math.pow(1 - Math.abs(self.betaS - scaledDistToSC), self.alphaS)`).

Now the utility is determined for each location for the resident agent.

Now run the model to make sure that the changes you have made work and the model runs as expected. Again the results should be similar to the previous exercises because you have not yet changed any of the parameter values.

- 6 Click Run in Agent Analyst, and then click the Initialize button on the Repast toolbar.

Note: If you receive an error, chances are the output rasters for the exercise already exist. Open ArcCatalog and navigate to C:\ESRIPress\AgentAnalyst\Chapter04\Output\ex04e and delete all the files in the maps subfolder (you may need to close ArcMap to delete the files). Then run the model again.

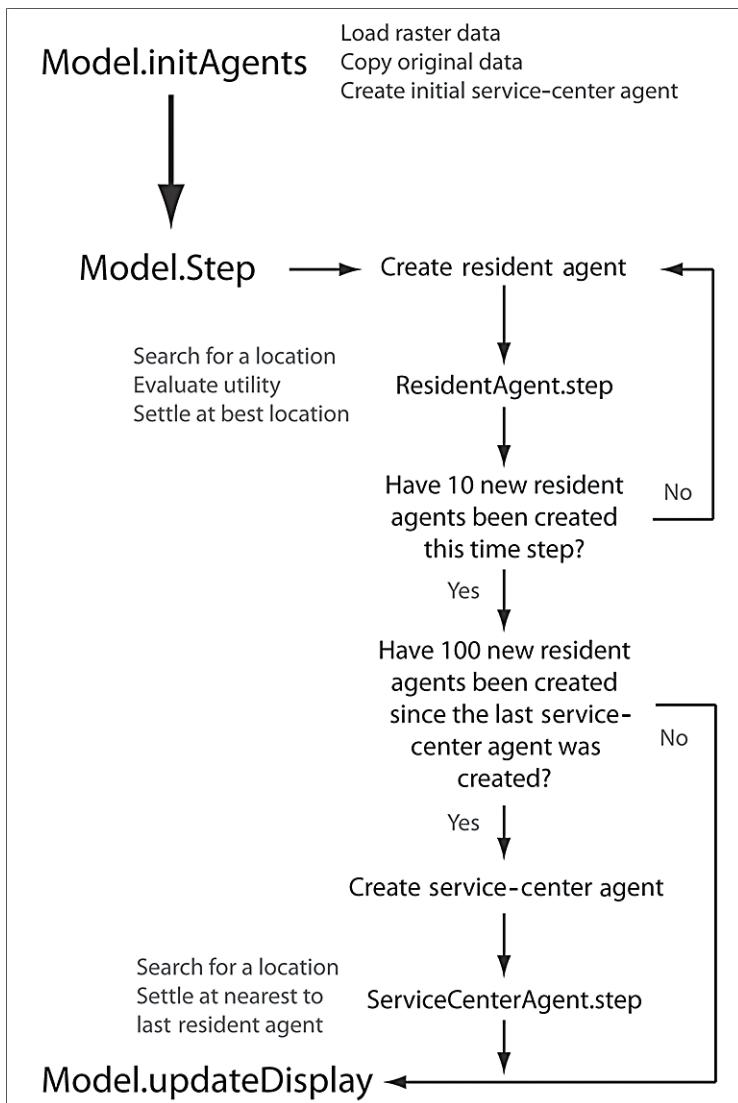
- 7 Click the Start button on the Repast toolbar.

Ensure that the tick count increments on the Repast toolbar.

- 8 Close Agent Analyst by clicking File and then Exit. Do not save any changes when prompted.

Building the SOME model

In the previous exercises you learned how to work with raster data in Agent Analyst. You learned the details of how to link to a raster and use it with Agent Analyst, query raster properties, reference and select individual raster cells, and read, write, and save raster data. In this exercise, you will use the techniques you acquired while learning the details of the code to better understand how the SOME model works and to build the SOME model. To give you a better idea of how the SOME model works, the following diagram illustrates the flow of the main SOME model actions and processes for a single step.



Exercise 4f

Load the ArcMap document

- 1 Start ArcMap. Navigate to C:\ESRIPress\AgentAnalyst\Chapter04 and open ex04f.mxd.

The landuse raster is displayed over the aesthetic raster. In ArcToolbox, the Chapter04Ex04f toolbox has been created for you. If the aesthetic raster displays as all gray, change its symbology to Stretched (compute Statistics if asked) with Type set to Standard Deviations (see step 1 of exercise 4a).

Load the Agent Analyst model

- 2 Right-click the Chapter04Ex04f toolbox, point to New, and select Agent Analyst Tool.
- 3 Once the Agent Analyst window appears, click File and select Import. Do not save the changes when prompted.
- 4 Navigate to C:\ESRIPress\AgentAnalyst\Chapter04\Models and open ex04f.sbp.

Create the initAgents action

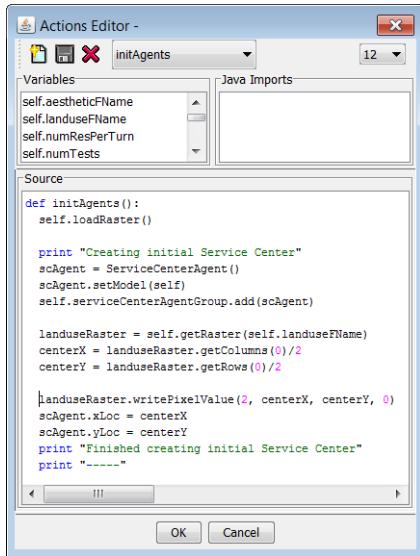
The first action called by the model is the initAgents action.

- 5 In the Environment panel, click SOME Model. In the Property panel, click the Edit button to the right of the Actions property to open the Actions Editor.
- 6 In the Actions drop-down list, select initAgents. Uncomment the four commented lines in this action so that it appears as shown in the following image. Make sure to save your changes. This action initializes the SOME model by loading in raster data and creating the initial landscape configuration (i.e., a service center at the center).

2

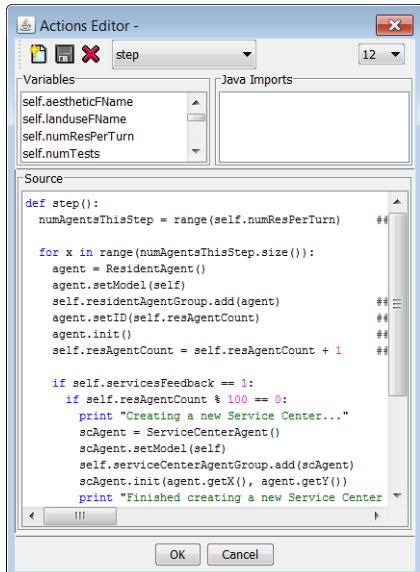
3

4



Since you have already worked with the loadRaster action in exercise 4b, you will develop the step action in the SOME model.

- In the Actions drop-down list, select step. Uncomment all five lines of commented code in this action so that it appears as shown in the following image. Make sure to save your changes.



Each time step, the model executes the step action. Some of the code should look familiar and some is quite new. Each line of code has comments to help you understand what is happening, but each line will also be described here in the text that follows.

The second line of code (`numAgentsThisStep = range(self.numResPerTurn)`) gets the parameter value for the number of new resident agents that should be created in the model step. A loop is then started (`for x in range(numAgentsThisStep.size()):`) to create new agents, which in this case is 10 per time step.

Within the loop a new resident agent is created and stored in the variable called `agent`. Next, that agent is told that it is associated with the SOME model by using `agent.setModel(self)`. Remember, because you are in the Actions Editor window of the SOME model, that `self` is referring to the SOME model. The new resident agent is added to the group of all resident agents (`self.residentAgentGroup.add(agent)`). The unique identification number for the agent is then set (`agent.setID(self.resAgentCount)`), which in this case also corresponds to the total number of agents in the model at that point in time. Then the agent calls its init action (`agent.init()`).

In exercise 4d you looked at the `ResidentAgent.init()` action and learned that this action initializes the resident agent parameters and has the agent find a location within the landscape at which to settle. The resident agent counter, which is maintained by the SOME model, is incremented (`self.ResAgentCount = self.ResAgentCount + 1`).

The second block of code checks to see whether the parameter for feedback by service centers is equal to 1 (i.e., included in the model run) or not (`if self.serviceFeedback == 1:`). If service-center feedbacks are included, then `if self.resAgentCount % 100 == 0:` checks to see whether 100 resident agents have been added since the last service-center agent. If this is the case, a new service-center agent is created and located in the landscape by using the same methods as you saw in exercise 4c.

As a result, the step action creates 10 new resident agents and places them onto the landscape according to the process in the `agent.init` action described in exercise 4d. When 100 new resident agents have been placed, a new service center is added to serve the growing population.

View the results of the time steps using the `updateDisplay` action

- 8 In the Actions drop-down list, select `updateDisplay`. Uncomment the following line of code in this action: `self.updateGISDisplay()`. Make sure to save your changes.

The `updateDisplay` action calls the Repast `updateGISDisplay` action, which is a Repast action that causes ArcGIS to refresh the map canvas so that you can observe any changes in the landscape.

2

3

4

Run the model

- 9 Click the Run button in Agent Analyst, and then click the Initialize button on the Repast toolbar.

Note: If you receive an error, chances are the output rasters for the exercise already exist. Open ArcCatalog and navigate to C:\ESRIPress\AgentAnalyst\Chapter04\Output\ex04f and delete all the files in the maps subfolder (you may need to close ArcMap to delete the files). Then run the model again.

- 10 Click the Start button on the Repast toolbar.

You will notice that the spatial patterns are similar to previous model runs.

- 11 Close Agent Analyst by clicking File and then Exit. Do not save any changes when prompted.

Copying raster data

Up to this point you have created a version of the SOME model. However, how to copy raster data was not explicitly discussed. There are two important reasons for copying your raster data. First, you should always make a copy of the original data that is unchanged for future use. You could do this manually, but that could be time-consuming and it would be more efficient to automate this task in code. Second, you may want to save intermediate output from your model runs. For example, you may want to save a raster of the model output after every 10 time steps. Saving a sequence of time steps of a model run allows you to use the ArcGIS animation tools to create animations and movies of your simulation for repetitive viewing and sharing with others (see chapter 10). The speed of these post-model-run movies will typically outperform the model, and many modelers find it more desirable to view output in this way.

2

3

4

Exercise 4g

To copy a raster using code, you need only complete three steps:

1. Locate the raster on your computer.
2. Connect to the raster dataset.
3. Save the raster dataset with a new name.

Load the ArcMap document

- 1 Start ArcMap. Navigate to C:\ESRIPress\AgentAnalyst\Chapter04. Open ex04g.mxd.

The landuse raster is displayed over the aesthetic raster. In ArcToolbox, the Chapter04Ex04g toolbox has been created for you. If the aesthetic raster displays as all gray, change its symbology to Stretched (compute Statistics if asked) with Type set to Standard Deviations (see step 1 of exercise 4a).

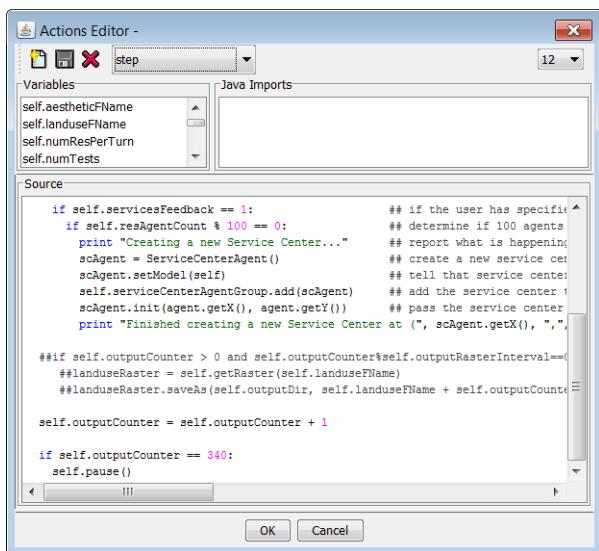
Load the Agent Analyst model

- 2 Right-click the Chapter04Ex04g toolbox, point to New, and select Agent Analyst Tool.
- 3 Once the Agent Analyst window appears, click File and select Import. Do not save the changes when prompted.
- 4 Navigate to C:\ESRIPress\AgentAnalyst\Chapter04\Models and open ex04g.sbp.

Develop the step action

The step action is performed within the SOME model.

- 5 In the Environment panel, click SOME Model. In the Property panel, click the Edit button to the right of the Actions property to open the Actions Editor.
- 6 In the Actions drop-down list, select step. Uncomment the three commented lines shown in the following image. This action creates a copy of the landuse raster data when it's called.



The first step is to create a variable (i.e., `landuseRaster`) to link to the raster data you are interested in copying. In this case the landuse raster is being copied. Earlier you added the landuse raster to the model, which enables you to ask the model to give you the raster associated with `self.landuseFName`. Since you are working on an action at the SOME model level, you can ask the SOME model (using the `self` command) to give you a specific raster as follows: `landuseRaster = self.getRaster(self.landuseFName)`.

The second step is to save a new copy of the raster you just stored in the variable `landuseRaster`. To do this you use the `<raster>.saveAs` method in the same way you created a copy of the original data in exercise 4b. However, because you want to create a copy of your output at multiple time steps, you add the tick count as a suffix to our file name so that you know at which time in the model run the landuse raster was created. To do this, use the following command: `landuseRaster.saveAs(self.outputDir, self.landuseFName + self.outputCounter)`. The second argument of the `saveAs` method identifies the output file name for the copy of the data. Here the time step counter variable (i.e., `outputCounter`) in the model (i.e., `self`) is concatenated to the model parameter `self.landuseFName`, which if you remember from earlier should be the string “abmlanduse”.

The third step is to tell the model when to make a copy of the landuse data. In the example code provided, the following if statement is used:

```
If self.outputCounter > 0 and self.outputCounter % ➔  
  ➔self.outputRasterInterval == 0:
```

This line of code tells the model to create a copy of the landuse data if the model is beyond the first time step (i.e., `self.outputCounter > 0`), and the model time step must be an interval of the `outputRasterInterval` parameter, which tells the model how frequently to create an output copy of the landuse raster. The second part of the if statement uses the modulo operator (i.e., `%`) that returns the remainder of the model time step (i.e., `outputCounter`) divided by the parameter value in `outputRasterInterval`. If that remainder is zero, the second clause in the if statement is true and the model will create a copy of the landuse raster using the code that follows. Without the if statement the model would create a new output at every time step.

2

3

4

Run the model

- 7 Click the Run button in Agent Analyst, and then click the Initialize button on the Repast toolbar.

Note: If you receive an error, chances are the output rasters for the exercise already exist. Open ArcCatalog and navigate to `C:\ESRIPress\AgentAnalyst\Chapter04\Output\ex04g` and delete all the files in the maps subfolder (you may need to close ArcMap to delete the files). Then run the model again.

- 8 Click the Start button on the Repast toolbar.

Confirm that the model is running. You'll see that the tick count will increment on the Repast toolbar and that the spatial patterns are similar to previous model runs.

If you want to alter which time steps of the landuse rasters are output

- 9 In the Environment panel, click SOME Model. In the Property panel, click the Edit button to the right of the Fields property to show the model-level parameters.

You can alter the `outputRasterInterval` variable value to output raster datasets every time step or at some interval you define. You could also change this variable at run time on the Parameters tab in the SOME Model Settings window. Remember, before rerunning the model with new parameter settings, you have to delete the files you just created.

Close Agent Analyst

- 10 Close Agent Analyst by clicking File and then Exit. Do not save any changes when prompted.

After running the model, open ArcCatalog and navigate to the following location: `C:\ESRIPress\AgentAnalyst\Chapter04\Output\ex04g\maps`. In this folder you will see a series of raster files created every 20 time steps.

- 11 Click File and then Exit. Do not save any changes when prompted.

Running computational experiments with the SOME model

You have now learned how to use Agent Analyst to access raster data and use that data in a simple model of urban growth. This exercise suggests one way to use ArcMap to create data that can be used with the model developed in exercises 4a through 4g. Included in the data for this chapter are two aesthetic-quality maps. As mentioned earlier, the first map, named *aesthetic*, is derived using real data from a township in southeastern Michigan (Robinson and Brown 2009). The second map, named *aestheticrand*, is a random assignment of aesthetic-quality values within the landscape. In this exercise, you will consider how landscape heterogeneity—that is, variation in the values of aesthetic quality—affect development patterns when compared to a landscape with no variation in aesthetic quality (i.e., a homogeneous landscape).

Exercise 4h

Load the ArcMap document

- 1 Start ArcMap. Navigate to C:\ESRIPress\AgentAnalyst\Chapter04. Open ex04h.mxd.

The landuse raster is displayed over the aesthetic raster and the *aestheticrand* raster. In ArcToolbox, the Chapter04Ex04h toolbox has been created for you. If the aesthetic raster displays as all gray, change its symbology to Stretched (compute Statistics if asked) with Type set to Standard Deviations (see step 1 of exercise 4a).

Load the Agent Analyst model

- 2 Right-click the Chapter04Ex04h toolbox, point to New, and select Agent Analyst Tool.
- 3 Once the Agent Analyst window appears, click File and select Import. Do not save the changes when prompted.
- 4 Navigate to C:\ESRIPress\AgentAnalyst\Chapter04\Models and open ex04h.sbp.

The two datasets provided both have variation in aesthetic-quality values. The first step is to use the Spatial Analyst Raster Calculator to create an aesthetic-quality map where all raster cells have the same value. If you have not already activated the Spatial Analyst extension, do so now (in ArcMap, open the Customize menu, select Extensions, select the Spatial Analyst check box in the Extensions dialog box, and then click Close).

- 5 In ArcToolbox, expand the Spatial Analyst Tools.
- 6 Find the Map Algebra toolbox and expand it to see the Map Algebra tools.
- 7 Right-click and open the Raster Calculator tool.

- 8 In the Expression panel in the center of the Raster Calculator dialog box, create the following expression by typing it or by selecting the different operators and data layers:

```
("aesthetic" * 0) + 1.0
```

This expression resets all the values in the aesthetic-quality layer to zero and then sets the value of each cell to 1.

- 9 Click the Browse button next to the Output Raster field. If necessary, navigate to C:\ESRIPress\AgentAnalyst\Chapter04\Data\ex04h. Type **aesthNoVar** in the Name box and click Save. Then click OK to run the Raster Calculator tool.
- 10 If the new data layer doesn't appear in ArcMap automatically, use the Add Data button and browse to C:\ESRIPress\AgentAnalyst\Chapter04\Data\ex04h. Then select the aesthNoVar raster dataset and click Add.

Run computational experiments

The SOME model that you have been building uses a random number generator. Whenever a random number generator is used in these types of dynamic models you can get different model output each time you run the model. Therefore, what modelers typically do is run the model a number of times (e.g., 30, 50, or thousands of times) and calculate summary statistics like the mean and standard deviation of each output metric to compare how the model behaves using one set of parameters versus another. In this exercise, you will run the model using each of the three different aesthetic-quality maps, which should provide enough difference in the results for you to observe the effect that landscape heterogeneity in aesthetic quality has on the development patterns created by the SOME model.

You will run the SOME model as you have done in the past, but this time you will change a number of parameters to see how they affect spatial patterns of residential development.

- 11 Click the Agent Analyst Run button to run the SOME model.
- 12 On the Parameters tab of the SOME Model Settings window, change the aestheticFName parameter to aesthNoVar.
- 13 Next, click the Initialize button on the Repast toolbar to register this parameter change with the model. Run the model for 340 time steps and observe the development pattern that the model produces.

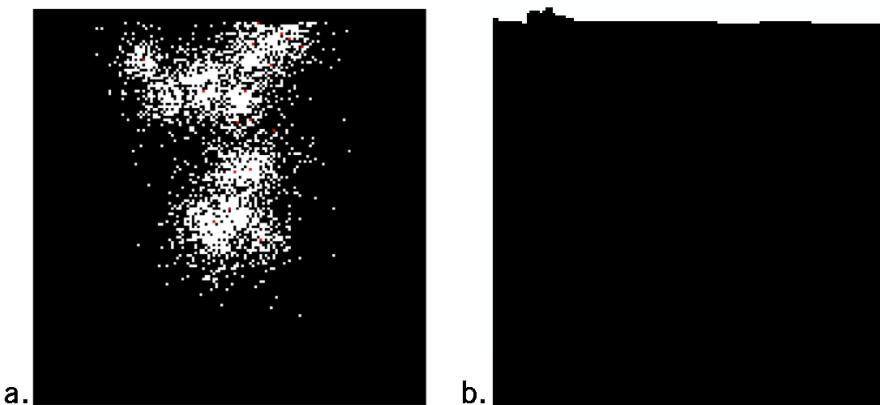
Note: If you receive an error, chances are the output rasters for the exercise already exist. Open ArcCatalog and navigate to C:\ESRIPress\AgentAnalyst\Chapter04\Output\ex04h and delete all the files in the maps subfolder (you may need to close Agent Analyst and ArcMap to delete the files). Then run the model again.

2

3

4

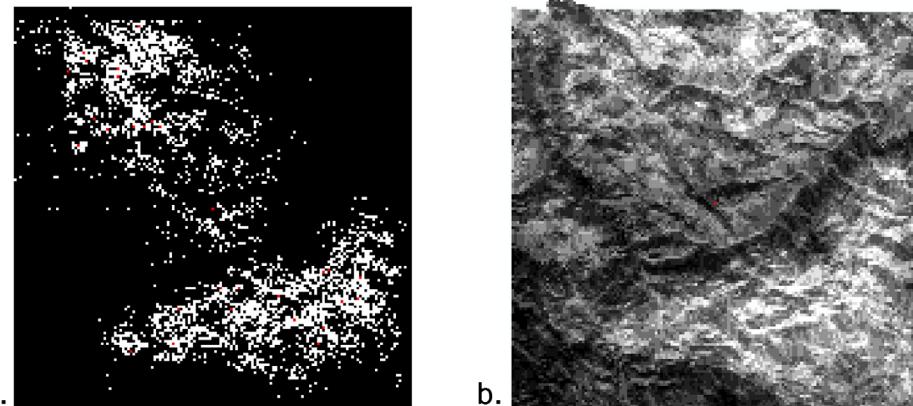
To view a sample of the output of your run, click the Add Data button and add one of the new land-use data layers the model created in your Output folder, C:\ESRIPress\AgentAnalyst\Chapter04\Output\ex04h\maps. The output file names will be of the form abmlanduse##, where ## is replaced by the time step number at which the output was created. Your results may look something like the following images.



The residential land-use development patterns are shown in white (a). All cells had the same value of aesthetic quality as shown by the uniform black background (b).

Because there is no variation in the aesthetic quality of the landscape, distance is the only factor affecting development patterns, and resident agents formed a fairly compact pattern of development. Next, you will use the aesthetic-quality map derived from real-world data to see how variation in aesthetic quality can affect residential development patterns.

- 14 Close the Repast model by closing the Repast toolbar or by closing each of the Repast windows.
- 15 Cut and paste the output files (i.e., abmlanduse##) to another directory or delete them if you no longer want to keep them. (You may need to close Agent Analyst and ArcMap to delete the files, and you will need to reimport the agent model.)
- 16 Run the SOME model a second time by clicking the Run button on the Agent Analyst toolbar.
- 17 If necessary, on the Parameters tab of the SOME Model Settings window, change the aestheticFName parameter to aesthetic.
- 18 Click the Initialize button on the Repast toolbar to register this parameter change with the model. Run the model for 340 time steps and observe the development pattern that the model produces. Your results may look something like the following images.



The residential land-use development patterns are shown in white (a). The variation in aesthetic quality (b) is derived by a number of environmental and geographic factors using real-world data. The methodology is described in detail in Robinson and Brown (2009).

When the model was run using the aesthetic map, the development pattern had a much greater level of dispersal than the results using the homogeneous aesthetic-quality map (see the results of step 13). Small clusters develop across the landscape, but generally the resident agents settle on the two areas of higher aesthetic quality, shown in a whiter color in the image above.

Note: If you add one of your final abmlanduse## layers to ArcMap and set the undeveloped land color (the 0 value) to No Color, you'll be able to observe the pattern of your development superimposed on the aesthetic-quality map.

Run the model using a randomly generated map

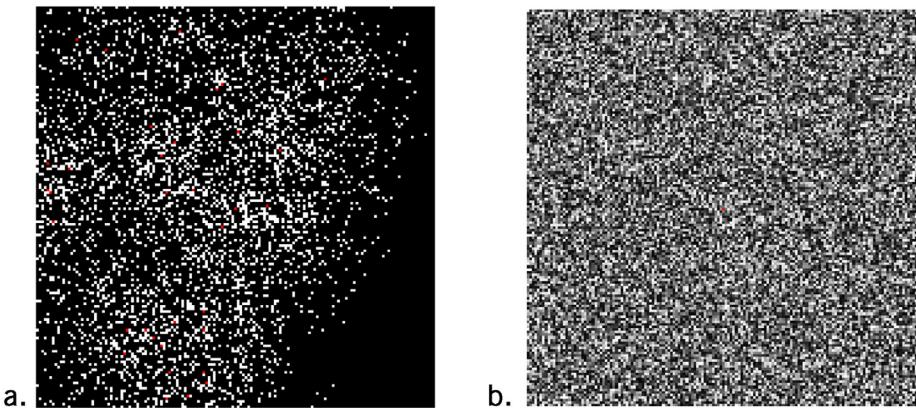
- 19 Close the Repast model by closing the Repast toolbar or by closing each of the Repast windows.
- 20 Cut and paste the output files (i.e., abmlanduse##) to another directory or delete them if you no longer want to keep them. (You may need to close Agent Analyst and ArcMap to delete the files, and you will need to reimport the agent model.)
- 21 Run the SOME model a third time by clicking the Run button on the Agent Analyst toolbar.
- 22 On the Parameters tab of the SOME Model Settings window, change the aestheticFName parameter to aestheticRand.
- 23 Next, click the Initialize button on the Repast toolbar to register this parameter change with the model. Run the model for 340 time steps and observe the

2

3

4

development pattern that the model produces. Your results may look something like the following screen shots.



The residential land-use development patterns (a) are shown in white. Variation in aesthetic quality (b) is derived by a randomly created aesthetic quality map.

As the amount of variation (or heterogeneity) is changed in the aesthetic quality of the landscape, development patterns become more dispersed. There are many drivers of land-use and land-cover change that act to pull development closer together or to push it farther apart. There are several other parameters that you might alter to investigate their effects on development patterns. Create your own experiments and investigate what the effects of having more information (using the numTests parameter), no new service-center development (using the servicesFeedback parameter), and other initial configurations of aesthetic quality or land-use patterns may have on subsequent development patterns.

Close Agent Analyst and ArcMap

- 24** In the Agent Analyst window, click File and then Exit. Do not save any changes when prompted.
- 25** In ArcMap, click File and then Exit. Do not save any changes when prompted.

In this chapter, you created and worked with a simple model of urban growth, named SOME. This simple model has formed the foundation for a number of research efforts that extend the SOME model to include other processes known to influence urban growth (e.g., other agent types and land-use policies—Brown et al. 2005a, Brown et al. 2005b, Brown and Robinson 2006, Brown et al. 2008, Zellner et al. 2008, Robinson and Brown 2009, and Robinson et al. in press). While this chapter has focused explicitly on using raster data with Agent Analyst and ArcGIS, it would be feasible to combine vector agents, as is shown in chapters 2 and 3, such that the vector agents interact with each other and with raster data.

GIS data, fields, and actions dictionaries

Table 4.1 Data dictionary of GIS datasets

| Dataset | Data type | Description |
|---------------|-----------|--|
| aesthetic | raster | Cell values from 0 to 1, where values close to 0 represent low aesthetic quality and values close to 1 represent high aesthetic quality. |
| landuse | raster | Land-use types. Values represent residential development (1) and service center (2) land uses. |
| aestheticRand | raster | Randomly assigned aesthetic-quality values. |

Table 4.2 Fields dictionary for the SOME model

| Attribute | Data type | Description |
|----------------------|-----------|---|
| SOME model | | |
| resAgentCount | integer | Total number of agents in the model. |
| numResPerTurn | integer | This is the number of new residential household agents entering into the simulation at each time step. |
| servicesFeedback | integer | This parameter is an integer flag that signals whether to allow new service centers to enter into the model (1) or to have only a single service center located at the center of the landscape (0). |
| aestheticFName | string | Name of the raster file that contains numerical values of aesthetic quality. Values in this data layer should be between 0 and 1. The resolution and extent of this layer should correspond to other layers. |
| landuseFName | string | Name of the raster file that contains categorical values for different land-use types across the landscape. In this simple model, 0 = undeveloped, 1 = residential development, 2 = service-center development. |
| rasterDir | string | The directory location of the folder that is holding the raster data (i.e., AestheticFName and LanduseFName). |
| numTests | integer | This parameter specifies the number of available locations that are evaluated before a resident agent selects a location for settlement. |
| outputRasterInterval | integer | The value specified by this parameter establishes the number of time steps after which a new raster is saved to the hard drive. A value of 0 results in no new output. A value of 10 creates copies of the raster data at time steps 10, 20, 30, and so on. |
| outputCounter | integer | Keeps track of time steps and when to stop the model run. |

Table 4.2 Fields dictionary for the SOME model (*continued*)

| Attribute | Data type | Description |
|-----------------------------|-----------|---|
| Resident agent | | |
| ID | integer | Unique identifier for the agent. |
| betaA | double | Preferred value of aesthetic quality. Set to 1, whereby agents prefer highest value of aesthetic quality. |
| betaS | double | Preferred value for distance from service centers. Set to 1, whereby agents prefer to be adjacent to service centers. |
| alphaA | double | Preference weight placed on aesthetic quality. $\alpha_A + \alpha_S = 1$. $0 \leq \alpha_A \leq 1.0$ |
| alphaS | double | Preference weight placed on distance from service centers. $0 \leq \alpha_S \leq 1.0$ |
| xLoc | integer | x-coordinate location for household agent. |
| yLoc | integer | y-coordinate location for household agent. |
| Service-center agent | | |
| xLoc | integer | x-coordinate location for service-center agent. |
| yLoc | integer | y-coordinate location for service-center agent. |

Table 4.3 Actions dictionary for the SOME model

| Declared actions | Description |
|-----------------------------|---|
| SOME model | |
| initAgents | Initializes the center location of the landscape with a service-center agent. It also loads the raster data and makes a copy of the original land-use data. |
| updateDisplay | Updates ArcMap display. |
| writeAgents | This action is not used in the SOME model. |
| loadRaster | Connects to raster data and registers that data with the SOME model. |
| step | Conducts routines associated with each tick count. Based on default parameters, this involves creating 10 new household agents and telling each agent to conduct its step action. |
| copyRaster | When called, this action creates a copy of the landuse raster and names the raster using the tick count as a suffix on the new raster file name. |
| Resident agent | |
| setID | Sets the unique identifier value for the resident agent. |
| init | Because the agent performs only one step, this action parameterizes the agent with initialization values as well as performs the actions to settle the agent in the landscape. |
| getUtility | Calculates the utility an agent would receive from a specific location. |
| getX | Retrieves the value of the agent along the x-axis. |
| getY | Retrieves the value of the agent along the y-axis. |
| Service-center agent | |
| init | Because the agent performs only one step, this action parameterizes the agent with initialization values as well as performs the actions to settle the agent in the landscape. |
| getX | Retrieves the value of the agent along the x-axis. |
| getY | Retrieves the value of the agent along the y-axis. |

2

3

4

References

- Brown, D. G., S. E. Page, R. Riolo, and W. Rand. 2004. "Agent-Based and Analytical Modeling to Evaluate the Effectiveness of Greenbelts." *Environmental Modelling & Software* 19: 1097–109.
- Brown, D. G., M. North, D. Robinson, R. Riolo, and W. Rand. 2005. "Spatial Process and Data Models: Toward Integration of Agent-Based Models and GIS." *Journal of Geographical Systems, Special Issue on Space-Time Information Systems* 7: 25–47.
- Brown, D. G., S. E. Page, R. Riolo, M. L. Zellner, and W. Rand. 2005. "Path Dependence and the Validation of Agent-Based Spatial Models of Land Use." *International Journal of Geographical Information Science, Special Issue on Land Use Dynamics* 19: 153–74.
- Brown, D. G., and D. T. Robinson. 2006. "Effects of Heterogeneity in Residential Preferences on an Agent-Based Model of Urban Sprawl." *Ecology and Society* 11: 46. <http://www.ecologyandsociety.org/vol11/iss41/art46/>.
- Brown, D. G., D. T. Robinson, L. An, J. I. Nassauer, M. Zellner, W. Rand, R. Riolo, S. E. Page, B. Low, and Z. Wang. 2008. "Exurbia from the Bottom-Up: Confronting Empirical Challenges to Characterizing a Complex System." *Geoforum* 39: 805–18.
- Robinson, D. T., and D. G. Brown. 2009. "Evaluating the Effects of Land-Use Development Policies on Ex-Urban Forest Cover: An Integrated Agent-Based GIS Approach." *International Journal of Geographical Information Science* 23: 1211–32.
- Robinson, D. T., S. Shipeng, M. Hutchins, R. L. Riolo, D. G. Brown, D. C. Parker, W. S. Currie, T. Filatova, and S. Kiger. In press. "Effects of Land Markets and Land Management on Ecosystem Function: A Framework for Modelling Exurban Land-Changes." *Environmental Modelling and Software*.
- Zellner, M. L., S. E. Page, W. Rand, D. G. Brown, D. T. Robinson, J. Nassauer, and B. Low. 2008. "The Emergence of Zoning Policy Games in Exurban Jurisdictions." *Land Use Policy* 26: 356–67.

Section 3: Advanced techniques for points, polygons, and rasters

Chapter 5

Moving point agents based on multiple-criteria decision making

by Kevin M. Johnston and Nicholson Collier

- ◆ An overview of the model
- ◆ Querying rasters for attributes of locations that will determine an agent's actions—evaluating the level of security for the eight surrounding cells for the security criterion
- ◆ Using probability and random numbers to determine stochastic events—making a kill and weighting the eight surrounding cells for the remaining with the kill criterion
- ◆ Using raster attributes and field observation data to determine the characteristics of an event—identifying the type of kill and how long it will take the cougar agent to consume it
- ◆ Detecting and tracking other agents using attractors—weighting the eight surrounding cells for the pursuing a female cougar agent criterion

In this chapter, we will add intelligence to the cougar point agents created in chapter 2. Instead of having the cougars move randomly, as was presented in chapter 2, you will learn how to move the cougar agents with intent and see how to instruct agents to make more informed decisions as they move. The cougar agents will choose to move into areas where they remain safe, have a higher chance of finding food, and can pursue a mate. To capture biological realism, the cougar agents will query the landscape around them, as represented by a raster. Using multiple rasters, each cell will store, among other things, the level of security realized at the location and the likelihood of finding food if the cougar enters the cell. Chapter 8 will add further complexity to the model to capture additional biological needs for the cougar.

The intent of this chapter is to introduce you to agent-based modeling principles, not how to create a cougar model. If the rules are not satisfactory, they can be changed.

The modeling scenario

In this model, at each time step a cougar agent will move, or not move, into one of eight surrounding cells. Which cell the agent moves into (if it moves) depends on what the cell contains or the direction of the cell relative to where the cougar wants to go.

The cougar agent evaluates the preference for each of the eight surrounding cells separately for six criteria:

- Security
- If it made a kill, remaining with the kill
- Pursuing a female cougar agent (for a male cougar agent)
- Staying within the home range
- Moving with intent toward good habitat/hunting
- Mating

For example, to evaluate each of the eight surrounding cells from the perspective of the security criterion, the cougar agent will query each of the cells, and the cell that contains the most cover will be the most preferred to move into based on this criterion. That cell receives the highest weight, indicating it is the most preferred relative to this criterion. The remaining surrounding cell locations may decrease in weight (preference) as the security levels decrease (determined from the attributes in the cells).

The cougar agent runs through the weighting process for the next criterion, remaining with the kill. Once the cougar makes a kill, it will eat the carcass until it is full. The cougar agent will bury the rest of the kill and wander, but it will continue to return to the kill until it finishes consuming it. As the cougar agent moves away from the unfinished carcass, it retains the

knowledge of the location of the kill. At each time step, when the cougar agent weights each of the eight surrounding cells relative to the remaining with the kill criterion, the cell that returns the cougar agent closest to the kill will be weighted the most preferred, with the cells decreasing in weight the farther the cell is from the kill location.

This weighting of each of the eight surrounding cells occurs for each of the six criteria. In this chapter, you will weight the eight surrounding cells for the first three criteria: security, remaining with the kill, and pursuing a female cougar (for a male cougar agent).

In chapter 8 you will complete the weighting of the eight surrounding cells for the remaining criteria (staying within the home range, moving with intent toward good habitat/hunting, and mating). For any particular time step, based on the state of the cougar (how hungry it is), one or more criteria will be more important than the others. For example, if the cougar agent is hungry during a time step, then the moving with intent toward good habitat/hunting criterion will be weighted as more important to the cougar agent during that time step. In chapter 8 you will learn how to apply this relative weighting for each criterion based on the energetic state of the cougar. Finally, in chapter 8 you will also see how a cougar agent selects a cell to move into while accounting for all six criteria and then moves into that cell.

5

6

7

8

9

Background information

The data providing the rules for this model's agents comes from literature, cougar experts at the USGS in Flagstaff, Arizona, and from the observation of collared cougars. As discussed in chapter 2, the areas around Flagstaff are experiencing an increase in human/cougar encounters. The goal of this model is to gain a better understanding of cougar movement in order to reduce these encounters.

The model agents

There are two primary types of agents in this model: male cougars and female cougars. To simplify the agents' activities, the male cougars are driven by a few primary behaviors. Male cougars either hunt, eat, wander, look for females, or avoid being killed. Female cougars have similar behaviors, but instead of pursuing male cougars they try to attract them. In addition, they raise young.

There are four other types of agents that support the multicriteria decision making of the cougar agents: cell, prey, HomeRange, and HabitatCentroid. To weight the eight cells surrounding a cougar agent each time step, each cell is defined as an agent, the cell agent. Once a kill is made, the type of kill needs to be determined. Each available prey type in the study area is defined as a prey agent (e.g., deer or elk). The possible prey at the location of the kill is identified and, based on observational data, the type of kill is determined.

In support of the fourth criterion, staying within the home range, the home range boundary for each cougar agent is a HomeRange agent. The center of the best habitat locations in the study area for hunting are identified as points derived from a GIS shapefile. Each habitat centroid is a

HabitatCentroid agent. When a cougar agent is weighting the eight surrounding cells for moving with intent toward good habitat/hunting, the cell closest to the habitat centroid that the cougar agent is moving toward is weighted the highest.

Weighting the eight surrounding cells—attractors

A cougar has long-term intent in its movement, so a random walk is not adequate. There are two main methods for the cougar agent to weight the eight surrounding cells relative to each criterion:

- Evaluate the attributes contained within each cell
- Evaluate the position of each cell relative to a goal to reach a desired location many cells away

Weighting the eight surrounding cells based on the attributes contained within each cell is a matter of querying each cell and determining the agent's preference for the cell for the criteria. This approach is implemented for the security criterion.

To account for long-term intent in the decision making of the cougar agents, the concept of attractors and repellants is incorporated into the model. Attractors and repellants drive or motivate "memory" and "intelligence." This model has five main attractors or repellants maintaining the long-term decision making of the cougar agent:

- Once a kill is made, the location of the kill is an attractor for remaining with the kill.
- The location of the detected female is an attractor for pursuing a female cougar agent (for the male cougar agent).
- The home range boundary polygon is a repellent for staying in the home range.
- The locations of the habitat centroids are attractors for moving with intent toward good habitat/hunting.
- The location of the female cougar agent that the male cougar agent is mating with is an attractor for the mating criterion.

The home range and the habitat attractors influence movement during each time step, while the kill and mating attractors are temporary and influence movement when present, but not every time step.

The weighting of the criteria relative to each other

This model is based on energetics. A cougar agent is motivated by the amount of energy in its system. Thus, the decision making for any time step is primarily influenced by how hungry the

cougar agent is. The energetic state of a cougar agent for a particular time step will determine how much to weight each criterion relative to one another. That is, if the cougar is very hungry, it will hunt intently, therefore moving with intent toward good habitat/hunting will receive a higher weighting relative to the other criteria that time step.

The weights determined for each of the eight surrounding cells for each criterion discussed previously will be multiplied by the appropriate weight assigned to that criterion based on the cougar agent's energetic state for that time step.

5

6

7

8

9

Energy gained

A cougar gains energy from eating a kill and loses energy through activity. To gain energy in a time step, the cougar agent needs to consume some or all of a kill that it made in this or an earlier time step. The type of prey killed determines how much energy was gained and the number of time steps required for the cougar agent to consume the kill. Both energy gained and the time it takes to consume the kill are based on observational data. Dividing the two provides the energy gained in a particular time step.

The cougar agents lose energy by doing any activity other than consuming prey. The more intensive the activity, the more energy used during that time step.

Making a move

The cougar agent in each time step will make a decision to move or not move into one of the eight surrounding cells based on considering what each neighboring cell offers relative to the six primary criteria: security, remaining with a kill, pursuing a female, staying within the home range, moving with intent toward good habitat/hunting, and mating. How important each criterion is during that time step is based on the cougar agent's energetic state. Once a cell is selected, the cougar moves in.

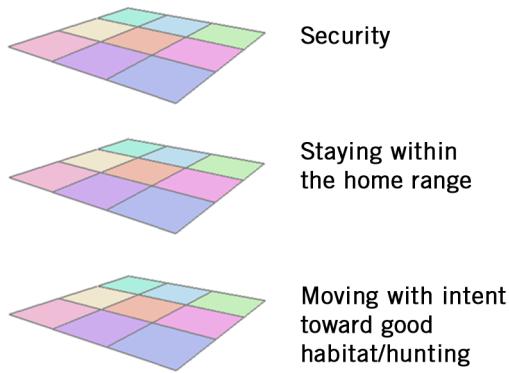
How each time step works

For each time step and for each agent, the model conceptually goes through the following steps to determine where the agent should move during the time step:

1. If the cougar agent is mating, the mating cougar will stay with its mate, therefore proceed to step 10. If the cougar agent is not mating, proceed to step 2 to determine which one of the eight surrounding cells the agent should move into.
2. Weight each of the eight cells surrounding the cougar agent based on the following three criteria:
 - a. Security
 - b. Staying within the home range

c. Moving with intent toward good habitat/hunting

A conceptual image of the weighting process of these three criteria is shown in the following figure.



3. Check to see whether the other criteria apply during the time step:
 - a. Has the male cougar agent detected a female cougar agent?
 - b. Has the cougar made a kill?
4. For the conditions that evaluate to true in step 3, weight the eight surrounding cells based on the criteria (pursuing a female cougar agent and/or remaining with the kill). For staying within the home range, weight the eight surrounding cells based on how far they are from the home range boundary.
5. Identify the cougar agent's energetic state for that time step (i.e., how hungry is the agent?). Based on the energetic state of the cougar agent, weight each of the six criteria relative to one another.
6. Evaluate the weights of the eight surrounding cells and the weights of each criterion and select a cell to move into.
7. Make a move or not.
8. If the male cougar agent has not identified a female cougar agent to pursue, then the cougar agent determines if it can detect a female cougar agent within a specified distance from the cell it just moved into. If a female is detected, the pursuing a female cougar agent criterion will be applicable in the subsequent time step.
9. If the male cougar agent detected a female agent in a previous time step and has been moving toward her, the male cougar agent determines if he has found her, and if so, the two agents will mate throughout 12 time steps. For the next time step, in step 1,

the cougar agent's mating status will evaluate to true and the agents will stay together, ignoring the other criteria.

10. Since the cougar is opportunistic, after the cougar agent makes a move, if it is not already with a kill, the model determines whether the cougar agent makes a kill in the cell it moved into. If it makes a kill, then the prey type is determined, the amount of calories the agent will gain each hour it consumes the kill is identified, and how long the cougar agent will stay with the kill is specified. In the subsequent time step, the remaining with the kill criterion will be applicable.
11. If the cougar agent is eating a kill, the appropriate energy is added to its energy reserve. If the cougar agent is not eating, then based on the activity it is performing during that time step, the appropriate energy is subtracted from the energy reserve.
12. Repeat the process for the specified number of time steps.

Before each move by each cougar agent, the cougar agent will make a trade-off between competing objectives within its immediate neighborhood.

Because a cougar will not always have full access to all information, there is an element of uncertainty in decision making. The model attempts to capture this uncertainty by adding some randomness into the decision-making process. As a result, the cell selected to move into may not always be the most preferred. The weights for each of the eight cells surrounding the cougar for each criterion are multiplied by each weight assigned to each criterion. The weights for each of the eight surrounding cells for each criterion multiplied by the weight assigned to the criteria are combined to create a distribution. A value is randomly selected from the cumulative distribution. The cells with the highest weight (the more preferred) for all the criteria based on what is at the location or the general direction toward an attractor, and with the greatest weights for the criterion, will have the greatest chance to be selected from the distribution. The selected cell identifies the cell the cougar agent will move into.

Scheduling time—the step interval

The length of the time intervals for the model is determined by how far the cougar can move in class III or running activity (see chapter 8, table 8.4) and the cell size or resolution. It would prove difficult to have the cougar move through multiple cells in one time step and still ensure that it is making optimal or near-optimal decisions because there are so many combinations of possible movement scenarios that must be examined before a move can be made.

5

6

7

8

9

An overview of the model

The following exercises will expand on the basic principles you learned in chapter 2 for point agents representing cougars. In these exercises you will focus on how to weight the eight cells surrounding the cougar agent for the following three criteria: security, remaining with a kill, and pursing a female cougar agent (for a male cougar agent). In chapter 8 you will take the three criteria developed in this chapter, add three other criteria (staying within the home range, moving with intent toward good habitat/hunting, and mating), weight the criteria relative to one another, combine the criteria to program your agents to make a move, and calculate the new energy levels for the cougars.

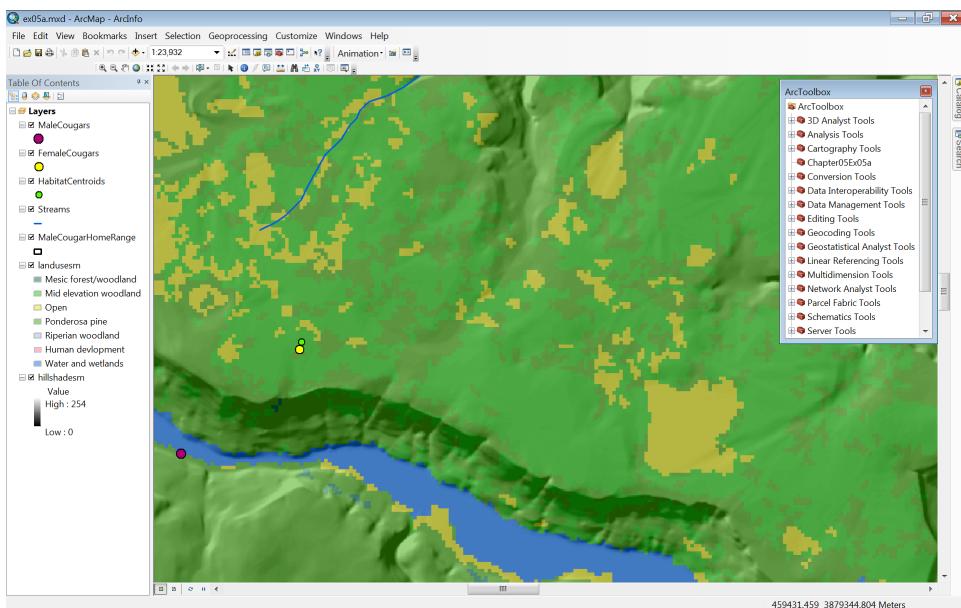
Exercise 5a

You were introduced to the cougar model in chapter 1. In chapter 2 you learned the basics for the model. In this chapter and in chapter 8 you will learn how to take the principles presented in chapter 2 and develop the model using advanced agent-based modeling (ABM) techniques to capture additional biological realism.

You will first explore the overall model in this exercise.

Load the ArcMap document

- 1 Start ArcMap. Navigate to C:\ESRIPress\AgentAnalyst\Chapter05. Select ex05a.mxd, and then click Open.

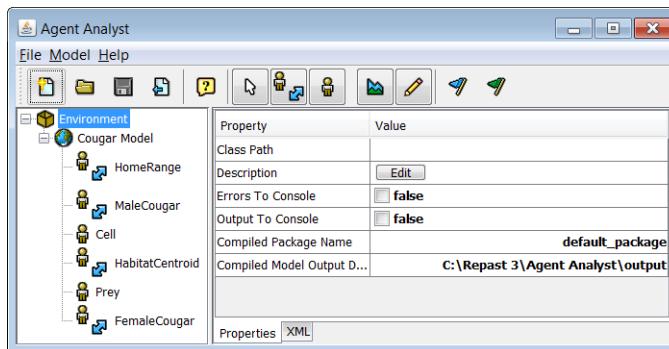


The map document opens. In the ArcMap display you will see land use displayed transparently on top of a hillshade. The purple dots represent the male cougars, the yellow dots the female cougars, and the light green dots are the habitat centroids. The black polygon outline represents a home range of a male cougar.

In ArcToolbox, the Chapter05Ex05a toolbox has been created for you.

Load the Agent Analyst model

- 2** Right-click the Chapter05Ex05a toolbox, point to New, and select Agent Analyst Tool.
- 3** Once the Agent Analyst window appears, click File and select Import. Do not save the changes when prompted.
- 4** Navigate to C:\ESRIPress\AgentAnalyst\Chapter05\Models and open ex05a.sbp. The Agent Analyst window for the cougar model appears.



There are six agent types in the cougar model as discussed earlier. There are MaleCougar and FemaleCougar agents. The HomeRange agent defines the home range boundary for the cougar agents for the staying within the home range criterion. The Cell agents are used in defining the eight cells surrounding a cougar agent and are used when making a movement decision. The HabitatCentroid agents are known locations of desirable habitats within the home range that will be used to guide the movement of the cougar agents for moving with intent toward good habitat/hunting. The Prey agents are the prey types the cougars will hunt.

Explore the male cougar agents

- 5** In the Environment panel, click MaleCougar. In the Property panel, click the Edit button to the right of the Data Source property to open the Data Source Editor.

You will see that the male cougar agents are created from the MaleCougars shapefile. The initial starting locations for the male cougars are established from each point (representing

5

6

7

8

9

a cougar) in the shapefile. Some of the fields or properties for the agents are established from the shapefile.

View the fields

- 6 In the Property panel, click the Edit button to the right of the Fields property to open the Fields Editor.

There are 23 fields that are generally used to monitor the status of the male cougar agent and its relationship to its surroundings. The current status of the cougar at a time step stored in these fields aids in the decision making for the time step. Notice that fields such as habCentroid, prey, targetFemale, energyReserve, homeRangeWeight, and HuntingWeight (as well as others) are related to the six criteria the male cougar evaluates each time step (see discussion in the section “Background information” earlier in the chapter).

Understand the actions

- 7 In the Property panel, click the Edit button to the right of the Actions property to open the Actions Editor. Click the arrow to view the actions in the Actions dropdown list. Scroll down the list.

Notice that the actions setMarblesForSecurity, setMarblesForHomeRange, setMarblesForHabitat, setMarblesForPrey, and setMarblesForFemale establish the weights for the eight surrounding cells for each criterion. The calcWeights action sets the relative weights for each criterion, and the calcMove action identifies the cell to move into. You will review the weighting of the eight surrounding cells for each of these criteria and see how they are combined to make a decision throughout the remaining exercises in this chapter and in chapter 8.

Explore the female cougar agents

- 8 In the Environment panel, click FemaleCougar. In the Property panel, click the Edit button to the right of the Data Source property to open the Data Source Editor.

The FemaleCougars shapefile is the data source for the female cougar agents. There are 12 female cougars in the shapefile, resulting in 12 female cougar agents.

Examine the actions

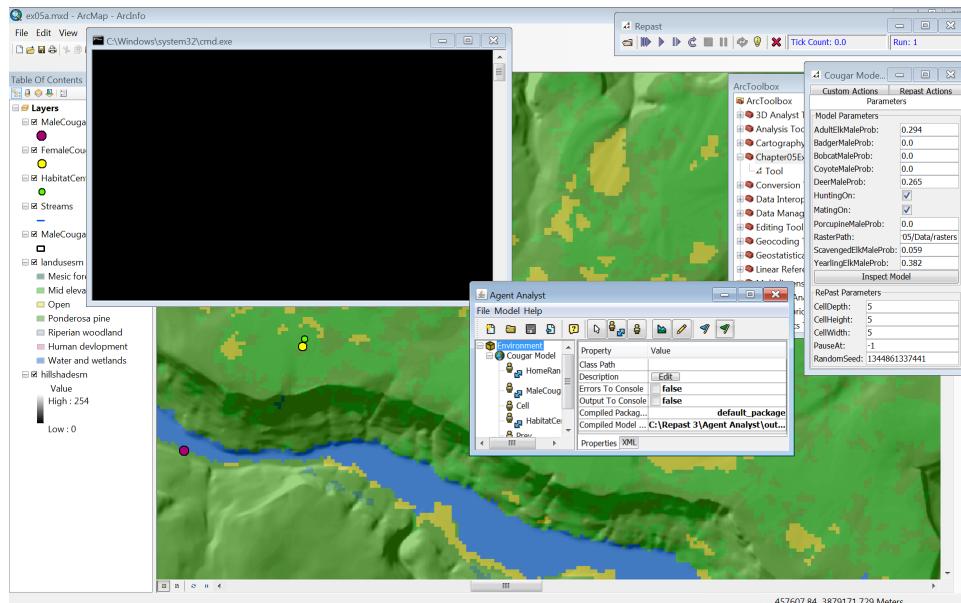
- 9 In the Property panel, click the Edit button to the right of the Actions property to open the Actions Editor. Click the arrow to view the actions in the Actions dropdown list.

To simplify the code for demonstration purposes, the female cougar agent has one criterion, moving with intent toward good habitat/hunting. The female cougar agent could

have similar actions to the male cougar agent, but the female agents would weight the eight surrounding cells for each criterion and the relative weights for the six criteria differently.

Run the model

- In the Agent Analyst window, click the Run button. The Repast toolbar and the Cougar Model Settings window appear.



The parameters that the user can change on the Parameters tab of the window include the weighting to give each potential prey (for example, AdultElkMaleProb to PorcupineMaleProb), and the path to where the series of prey distribution rasters are stored.

- On the Repast toolbar, click the Start button. Observe the cougars' movement.

A dialog box appears identifying the energy level of the male cougar agent (there is only one male cougar in this model run) and when certain events occur. The model will pause when the male cougar agent detects a female, makes a kill, finishes a kill, or is mating (identified in the dialog box). Click the Start button again to continue the model.

Notice that when a male cougar agent makes a kill, he stays with it (wandering a short distance away, but always returning to the kill) until the kill is consumed. The male cougar agent will detect a female agent and will move toward her until they eventually mate.

- Click the Stop button on the Repast toolbar.

5

6

7

8

9

Note: Death has not been implemented in this model demonstration. If you let your model run long enough, you may see negative energy values associated with the cougar agent. It would be easy enough to add a condition to kill the cougar agent when it goes below a specified survival threshold. Take this on as a challenge once you complete chapter 8.

- 13 Close the model and close ArcMap. Do not save any changes.

You now have seen the entire model run. In the following exercises, you will explore how the model is created.

Querying rasters for attributes of locations that will determine an agent's actions—evaluating the level of security for the eight surrounding cells for the security criterion

The first criterion you will add to the model is the cougar's desire to move through secure areas: the security criterion. The USGS biologists have created a raster layer identifying the relative security each cell provides. The values range from 8.81569 down to 0.959982. The higher the value, the more secure and more desirable the cell location will be for the cougar.

The surface was created by applying logistic regression formulas to two distinct Euclidean Distance rasters. Female cougars prefer security in environments away from major roads. Adult male cougars prefer security in environments distant from roads only, whereas juvenile male cougars prefer security in environments distant from both roads and urban areas. Probability surfaces were created individually for each cougar, then averaged to create the final security raster surfaces.

Exercise 5b

In this exercise you will see how the cougar agent weights the eight cells surrounding it relative to the security criterion that will later be used in its decision making.

Load the ArcMap document

- 1 Start ArcMap. Navigate to C:\ESRIPress\AgentAnalyst\Chapter05. Open ex05b.mxd.

In the ArcMap display you will see the security raster displayed over a hillshade with the male cougars (purple dots), female cougars (yellow dots), and the habitat centroids (light green dots that are somewhat hidden by the female cougars). The brightest green cells represent the most secure areas: the brighter the green, the more secure the area.

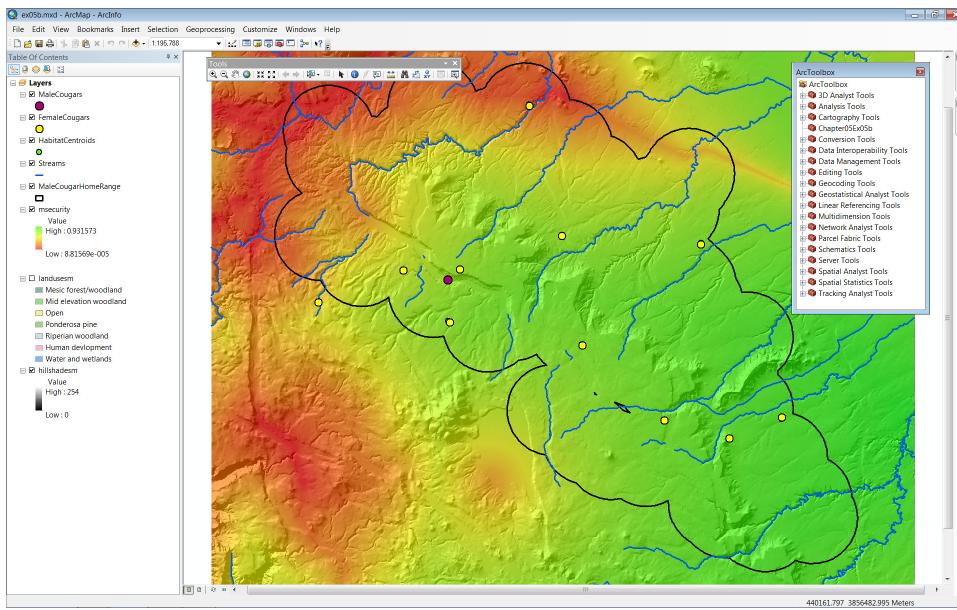
5

6

7

8

9



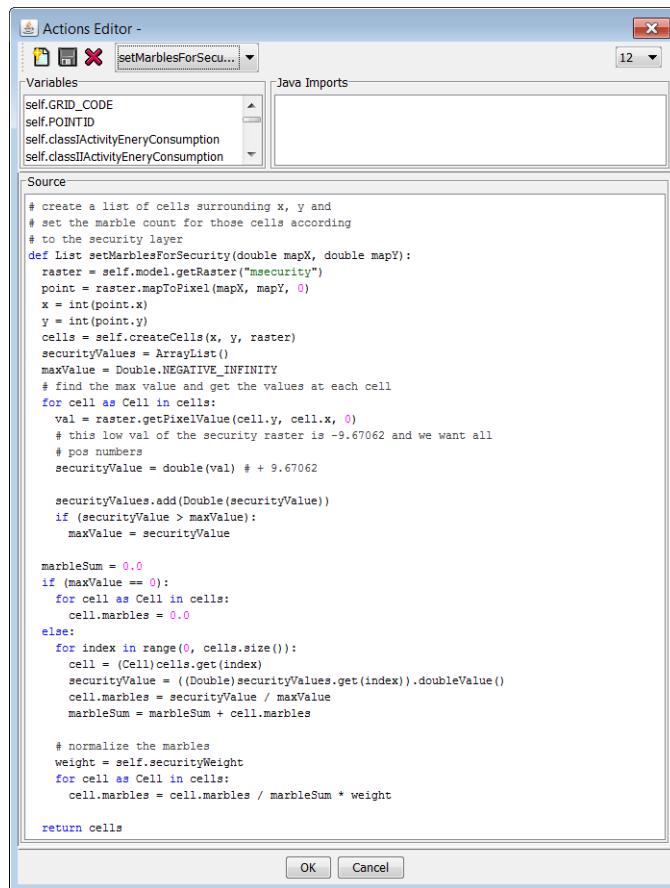
In ArcToolbox, the Chapter05Ex05b toolbox has been created for you.

Load the Agent Analyst model

- 2** Right-click the Chapter05Ex05b toolbox, point to New, and select Agent Analyst Tool.
- 3** When the Agent Analyst window appears, click File and select Import. Do not save the changes when prompted.
- 4** Navigate to C:\ESRIPress\AgentAnalyst\Chapter05\Models and open ex05b.sbp.

Examine the action used by the male cougar to determine and weight security

- 5** In the Environment panel, click MaleCougar. In the Property panel, click the Edit button to the right of the Actions property to open the Actions Editor.
- 6** In the Actions drop-down list, select setMarblesforSecurity. Uncomment the code in the Source panel so that it matches the code in the following image.



In the code, the current location of the agent is first passed to the action (`List setMarblesForSecurity(double mapX, double mapY)`). The raster defining the security values for each cell is identified (`raster = self.model.getRaster ("msecurity")`). The current cell location of the map position of the agent is set to a variable (`point = raster.mapToPixel(mapX, mapY, 0)`). The code ensures that the x and y location values are integer values (`x = int(point.x); y = int(point.y)`). The `createCells` action is called to identify the eight surrounding cells and returns their map coordinate locations, storing them in the `cells` array. The current location and the name of the security raster is also passed to the action (`cells = self.createCells(x, y, raster)`). You will examine the `createCells` action in step 7 of this exercise.

5

6

7

8

9

Creating a probability distribution for determining the movement of an agent into surrounding cells

In the cougar model, during each time step a cougar agent will either move into one of the eight surrounding cells or not. Which cell the agent moves into is determined by a trade-off of multiple criteria (security, remaining with the kill, pursuing a female cougar agent, staying within the home range, moving with intent toward good habitat/hunting, or mating). For each criterion, each of the eight surrounding cells has varying degrees of preference for the cougar to move into the cell based on the attributes of the cell or the relative position or direction of the cell. For example, the surrounding cell with the highest security will be most preferred for the security criterion, with decreasing preference for cells as they decrease in security.

Each of the eight surrounding cells is weighted for each criterion and then adjusted by multiplying the weight determined for the cell by the weight (the importance) assigned to that criterion during that time step. For each cell, the weights for all the criteria are added together. A probability distribution is created from the weights assigned to each cell. Since decision making has uncertainty associated with it—because complete knowledge of all the elements is never known—a random value is extracted from this distribution. The cells that are most preferred by all criteria will be represented more in the distribution and therefore have a higher probability of being selected. However, because this is a random selection based on weighted values, any cell has a chance of being selected, with the least preferred having less probability. Generally, instead of creating one optimal deterministic run of a model, agent-based models are run hundreds of times, and the results are added together to produce probability maps of movement.

Conceptually, the algorithm for creating the distribution and cell selection for cougar movement is implemented as follows: Visualize a bucket within each of the eight cells surrounding the cougar agent. Within each bucket you will place marbles identifying the relative importance of the cell for the criterion. The more preferred the cell is for that criterion, the more marbles will be placed in that bucket. The number of marbles in each bucket is adjusted by multiplying the number of marbles in the bucket by the weight assigned to that criterion during that time step.

This process of adding marbles to buckets continues for each criterion. After completing this process for all the criteria, all the marbles are added together (creating the distribution) into one large bucket. Each marble in the large bucket knows which small bucket (the cell) it came from. You randomly pick one of the marbles out of the large bucket. Which smaller bucket (or cell) the marble you select originated from is the cell the cougar will move into. The cells that are preferred most for all criteria will have more marbles in the smaller buckets, therefore those cells will have more marbles represented in the larger bucket. The odds of having one of the marbles from the more preferred smaller buckets be selected is greater.

An array is created to store the security values of the eight surrounding cells (`securityValues = ArrayList()`). A variable storing the highest security value of the eight surrounding cells is initialized to negative infinity (`maxValue = Double.NEGATIVE_INFINITY`). For each of the eight surrounding cells identified in the `cells` array, the security value will be identified and the maximum security value for the eight surrounding cells will be tracked (`for cell as Cell in cells:`). In this loop, the security value is identified from the security raster (`val = raster.getPixelValue(cell.y, cell.x, 0)`). The security value is converted to a double (`securityValue = double(val)`). The security value for the cell is added to the array holding the security values for all eight cells (`securityValues.add(Double(securityValue))`). The security value for the cell is tested to determine if it contains the highest security value (`if (securityValue > maxValue)`), and if it does it is set to the variable storing the highest value (`maxValue = securityValue`).

The remaining code in the action creates the probability distribution for the criteria using the marble approach. (See the sidebar “Creating a probability distribution for determining the movement of an agent into surrounding cells.”) If the `maxValue` for all the cells is 0 (`if (maxValue == 0) :`), then none of the cell buckets will receive any marbles (`for cell as Cell in cells:; cell.marbles = 0.0`). The typical evaluation is `maxValue` is not equal to 0.0. If this is the case, then for each of the eight cells (`for index in range(0, cells.size()) :`), the security value is identified from the `securityValues` array (`securityValue = ((Double) securityValues.get(index)).doubleValue()`). The number of marbles for each cell based on security is divided by the maximum security value (`cell.marbles = securityValue / maxValue`). The total number of marbles for each cell is normalized so that it will be comparable to the other criteria (`cell.marbles = cell.marbles / marbleSum * weight`).

As a result of the `setMarblesForSecurity` action, based on the security contained within each of the eight surrounding cells, the appropriate number of marbles are put into the associated bucket. The `setMarblesForSecurity` action calls the `createCells` action to identify the eight cells surrounding the agent.

Look at the `createCells` action

- 7 In the Actions Editor, select `createCells` from the Actions drop-down list. The code identifies the eight cells surrounding the agent. The action is called from the `setMarblesForSecurity` action.

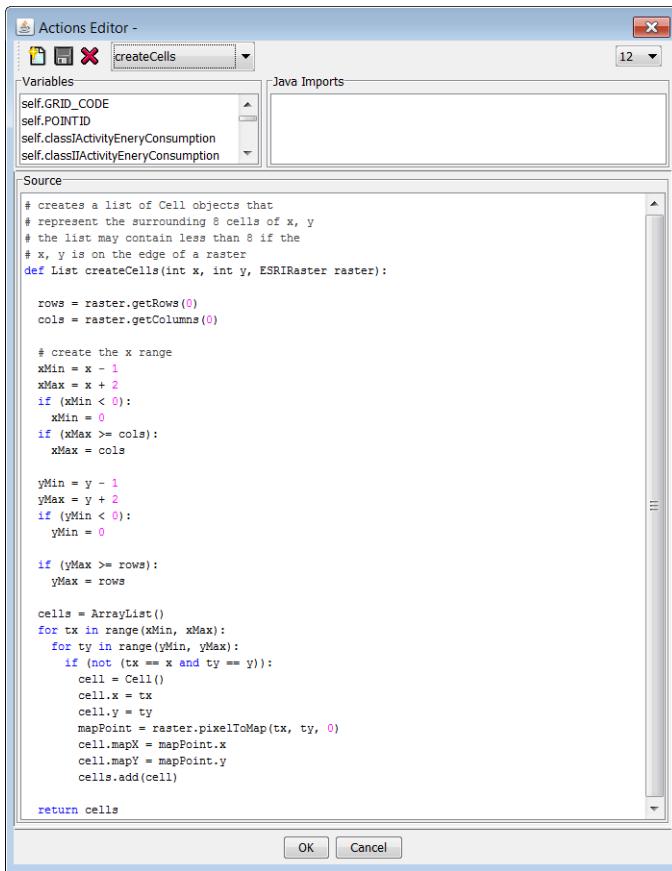
5

6

7

8

9



The `createCells` action is called by the `setMarblesForSecurity` action to identify the map coordinates of the eight cells surrounding the agent. The `x` and `y` row column of the cell the agent is in and the name of the security raster are passed to the `createCells` action (`def List createCells(int x, int y, ESRIRaster raster)`). The number of rows and columns in the security raster is determined to track whether the cell resides in a boundary cell (`rows = raster.getRows(0); cols = raster.getColumns(0)`). The extent in the `x` direction for the neighborhood cells is determined (`xMin = x - 1; xMax = x + 2`). (Note: `xMax` equals `x + 2` not `x + 1` because it is used in the `range` function later in the action, and the `range` function does not evaluate inclusively for the specified upper range.) If the neighborhood cells are beyond the study area's extent, then the neighborhood cell will be set to the minimum (`if (xMin < 0):; xMin = 0`) or the maximum, which is the maximum number of columns (`if (xMax > cols): xMax = cols`). The same procedure is determined for the `y` coordinate.

Each of the eight cells surrounding the agent is identified through a loop within a loop (`for tx in range(xMin, xMax):; for ty in range(yMin, yMax):`). A cell object is created for each cell (`cell = Cell()`) and the

`y(cell.y = ty)` cell location for each of the eight surrounding cells are identified. From each cell location, the map coordinates for each cell are identified (`mapPoint = raster.pixelToMap(tx, ty, 0)`). The x map coordinate is set to the `mapX` field (or property) for the associated cell object (`cell.mapX = mapPoint.x`) and the y map coordinate is set to the `mapY` property (`cell.mapY = mapPoint.y`). Each cell object is added to the `cells` array (`cells.add(cell)`). Once the map coordinates for each of the eight surrounding cells are added to the `cells` array, the array is returned to the calling action, `setMarblesForSecurity` (`return cells`). The `setMarblesForSecurity` action now registers the x, y map position for each of the eight cells surrounding the agent and can then identify the security of those cells.

- 8** Close the Actions Editor.
- 9** Close the Agent Analyst model. Do not save any changes.
- 10** Close ArcMap. Do not save any changes.

Through the `setMarblesForSecurity` action, the cougar agents will now be aware of the security values for each of the eight cells surrounding them and weight them appropriately for the criterion that will be used when determining where to move.

5

6

7

8

9

Using probability and random numbers to determine stochastic events—making a kill and weighting the eight surrounding cells for the remaining with the kill criterion

The process of placing marbles in buckets located in the eight cells surrounding a cougar agent to determine its movement occurs for each criterion. The second criterion that you will explore is remaining with the kill (if a kill was made).

There are three aspects for the remaining with the kill criterion:

- Making a kill
- How long it takes to consume the kill
- Weighting the eight surrounding cells each time step that the cougar agent is consuming the kill to keep the cougar agent near the carcass.

In the step action for the cougar agent, the cougar agent will first weight the eight surrounding cells based on the remaining with the kill criterion if a kill was made in a previous time step and if the cougar agent has not finished consuming the kill. The cougar agent will stay with that kill until it is consumed. For each time step the cougar agent is with the kill, the remaining with the kill criterion is applicable.

If the cougar agent is not remaining with a kill, this criterion does not apply for that time step. After the cougar agent makes a move in the time step based on the other criteria, the cougar agent will then hunt in the cell it moves into. If it is successful and makes a kill, then the type of prey that was killed is determined and the appropriate fields set so the remaining with the kill criterion will be valid for the appropriate number of subsequent time steps.

Exercise 5c

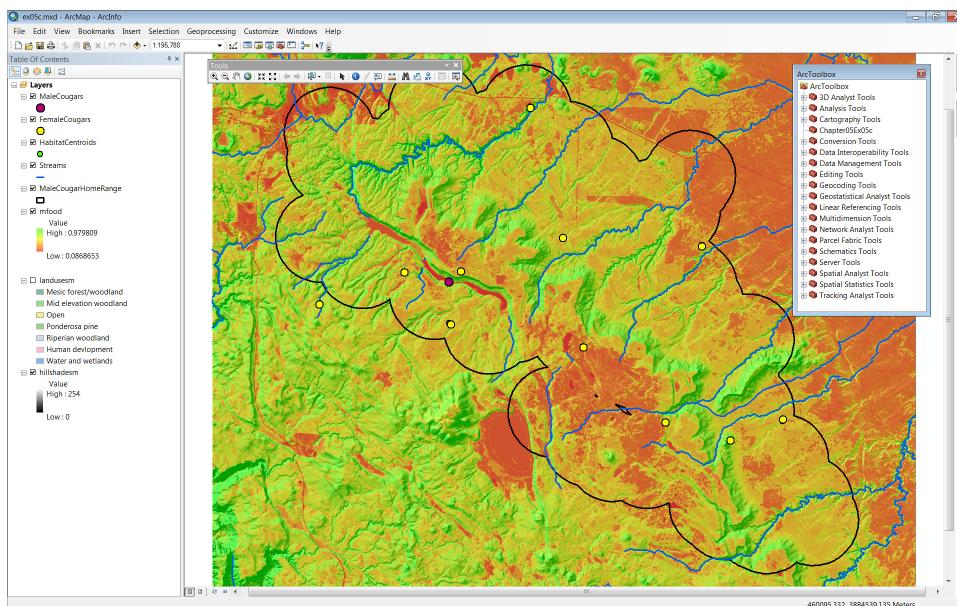
In this exercise you will first examine the code for weighting the eight cells surrounding an agent for the remaining with the kill criterion if the cougar agent has made a kill in a previous time step and is still consuming it. Then you will explore how the cougar agent hunts in the cell it has moved into if it is not with a kill. You will see how random numbers are used when determining if the cougar agent actually makes a kill or not.

First, examine the process of weighting the eight surrounding cells based on the remaining with the kill criterion if a kill has been made in an earlier time step.

Load the ArcMap document

- 1 Start ArcMap. Navigate to C:\ESRIPress\AgentAnalyst\Chapter05. Open ex05c.mxd.

In the ArcMap display you will see the mfood layer identifying the probability of making a kill at each location overlaid on top of the hillshade. The purple dots represent the male cougars, the yellow dots the female cougars, and the light green dots are the habitat centroids (the centroids are somewhat hidden by the female cougars). In ArcToolbox, the Chapter05Ex05c toolbox has been created for you.



The mfood layer was created by applying logistic regression formulas for each male cougar. These raster probability surfaces reflect the male preference to secure food resources by using rugged terrain in mesic forests, pinyon and juniper woodlands, and open or non-forested areas. Probability surfaces were created individually for each cougar and then averaged to create the final mfood raster surface.

The distribution layers for each species are based on field observations.

Load the Agent Analyst model

- 2 Right-click the Chapter05Ex05c toolbox, point to New, and select Agent Analyst Tool.
- 3 When the Agent Analyst window appears, click File and select Import. Do not save the changes when prompted.
- 4 Navigate to C:\ESRIPress\AgentAnalyst\Chapter05\Models and open ex05c.sbp.

5

6

7

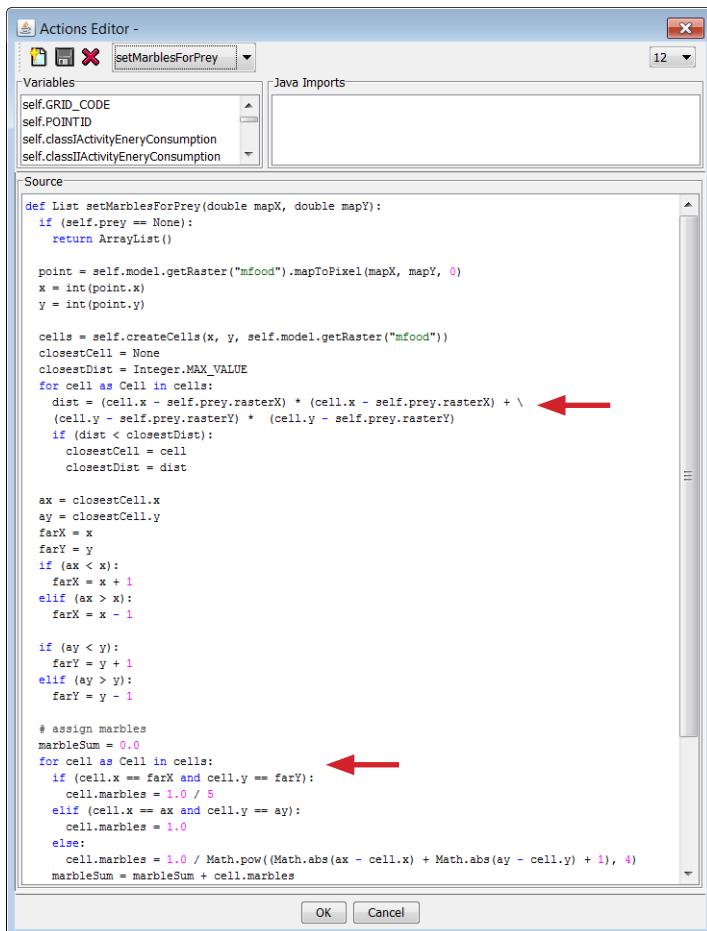
8

9

If the cougar agent made a kill in a previous time step and is still consuming it, the setMarblesForPrey action is used to allocate the appropriate marbles to each bucket in the eight surrounding cells for the remaining with the kill criterion. This action is similar to the setMarblesForSecurity action, which allocates the marbles for the security criterion.

View the action

- 5 In the Environment panel, click MaleCougar. In the Property panel, click the Edit button to the right of the Actions property to open the Actions Editor.
- 6 In the Actions Editor, select setMarblesForPrey from the Actions drop-down list. The action weights each of the eight surrounding cells for the remaining with the kill criterion. Notice the code that the arrows are pointing to.



The first line of the code determines if the cougar agent is with a kill (if `(self.prey == None)`). If it is not, then do not apply the setMarblesForPrey action (`return`

`ArrayList()`), which means the remaining with the kill criterion is not applicable for this time step. If the cougar agent is with a kill, the remaining code in the `setMarblesForPrey` action will be implemented.

The code is very similar to the `setMarblesForSecurity` action. The action identifies the current cougar agent location, calls the `createCells` action to determine the eight cells surrounding the cougar agent, and allocates the appropriate number of marbles to each bucket. The main difference between the `setMarblesForSecurity` action and the `setMarblesForPrey` action is in how the weights for the eight surrounding cells are determined. In `setMarblesForSecurity`, the cell that provides the highest security is identified from a raster and receives the highest weight (the most marbles), with decreasing weights being applied to the cells as their relative security levels decrease. In `setMarblesForPrey`, the distance of the closest cell to the kill location is identified, and cells moving farther away from the kill decrease in preference. This code is identified by the top red arrow in the preceding image.

In the code, for each of the eight surrounding cells (`for cell as Cell in cells`), the x location of the kill (`self.prey.rasterX`) is subtracted from the x position (`cell.x`) of the cell being processed. The same occurs for the y coordinate. To determine the squared distance (the hypotenuse squared) of the cell that is being processed to the location of the current kill, the two differences are squared and added together (`dist = (cell.x - self.prey.rasterX) * (cell.x - self.prey.rasterX) + (cell.y - self.prey.rasterY) * (cell.y - self.prey.rasterY)`). Each cell is tested to see if it is the closest cell to the prey (`if (dist < closestDist)`), and the closest cell (`closestCell = cell`) and the closest distance (`closestDist = dist`) are tracked.

The eight surrounding cells are then weighted by the code identified by the lower red arrow (`for cell as Cell in cells:`). The farthest cell (`if (cell.x == farX and cell.y == farY) :`) is assigned a weight of 0.2 (`cell.marbles = 1.0 / 5`). The closest cell (`elif (cell.x == ax and cell.y == ay) :`) is assigned a weight of 1 (`cell.marbles = 1.0`). The remaining cells are weighted based on a decreasing function as they move farther from the closest cell (`cell.marbles = 1.0 / Math.pow((Math.abs(ax - cell.x) + Math.abs(ay - cell.y) + 1), 4)`). Finally, the weights are normalized so they can be compared to the other criteria (`cell.marbles = cell.marbles / marbleSum * weight`).

When a cougar is hunting, whether it kills the prey is dependent on the availability of the prey, the probability of catching the prey (determined by the roughness of the terrain), and how hungry the cougar is (its energetic state). Not only is it important to have prey available, the cougar must be able to catch it. Whether the cougar can catch the prey is based on the cougar's hunting advantage. The more rugged the landscape, the more hunting advantage the cougar will have since it can pounce on the prey. The `mfood` raster was created to weight each cell for the potential for a cougar agent to make a kill at that location.

5

6

7

8

9

The hunger level of the cougar will dictate the determination with which it pursues the prey. The hungrier the cougar is (the lower the energetic state), the more intently it will hunt (increasing the probability of a kill), resulting in higher success.

If the cougar agent is not with a kill and the cougar agent is not full, it will hunt in the cell it moves into during that time step. If the cougar does hunt, and a kill is made, the types of prey that are available at the location are identified and, based on distribution of prey killed in the wild, the type of prey killed is determined.

Once the cougar agent makes a move within a time step, you need to determine if the cougar agent will hunt and make a kill in the cell it moved into.

- 7 In the Actions Editor, select step from the Actions drop-down list. Notice the code that the arrows are pointing to.

```

Actions Editor - step
Variables: self.GRID_CODE, self.POINTID, self.classIActivityEnergyConsumption, self.classIActivityEnergyConsumption
Java Imports: 12
Source:
if (self.prey != None):
    timeDiff = self.model.getTickCount() - self.preyTimeStart ←
    # Stop eating if the prey is all gone
    if (timeDiff >= self.prey.consumeDuration):
        self.prey = None
        self.model.messageDisplay.addAlert("Finished eating")
        self.model.pause()
    ## Eating part of the prey - adding the prey's calorie amount / prey's consumeDuration, up to the
    ## cougars max energy reserve amount (when that happens, discard the rest of the prey)
    else:
        remainingRoom = self.maximumEnergyReserve - self.energyReserve
        self.model.messageDisplay.addAlert("Room left: " + remainingRoom)
        if (remainingRoom > self.prey.hourlyConsumptionAmount):
            self.energyReserve = self.energyReserve + self.prey.hourlyConsumptionAmount
            self.model.messageDisplay.addAlert("Energy added: " + self.prey.hourlyConsumptionAmount)
        else:
            self.energyReserve = self.energyReserve + remainingRoom
            self.model.messageDisplay.addAlert("Energy added: " + remainingRoom)
            self.prey = None
            self.model.messageDisplay.addAlert("Full - Finished eating")
            self.model.pause()

    self.model.messageDisplay.addAlert("Energy left: " + self.energyReserve)

## The cougar hunts when he has finished consuming the prey previously hunted and he is getting hungry
if (self.model.huntingOn and (self.prey == None and self.energyReserve < self.energyReserveForHuntingInterest)):
    self.hunt(curX, curY) ←

## Formula for energy consumed during this step
# If eating prey, use Class I activity consumption
# If <= 25,000 kcal : calories_used = 500
# If > 25,000 kcal : calories_used = - 1/156.25(kcal - 25,000) + 500
if (self.prey != None):
    energyConsumed = self.classIActivityEnergyConsumption
elif (self.energyReserve <= 25000.0):
    energyConsumed = 500.0
else:
    energyConsumed = 500 - (self.energyReserve - 25000)/156.25

```

The step action is called each time step. The upper arrow points to the code managing the fields for an existing kill, and the lower arrow points to the code that determines if the cougar will hunt.

Examining the code a little more closely, you can see that the code highlighted by the upper arrow checks to see if the cougar agent has made a kill in a previous time step and is not done consuming it (`if (self.prey != None)`). If the cougar is with a kill, the prey field is not None, indicating the cougar is with a kill. Therefore, the code calculates how long the cougar has been consuming the kill (`timeDiff = self.model.getTickCount() - self.preyTimeStart`). If the time the cougar agent has been consuming the kill is greater than how long it takes to consume that kill (`if (timeDiff >= self.prey.consumeDuration)`), then a series of fields are set indicating that the cougar agent is done consuming the kill. If it is not done eating the kill, the cougar agent will continue eating in the else part of the if construct. If the cougar agent is to continue to eat, the amount of room that is left in its stomach is calculated (`remainingRoom = self.maximumEnergyReserve - self.energyReserve`). If the cougar has enough room (`if (remainingRoom > self.prey.hourlyConsumptionAmount)`), it will continue to eat, and the appropriate amount of energy for consuming the kill that time step is added to the cougar agent's energy reserve (`self.energyReserve = self.energyReserve + self.prey.hourlyConsumptionAmount`).

The code highlighted by the lower arrow in the preceding image determines if the hunting parameter is on, if the cougar agent does not have a kill it is consuming, and if its energy level is low (`if (self.model.huntingOn and (self.prey == None and self.energyReserve < self.energyReserveForHuntingInterest))`). If all these conditions are true, the cougar agent will hunt. The hunt action is called, passing the current location of the cougar agent to the action (`self.hunt (curX, curY)`).

If the cougar is going to hunt, determine whether the agent kills a prey

- 8 In the Actions Editor, select hunt from the Actions drop-down list. View the following code in the Source panel. The hunt action determines whether a kill is made and the type of prey killed.

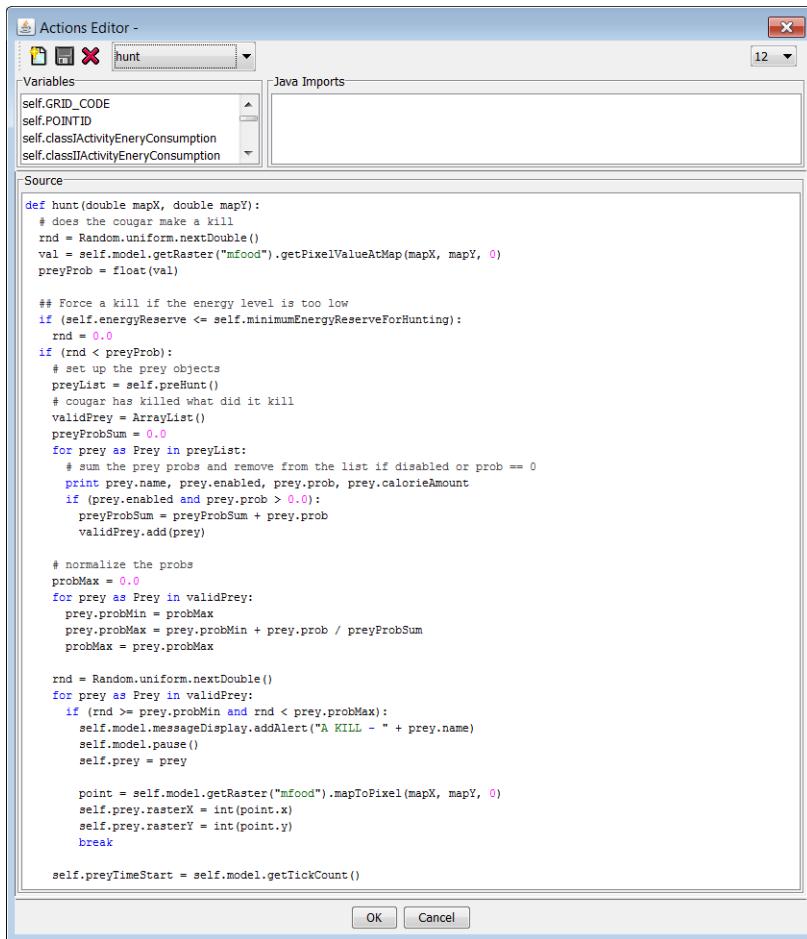
5

6

7

8

9



In the hunt code, the map coordinates of the location of the agent are passed to the action (`hunt(double mapX, double mapY)`). Whether the agent will kill or not will be based on a random number, therefore a random number must be created (`rnd = Random.uniform.nextDouble()`). Before the model was run, a raster layer called mfood was created from field data. It stores the likelihood of finding prey at each cell. The probability of finding prey at the location where the agent moved is identified (`val = self.model.getRaster("mfood").getPixelValueAtMap(mapX, mapY, 0)`). The probability value identified from the mfood raster dataset is converted to a floating-point value (`preyProb = float(val)`). If the random number that was created earlier is less than the probability identified from the mfood raster, then the agent makes a kill during the time step (`if (rnd < preyProb) :`). Cells where the cougar agent has a better chance of making a kill, or has a hunting advantage, are captured in the mfood raster. Because the selected random number (`rnd`) is evaluated based on the mfood value (`preyProb`) at the location, the greater the probability of killing prey, the higher the value of `preyProb` will be, and therefore the better the chance `rnd` will be lower. If a kill is made, the following code is implemented.

The preHunt action is called, and the action returns an array of prey agents that identify the type of prey that might be encountered in the study area (`preylist = self .preHunt ()`). For each prey agent in the preylist, the preHunt action also determines if that prey might be encountered at the cougar agent's current location. If so, the enable field on the prey agent is set to TRUE. The details of this action are discussed later in this chapter.

The hunt action produces another array (`validPrey = ArrayList ()`) from the preylist returned from the preHunt action, which will contain only the prey agents that might be encountered at the cougar agent's current location. To populate the validPrey array, for each prey agent in the preyList (`for prey as Prey in preyList:`), the hunt action determines if the prey might be encountered at the cougar agent's cell location by querying whether the enabled field is set to TRUE for the prey agent. If the field is set to TRUE and the probability of encountering the prey is greater than 0 (`if (prey .enabled and prey.prob > 0.0):`), that prey probability is added to a running sum (`preyProbSum = preyProbSum + prey.prob`). Since the prey that is being tested can be encountered, it is a valid prey to consider for the agent to kill, and therefore it will be added to the array storing the available prey (`validPrey.add(prey)`).

You must now determine which of the available prey in the cell is actually killed in the model. Since the sum of the probabilities must equal 1, the probabilities for each of the possible prey are normalized to provide the appropriate relative weighting for each species (`for prey as Prey in validPrey:`). The available probabilities will be distributed in a 0 to 1 range. The higher the probability that a species of prey will be encountered in the wild, the greater its range in the 0 to 1 range. The minimum probability value for the start of the first available prey is set in the 0 to 1 probability range for the prey (`prey .probMin = probMax`). Notice that probMax was initialized to 0.0 prior to entering the for loop. The maximum value for the prey probability is set in the 0 to 1 range for the prey (`prey.probMax = prey.probMin + prey.prob / preyProbSum`). The minimum probability value for the next available species is set to the maximum value for the previous potential prey. This process continues for each of the potential prey that might be encountered at the location. As a result, a 0 to 1 probability distribution is created of potential prey the cougar might kill in the time step. For example, 0 to .32 will represent a rabbit, .32 to .70 a yearling elk, and .70 to 1 an adult deer. The yearling elk has the greatest probability of being selected since it is represented by a larger range (.38).

Note: This is a distribution range and does not represent a preference scale. The numbers 0 to 1 represent the range of probability but do not imply a preference value (i.e., 1 is not better than 0).

To identify which prey will be killed by the agent in the cell it moves into in that time step, a random number between 0 and 1 is determined (`rnd = Random.uniform.nextDouble ()`). The random number is tested (`for prey as Prey in validPrey:`) against the probability ranges defined in the 0 to 1 range created for each species (`if (rnd >= prey.probMin and rnd < prey.probMax):`). If the random number falls within a range of one of the species, that species becomes the one that is killed in that

5

6

7

8

9

time step, the program sends a message to the user (`self.model.messageDisplay.addAlert("A KILL - " + prey.name)`), and the prey field for the cougar agent is set to that species (`self.prey = prey`). In our example, if the selected random number is, for example, set to .56, then a yearling elk will be killed.

The location of the prey in map units is set to the appropriate fields for the cougar agent (`self.prey.rasterX = int(point.x); self.prey.rasterY = int(point.y)`). This location will become an attractor for remaining with the kill until the agent finally consumes the kill. Therefore, the time of the kill has to be registered (`self.preyTimeStart = self.model.getTickCount()`).

9 Close the Agent Analyst model. Do not save any changes.

10 Close ArcMap. Do not save any changes.

In this exercise you set the weights for the eight cells surrounding a cougar relative to the remaining with the kill criterion, when a kill was made in a previous time step and the kill has not been fully consumed. If the cougar agent is not with a kill, once the cougar made a move (movement will be fully explained in chapter 8), you learned how the cougar agent makes, or doesn't make, a kill. In the next exercise you will learn how the type of prey that is killed is identified (if a kill is made).

Using raster attributes and field observation data to determine the characteristics of an event—identifying the type of kill and how long it will take the cougar agent to consume it

In the previous exercise you saw how a kill was made from the available prey, but how the available prey was identified in the model was not addressed. The type of prey that is killed influences how long the cougar agent will stay with the kill, which is realized in the model through the remaining with the kill criterion. You will see in chapter 8 that the type of prey that is killed also affects the energy gained per hour, which in turn affects the weighting of the criteria in subsequent steps.

Exercise 5d

In this exercise you will see how the types of prey are identified and how the fields for this action are set to define the characteristics of the prey that will be used in the decision making of the cougar agent.

Because of the complexity of modeling the prey individually, the prey are not represented as agents but by seasonal raster distribution surfaces of the known locations of the individual prey species. If the species might be present at a particular location during that season, the cell is assigned a 1. Otherwise, the cell is assigned a 0. There are multiple layers for each prey species representing the seasonal distribution of the species. The prey agents used in this model store only the available prey at a particular location, which is determined from these distribution rasters (as seen in the previous exercise). These prey agents are not agents that move and make decisions like the cougar agents.

Once the cougar agent makes a kill in the cell it moves into, the individual prey-species distribution layers are queried to see what potential prey might have been killed. Based on collared cougar kills (see table 5.5 at the end of this chapter), probabilities are assigned to each available prey. The more the prey type is killed by a cougar in the wild, the higher the probability is that the prey type will be selected (if available) by the cougar agent.

You will now determine the prey that is available in the study area and which is possibly available at the current location of the cougar agent.

Load the ArcMap document

- 1 Start ArcMap. Navigate to C:\ESRIPress\AgentAnalyst\Chapter05. Open ex05d.mxd.

5

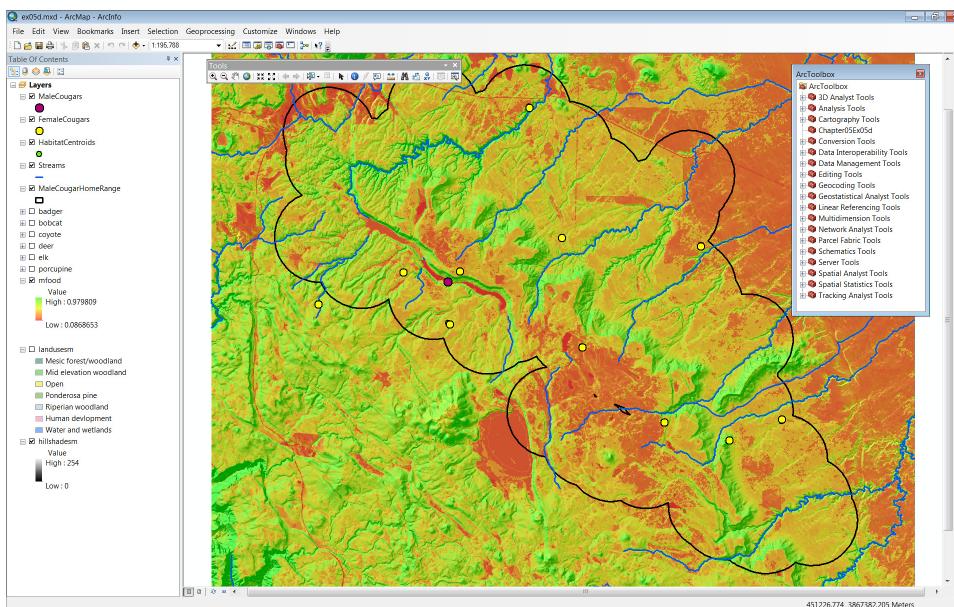
6

7

8

9

In the ArcMap display you will see the mfood layer identifying the probability of making a kill at each location (based on the terrain and vegetation types) overlaid on top of the hillshade. The purple dots represent the male cougars, the yellow dots the female cougars, and the light green dots are the habitat centroids (the habitat centroids are somewhat hidden by the female cougars). Explore the distributions of each prey found in the table of contents by turning them on and off. These prey distribution layers will be used to determine which prey type is killed at a particular location when a cougar makes a kill. If a prey type is not available at a particular location where a cougar makes a kill, then that prey type cannot be considered in that kill.



In ArcToolbox, the Chapter05Ex05d toolbox has been created for you.

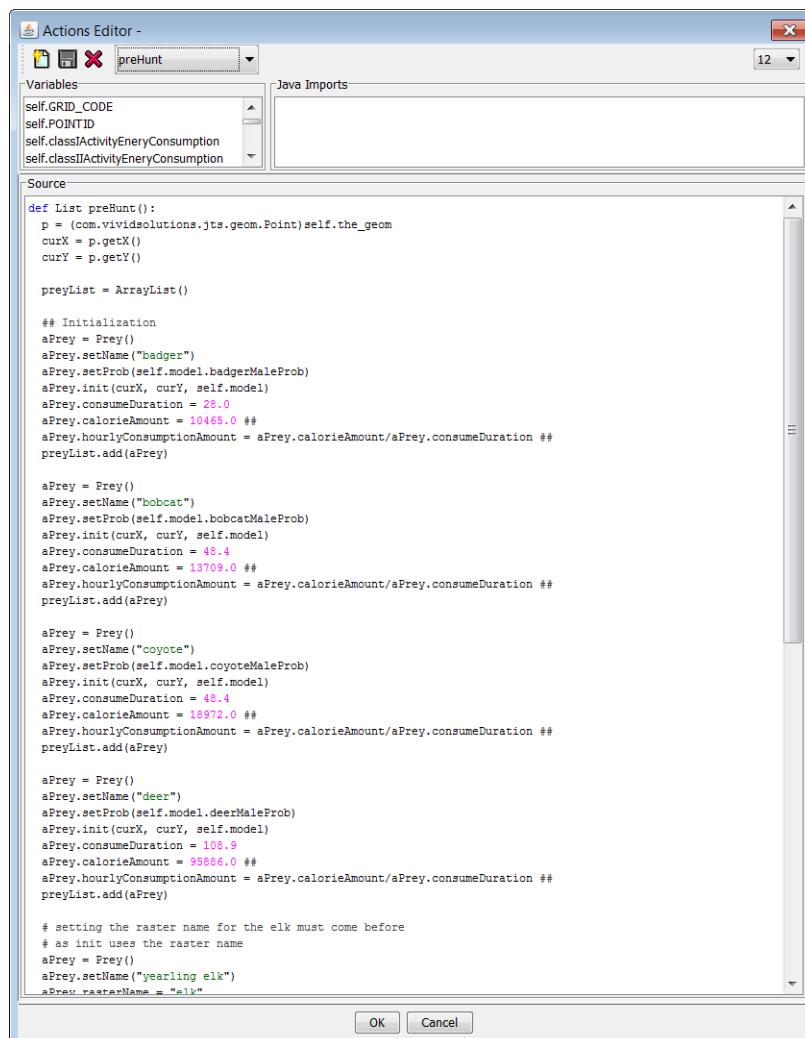
Load the Agent Analyst model

- 2 Right-click the Chapter05Ex05d toolbox, point to New, and select Agent Analyst Tool.
- 3 Once the Agent Analyst window appears, click File and select Import. Do not save the changes when prompted.
- 4 Navigate to C:\ESRIPress\AgentAnalyst\Chapter05\Models and open ex05d.sbp.

You will view the preHunt action, which determines the available prey in the study area, the prey available at the cougar agent's current location, and the characteristics of the prey (e.g., the probability the prey is encountered in the diet of cougars in the wild, the amount of calories gained by eating the prey, the amount of time it takes to consume the prey, and the amount of calories gained per time step when consuming the prey). Each of these

characteristics is determined from observational data. This action is called after a kill is made from the hunt action discussed in exercise 5c. The hunt action in exercise 5c narrows the list of available prey in the study area to the possible prey in the cell the cougar is hunting in based on the enabled field on the prey agent set in this exercise. If a kill is made, the characteristics associated with the prey are set and used to influence decision making in subsequent steps.

- 5 In the Environment panel, click MaleCougar. In the Property panel, click the Edit button to the right of the Actions property to open the Actions Editor.
- 6 In the Actions Editor, select preHunt from the Actions drop-down list. View the following code in the Source panel. The action identifies the possible prey at the study site and sets a series of fields or properties for each prey type.



The screenshot shows the 'Actions Editor' dialog box. At the top, the title bar says 'Actions Editor -' and the dropdown menu shows 'preHunt'. Below the title bar are two tabs: 'Variables' and 'Java Imports'. The 'Variables' tab lists several variables: self.GRID_CODE, self.POINTID, self.classIActivityEnergyConsumption, and self.classIIActivityEnergyConsumption. The 'Java Imports' tab shows '12' imports. The main area is the 'Source' tab, which contains the following Python code:

```

Actions Editor - 
Variables Java Imports 12
preHunt
self.GRID_CODE
self.POINTID
self.classIActivityEnergyConsumption
self.classIIActivityEnergyConsumption

Source
def List preHunt():
    p = (com.vividsolutions.jts.geom.Point)self.the_geom
    curX = p.getX()
    curY = p.getY()

    preyList = ArrayList()

    ## Initialization
    aPrey = Prey()
    aPrey.setName("badger")
    aPrey.setProb(self.model.badgerMaleProb)
    aPrey.init(curX, curY, self.model)
    aPrey.consumeDuration = 25.0
    aPrey.calorieAmount = 10465.0 #
    aPrey.hourlyConsumptionAmount = aPrey.calorieAmount/aPrey.consumeDuration ##
    preyList.add(aPrey)

    aPrey = Prey()
    aPrey.setName("bobcat")
    aPrey.setProb(self.model.bobcatMaleProb)
    aPrey.init(curX, curY, self.model)
    aPrey.consumeDuration = 45.4
    aPrey.calorieAmount = 13709.0 #
    aPrey.hourlyConsumptionAmount = aPrey.calorieAmount/aPrey.consumeDuration ##
    preyList.add(aPrey)

    aPrey = Prey()
    aPrey.setName("coyote")
    aPrey.setProb(self.model.coyoteMaleProb)
    aPrey.init(curX, curY, self.model)
    aPrey.consumeDuration = 45.4
    aPrey.calorieAmount = 18972.0 #
    aPrey.hourlyConsumptionAmount = aPrey.calorieAmount/aPrey.consumeDuration ##
    preyList.add(aPrey)

    aPrey = Prey()
    aPrey.setName("deer")
    aPrey.setProb(self.model.deerMaleProb)
    aPrey.init(curX, curY, self.model)
    aPrey.consumeDuration = 108.9
    aPrey.calorieAmount = 95886.0 #
    aPrey.hourlyConsumptionAmount = aPrey.calorieAmount/aPrey.consumeDuration ##
    preyList.add(aPrey)

    # setting the raster name for the elk must come before
    # as init uses the raster name
    aPrey = Prey()
    aPrey.setName("yearling elk")
    aPrey.rasterName = "elk"

```

At the bottom of the dialog box are 'OK' and 'Cancel' buttons.

5

6

7

8

9

The code identifies the types or species of potential prey a cougar agent might encounter in the study area. For each prey species, a prey agent is created (`aPrey = Prey()`). For the species, a series of fields or properties are defined for the prey agent. First the name is specified (`aPrey.setName("badger")`). Then the probability of killing that prey is set (`aPrey.setProb(self.model.badgerMaleProb)`). The probability of the prey in a cougar's diet in the wild was derived by examining the distribution of prey in the diet of actual male cougars. Next, the init action is called for the prey agent to query the prey distribution raster to determine if the prey might be present at the current location of the cougar agent (`aPrey.init(curX, curY, self.model)`). If the prey type is possibly present, the enabled field is set to TRUE (see step 11 later in this exercise). The time it will take a cougar to consume the particular prey is set (`aPrey.consumeDuration = 28.0`). The length of time it will take to consume the prey is derived from field observation. The amount of calories gained each hour the cougar agent feeds on the prey is set (`aPrey.hourlyConsumptionAmount = aPrey.calorieAmount/aPrey.consumeDuration`). This calorie amount is added to the cougar's energetics level each hour. This information can be found in table 5.4: Energy obtained per prey item—Field data from collared cougars in the Flagstaff Uplands.

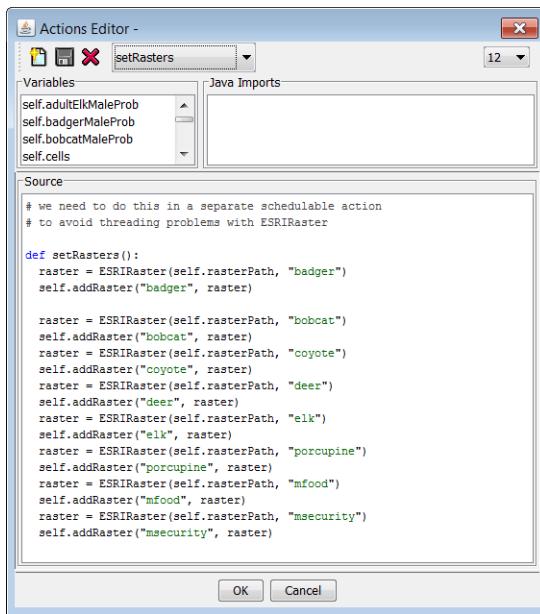
We now know which prey are available in the study area and their characteristics, but we also need to know where each prey exists within the study area. There can be a separate raster for each prey for different seasons identifying the distribution of the prey in the seasonal time frame. For simplicity, in this model only one set of distribution rasters is used. If prey is present, the cell will be assigned a 1, and if prey is not present, a 0 is assigned. The rasters are identified in the initialization process of the model. The `setRasters` action is called by the scheduler at tick 0.1, therefore the action becomes part of the initialization process of the model.

View the initialization process

- 7 In the Environment panel, click Cougar Model. In the Property panel, click the Edit button to the right of the Schedule property to open the Schedule Editor. Notice in the Action Execution panel that `setRasters` is scheduled to Tick 0.1.

View the `setRasters` action

- 8 Close the Schedule Editor. In the Environment panel, click Cougar Model if it isn't already selected. In the Property panel, click the Edit button to the right of the Actions property to open the Actions Editor.
- 9 In the Actions Editor, select `setRasters` from the Actions drop-down list. View the Source panel. The distribution rasters for each prey agent are identified in the code.

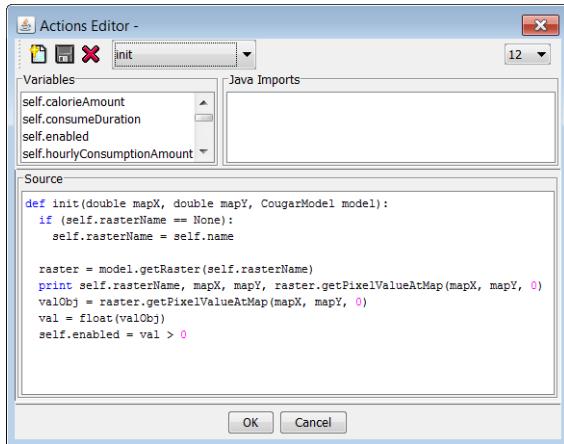


In the code, the names of the raster datasets containing the distributions for each prey are identified. These rasters will be queried to determine if the species is possibly in the cell the cougar agent moved into. This action is called only once at the start of the model.

As mentioned earlier, the preHunt action calls the init action on the prey agent (`aPrey.init(curX, curY, self.model)`) to determine if the prey is available at the current location of the cougar agent. If the prey might be there, that prey has a possibility of being killed as determined by setting the prey.enabled field to TRUE.

View the init action on the prey agent, used to determine the available prey type at the cell the cougar agent moved into

- 10** In the Environment panel, click Prey. In the Property panel, click the Edit button to the right of the Actions property to open the Actions Editor.
- 11** In the Actions Editor, select init from the Actions drop-down list. View the following code in the Source panel. Each raster representing the distribution for a particular prey indicates whether the prey might be at the cell the cougar moved into.



In this code, the current location of the cougar agent is passed to the action in map units (def init(double mapX, double mapY, CougarModel model):). The associated raster for the prey is identified (raster = model.getRaster(self.rasterName)). The value from the prey distribution raster for the location of the cougar agent is identified (valObj = raster.getValueAtMap(mapX, mapY, 0)). The value from the raster is converted to a floating-point value (val = float(valObj)). If the value is greater than 0, the prey is possibly in the cell, and the enabled field is set to TRUE (self.enabled = val > 0).

From these actions, the possible prey that the cougar might kill and which prey is killed will be identified.

When a kill is made, it takes time for the cougar agent to consume the kill. The cougar agent will remain with the kill and not wander far from it until it is consumed.

12 Close the model. Do not save the changes.

13 Close ArcMap. Do not save the changes.

Thus far in this chapter, you have set the weights for preference for two criteria, security and remaining with the kill. Once the cougar makes a move, and if the cougar is not already consuming a carcass, you determined if a kill is made in the cell the cougar agent moved into. If a kill is made, you identified the type of prey that was killed. In the next exercise you will weight the eight cells surrounding a male cougar relative to the third criterion, pursuing a female cougar agent.

Detecting and tracking other agents using attractors—weighting the eight surrounding cells for the pursuing a female cougar agent criterion

A male cougar must first sense a female cougar before he can pursue mating. Field data indicates that he can sense a female if she is within 3,000 meters of him. The closer she is, the greater possibility he will sense her. Once he detects her, he then needs to locate her.

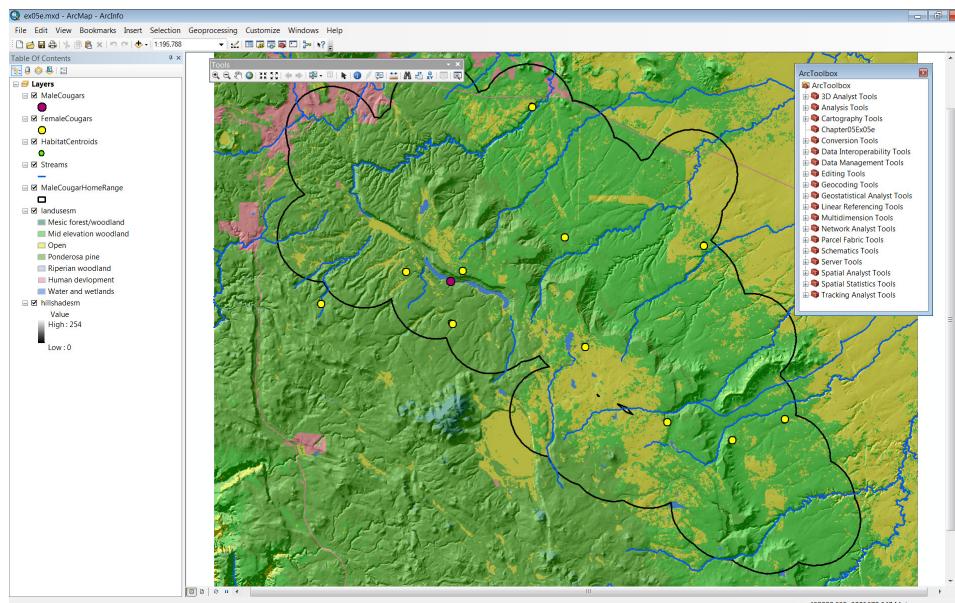
Once the male cougar agent detects a female cougar agent, depending on his energetic state and other attractors that are influencing his decisions, he may go looking for the female with the intent to mate. Once the male cougar agent finds the female, she becomes a strong attractor for 12 hours (again based on field data).

Exercise 5e

Add the pursuing a female cougar criterion to the model.

Load the ArcMap document

- 1 Start ArcMap. Navigate to C:\ESRIPress\AgentAnalyst\Chapter05. Select ex05e.mxd, and click Open.



In the ArcMap display you will see land use displayed transparently on top of a hillshade. The purple dot represents a male cougar and the yellow dots represent female cougars.

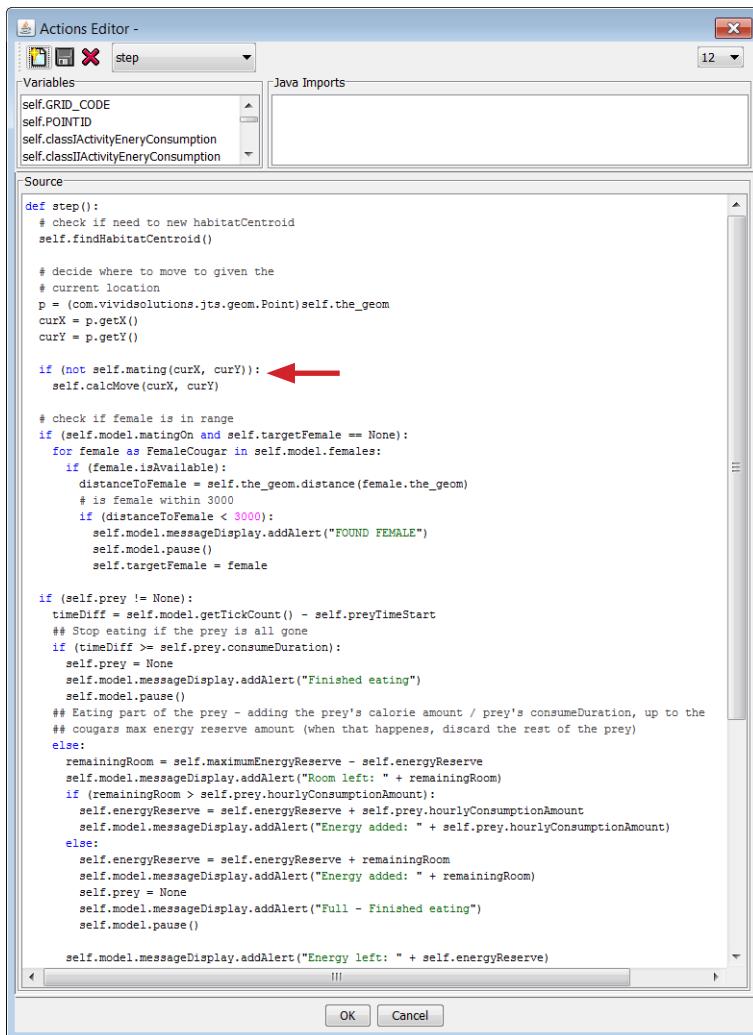
In ArcToolbox, the Chapter05Ex05e toolbox has been created for you.

Load the Agent Analyst model

- 2 Right-click the Chapter05Ex05e toolbox, point to New, and select Agent Analyst Tool.
- 3 Once the Agent Analyst window appears, click File and select Import. Do not save the changes when prompted.
- 4 Navigate to C:\ESRIPress\AgentAnalyst\Chapter05\Models and open ex05e.sbp.

Add the action for detecting a female cougar

- 5 In the Environment panel, click MaleCougar. In the Property panel, click the Edit button to the right of the Actions property to open the Actions Editor.
- 6 In the Actions Editor, select step from the Actions drop-down list. View the code in the Source panel. In the action, the male cougar will determine if it is mating, and if it is not mating, it will make a move. After the move, the male cougar agent will determine if it has detected a female.

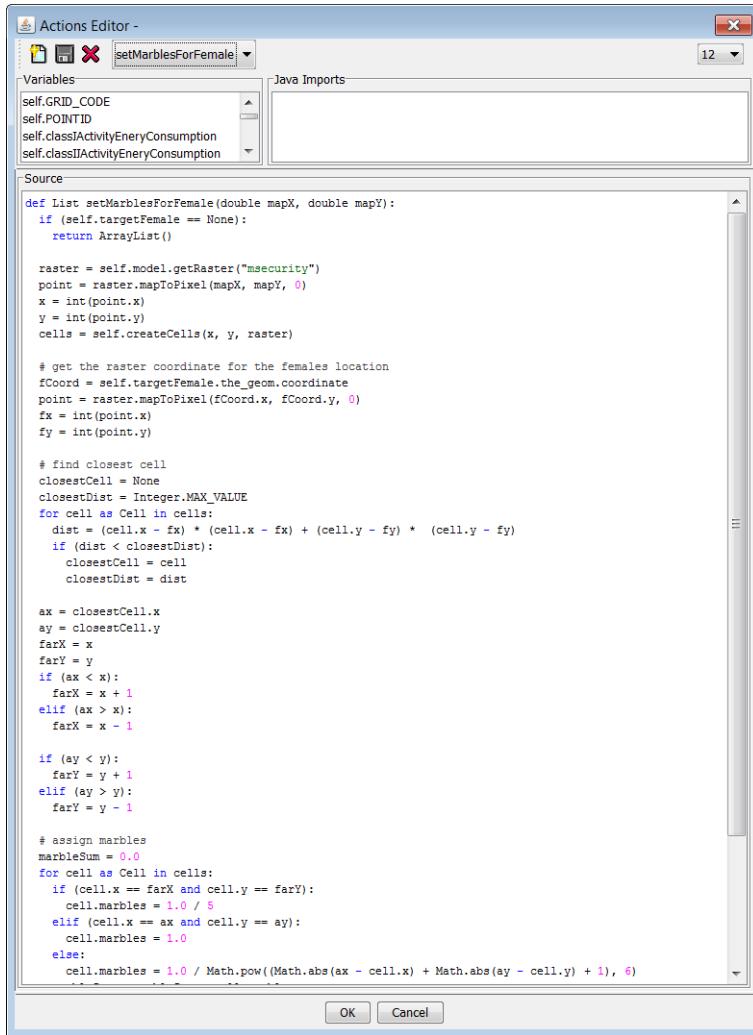


In this code, the arrow points to the code that determines if the cougar agent is currently mating or not (`if (not self.mating(curX, curY)):`). To determine if a male cougar agent is mating, the mating action is called, which will be discussed in chapter 8. If the male cougar agent is not mating, it will decide which of the eight surrounding cells it should move into (`self.calcMove(curX, curY)`). If it is mating, it will stay with the female, and all other criteria will be ignored (security, staying within the home range, and so on). If the male cougar agent is not mating, after he moves, he checks to see if he has

detected a female cougar in a previous step and is moving toward it (`if (self.model.matingOn and self.targetFemale == None) :)`. (This check will be done by the male cougar agent even if he is mating, but the check will be irrelevant since he is currently with a female cougar.) If the male cougar has not identified a female cougar to move toward, then the model will query each female cougar (`for female as FemaleCougar in self.model.females :)`). If a female is not mating with another male cougar (`if (female.Available) :)` and is within 3,000 meters (`if distanceToFemale < 3000) :)`, then that female is set as the target female that the male cougar agent will move toward (`self.targetFemale = female`). The pursuing a female cougar agent criterion will be applied in the next step.

Weight the eight surrounding cells relative to the pursuing a female cougar agent criterion

- 7 In the Actions Editor, select `setMarblesForFemale` from the Actions drop-down list. Uncomment the code in the Source panel so that it matches the code in the following image. The action weights the eight cells surrounding a male cougar agent based on how close the cell is to the detected female cougar agent.



The screenshot shows the 'Actions Editor' dialog box with the title 'Actions Editor - setMarblesForFemale'. The interface includes tabs for 'Variables' (containing self.GRID_CODE, self.POINTID, self.classIActivityEnergyConsumption, self.classIICactivityEnergyConsumption), 'Java Imports' (empty), and 'Source' (containing the provided Java code). The code implements a method to calculate the closest cell to a target female and assign marbles based on its distance from the male agent's current position.

```

Actions Editor - setMarblesForFemale
Variables
self.GRID_CODE
self.POINTID
self.classIActivityEnergyConsumption
self.classIICactivityEnergyConsumption
Source
def List setMarblesForFemale(double mapX, double mapY):
    if (self.targetFemale == None):
        return ArrayList()

    raster = self.model.getRaster("msecurity")
    point = raster.mapToPixel(mapX, mapY, 0)
    x = int(point.x)
    y = int(point.y)
    cells = self.createCells(x, y, raster)

    # get the raster coordinate for the females location
    fCoord = self.targetFemale.the_geom.coordinate
    point = raster.mapToPixel(fCoord.x, fCoord.y, 0)
    fx = int(point.x)
    fy = int(point.y)

    # find closest cell
    closestCell = None
    closestDist = Integer.MAX_VALUE
    for cell as Cell in cells:
        dist = (cell.x - fx) * (cell.x - fx) + (cell.y - fy) * (cell.y - fy)
        if (dist < closestDist):
            closestCell = cell
            closestDist = dist

    ax = closestCell.x
    ay = closestCell.y
    farX = x
    farY = y
    if (ax < x):
        farX = x + 1
    elif (ax > x):
        farX = x - 1

    if (ay < y):
        farY = y + 1
    elif (ay > y):
        farY = y - 1

    # assign marbles
    marbleSum = 0.0
    for cell as Cell in cells:
        if (cell.x == farX and cell.y == farY):
            cell.marbles = 1.0 / 5
        elif (cell.x == ax and cell.y == ay):
            cell.marbles = 1.0
        else:
            cell.marbles = 1.0 / Math.pow((Math.abs(ax - cell.x) + Math.abs(ay - cell.y) + 1), 6)

```

This code first identifies the cell the male cougar agent moved into (`point = raster.mapToPixel(mapX, mapY, 0)`). If no female agent has been detected, then do nothing (`if (self.targetFemale == None):; return ArrayList()`) and weight the next criterion.

If a female has been detected, then the eight cells surrounding the male cougar agent are identified (`cells = self.createCells(x, y, raster)`). The map coordinates for the identified female agent from the previous step are identified (`fCoord = self.targetFemale.the_geom.coordinate`) and then its cell location (`point = raster.mapToPixel(fCoord.x, fCoord.y, 0)`). For each of the eight cells surrounding the male cougar agent (`for cell as Cell in cells:`), the squared

5

6

7

8

9

distance from the surrounding cell to the female cougar is calculated (`dist = (cell.x - fx) * (cell.x - fx) + (cell.y - fy) * (cell.y - fy)`). The distance is tested to see if it is the closest to the female (`if (dist < closestDist) :`), then the closest cell is identified (`closestCell = cell`), and then the distance to the cell is identified as well (`closestDist = dist`). The position in map units identifying the closest of the eight cells surrounding the male cougar agent to the female is established and stored in the variables `farX` and `farY`. Each of the eight surrounding cell's (`for cell as Cell in cells:`) weight—or the number of marbles that will be placed in each bucket for moving into that cell—is established. If the surrounding cell is the farthest from the identified female cougar agent (`if (cell.x == farX and cell.y == farY) :`), it is set to 0.2 (`cell.marbles = 1.0 / 5`). If the surrounding cell is closest to the female cougar agent (`elif (cell.x == ax and cell.y == ay) :`), it is set to 1 (`cell.marbles = 1.0`). The cells farther from the target female cougar agent will receive decreasingly lower weights (`cell.marbles = 1.0 / Math.pow((Math.abs(ax - cell.x) + Math.abs(ay - cell.y) + 1), 6)`). The cell weights are normalized by the sum of marbles times the weight for the pursuing the female cougar criterion (`cell.marbles = cell.marbles / marbleSum * weight`).

Run the model

- 8** Click OK to save your changes in Actions Editor.
- 9** Click Bookmarks on the ArcMap menu and click ZoomToMale. (Depending on your previous actions and model runs, you may not see any change in the extent, and the male cougar may not be centered within the ZoomToMale extent. However, when you run the model, the male cougar will be repositioned).
- 10** Run the Cougar model.

When a female is found, the message FOUND FEMALE is displayed in the Cougar Messages window and execution pauses. If you click Start again, the cougar might continue to search for the female if not interrupted by hunting. The model will pause again when mating begins.

- 11** Close the model. Do not save any changes.
- 12** Close ArcMap. Do not save any changes.

Three of the six criteria have now been implemented for the decision making for a male cougar agent. In chapter 8 you will complete the final three criteria for the cougar model and then add the code for the cougar agent to evaluate the six criteria and, based on their current state, learn how the agents make a decision.

GIS data, fields, and actions dictionaries

Table 5.1 Data dictionary of GIS datasets

| Dataset | Data type | Description |
|---------------------|-------------------|---|
| badger | raster | Distribution of badgers |
| bobcat | raster | Distribution of bobcats |
| coyote | raster | Distribution of coyotes |
| deer | raster | Distribution of deer |
| elk | raster | Distribution of elk |
| FemaleCougars | shapefile—point | The locations of the female cougar agents each time step. |
| fire | raster | The location of a burning fire. It is not actively used in the exercises. |
| firecostsm | raster | The susceptibility surface the fire moves across. It is not actively used in the exercises. |
| firestart | raster | The starting location of the fire. It is not actively used in the exercises. |
| HabitatCentroids | shapefile—point | The location of the habitat attractors. Derived from the probability of a kill dataset (mfood). |
| hillshade | raster | The hillshade of the study area |
| landusesm | raster | The land use types |
| MaleCougarHomeRange | shapefile—polygon | The home ranges of the male cougars |
| MaleCougars | shapefile—point | The locations of the male cougar agents each time step |
| mfood | raster | Probability of a kill; combines the hunting advantage for a cougar with prey density to determine for each location the probability of obtaining a kill. The roughness of the terrain is calculated by internal software from the USGS. The rougher terrain provides more hunting advantage to the cougars. |
| msecurity | raster | The level of security at each cell. The dataset defines the level of security for the cougar. The layer is mainly a combination of vegetation and distance from human activity. |

5
6
7
8
9

Table 5.1 Data dictionary of GIS datasets (*continued*)

| Dataset | Data type | Description |
|---------------|-------------------|--------------------------------|
| porcupine | raster | Distribution of porcupines |
| Streams | shapefile—lines | The streams |
| StudyBoundary | shapefile—polygon | The boundary of the study site |

Table 5.2 Fields dictionary for the cougar model

| Field | Data type | Description |
|----------------------|---|--|
| Cougar model | | |
| rasterPath | java.lang.String | The relative path to the animal distribution rasters |
| badgerMaleProb | double | The probability of badger in the male cougar diet |
| bobcatMaleProb | double | The probability of bobcat in the male cougar diet |
| coyoteMaleProb | double | The probability of coyote in the male cougar diet |
| porcupineMaleProb | double | The probability of porcupine in the male cougar diet |
| deerMaleProb | double | The probability of deer in the male cougar diet |
| yearlingElkMaleProb | double | The probability of a yearling elk in the male cougar diet |
| adultElkMaleProb | double | The probability of male elk in the cougar diet |
| scavengerElkMaleProb | double | The probability of a scavenger male elk in the cougar diet |
| matingOn | boolean | Whether the mating criteria is active |
| huntingOn | boolean | Whether the hunting criteria is active |
| messageDisplay | uchicago/src. simbuilder.util. MessageDisplay | A field for the message to display |

Table 5.2 Fields dictionary for the cougar model (*continued*)

| Field | Data type | Description |
|------------------------------------|--|---|
| MaleCougar agent | | |
| homeRangeBoundary | com.vividsolutions.jts.GeometryFactory | The polygon defining the home range boundary for the male cougar agent |
| factory | com.vividsolutions.jts.GeometryFactory | A geometry factory for storing the home range geometry |
| sharedPoint | com.vividsolutions.jts.GeometryFactory | The coordinates of a specified point geometry. Used in measuring the distance from the current cougar agent position to the home range polygon. |
| needHabCentroid | boolean | Identifies whether the male cougar agent needs to identify a habitat centroid to move toward |
| habCentroidPointX | integer | The x coordinate of the habitat centroid for the male cougar agent to move toward |
| habCentroidPointY | integer | The y coordinate of the habitat centroid for the male cougar agent to move toward |
| prey | Prey | The type of prey that was killed |
| preyTimeStart | double | Identifies the starting time for consuming a kill |
| targetFemale | FemaleCougar | The female cougar agent the male cougar agent has detected and will move toward |
| mating | integer | Indicates if the cougar agent is mating |
| matingTimeCount | integer | Identifies the starting time the cougar agents starts mating |
| energyReserve | double | Monitors how much energy the cougar agent has at each time step |
| minimumEnergyReserve-ForHunting | double | The minimum kcal threshold below which the cougar agent must focus all its effort on hunting to survive |
| classIActivityEnergy-Consumption | double | The amount of calories used each time step when performing class I activity |
| classIIActivityEnergy-Consumption | double | The amount of calories used each time step when performing class II activity |
| classIIIActivityEnergy-Consumption | double | The amount of calories used each time step when performing class III activity |

5

6

7

8

9

Table 5.2 Fields dictionary for the cougar model (*continued*)

| Field | Data type | Description |
|----------------------------------|-----------|---|
| MaximumEnergyReserve | double | Identifies the maximum amount of energy the cougar can have |
| securityweight | double | The weight determined for the security criterion for a given time step |
| homeRangeweight | double | The weight determined for the staying within the home range criterion for a given time step |
| huntingWeight | double | The weight determined for the hunting criterion for a given time step |
| femaleWeight | double | The weight determined for the pursuing a female criterion for a given time step |
| killedPreyWeight | double | The weight determined for the remaining with the kill criterion for a given time step |
| energyReserveFor-HuntingInterest | double | The minimum kcal threshold below which the cougar agent loses interest in hunting |
| Cell agent | | |
| x | integer | The x raster position of the cell agent |
| y | integer | The y raster position of the cell agent |
| mapX | double | The x map position of the cell agent |
| mapY | double | The y map position of the cell agent |
| Marbles | double | The number of marbles assigned to the cell agent |
| probMin | double | The minimum probability for a surrounding cell agent when calculating a distribution for determining a move |
| probMax | double | The maximum probability for a surrounding cell agent when calculating a distribution for determining a move |

Table 5.2 Fields dictionary for the cougar model (*continued*)

| Field | Data type | Description |
|---------------------------|------------------|---|
| Prey agent | | |
| prob | double | The probability of the prey in the male cougar diet |
| name | java.lang.String | The name of the prey agent |
| enabled | integer | Determines if the prey is available when a kill is made |
| rasterX | integer | The x location of a current kill |
| rasterY | integer | The y location of a current kill |
| rasterName | java.lang.String | The name of the raster containing the distribution of the prey |
| probMin | double | The minimum probability for the available prey agent when calculating a distribution for determining which prey is killed |
| probMax | double | The maximum probability for the available prey agent when calculating a distribution for determining which prey is killed |
| consumeDuration | double | How long it takes to consume the prey |
| calorieAmount | double | The total amount of calories gained from consuming the prey |
| hourlyConsumptionAmount | double | The amount of calories gained per hour consuming the prey |
| FemaleCougar agent | | |
| homePointX | double | The x coordinate of the habitat centroid the female cougar will move relative to |
| homePointY | integer | The y coordinate of the habitat centroid the female cougar will move relative to |
| isAvailable | integer | Indicates whether the female cougar agent is available for mating |
| mating | integer | Identifies whether the female cougar agent is mating |

5

6

7

8

9

Table 5.3 Actions dictionary for the cougar model

| Declared actions | Description |
|-------------------------|---|
| Cougar model | |
| initAgents | Initializes the cougar model |
| updateDisplay | Updates the ArcMap display |
| writeAgents | Writes the location of the agents to the identified shapefile |
| setRasters | Identifies the location of a series of species distribution rasters that will be used for the hunting criteria |
| showMessages | Displays messages to the user |
| setupHook | Deletes previous messages and establishes the “hook” into the mechanism that allows user code to be run |
| fire | Creates the fire model using ArcGIS Spatial Analyst |
| HomeRange | |
| step | The action to perform on the HomeRange agent each time step |
| Cougar agent | |
| step | The action that is performed on each male cougar agent each time step |
| setFactory | Sets a geometry factory and is necessary to create geometry |
| setSharedPointCoords | Sets the coordinates of self.sharedPoint. When self.sharedPoint is used after calling this action, self.sharedPoint will then have those coordinates. |
| move | Moves the male cougar agent |
| int | Initializes the male cougar agent |
| createCells | Identifies each of the eight surrounding cells |
| setMarblesForSecurity | Sets the weights for the eight surrounding cells relative to the staying secure criterion |
| setMarblesForHomeRange | Sets the weights for the eight surrounding cells relative to the staying within the home range criterion |
| findHabitatCentroid | Locates the habitat centroid that will act as the habitat attractor for the male cougar agent |
| setMarblesForHabitat | Sets the weights for the eight surrounding cells relative to the moving toward good habitat criterion |
| calcMove | Performs the calculations for determining the move for the male cougar agent |

Table 5.3 Actions dictionary for the cougar model (*continued*)

| Declared actions | Description |
|------------------------------|---|
| hunt | Determines if a prey is killed and the type of prey that is killed |
| preHunt | Initializes the prey characteristics and determines if the prey is available at the site of a kill |
| setMarblesForPrey | Sets the weights for the eight surrounding cells relative to the hunting criterion |
| setMarblesForFemale | Sets the weights for the eight surrounding cells relative to the pursuing a female criterion |
| mating | If the male agent is mating, the number of hours that the male agent is with the female agent is incremented. If the male agent is pursuing a female agent, the action determines if the male agent moved into position to mate with a female cougar agent. |
| calcWeights | Calculates the weights for each criterion based on the state of the male cougar agent |
| HabitatCentroid agent | |
| step | The action that is performed each time step on the HabitatCentroid agents |
| FemaleCougar agent | |
| step | The action that is performed each time step on the female cougar agents |
| setMarblesForHabitat | Sets the weights for the eight surrounding cells relative to the moving toward good habitat criterion. For simplicity, this is the only criterion used for determining the female cougar agent movement. |
| createCells | Identifies each of the eight surrounding cells |
| move | Moves the female cougar agent |

5

6

7

8

9

Table 5.4 Energy obtained per prey item—
Field data from collared cougars in the Flagstaff Uplands

| | N | Time (hr) | Mass (kg) | Cmass (kg) | % | kg/hour | kg/day | kcal | assim kcal |
|-------------------|----|--------------|--------------|---------------|------|---------|--------|--------|---------------|
| Prey | | | | | | | | | |
| Rabbit | 3 | 28.7 | 1.8 | 1.44 | 0.88 | 0.0502 | | 2203 | 1763 |
| Prey group | | | | | | | | | |
| Porcupine | 1 | 24 | 8.6 | 6.88 | 0.8 | 0.2867 | 6.88 | 10526 | 8421 |
| Badger | 1 | 32 | 8.5 | 6.8 | 0.8 | 0.2125 | 5.1 | 10404 | 8323 |
| Average | | 28 | | | | 0.2496 | 5.99 | 10465 | 8372 |
| Prey group | | | | | | | | | |
| Bobcat | 1 | 46 | 11.2 | 8.96 | 0.8 | 0.1948 | 4.6748 | 13709 | 10967 |
| Coyote | 13 | 50.7 | 15.5 | 12.4 | 0.8 | 0.2446 | 5.8698 | 18972 | 15178 |
| Average | | 48.4 | | | | 0.2197 | 5.2723 | 16340 | 13072 |
| Prey group | | | | | | | | | |
| Male deer | 11 | 94.2 | 74 | 51.8 | 0.7 | 0.5499 | 13.197 | 97902 | 84196 |
| Female deer | 8 | 140 | 59 | 47.2 | 0.8 | 0.3359 | 8.0626 | 89208 | 76719 |
| Adult deer | 8 | 92 | 66.5 | 53.2 | 0.8 | 0.5783 | 13.878 | 100548 | 86471 |
| Average | | 108 | | | | 0.5641 | 13.537 | 95886 | 82462 |
| Prey group | | | | | | | | | |
| Yearling elk | 14 | 71.9 | 120 | 40.26 | 0.34 | | | 76099 | 65445 |
| Female elk | 9 | 80.2 | 238 | 44.91 | 0.19 | | | 84884 | 73000 |
| Male elk | 4 | 91 | 325 | 50.96 | 0.16 | | | 96314 | 82830 |
| Average | | 85.6 | | | | | | 90599 | 77915 |
| Prey | | | | | | | | | |
| Scavenger elk | 1 | 32 | 238 | 17.92 | 0.8 | | | 33869 | 29127 |

Table 5.5 Proportional distribution of kill among prey types—
Field data from collared cougars in the Flagstaff Uplands

| Prey | Females | | Males | |
|------------------|-----------|------------|-----------|------------|
| | n | Proportion | n | Proportion |
| Rabbit | | | | |
| Porcupine/badger | 2 | 0.026 | 0 | 0 |
| Coyote/bobcat | 22 | 0.289 | 0 | 0 |
| Adult deer | 25 | 0.329 | 9 | 0.265 |
| Yearling elk | 13 | 0.171 | 13 | 0.382 |
| Scavenger elk | 3 | 0.039 | 2 | 0.059 |
| | 76 | | 34 | |

5

6

7

8

9

Section 3: Advanced techniques for points, polygons, and rasters

Chapter 6

Moving agents on representative networks

by Elizabeth R. Groff and Mary Jo Fraley

- ◆ Introduction to a basic vector movement model
- ◆ Representing the network
- ◆ Recognizing the representative network in Agent Analyst
- ◆ Using random number generators to simulate random movement
- ◆ Using the cartographic capabilities of the ArcMap interface to represent agent movement

The agents in previous chapters were not physically restricted in their movement across a landscape. The decision to move or not move to a location was driven by the attributes contained at the location. In this chapter, you will model agent movements that are restricted to a predefined network. Moving agents on a vector network is representative of many types of activities across a variety of applications.

A sample of applications include the following:

- Transportation modeling (e.g., buses, taxis, light rail, and cars all travel along a network)
- Medical modeling (e.g., blood flow through veins or electric pulses through nerves)
- Utilities (e.g., electricity and water systems)
- Disease modeling (e.g., the spread of a disease as based on a functional contact network)
- Wildlife modeling (e.g., corridors between habitat patches)

This model focuses on representing human activity on a network. You begin with the most basic representation of human activity—agents traveling on roads. All the roads in this simple model have the same speed limit.

The model presented here is a simple example. However, with additional programming the model can become more complex. One potential enhancement would be to incorporate different modes of transportation (e.g., driving, walking, biking, public transportation, or some combination). The mode of transportation would then influence the exact routes taken and the duration of the trip. Constraints on travel related to mode of transportation could then be incorporated so that they influenced the choices made by an agent. Taking complexity in a different direction, a model could also have bus agents that move along a route while human agents board and disembark at specified locations. To illustrate the basic concepts of moving agents on a network, this example focuses on moving agents representing humans along a street network.

Currently, Agent Analyst does not support connections with a geodatabase or with network data files. Thus, the method presented here relies on geoprocessing the streets and street intersection nodes in a GIS outside Agent Analyst (prior to running the model), and then importing them into the model at initialization. To enhance speed of processing, the method makes use of a representative network in which agents move from street intersection to street intersection rather than along actual streets. This approach also enables a point shapefile representing street intersections to be used for display of agent travel and decision outcomes. For example, the values of the nodes representing street intersections can be changed to reflect the number of times the place is visited or the number of robberies that occur there.

The intent of this chapter is to introduce how to model agents along a defined network, not how to create a crime model. The rules related to crime represent one theoretical view, and they can be changed to incorporate others.

The “vector network” used in this chapter is a representation of the connections between street segments participating in a street centerline layer. It is not a network in the sense of a network dataset or a geometric network.

The modeling scenario

In this model you will examine street robberies in Seattle, Washington. For a street robbery to occur three elements are necessary: a motivated offender, a suitable target (in this case another person), and the lack of other people around who might interfere with the crime (Cohen and Felson 1979). Since people in an urban environment typically travel along streets, our agents do so also. When motivated offenders end up at the same place and time with suitable targets, they evaluate the situation (i.e., how attractive the target is and the likelihood that other agents at the same place will interfere). If the potential risk outweighs the potential reward, no crime takes place; if it does not, a crime is committed.

Background information

The purpose of this model is to explore a popular theory in criminology that states that crime requires motivated offenders and suitable targets, without people present who might act as capable guardians (e.g., other people, a parking attendant, a police officer, and so on). (For complete documentation of the full robbery models, please see Groff, 2006, 2007a, 2007b). When these three elements converge in space/time, the motivated offender considers the relative risk of committing a street robbery. Routine activity theory (Cohen and Felson 1979) incorporates the effect of human activity patterns on crime by hypothesizing that if people spend more time away from home, the elements necessary for a crime to occur will converge more often and crime will increase. This model allows us to represent the activities of people (both citizens and police) in the model by having them travel from street intersection to street intersection along the streets in Seattle.

5

6

7

8

9

Introduction to a basic vector movement model

The exercises in this chapter provide you with the basics of moving agents on a vector network. You are using street centerline data here, but any set of connected line segments can be used. The exercises start with an introduction to a completed model that enables vector movement.

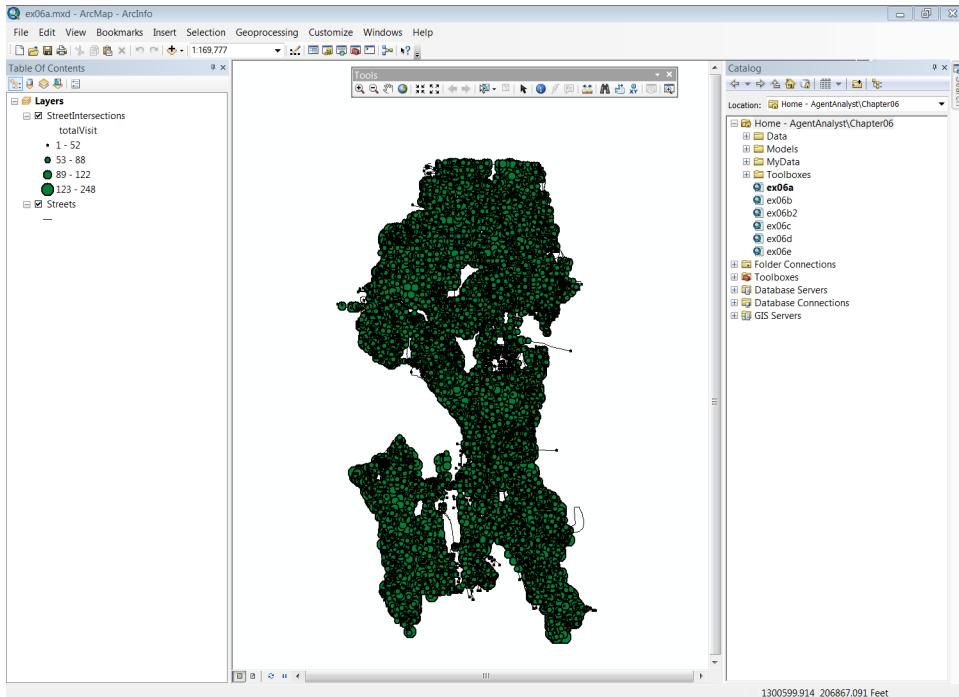
Exercise 6a

This exercise introduces you to the basics of a model that enables movement along a vector network. You will learn how to create this model in exercises 6b–e.

Load the ArcMap document

- 1 Start ArcMap. Navigate to C:\ESRIPress\AgentAnalyst\Chapter06. Open ex06a.mxd.

The map document opens. In the ArcMap display you will see the Seattle Streets (lines) shapefile and a StreetIntersections shapefile (which uses points). In ArcToolbox, the Chapter06Ex06a toolbox has been created for you.



The StreetIntersections shapefile contains the points that represent street intersections in the model. Agents move from intersection to intersection rather than along street segments. Each time an agent moves, it determines its location randomly by selecting a node

from a list of street intersections that are connected to its current intersection. This design reduces processing time because it does not involve spatial analysis.

StreetIntersections is also the shapefile to which the model writes output information related to agent movement among places. You'll now explore the street intersection shapefile.

- 2** Right-click the StreetIntersections layer, and then click Open Attribute Table. Note the fields in the attribute table, in particular totalVisit.

A cumulative total of citizen agent visits to each node is kept during the model run and written out periodically to the totalVisit field in the shapefile. The values in the totalVisit field are from the previous model run. They are zeroed out when the model is run again. Zoom in on a section of Seattle. Notice how some streets have many more visits than others. Those streets are most likely the arterial roads or highways.

Open the Agent Analyst model to move agents along the street network

- 3** Right-click the Chapter06Ex06a toolbox, point to New, and select Agent Analyst Tool.
- 4** Once the Agent Analyst window appears, click File and select Import. Do not save the changes when prompted.
- 5** Navigate to C:\ESRIPress\AgentAnalyst\Chapter06\Models and open ex06a.sbp. The Agent Analyst window appears.
- 6** Run the model by clicking the Run button. The Repast toolbar appears with the Simple Vector Movement Settings window.

Since this is a stripped-down version of a robbery model, the only functionality implemented is movement.

- 7** Click the Start button on the Repast toolbar.

The graduated symbol classification scheme in ArcMap was created using data from a day's worth of agent movement. The classification scheme persists and can be used to examine the results of agent movement in subsequent runs. Observe how the graduated symbols representing agent movement begin to appear and grow larger. You should see the map refresh after each 100 ticks of the model.

- 8** Close Agent Analyst by clicking File and Exit. Do not save any changes when prompted.

5

6

7

8

9

Representing the network

This exercise is divided into two sections. The first (exercise 6b-1) describes how to take a street centerline and create the representative network of nodes (i.e., street intersections) using a geo-processing model to automate the steps. Using a list of nodes is a very computationally efficient method for moving agents along a street network. The second section (exercise 6b-2) walks you through how to identify the list of neighboring nodes for each node by using a Python script. These steps build the data foundation for implementing movement on a street network. When the steps are complete you will have a set of nodes representing the intersections in Seattle and a file that contains the list of neighbors for each node.

Exercise 6b-1

Deriving a representative network from a street file

Since Agent Analyst currently does not support network datasets from ArcGIS, you need a way to simulate the movement of agents on a network. You can accomplish this by using street intersections (which you create in this exercise) and a file containing the list of neighbors for each node (created in exercise 6b-2). To create these files, you start with a street file.

In this exercise you will take a network of streets and create a shapefile of nodes (representing street intersections) that can be read in Agent Analyst as vector agents. You will use ArcGIS ModelBuilder to establish the network relationships.

Note: This exercise requires the ArcInfo license of ArcGIS. If you do not have a license, read through the exercise and proceed to exercise 6b-2.

Load the ArcMap document

- 1 Start ArcMap. Navigate to C:\ESRIPress\AgentAnalyst\Chapter06 and open ex06b.mxd.

In the ArcMap display you will see the Seattle Streets (lines) shapefile. In ArcToolbox, the Chapter06Ex06b1 toolbox has been created for you.

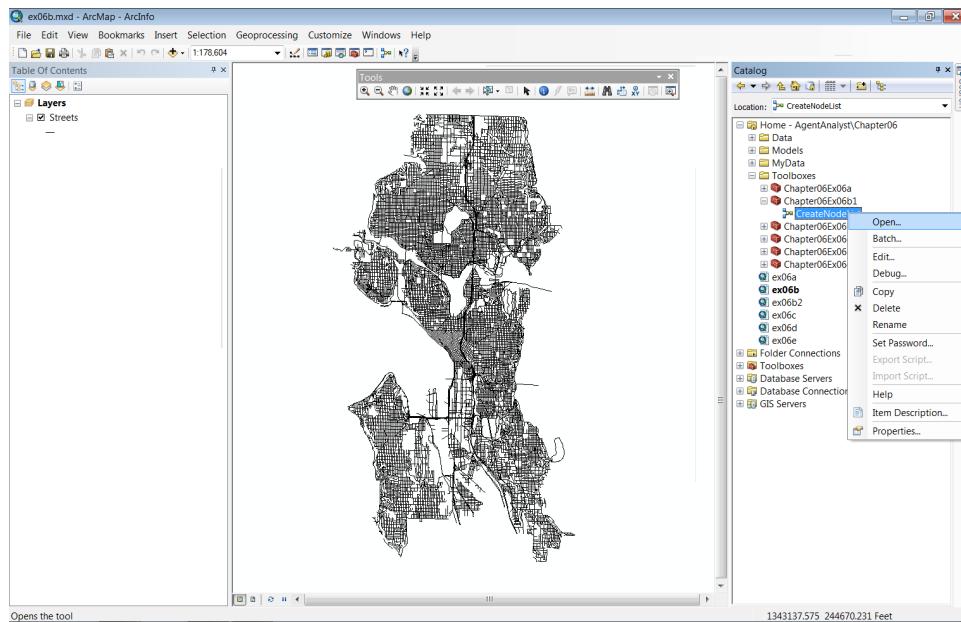
The toolbox contains the CreateNodeList ModelBuilder model to convert the Streets shapefile into a shapefile of nodes. The model will automatically create nodes wherever two or more lines meet.

Geoprocessing models

Geoprocessing models allow complicated work flows to be automated. Put simply, they allow the chaining together of tools, using the output of one tool as the input for another. The use of models will ensure that the same steps are executed in the same order without human intervention. For more information about geoprocessing models, please see the ArcGIS Desktop Help.

Run the model

- 2 Expand the Chapter06Ex06b1toolbox if it is not already expanded. Right-click the CreateNodeList tool, and then click Open.



Note: If the Tool Not Licensed message box appears, it indicates you are not licensed for the ArcInfo version of ArcGIS. Read through the exercise steps and then continue with exercise 6b-2.

You will see that the geoprocessing model takes two parameters:

- Input feature class
- Final Output with Unique field that will contain the street nodes, which are the street intersections

5

6

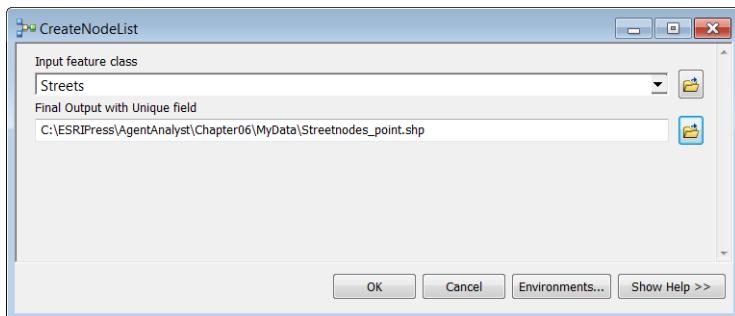
7

8

9

Fill in the parameters and run the model

- 3 Fill in the parameters to match those in the following image. In the Input feature class parameter, click the drop-down list to the right of the parameter and select the Streets layer if it is not already selected. For the name of Final Output with Unique field, store the resulting shapefile in the C:\ESRIPress\AgentAnalyst\Chapter06\MyData directory (which should be the default).

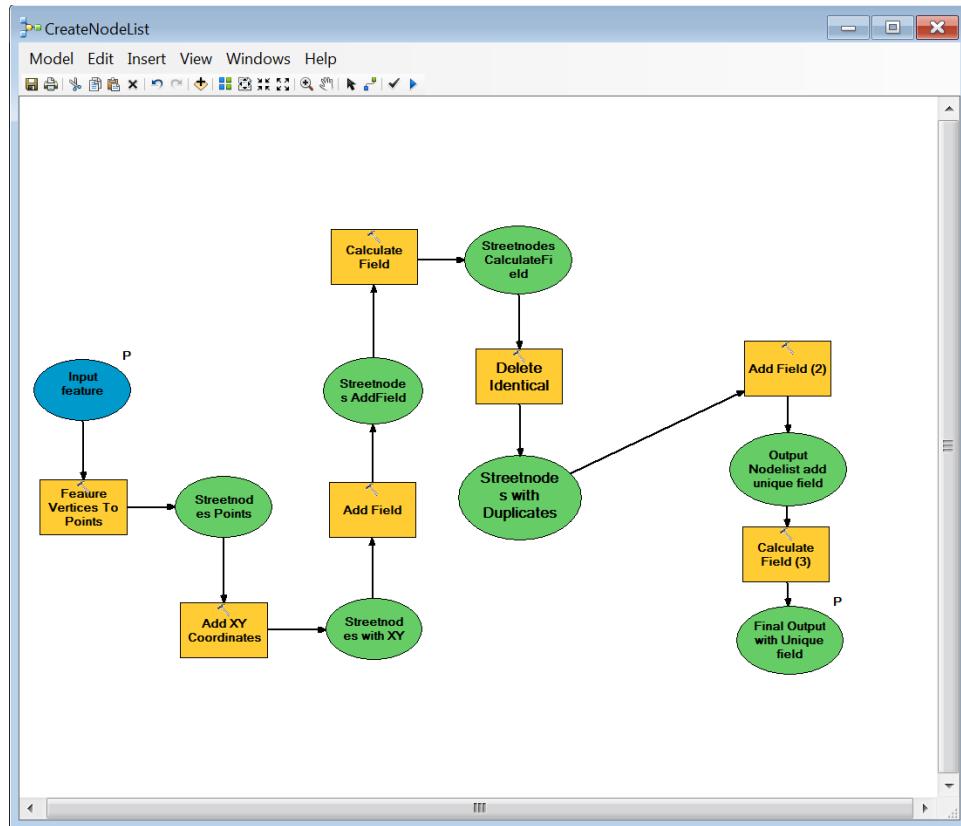


The CreateNodeList ModelBuilder model converts the Streets shapefile into a shapefile of nodes.

Model parameters

Parameters are the variables in a model for which you supply the values and thereby affect the outcome of the model. Previous chapters have referred to parameters in Not Quite Python (NQPY) code, but you also have parameters for geoprocessing models.

- 4 Click OK to create the feature class containing the node list.
- 5 It will be valuable to understand what the geoprocessing model does, so the next section explains the steps in the model. To view the model in this mode, right-click the CreateNodeList model and click Edit.



The geoprocessing model takes the Streets feature class that you designated as the input and creates a point output feature class (Streetnodes Points), which contains the vertices of the streets. Next it adds the x,y coordinates as fields to the output point feature class (Streetnodes with XY). Then the model adds a new field, called Comp_Dup (Streetnodes AddField), and calculates this field to be the concatenation of the x and y fields (added in the previous tool) (Streetnodes CalculateField). Next we use Delete Identical, which deletes the duplicate records for the field Comp_Dup (Streetnodes with Duplicates). Lastly, the model creates a field (Output Nodelist add unique field) that contains a unique ID for each node (Final Output with Unique field).

5

6

7

8

9

Exercise 6b-2

Creating a list of neighboring nodes for routing

The next step in implementing random movement on a network uses the node representation of a network that you created in exercise 6b-1 to define the set of nodes that can be reached from each node. The goal is to provide each agent with a set of alternatives that represents possible moves to adjacent streets from the agent's current position. The agent chooses its next move destination from this set of nodes. To create the set of adjacent nodes, you have to identify which street intersections are connected to each other (i.e., which are neighbors). Once you know this information, it is possible to have agents move from their current node to a randomly selected adjacent node. To assemble a list of neighbor nodes you need to create a comma-delimited (.csv) text file that contains a list of the adjacent nodes for each location. The following example uses a Python script to identify adjacent nodes. The script requires several parameters that are explained in the upcoming exercise steps.

This particular step can take a long time to process for an entire city's street feature class. To save time, an existing nodeneighbors.csv file is located in C:\ESRIPress\AgentAnalyst\Chapter06\Data. If you decide to use nodeneighbors.csv, read through the next steps and then go to exercise 6c.

Load the ArcMap document

- 1 Start ArcMap. Navigate to C:\ESRIPress\AgentAnalyst\Chapter06. Open ex06b2.mxd.

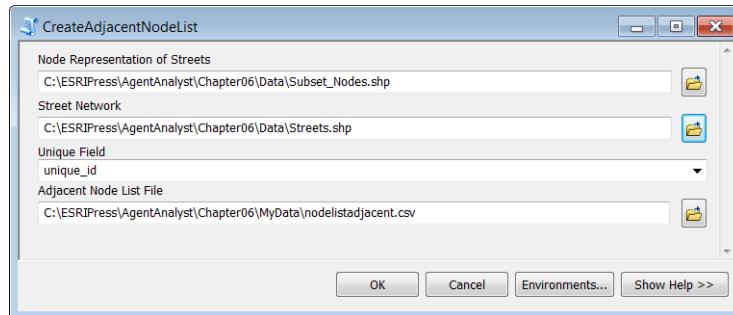
In the ArcMap display you will see Subset_Nodes and Streets displayed. In ArcToolbox, the Chapter06Ex06b2 toolbox has been created for you. In the toolbox you will find the CreateAdjacentNodeList Python script.

Run the CreateAdjacentNodeList Python script to create a text file containing each node and a list of adjacent nodes

The CreateAdjacentNodeList Python script tool will create a comma-delimited text file containing each node and a list of adjacent nodes.

2 Expand the Chapter06Ex06b2 toolbox, and then double-click the CreateAdjacentNodeList Python script to open the CreateAdjacentNodeList dialog box. Populate the following parameters for the model as shown in the following figure:

- Node Representation of Streets—Fill in with the node feature class called Subset_Nodes.shp, which is located in the C:\ESRIPress\AgentAnalyst\Chapter06\Data folder.
- Street Network—Type in or browse to the location of the original street network, Streets, from exercise 6b-1.
- Unique Field—This is the same unique field created in exercise 6b-1, unique_id.
- Adjacent Node List File—This is the output text file that contains the adjacent nodes list for each node in the node representation of the Streets feature class. This file will be used by Agent Analyst to randomly select the agent's movement. Name this file nodelistadjacent.csv and save it to the MyData folder for Chapter 6.

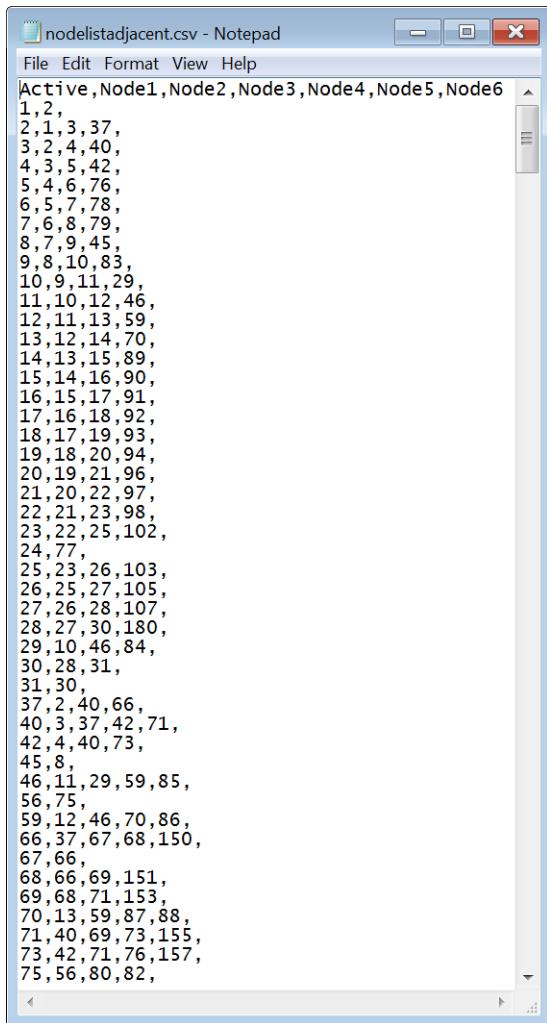


3 Click OK to run the script.

The script will look at each node in the node feature class, locate all the nodes that are adjacent to that particular node, and write the information to a text file. This information allows you to create a list of adjacent nodes for network movement. The output text file can be viewed in Notepad. The first field, labeled Active, contains the node ID for the active node (the one for which adjacency is being identified). Subsequent fields (e.g., Node1, Node2, and so on to Node6) contain the node IDs for adjacent nodes.

5
6
7
8
9

The output of the script will look like that shown in the following figure:



A screenshot of a Windows Notepad window titled "nodelistadjacent.csv - Notepad". The window contains a list of node IDs separated by commas. The list starts with "Active,Node1,Node2,Node3,Node4,Node5,Node6" and continues with a series of node numbers from 1 to 153. The nodes are listed in pairs or groups, indicating adjacent nodes. For example, node 1 is adjacent to nodes 2, 3, and 37; node 2 is adjacent to nodes 1, 3, and 37; and so on. The list ends with node 75 being adjacent to nodes 56, 80, and 82.

```
Active,Node1,Node2,Node3,Node4,Node5,Node6
1,2,
2,1,3,37,
3,2,4,40,
4,3,5,42,
5,4,6,76,
6,5,7,78,
7,6,8,79,
8,7,9,45,
9,8,10,83,
10,9,11,29,
11,10,12,46,
12,11,13,59,
13,12,14,70,
14,13,15,89,
15,14,16,90,
16,15,17,91,
17,16,18,92,
18,17,19,93,
19,18,20,94,
20,19,21,96,
21,20,22,97,
22,21,23,98,
23,22,25,102,
24,77,
25,23,26,103,
26,25,27,105,
27,26,28,107,
28,27,30,180,
29,10,46,84,
30,28,31,
31,30,
37,2,40,66,
40,3,37,42,71,
42,4,40,73,
45,8,
46,11,29,59,85,
56,75,
59,12,46,70,86,
66,37,67,68,150,
67,66,
68,66,69,151,
69,68,71,153,
70,13,59,87,88,
71,40,69,73,155,
73,42,71,76,157,
75,56,80,82,
```

The nodelistadjacent.csv file stores all the nodes that are adjacent to specific nodes. Your output file will be smaller than the preprocessed .csv file that comes with the chapter. You ran the script on a subset of the data. If you were not able to complete exercise 6b-1, the script may not run correctly.

You have now created the foundation for using a representative network in Agent Analyst.

Recognizing the representative network in Agent Analyst

Now that you have appropriate data to describe your road network and a neighbor matrix so you know which nodes are neighbors, you are ready to start building an Agent Analyst model. Since you have already learned in previous chapters how to create vector agents and generic agents, you will start with a model that already has a few agents defined for you. In this exercise you will go through the steps of implementing the movement within the model.

5

6

7

8

9

Exercise 6c

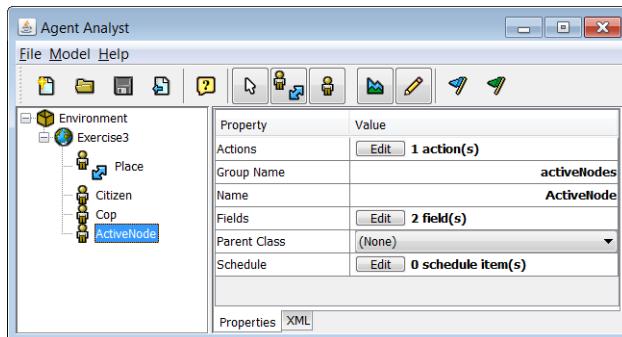
Load the ArcMap document

- 1 Start ArcMap. Navigate to C:\ESRIPress\AgentAnalyst\Chapter06 and open ex06c.mxd.

In the ArcMap display you will see the Streets layer displayed. In ArcToolbox, the Chapter06Ex06c toolbox has been created for you.

Open the Agent Analyst model

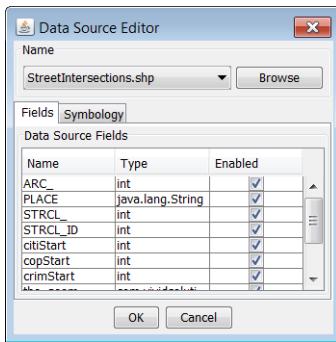
- 2 Right-click the Chapter06Ex06c toolbox, point to New, and select Agent Analyst Tool.
- 3 Once the Agent Analyst window appears, click File and select Import. Do not save the changes when prompted.
- 4 Navigate to C:\ESRIPress\AgentAnalyst\Chapter06\Models and open ex06c.sbp. The Agent Analyst window appears.



Create the network framework

Set the data source for the place agent to the StreetIntersections shapefile (which is similar to the one you created in exercise 6b-1).

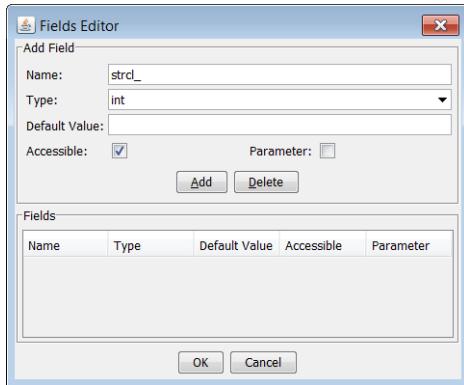
- 5 In the Environment panel, click Place. In the Property panel, click the Edit button to the right of the Data Source property. In the Data Source Editor, click Browse, navigate to C:\ESRIPress\AgentAnalyst\Chapter06\MyData\StreetIntersectionsEx2_2a.shp, and then click Open. Once you add the shapefile, you can see the fields in the editor.



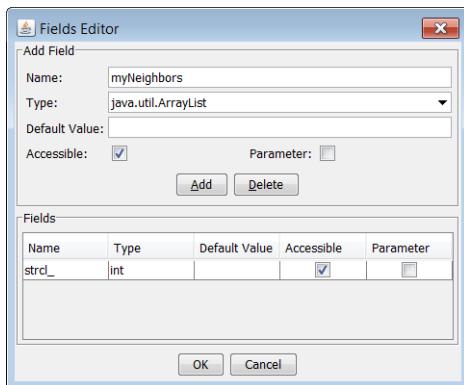
- 6 Click OK to close the Data Source Editor.

After you add the shapefile, you need to add two fields in Agent Analyst to the place vector agent.

- 7 In the Environment panel, click Place. In the Property panel, click the Edit button to the right of the Fields property to open the Fields Editor.
- 8 In the Fields Editor, add a field to hold the unique ID. Type **strcl_** in the Name box, click int in the Type list, and select the Accessible check box. Click Add.



- 9 Add another field to hold the contents of the text file that contains the adjacent node list. Name the field **myNeighbors**, and since a text file is used for the adjacent node list, type **java.util.ArrayList** as the Type. Select the Accessible check box, and then click Add.



Define the place agents and establish their context in the network

Next you need to create an action through which Agent Analyst recognizes the Street Nodes feature class as a vector agent class called Place. This is done at the model level (i.e., Exercise3) in the code shown in step 11.

- 10 In the Environment panel, click Exercise3. In the Property panel, click the Edit button to the right of the Actions property.

5

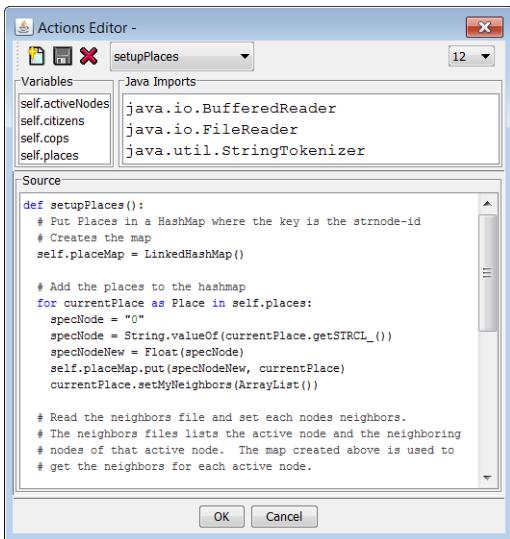
6

7

8

9

- 11** In the Actions Editor, select setupPlaces from the Actions drop-down list. In the Source panel, uncomment just the code shown in the following figure (more code will be uncommented later). This action associates the Street Nodes feature class to place agents.



Three modules need to be imported to support certain commands in the model code. In the Java Imports box, the modules were added so that you can call the needed commands for reading files:

- `java.io.BufferedReader`
- `java.io.FileReader`
- `java.util.StringTokenizer`

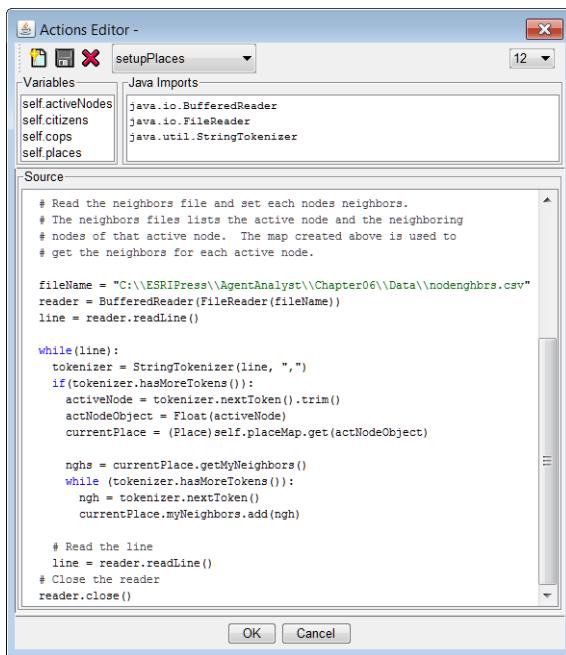
The `BufferedReader` module allows you to read from a character input stream. The `FileReader` module allows you to open a file for reading. The `StringTokenizer` module allows a string to be broken into tokens.

The preceding code first creates an empty hash map (`self.placeMap = LinkedHashMap()`). A hash map is a way of associating identifiers to values. Creating a hash map allows you to identify all the nodes (representing Places) by names. Once the hash map is created, it is populated with the places or nodes that the agents can move to with a unique identifier, which is a field from the shapefile that was used to create the places (`self.placeMap.put(specNodeNew, currentPlace)`). The variable `specNode` is set to the unique identifier for the current place (`specNode = String.valueOf(currentPlace.getSTRCL_())`) and then put into the hash map as the identifier for each place. The values for each identifier (i.e., consisting of

the adjacent nodes for each place) will be held in an ArrayList (`currentPlace.setMyNeighbors(ArrayList())`).

Now that you have a class of vector agents (place) that contains the street nodes, you need to add the neighbor node information.

- 12** Uncomment the code in the Source panel at the end of the `setupPlaces` action so that it looks like the code in the following image. Change the path in the `fileName =` line to `C:\ESRIPress\AgentAnalyst\Chapter06\Data\nodengbrs.csv`, as shown in the figure. This section of code will associate the neighboring node information for each place agent. Click OK to save your work.



This code does the following: The first line allows you to access the file containing the adjacent node list that you created in exercise 6b-2. This node list contains all the nodes adjacent to a specific node. The next line creates a buffered reader (`reader = BufferedReader(FileReader(fileName))`) to allow the reading of each line of the file (`line = reader.readLine()`). A while loop is used to read each line of the file. The string tokenizer (`tokenizer = StringTokenizer(line, ",")`) allows each line to be parsed to get the nodes that are adjacent to an active node. At this point, each place or active node now has a list of neighbor nodes inside Agent Analyst (`currentPlace.myNeighbors.add(ngh)`). Once the process has completed reading in the file and processing, the buffered reader is closed (`reader.close()`).

At this point, the place vector agents in the model know where they are and who their neighbors are.

5

6

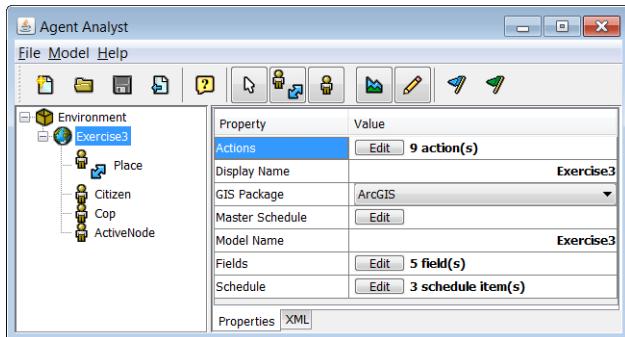
7

8

9

You are almost ready to simulate movement, but you still need to create the agents that will be moving and specify a methodology for deciding to which of their adjacent nodes they will move.

In this model you are interested in two types of moving agents, citizens and cops. Both types of agents will have simple random movements in this model.



There are two generic agent classes that have already been added to the model. One class is citizen and the other is cop. (See chapters 2 and 3 for how to add generic agents.)

You need to add five fields at the model level as specified in table 6-1. These fields set some model-level variables that can be used by any of the classes.

- 13** In the Environment panel, click Exercise3. In the Property panel, click the Edit button to the right of the Fields property to open the Fields Editor.
- 14** In the Fields Editor, add the following fields, and then click OK.

Table 6.1 Fields Editor data

| Name | Type | Default value | Accessible |
|------------|-------------------|---------------|------------|
| placeMap | java.util.HashMap | | Yes |
| modelStep | int | | Yes |
| MODEL_DAY | int | | Yes |
| NUM_PLACES | int | 16035 | Yes |
| COPS | int | 200 | Yes |

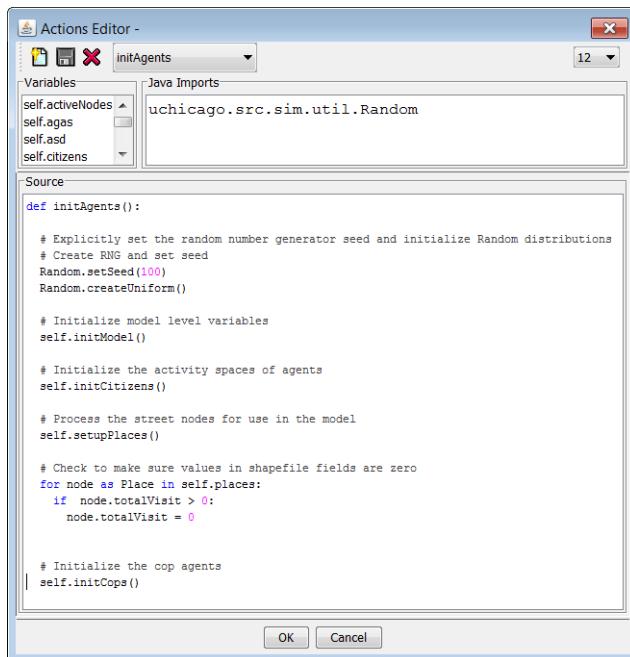
Note: The java.util.HashMap type is not a selection in the drop-down list and must be typed into the box.

The placeMap field holds the contents of the hash map (the information about street intersections). The modelStep field holds the incremented value representing the number

of model steps that have occurred. The MODEL_DAY field holds the number of ticks in one day ($n = 1,440$). The NUM_PLACES field contains the total number of street intersections in Seattle. Finally, the COPS field contains the number of cops in the model.

There are a few more actions that you need to develop to make the model work. Three actions are automatically added to each new Agent Analyst model when it is created: initAgents, updateDisplay, and writeAgents. The initAgents action handles the setup of the model.

- 15** In the Environment panel, click Exercise3. In the Property panel, click the Edit button to the right of the Actions property to open the Actions Editor. Select initAgents from the Actions drop-down list. In the Java Imports panel, the following class has been added: uchicago.src.sim.util.Random.
- 16** Uncomment the code in the Source panel so that it looks like the code in the following image. The action initializes the parameters for the model.



These commands do a series of actions. First, the Random function creates a uniform random number distribution for use in the model (`Random.createUniform()`). Using the `setSeed` function (`Random.setSeed(100)`) allows you to control the starting “seed” number for the distribution. Each time the same seed is used, the same sequence of numbers is generated. The ability to generate the same random number distribution is essential to running multiple experiments in which only one aspect of the model is changed and everything else is held constant. If you could not generate exact duplicates of random number distributions, you could not do these types of experiments.

5

6

7

8

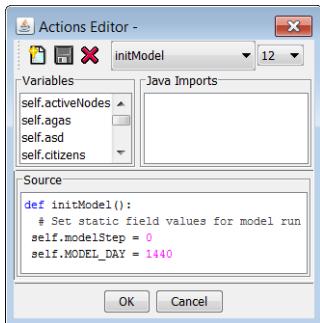
9

The action then calls other actions to initialize the model (`self.initModel()`), the citizen agents (`self.initCitizens()`), the cop agents (`self.initCops()`), and the places (`self.setupPlaces()`). Finally, the action checks to make sure the values in the `totalVisit` field are zero at the start of the run (`if node.totalVisit > 0 : node.totalVisit = 0`). This allows you to keep model results from one run separate from subsequent runs.

The `totalVisit` field in the `StreetIntersections2_2a` shapefile needs to be included or the model will not compile. This field is already added to the shapefile and will be used to hold data that the model will generate.

Open the `initModel` action that you call in the previous code block

- 17 In the Actions Editor, select `initModel` from the Actions drop-down list. Uncomment the code in the Source panel so that it looks like the code in the following image. The action initializes some model fields before the model is run. Click OK to save your work.



This action sets the initial values for fields used in the model.

At this point you have created the framework for movement.

- 18 Compile the model by clicking the Compile button. Compiling the model will allow you to check for any errors before running it. You may receive some errors that you will correct by adding additional code in the next exercise, which is focused on using the random number generator.
- 19 Close Agent Analyst by clicking File and then Exit. Do not save any changes when prompted.

Using random number generators to simulate random movement

In this model, you want the agents to move at every time step of the model. You will identify the best place (i.e., in which action) to put the code you are writing. In this case, the step action is the most appropriate because the step action is called during each tick of the model. This section explains how to move the citizen and cop agents from node to node.

5

6

7

8

9

Exercise 6d

Load the ArcMap document

- 1 Start ArcMap. Navigate to C:\ESRIPress\AgentAnalyst\Chapter06. Open ex06d.mxd.

In the ArcMap display you will see Streets displayed. In ArcToolbox, the Chapter06Ex06d toolbox has been created for you.

Open the Agent Analyst model

- 2 Right-click the Chapter06Ex06d toolbox, point to New, and select Agent Analyst Tool.
- 3 Once the Agent Analyst window appears, click File and select Import. Do not save the changes when prompted.
- 4 Navigate to C:\ESRIPress\AgentAnalyst\Chapter06\Models and open ex06d.sbp. The Agent Analyst window appears.

Add two fields that will be used in moving the cop agents

- 5 In the Environment panel, click Cop. In the Property panel, click the Edit button to the right of the Fields property to open the Fields Editor. Add the fields shown in table 6.2.

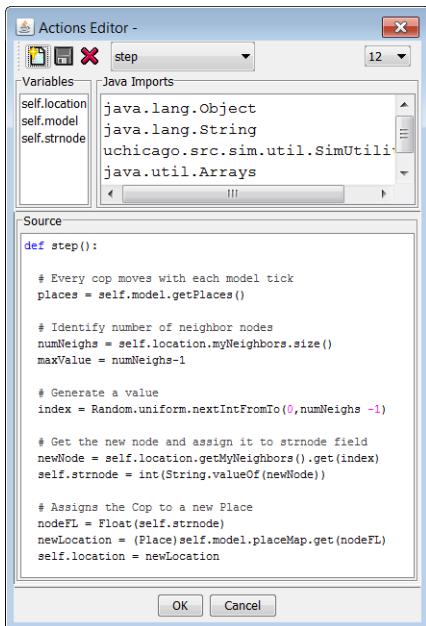
Table 6.2 Fields Editor data

| Name | Type | Default value | Accessible |
|----------|-------|---------------|------------|
| strnode | int | | Yes |
| location | Place | | Yes |

The strnode field is the unique ID of a node, and the location field is its current location or node. For both fields, select the Accessible check box.

Agent Analyst code to move the cop agent from node to node is written in the Actions Editor. This involves identifying the place where a cop agent is located, randomly choosing a neighbor node, and then moving the cop to the new node.

- 6 In the Environment panel, click Cop. In the Property panel, click the Edit button to the right of the Actions property to open the Actions Editor. There is only one action for the cop agent, step.
- 7 In the Source panel, uncomment the step action code so that it looks like the code in the following image. The action randomly moves the cop agent to one of its neighboring nodes.



In this code you are setting up the ability to move the agents randomly. First, you set a variable, which is the group name Places (`places = self.model.getPlaces()`). Then the code finds the total number of neighbors for each place agent. Since each neighboring node has an index, the index can be used to randomly select a position between 0 and the total number of nodes for the target street node. In this example, a uniform random distribution is used to ensure that each node has an equal chance of being selected (`index = Random.uniform.nextIntFromTo(0, numNeighs - 1)`). The random index variable is used to select the adjacent node from the `myNeighbors` list. The final step is to assign the `strnode` field (`self.strnode = int(String.valueOf(newNode))`).

However, if you stop here, the cop agent will know where to go but will not be moved to the new place. To “move” the cop agent, you need to associate it with the new street node, which is in the place agent (see the section of code below the comment line “Assigns the Cop to a new Place”).

- 8 Click OK.
- 9 As you did for the cop agent class, define the following fields for the citizen agent class. In the Environment panel, click Citizen. In the Property panel, click the Edit button to the right of the Fields property to open the Fields Editor. Add the same two fields related to movement as you did for the cop agent class plus one additional field. The fields’ definitions are shown in table 6.3.

5

6

7

8

9

Table 6.3 Fields Editor data

| Name | Type | Default value | Accessible |
|----------|------------------|---------------|------------|
| strnode | int | | Yes |
| location | Place | | Yes |
| name | java.lang.String | | Yes |

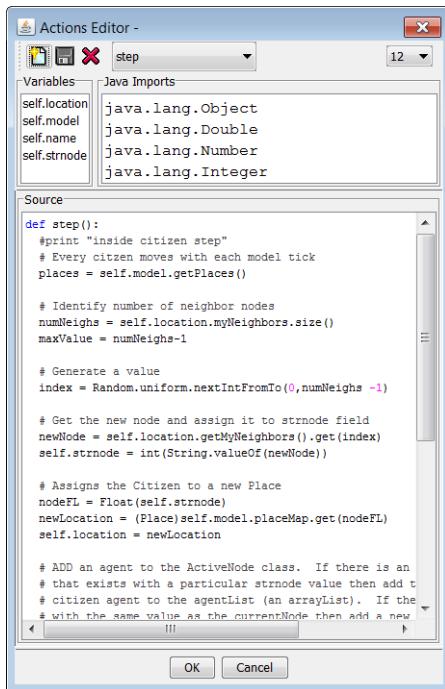
The strnode field is the unique ID, location is the current location or node, and name is needed for citizen agents because you want to be able to track their attributes.

Develop the step action for the citizen agents

- 10 In the Environment panel, click Citizen. In the Property panel, click the Edit button to the right of the Actions property to open the Actions Editor.

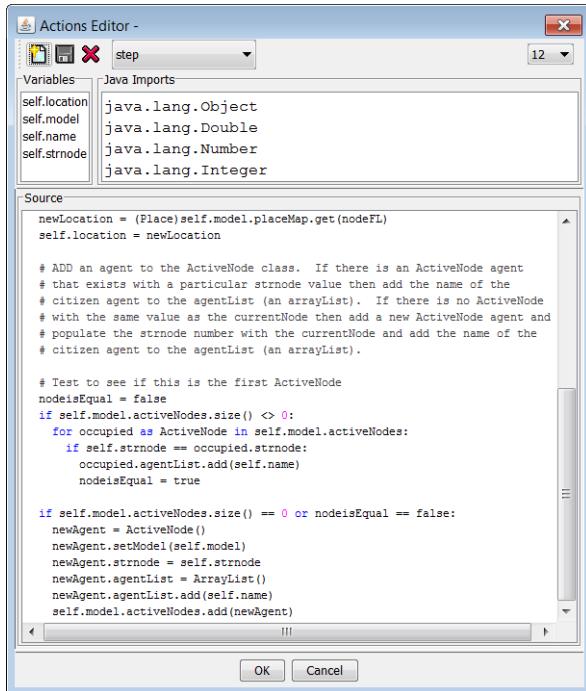
There is only one action, step, for the citizen agent, so it is already selected.

- 11** In the Source panel, uncomment the code so that it looks like the code in the following image.



If you compare the step action for cops with the step action for citizens, you will notice some additional code in the citizen step action. The additional code is needed to assign the presence of a citizen on a node via the ActiveNode agent.

- 12** Uncomment the additional code for the step action as shown in the following figure. Remember to click OK to save your changes.



The portion of code shown in the figure allows you to keep track of which agents are present on a node at each tick and to count the total number of agents to visit each node. If the activeNodes array has some values, you add the current citizen agent to the list of agents at that particular node (`if self.model.activeNodes.size() <> 0:`). Otherwise, if the activeNodes array is currently empty, you want to add the current agent to a new agent list (`if self.model.activeNodes.size() == 0 or nodeisEqual == false:`).

Two remaining fields are needed to track the number of times a specific node is visited by citizen agents.

Add fields to the ActiveNode

- 13** In the Environment panel, click ActiveNode. In the Property panel, click the Edit button to the right of the Fields property. In the Fields Editor add the fields shown in table 6.4. The `java.util.ArrayList` is not a selection in the Type drop-down list and must be typed into the box.

5

6

7

8

9

Table 6.4 Fields Editor data

| Name | Type | Default value | Accessible |
|-----------|---------------------|---------------|------------|
| strnode | int | | Yes |
| agentList | java.util.ArrayList | | Yes |

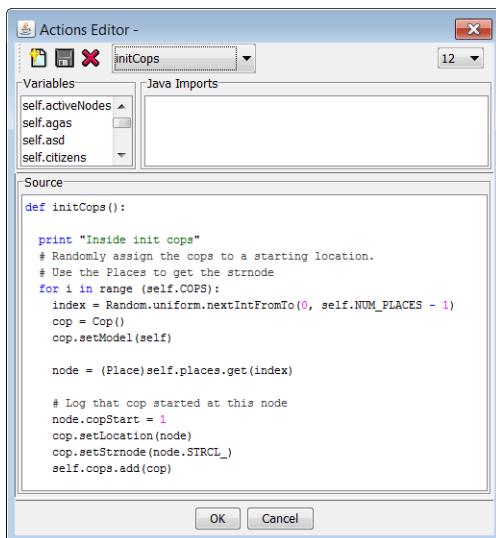
The strnode field is the unique ID, and agentList is the list of agents at a particular node.

- 14** Click OK.

You need to create actions for the citizen and cop agents to initialize the variables for each at the model level.

Add an action to initialize the cop agents

- 15** In the Environment panel, click Exercise3. In the Property panel, click the Edit button to the right of the Actions property to open the Actions Editor.
- 16** In the Actions list, select initCops. Uncomment the code in the Source panel so that it looks like the code in the following image. The code creates cop agents and assigns them to starting nodes/places.

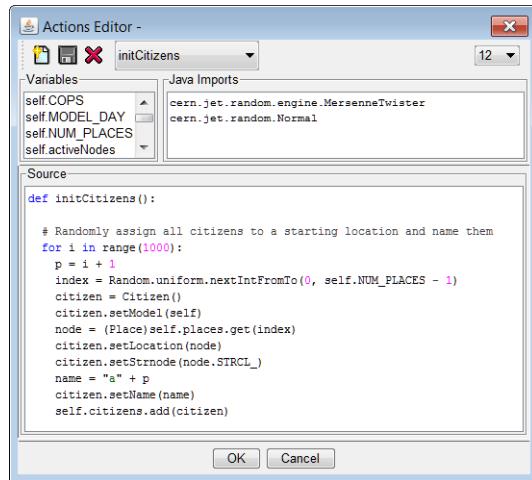


The preceding code, designed to initialize the cop agents, does the following. The starting location of each cop is randomly assigned by using a random number generator. The starting location for each cop is also captured.

Initialize the citizen agents

You next need to add code to create citizen agents, assign them unique names, and allocate starting nodes/places.

- 17** Select initCitizens in the Actions drop-down list. In the Source panel, uncomment the code so that it looks like the code in the following image.



The action randomly assigns each citizen agent to a node as its starting location for the model run.

- 18** Click OK to close the Actions Editor.
- 19** Close Agent Analyst by clicking File and then Exit. Do not save any changes when prompted.

In the next exercise, you will simulate movement of the citizen and cop agents.

5

6

7

8

9

Using the cartographic capabilities of the ArcMap interface to represent agent movement

In this exercise, you learn how to write data generated in the model to the shapefile and then display it using ArcMap's cartographic capabilities. You can do this while the model is running or at the end of the model (see chapter 9). There is a performance trade-off that occurs between the frequency of write and display operations and the speed of the model run. As the frequency of write and display operations increases, the speed of the model run decreases. If you are interested in running large models, a good strategy is to use the ArcMap display while creating the model but turn it off for the final model runs.

One way to display the results of agent movement in ArcMap is to collect information about how many agents visit each node. In earlier exercises in this chapter, you learned about a field called totalVisit that's included the shapefile you will use for display. To display the total visits agents make to nodes, you need to make sure the shapefile to which you are going to write the data is added to the ArcMap session.

Exercise 6e

Load the ArcMap document

- 1 Start ArcMap. Navigate to C:\ESRIPress\AgentAnalyst\Chapter06. Open ex06e.mxd.

In the ArcMap display you will see the Streets layer displayed. In ArcToolbox, the Chapter06Ex06e toolbox has been created for you.

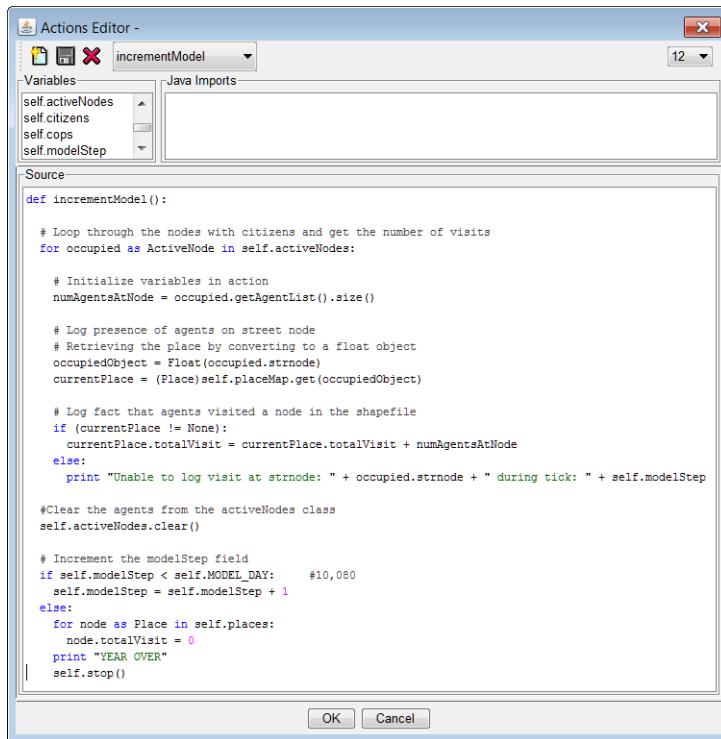
Open the Agent Analyst model

- 2 Right-click the Chapter06Ex06e toolbox, point to New, and select Agent Analyst Tool.
- 3 Once the Agent Analyst window appears, click File and select Import. Do not save the changes when prompted.
- 4 Navigate to C:\ESRIPress\AgentAnalyst\Chapter06\Models and open ex06e.sbp. The Agent Analyst window appears.

Add the final action to gather information about cumulative data to describe what occurred in the model run

- 5 In the Environment panel, click Exercise3. In the Property panel, click the Edit button to the right of the Actions property to open the Actions Editor.

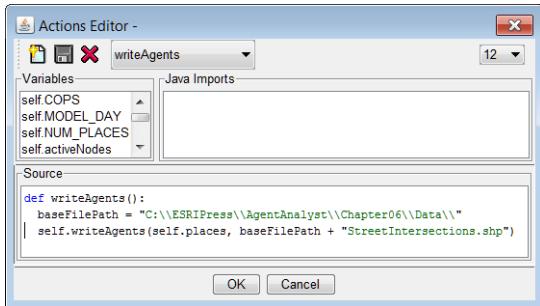
- 6** From the Actions drop-down list, select incrementModel. This is the last action to run, so it is the place to gather cumulative information. In the Source panel, uncomment the code lines by removing all the comment markers that fall in the first column, so that it looks like the code in the following image. The action finds all the nodes (places) in the model with at least one agent. Click OK to save your changes.



You need code to assign values to the attributes of the place agents and to populate a field in a vector agent feature class. The preceding code checks each of the nodes to find out if there is an agent present at the node. Every street node (place agent) is evaluated, selecting only those nodes that have one or more agents present (`currentPlace != None`). To take into account the number of agents that have visited that node since the model run started, you calculate a field in the feature class via the place agent (`currentPlace.totalVisit = currentPlace.totalVisit + numAgentsAtNode`). This adds the number of agents at the node in this model step (`numAgentsAtNode`) to the cumulative total that have visited the node (`currentPlace.totalVisit`).

Now you need to write the values to the shapefile and then tell ArcMap to refresh the display. To write the new values to the `totalVisit` field in the shapefile attribute table you will use the existing `writeAgents` action.

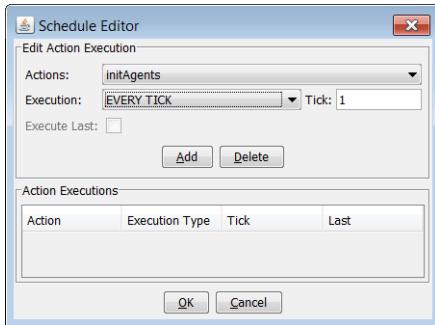
- 7 In the Environment panel, click Exercise3. In the Property panel, click the Edit button to the right of the Actions property to open the Actions Editor.
- 8 In the Actions Editor, select writeAgents from the Actions drop-down list, and then uncomment the two lines of code in the Source panel so that it looks like the code in the following image.



- 9 Click OK to save your changes and close the Actions Editor.

Schedule the actions

- 10 In the Environment panel, click Exercise3. In the Property panel, click the Edit button to the right of the Schedule property.

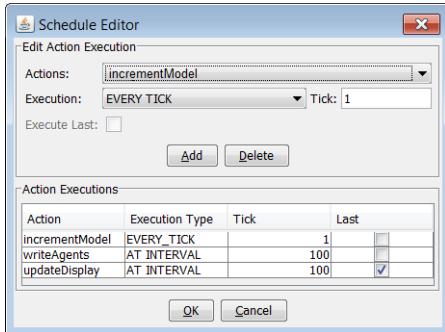


- 11 In the Schedule Editor, open the Actions list and select writeAgents. From the Execution list, select AT INTERVAL so that the display is updated only at a user-specified interval. In the Tick parameter, type **100** to set an interval of 100 ticks. Click the Add button.

Schedule two more actions, including the final action

- 12 From the Actions list, select updateDisplay. Set Execution to AT INTERVAL and type **100** in Ticks. Select the Execute Last check box. Click Add. The ArcMap display will refresh every 100 ticks.

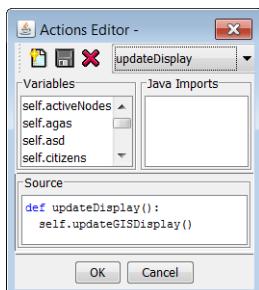
- 13** From the Actions drop-down list, select incrementModel. This action runs every tick, so select EVERY_TICK as the execution type and type **1** in the Tick box. Clear the check box for Execute Last. Click the Add button. You will see the three actions you scheduled in the Action Executions list as shown in the following figure.



- 14** Click OK to save your changes and close the Schedule Editor.

At this point, the updateDisplay action will run at 100 tick intervals. To display the changes in values in ArcMap as you run the model, you need to instruct ArcMap which shapefile to update. In this case, it is StreetIntersectionsEX2_2a.shp.

- 15** In the Environment panel, click Exercise3. In the Property panel, click the Edit button to the right of the Actions property to open the Actions Editor.
- 16** In the Actions Editor, select updateDisplay from the Actions drop-down list. In the Source panel, uncomment the line of code so that it looks like the code in the following image. Click OK.



The updateGISDisplay action is provided by Agent Analyst for updating the map in ArcMap for symbolization.

5

6

7

8

9

Add StreetIntersections to ArcMap

To visualize the updating of the shapefile, the layer must be displayed in ArcMap and it needs to be symbolized appropriately.

- 17 Click the Add Data button on the ArcMap toolbar and navigate to C:\ESRIPress\AgentAnalyst\Chapter06\Data. Select StreetIntersections.shp, and then click Add.
- 18 Right-click the StreetIntersections layer in the ArcMap Table Of Contents, select Properties..., and then select the Symbology tab. In the Show pane, select Quantities and then Graduated Symbols. For Value, select the totalVisit field (if you receive a warning dialog box, click OK). If necessary, type **2** in the Symbol Size From box and **12** in the To box. In the Classification pane, set Classes to 4. The class size breaks should default to breaking at 52, 88, 122, and 248. If not, click the Classify button, and in the Break Values box, enter the break values **52, 88, 122, and 248**, and then click OK to close the Classification dialog box. Click OK to close the Layer Properties dialog box.
- 19 In the Agent Analyst window, click Run. Then click Start on the Repast Toolbar to run the model.

You will see graduated symbols update every 100 ticks, based on the totalVisit field. The model will run to 1,441 ticks and then stop.

- 20 Close Agent Analyst without saving any changes, and then close ArcMap without saving any changes.

In chapter 9 you will build on the framework developed in this chapter to further explore how routine human activity affects the pattern of where crimes might occur.

GIS data, fields, and actions dictionaries

Table 6.5 Data dictionary of GIS datasets

| Dataset | Data type | Description |
|-------------------------|-----------|--|
| Streets.shp | line | Streets in Seattle |
| StreetIntersections.shp | point | Street intersections in Seattle |
| Subset_Nodes.shp | point | Subset of the above street intersections |

Table 6.6 Fields dictionary for the crime model

| Fields | Data type | Description |
|--|---------------------|--|
| Simple vector model agent classes | | |
| place | vector agent | Vector agents representing street intersections |
| citizen | generic agent | Citizens |
| cop | generic agent | Police officers |
| ActiveNodes | generic agent | Nodes with agents on them |
| Simple vector model fields | | |
| placeMap | java.util.HashMap | A hash map used to store the list of all the places with strnode-id as the key |
| modelStep | integer | Variable to hold the incremented model step |
| MODEL_DAY | integer | The number of ticks in a model day |
| NUM_PLACES | integer | The total number of intersection nodes |
| COPS | integer | The total number of cop agents in the model |
| Place data source and fields | | |
| Data source | string | Path to the street nodes/intersections shapefile |
| Strcl_ | string | The unique ID of the street node |
| myNeighbors | java.util.ArrayList | Each street node and its neighbors |
| Cop fields | | |
| strnode | integer | The node where a police officer is located as an integer |
| location | place | The place object that corresponds to that integer |
| Citizen fields | | |
| strnode | integer | The node where a police officer is located as an integer |
| location | place | The place object that corresponds to that integer |
| name | java.lang.String | The unique name of the agent ($a_1 \dots a_n$) |
| ActiveNode fields | | |
| strnode | integer | The node where a visitor is located as an integer |
| agentList | java.util.ArrayList | The array of agents that are currently occupying the place object |

5

6

7

8

9

Table 6.7 Actions dictionary for the crime model

| Declared actions | Description |
|----------------------------|---|
| Simple vector model | |
| initAgents | Sets the random number seed generator and initializes the random number distribution. Calls the initModel, initCitizens, initCops, and setupPlaces actions. |
| setupPlaces | Initializes a hash map and identifies the neighbors of each place for random movement. First, a hash map is created to store a list of all the places with strnode-id as the key. Next, the action reads the nodeNeighbors.csv file and associates the set of neighbors with the correct Place (i.e., it populates the field myNeighbors in the Place class). |
| initModel | Sets values for constants and static variables in the model, some of which are parameters and can be changed through the Repast user interface at model run time. |
| initCitizens | Creates citizen agents, names them, and assigns them to a strnode (number) and a location (Place). This action uses a uniform distribution to select the nodes on which to place the citizens at the start of the model. |
| initCops | Creates cop agents and assigns them to a strnode (number) and a location (Place). This action uses a uniform distribution to select the nodes on which to place the cops at the start of the model. |
| updateDisplay | Changes the display in ArcMap. The action is currently scheduled to be called every 100 ticks. |
| incrementModel | Counts the cumulative number of agents at each node and stores the information. Keeps count of the model step number and resets the totalVisit field values to zero at the start of each model run. |
| writeAgents | Writes the values of the counter for each place node to the totalVisit field in the strnodes2 shapefile. The action is currently scheduled to be called every 100 ticks. |
| Citizens | |
| step | First the action gets the list of all the places in the Place class. For each node that has a citizen, the list of neighbor nodes is shuffled, and then the citizen is assigned to the first position in the node list of possible moves. The strnode and the location fields are changed to reflect the citizen's new position. |
| Cops | |
| step | First the action gets the list of all the places in the Place class. For each node that has a cop, the list of neighbor nodes is shuffled, and then the cop is assigned to the first position in the node list. The strnode and the location fields are changed to reflect the cop's new position. |

References

- Cohen, L. E., and M. Felson. 1979. "Social Change and Crime Rate Trends: A Routine Activity Approach." *American Sociological Review* 44: 588–608.
- Groff, E. R. 2006. "Exploring the Geography of Routine Activity Theory: A Spatio-temporal Test Using Street Robbery." Unpublished dissertation, University of Maryland, College Park.
- . 2007a. "Simulation for Theory Testing and Experimentation: An Example Using Routine Activity Theory and Street Robbery." *Journal of Quantitative Criminology* 23(2): 75–103.
- . 2007b. "'Situating' Simulation to Model Human Spatio-Temporal Interactions: An Example Using Crime Events." *Transactions in GIS*, 11(4): 507–530.

5

6

7

8

9

Section 3: Advanced techniques for points, polygons, and rasters

Chapter 7

Adding complexity to polygon agents using an urban growth model

by Naicong Li

- ◆ Exploring the complete urban growth model
- ◆ Using GIS data to add the landscape to the model
- ◆ Making decisions at each step based on multiple criteria
- ◆ Building probability and randomness into the agent's behavior
- ◆ Initializing and keeping track of parcel agents' permanent and dynamic conditions
- ◆ Tracking changes at the model level due to the parcel agents' actions
- ◆ Building scenarios using different weights on suitability criteria

In chapter 3, you learned the basics of modeling with polygon agents. You learned how to initialize and work with polygon agents, how to query and update their attributes, and how to formulate decision rules for polygon agents. Building on these concepts, in this chapter you will add complexity to the polygon agent decision-making process by adding restrictions, constraints, and probability.

Adding complexity to polygon attribute changes can be used in a variety of models to simulate real-world situations. This chapter presents a simplified urban growth model in which agents are land parcels. An agent can change its development status from undeveloped to developed or to developed with higher density at each time step. The decision of an agent to change its status is based on multiple development suitability criteria and various constraints, evaluated against the respective properties of this agent. Through this model you will learn how to use GIS data representing the environmental or policy constraints on development to influence polygon agents' decision making. You will also learn how to represent various suitability criteria in your model and how to evaluate a parcel agent's properties against these suitability criteria. Finally, you will learn how to simulate different development scenarios by altering the weights on suitability criteria.

The intent of this chapter is to introduce you to agent-based modeling (ABM) principles, not how to create an urban growth model. The rules are simplified for the exercises. If they are not satisfactory, they can be changed.

The modeling scenario

You are given data for a small city with some heavily populated areas (town and village centers) and some undeveloped areas. The population of the city is expected to grow in the next 20 years, putting pressure on the undeveloped parcels to become developed. You want to build a suitability-based model to simulate the future urban growth patterns that could happen in this city, for the purpose of, for example, helping the city planning department make more informed decisions in terms of smart growth. Your model will run 20 time steps, each representing the time period of a year. At each time step, a parcel agent reevaluates its current conditions against the current suitability criteria set for that time step and decides, based on its behavioral rules, whether it should change its development status. Some of the suitability criteria may also change based on the cumulative results of agent actions, which in turn will affect the agent behavior during the following time step.

Background information

In this model, parcel agents determine their development status each time step based on their behavioral rules and their relationship with the environment.

Parcel agents and their behavior

In this sample urban growth model, you will treat each parcel in the city as an agent. (You can also consider the agents as representing the parcel owners; of course, in this case, you need to be aware that you are not considering the one-to-many, many-to-one, and many-to-many relationships between property owners and parcels.) The main attribute of these parcel agents is their development status. For simplicity in the exercises, the development status has three possible values:

- Undeveloped
- Developed
- Developed-High-Density (e.g., the parcel has multifamily housing dwellings on it)

Also for the purpose of exercise simplicity, you are not distinguishing developments of different land-use types (e.g., commercial development versus residential development).

This urban growth model is a suitability-based model. At each time step (year), a parcel agent evaluates the current situation and decides whether to change its development status (from undeveloped to some developed status), based on its current conditions, its current environment, and a set of suitability criteria. For the purpose of this exercise, we will consider only the following relevant factors in a parcel agent's decision about changing its development status:

- Physical constraints such as steep slope (> 20 percent), stream buffer area, fault line buffer area, and flood zones. The area that has such physical constraints is considered not buildable.
- Suitability factors such as ownership (private versus public), accessibility to local roads, distance to major roads, distance to town, and so on.
- Conservation land-use policy considerations, such as whether the parcel falls in a wildlife corridor protection zone.
- Neighboring parcels' development status.

Based on these factors, your parcel agents will have the following behavioral rules for changing their undeveloped status to developed status:

- If its ratio of unbuildable area (due to some physical constraints) to total area exceeds a predetermined threshold value, it will not be developed.
- If its ownership is public, it will have a very slim chance of being developed.

5

6

7

8

9

- If it does not have access to a local road, either through its own access or through that of a neighbor, it will not be developed.
- If its distance to a major road is above the current threshold, it is less likely to be developed.
- If its distance to the nearest town/village center is above the current threshold, it is less likely to be developed.
- If it falls in a conservation land-use constraints area, it is less likely to be developed.
- If a large percentage of its neighbors are developed, it is more likely to be developed.

Interaction between the agents and the environment

Agents act according to their behavioral rules. The behavioral rules are formulated partially based on some environmental factors. The current status of the environmental factors (which may change from time step to time step) affect the agents' behavior. Conversely, the results of the agents' behavior may affect the environmental factors. In this exercise model, the environmental factors are represented at the simulation model level. The interactions between agent behavior and the environment are expressed as follows:

- The various physical constraints mentioned earlier are precalculated into a combined constraints raster layer for nonbuildable areas, which is maintained by the model and which the parcel agents consult at initialization to determine their buildable area. The size of the buildable area is a crucial constraint on whether a parcel can be developed. Besides physical constraints, lack of access to local roads is also considered a constraint in this model. The distance between a parcel and the nearest local road, which is a factor in determining whether the parcel has access to local roads, is also precalculated and included in the model as a raster layer. The parcel agents consult this layer to get their distance to the nearest local road, and based on this distance an agent determines whether it has access to local roads. Note that this is a simplified function for determining road access for the purpose of this exercise.
- The overall development suitability rating of a parcel depends on, among other things, the parcel's conditions in relation to some suitability criteria, such as distance to major roads, distance to town/village centers, and not overlapping with conservation land-use constraint areas. Such conditions of the parcels are precalculated into raster layers based on environment datasets such as highways, town and village center boundaries, and conservation areas. These parcel condition raster layers are incorporated in the model at model initialization, and the parcel agents consult them to extract their condition values (e.g., their distance to town/village centers) to determine their development suitability score.

- Weighting the suitability criteria could amplify or reduce a parcel agent's overall development suitability score. These weights can be adjusted during run time in the model's user interface.
- At any time step, the model maintains a threshold of a suitability score that must be met by a parcel agent for it to become developed. This threshold value becomes lower as time steps move forward, simulating the trend that more suitable parcels will be developed first, with the less suitable parcels becoming candidates for development in later years.
- The model maintains parcel adjacency information through a list of parcel neighbors, which the parcel agents consult from time to time to check the percentage of its developed neighbors in the current time step or to see whether a neighbor or an indirect neighbor has been developed and therefore could grant access to a local road (if a parcel does not have direct access to a local road). Note that the development status of one's neighbors is dynamically changing from time step to time step. The percentage of a parcel's developed neighbors is a suitability factor as well, but it is not represented by a raster, and there is no associated weight on this factor in this model.
- Whenever a parcel agent changes its development status, it reports its change to the model so that the model can do some summary calculation at the end of each time step, such as the total number of parcels developed during this time step, and so on.

It should be noted that you are working on an exercise model to show how the functionalities of Agent Analyst can be used in urban growth modeling. The model uses only a couple of example constraints and suitability factors, and the algorithms used in calculating the suitability ratings of the parcel are often placeholders that you can replace or improve in your own model.

5

6

7

8

9

Exploring the complete urban growth model

In the following exercises you will learn more techniques to model various aspects of polygon agents' decision making. As in each chapter, in this first exercise you will be exposed to the final model, and in the subsequent exercises you will build the individual components.

Exercise 7a

In this exercise, you explore the completed urban growth model to become familiar with its structure.

Note: This is an exploration exercise. Do not run the model because it will process a large dataset, which takes a long time. You will run the model on a smaller dataset in subsequent exercises.

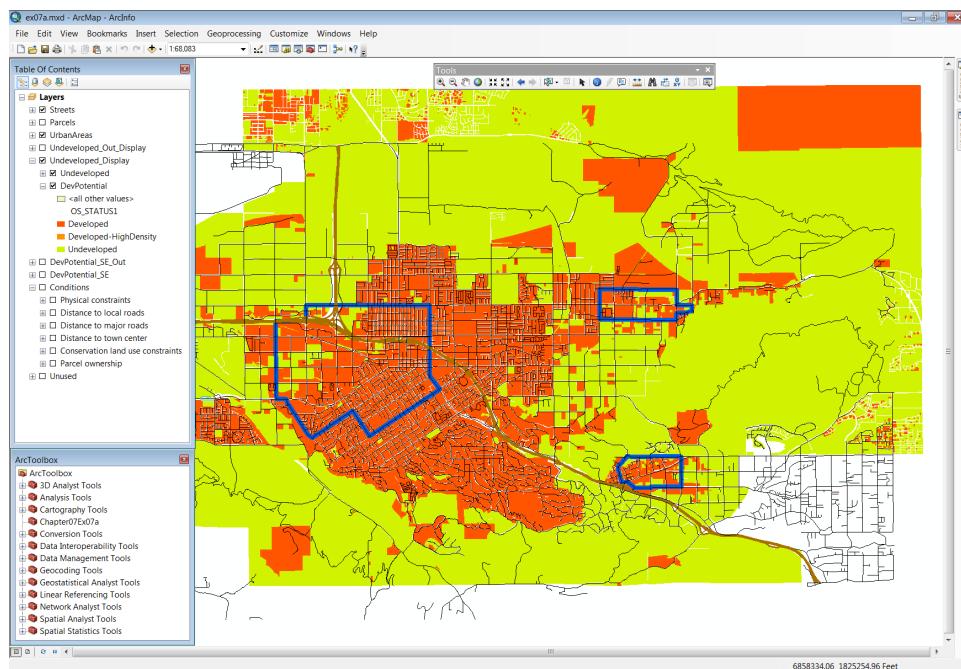
Load the ArcMap document

- 1 Start ArcMap. Navigate to C:\ESRIPress\AgentAnalyst\Chapter07. Open ex07a.mxd.

The map document opens. This map document has relevant data layers added to the display and includes the Chapter07Ex07a toolbox in ArcToolbox.

The data layers include parcels with their development status, streets, urban center boundaries, physical constraints on development, distance to urban/town centers, and so on.

Streets, urban areas, and the development potential are displayed. The red polygons are developed parcels, the orange polygons are higher-density development parcels, and green polygons are undeveloped. More detailed descriptions about these layers are given later when their use is discussed.



Load the Agent Analyst model

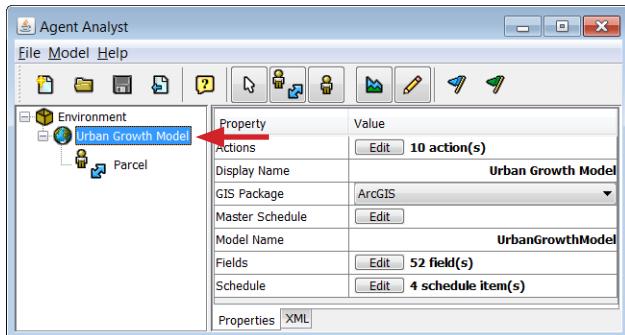
- 2** Right-click the Chapter07Ex07a toolbox, point to New, and select Agent Analyst Tool.
- 3** Once the Agent Analyst window appears, click File and select Import. Do not save the changes when prompted.
- 4** Navigate to C:\ESRIPress\AgentAnalyst\Chapter07\Models and open ex07a.sbp. The Agent Analyst window appears.

Note: If you receive an error, you need to reset the data source path for the parcel agents. To do so, in the Environment panel in the Agent Analyst window, click Parcel. In the Property panel, click the Edit button to the right of the Data Source property. In the Data Source Editor, click the Browse button and navigate to C:\ESRIPress\AgentAnalyst\Chapter07\Data. Select DevPotential.shp, and then click Open. Click OK in the Data Source Editor.

This sample urban growth model has two main components:

- Urban Growth Model
- Parcel agents

5
6
7
8
9



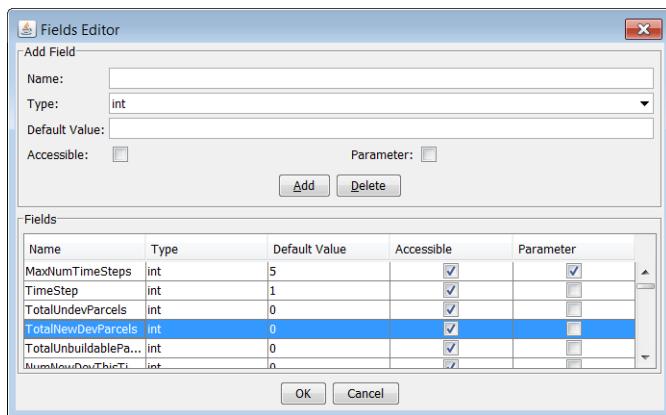
The model name is UrbanGrowthModel, and the display name is Urban Growth Model. Like most of the Agent Analyst models, this model is responsible for the following tasks:

- Set up the input environment data layers.
- Initialize the parcel agents.
- Schedule the agents' activities in terms of the time steps.
- Update and manage information at each time step and adjust the model state based on results of agent actions during that time interval.
- Interface with ArcGIS for accessing data and for updating the map with the model run results at each time step.
- Perform summary analysis and produce reports.

Explore the fields

- 5 In the Environment panel, click Urban Growth Model. In the Property panel, click the Edit button to the right of the Fields property to bring up the Fields Editor.

You will see the fields (attributes) for the model:



These fields are used to record information about the environment in which the parcel agents live. Some fields will be consulted by parcel agents at each time step when they make decisions about whether they should change their development status. Some of the fields will be discussed in more detail in the following exercises, and a list of these fields and their descriptions is provided at the end of this chapter in a data dictionary.

6 Close the Fields Editor.

The urban growth model has 10 actions. Some of them are called from within the `initAgents` action when it executes. Some of them are scheduled to run at a certain point in the simulation by the scheduler of Agent Analyst. In the following exercises, we will go over some of these actions.

Examine the `initAgents` action

- 7 In the Environment panel, click Urban Growth Model. In the Property panel, click the Edit button to the right of the Actions property. The Actions Editor appears.

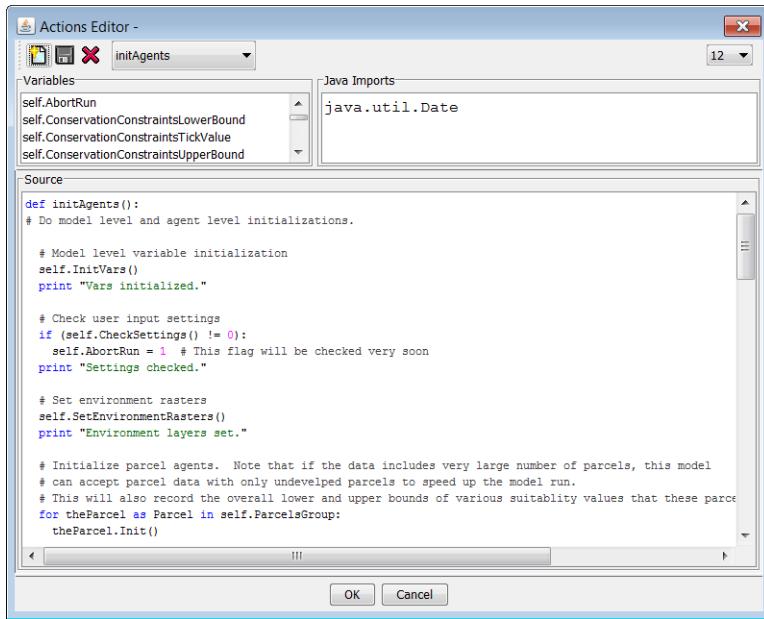
5

6

7

8

9



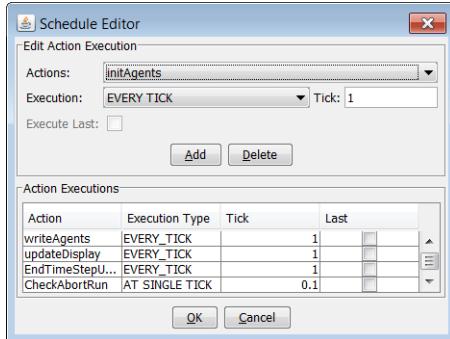
The `initAgents` action is predeclared by Agent Analyst, and the model will execute this action first at the start of the simulation.

- 8** Close the Actions Editor.

View the scheduler

- 9** In the Environment panel, click Urban Growth Model. In the Property panel, click the Edit button to the right of the Schedule property.

In the Schedule Editor, notice that the `initAgents` action is scheduled for EVERY TICK.

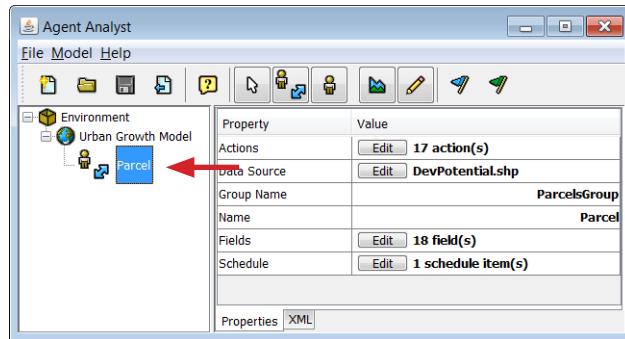


All the actions listed in the Actions Editor that are not called by the initAgents action need to be scheduled here to be executed. The writeAgents and updateDisplay actions are used to update the parcel representation in ArcMap each time step.

10 Close the Schedule Editor.

The second major component of this model (located in the Environment panel) is the parcel agents. The collection of the individual parcel agents is called ParcelsGroup. Each member of this group (an agent) is of type Parcel. For your convenience, this agent group is already created in the sample model, as shown in the following figure.

11 In the Environment panel, click Parcel. This activates the parcel agent definitions.



Notice that the data source for parcel agents is from DevPotential.shp. This shapefile layer should already be displayed and zoomed to in ArcMap. If it is not, click the layer in the ArcMap Table Of Contents. Then, right-click the layer, and click Zoom To Layer. The DevPotential layer is displayed with Streets, UrbanAreas, and Undeveloped in ArcMap. The DevPotential layer is the source for the parcel agents.

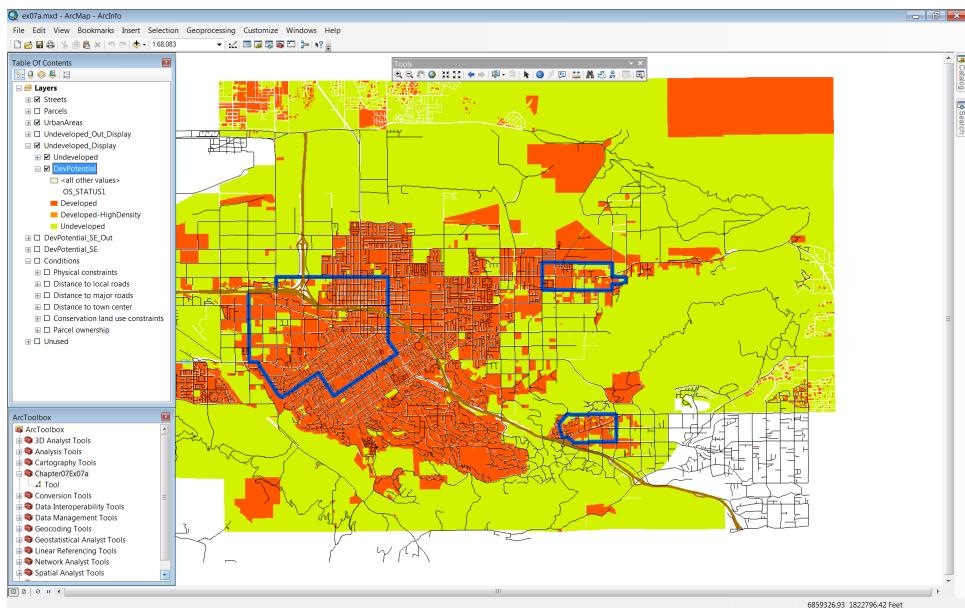
5

6

7

8

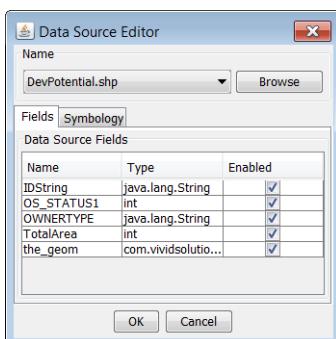
9

Exercise 7a

The red polygons are parcels that are already developed, the green polygons are parcels that are undeveloped. There are a couple of highways going through this area. The blue boundary polygons indicate the boundaries of towns/village centers, which are created for exercise purposes. Local streets are also shown. Because of the large number of parcels, the parcel boundaries are not shown to reduce clutter in the image.

Explore how the attributes of these agents are defined

- 12** In the Agent Analyst Environment panel, click Parcel. In the Property panel, click the Edit button to the right of the Data Source property to open the Data Source Editor. Notice that DevPotential is identified as the data source for the parcel agents.



The agents are created by Agent Analyst using the input shapefile DevPotential.shp. Each feature in the shapefile becomes a parcel agent. In the Data Source Editor you also see

attributes of the parcel agents: IDString, OS_STATUS1, and so on. These attributes are the fields in the shapefile DevPotential.shp.

To see these fields and their values, bring up the shapefile's attribute table using ArcMap.

- 13** Right-click the DevPotential layer in ArcMap, and then click Open Attribute Table.

| FID | Shape | OS_STATUS1 | OWNERTYPE | IDStrir |
|------|---------|------------|-----------|---------|
| 7785 | Polygon | 1 | Private | 7785 |
| 7786 | Polygon | 1 | Private | 7786 |
| 7787 | Polygon | 2 | Private | 7787 |
| 7788 | Polygon | 2 | Private | 7788 |
| 7789 | Polygon | 3 | Private | 7789 |
| 7790 | Polygon | 2 | Private | 7790 |
| 7791 | Polygon | 1 | Public | 7791 |
| 7792 | Polygon | 1 | Private | 7792 |
| 7793 | Polygon | 1 | Private | 7793 |
| 7794 | Polygon | 1 | Private | 7794 |
| 7795 | Polygon | 3 | Private | 7795 |
| 7796 | Polygon | 1 | Private | 7796 |
| 7797 | Polygon | 1 | Private | 7797 |
| 7798 | Polygon | 1 | Private | 7798 |
| 7799 | Polygon | 1 | Private | 7799 |
| 7800 | Polygon | 1 | Private | 7800 |

The relevant fields that become the attributes of the parcel agents in the urban growth model include the following:

- OS_STATUS1, the development status of a parcel:
 - 1 = developed
 - 2, 3 = undeveloped
 - 4 = developed; high density
- OWNERTYPE, the ownership type of the parcel (public vs. private)
- IDString, the parcel ID
- TotalArea, the total square footage of the parcel

You can refer to these fields in the parcel agent code. In this model, the agents require more attributes than those just listed. You can define additional fields (attributes) for the parcel agents, as shown later.

- 14** Click Cancel to close the Data Source Editor, and close the attribute table for the shapefile in ArcMap.

5

6

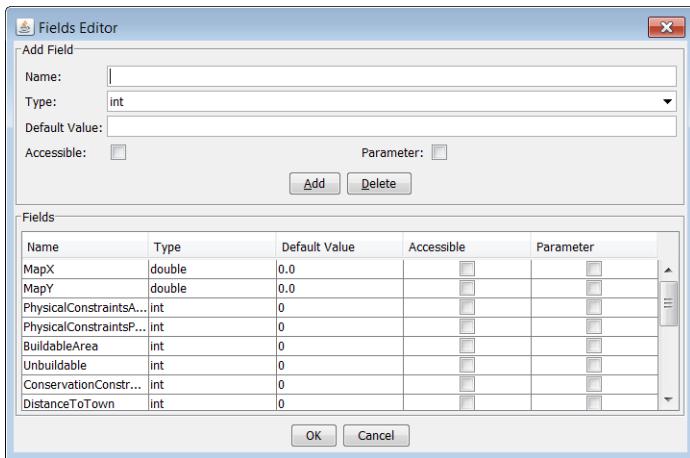
7

8

9

Explore the additional fields that were added to the parcel agents

- In the Environment panel, click Parcel. In the Property panel, click the Edit button to the right of the Fields property to open the Fields Editor.



The Fields Editor allows you to define and modify the parcel agent attributes that are not already defined in the parcel shapefile. These attributes and their descriptions will be shown in the data dictionary at the end of this chapter. The values for these attributes will be filled in with data from layers other than the parcel data.

- Click Cancel to close the Fields Editor.

Look at the actions for the parcel agents

- In the Environment panel, click Parcel. In the Property panel, click the Edit button to the right of the Actions property to open the Actions Editor.

Seventeen actions are defined for the parcel agents based on the agents' behavioral rules. You can achieve the desired behavior of your parcel agents by creating these actions. Among these actions, the step action is the main action and is scheduled to run at every time step by all the parcels. The other actions are called by the step action or by some other action in the model. You will go over some of the actions later.

- Click Cancel to close the Actions Editor.
- Close Agent Analyst and then close ArcMap without saving.

In this exercise you explored the urban growth model using the entire dataset. You will not run the model at this time because doing so would take too long. In later exercises you will run the model on a subset of the data.

Using GIS data to add the landscape to the model

Agent behavior rules are often formulated based on some environmental conditions. In this model, the environment is established by adding a landscape using the available GIS data. For the purpose of this exercise, you will consider the following environmental factors:

- Physical constraints (combining constraints including steep slope, stream buffer area, fault line buffer area, and flood zones)
- Accessibility to local roads
- Distance to major roads
- Distance to town/village centers
- Conservation land-use constraints

5

6

7

8

9

Each of these factors is represented as a raster layer, which is already prepared for you.

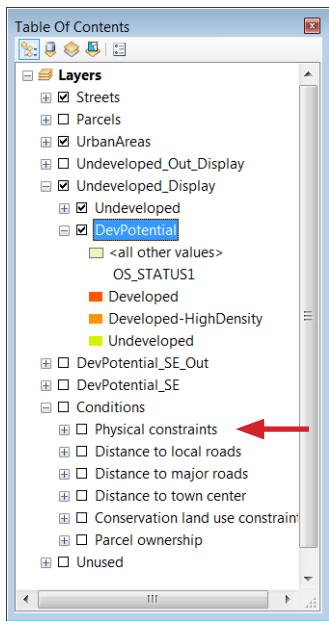
Exercise 7b

Load the ArcMap document

- 1 Start ArcMap. Navigate to C:\ESRIPress\AgentAnalyst\Chapter07. Open ex07b.mxd.

In the ArcMap display, a series of layers have been loaded. In ArcToolbox, the Chapter07Ex07b toolbox has been created for you.

In ArcMap you can see these layers in the table of contents of your project, as shown in the following figure:



The environmental conditions layers will be used in the urban growth model.

To explore the data layers, you may want to display your layers in list mode if the table of contents is not in that view. To do so, click the List By Drawing Order button at the top of the table of contents (the left button just above Layers). To get a sense of the environmental conditions, turn them on one at a time. You need to turn on both the group layer Conditions and the raster layer you want to look at. Click the raster plus sign to see the symbology.

There is an extra, nonraster layer included under the Conditions group layer. This is represented here so that you can have a clear view about the ownership situation of the parcels.

To use these layers, you need to first add them to the model. Instead of hard-coding the raster names and workspace path (assuming they all reside in the same workspace) within the model code, you add them to the model as model-level fields. This makes it easier if you want to replace an existing layer with a different one.

Load the Agent Analyst model

- 2 Right-click the Chapter07Ex07b toolbox, point to New, and select Agent Analyst Tool.
- 3 In the Agent Analyst window, click File and select Import. Do not save the changes when prompted.
- 4 Navigate to C:\ESRIPress\AgentAnalyst\Chapter07\Models and open ex07b.sbp.

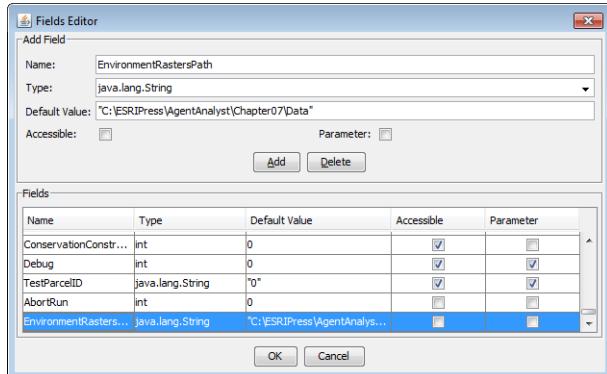
Note: If you receive an error, see the note after step 4 of exercise 7a.

Note: Do not run the model because it will process a large dataset, which will take a long time. You will be running the model on a smaller dataset in subsequent exercises. Also, the model cannot be run until you finish adding the environmental layers to the model.

Add the layer names and path to the model

- 5 In the Environment panel, click Urban Growth Model. In the Property panel, click the Edit button to the right of the Fields property to open the Fields Editor.
- 6 In the Fields Editor, define the following field for the raster workspace path:
 - Name: EnvironmentRastersPath
 - Type: java.lang.String
 - Default value: "C:\ESRIPress\AgentAnalyst\Chapter07\Data"
- 7 Leave the Accessible and Parameter check boxes cleared. Click Add.

You will see the new attributes in the Fields list. Notice in the following image that the field you added is highlighted.



5

6

7

8

9

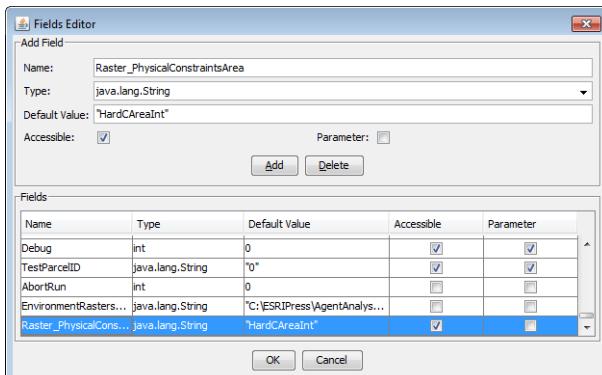
Add a field for the physical constraints layer

- 8 In the Fields Editor, create the following field:

- Name: Raster_PhysicalConstraintsArea
Type: java.lang.String
Default value: "HardCAreaInt"
Accessible: Selected
Parameter: Not selected

With Accessible selected, this field can be accessed by other components of the model, such as parcel agents.

- 9 Click Add, and you will see this raster layer name recorded in the Fields list. Notice in the following image that the field for the Constraints raster that you just added is highlighted.



- 10 Repeat the preceding steps to add the rest of the raster layer names as fields using the information in table 7-1.

Table 7.1 Fields Editor data

| Name | Type | Default value | Accessible | Parameter |
|------------------------------------|------------------|---------------|------------|--------------|
| Raster_LocalRoadDistance | java.lang.String | LocRdDist_zn | Selected | Not selected |
| Raster_MajorRoadDistance | java.lang.String | MajRdDist_zn | Selected | Not selected |
| Raster_DistanceToUrban | java.lang.String | UrbanDistInt | Selected | Not selected |
| Raster_ConservationConstraintsPerc | java.lang.String | SoftCPercInt | Selected | Not selected |

- 11 Click OK to close the Fields Editor.

Add code to initialize these rasters

To modularize the handling of this task, you will use a separate action called SetEnvironmentRasters, which you will call from the initAgents action. This action is already defined for you. You need to uncomment the code to activate it.

- 12 In the Environment panel, click Urban Growth Model. In the Property panel, click the Edit button to the right of the Actions property to open the Actions Editor.
- 13 In the Actions Editor, select initAgents in the Actions drop-down list. Uncomment the following line in the initAgents action:

```
self.SetEnvironmentRasters()
```

Now the SetEnvironmentRasters action will be called from the initAgents action.

- 14 In the Actions Editor, select SetEnvironmentRasters from the Actions drop-down list to see the code in the action.

Observe that for each raster in the SetEnvironmentRasters action, a raster object is created from the values of the EnvironmentRastersPath field and the raster name field defined earlier. The addRaster action (a predefined Agent Analyst action) is called to load the raster into the model with a specified name (in this case the raster dataset name defined above is used) and this name will be used as a reference within the model.

- 15 Click OK to save your changes and close the Actions Editor.

Now that the raster layers are registered with the model, you can refer to them elsewhere in the code. Follow the next set of steps to examine how they are used.

- 16 In the Environment panel, click Parcel. In the Property panel, click the Edit button to the right of the Actions property to open the Actions Editor.
- 17 In the Actions Editor, select Init from the Actions drop-down list. You will see the following code in the Source panel.

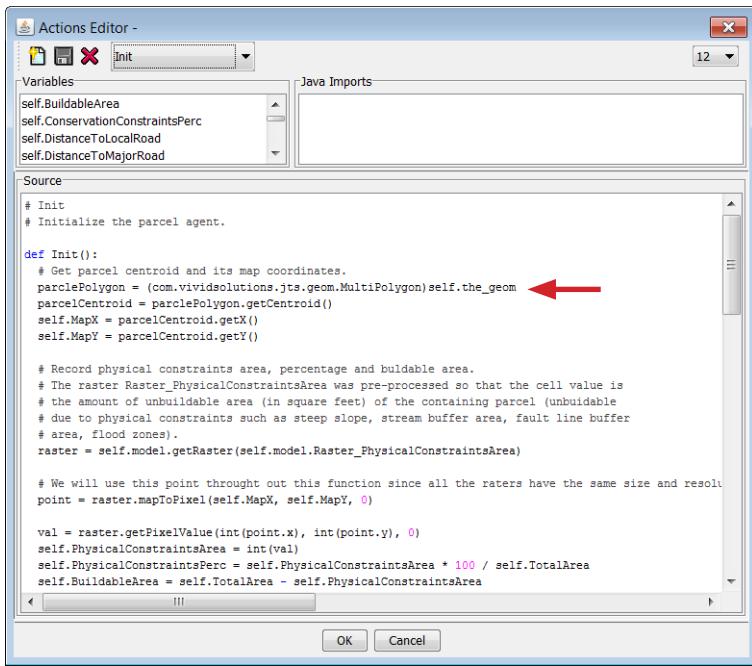
5

6

7

8

9



In the Init action, the parcel agent consults the raster layers to get information about itself and stores the information if necessary. For example, the parcel agent would want to know the total area within its boundary that overlaps with the physical constraints area. To do this, the parcel consults the value of the physical constraints raster at the location of the parcel's centroid; the code is highlighted by the red arrow.

The code finds the parcel's centroid and identifies its coordinates by first getting the geometry of the parcel (`parcelPolygon = (com.vividsolutions.jts.geom.MultiPolygon)self.the_geom`). The centroid of the parcel is determined next (`parcelCentroid = parcelPolygon.getCentroid()`), and the X centroid coordinate is assigned to a variable (`self.MapX = parcelCentroid.getX()`), as is the Y centroid coordinate (`self.MapY = parcelCentroid.getY()`).

Once the location of the centroid is determined, the parcel can consult the raster layer to get the value at that location in the second section of code.

The code first gets the raster of interest. The parcel agent can identify the raster by the proper model field, which we added earlier. In this case, the field is `Raster_PhysicalConstraintsArea` (`raster = self.model.getRaster(self.model.Raster_PhysicalConstraintsArea)`). The coordinates of the centroid of the parcel are then mapped to the raster pixel (`point = raster.mapToPixel(self.MapX, self.MapY, 0)`). The value for the pixel (for the parcel centroid) is identified (`val = raster.getValue(int(point.x), int(point.y), 0)`).

The raster layer values were preprocessed so that the cell value equals the amount of unbuildable area (in square feet) of the containing parcel (unbuildable due to physical constraints).

In the same way, the parcel agents call `self.model.getRaster` to get the relevant value from the other raster layers that you registered with the model earlier (see the rest of the code in the `Init` action): `self.model.Raster_ConservationConstraintsPerc`, `self.model.Raster_DistanceToUrban`, `self.model.Raster_LocalRoadDistance`, and `self.model.Raster_MajorRoadDistance`.

Notice that the point variable representing the parcel centroid in the preceding code can be reused for all these rasters because these rasters have the same resolution.

18 Click Cancel to close the Actions Editor.

19 Close the model and then close ArcMap. Do not save any changes.

As mentioned earlier, the raster layers used in this sample model are preprocessed and contain values that can be directly used as environmental factors by the parcel agents. For example, the model does not directly include major roads as an environmental layer, but rather a raster with precalculated distances between the parcels and major roads is included as an environment layer. You could, of course, include the major roads information directly in the model and calculate the distances between the parcels and major roads during the model run. This would mean that the model would have to do a lot more work, and do it every time you run the model. This would have an impact on the model run-time speed, at least during the model's initialization. Therefore, certain environmental conditions are treated as permanent and are precalculated.

On the other hand, if an environmental condition is likely to change over time—for example, the boundaries of town/village centers might change over time (or new roads may be added)—then you should include these factors directly in the model instead of capturing the relationship in a precalculated layer. This lets you control the change of these boundaries and recalculate the distance every time the boundaries change. (The same argument could be used for directly including the major roads layer because there may be new major roads emerging after a certain period of time.) However, in this simplified sample model these environmental conditions are treated as permanent, although there is some placeholder code to simulate the effect of expanding town/village center boundaries in exercise 7f.

5

6

7

8

9

Making decisions at each step based on multiple criteria

A parcel agent makes its decision on whether to change its development status at each time step (year). The parcel agent bases its decision on a set of criteria that are incorporated into a set of agent behavioral rules, described earlier in the section “Background information.” These agent behavioral rules are implemented in the parcel agent’s actions. Follow the steps in this exercise to examine these actions.

The parcel shapefile you saw earlier, DevPotential.shp, has more than 38,000 parcels and is too large for repeated model runs during model development—as well as for the purposes of this exercise. To obtain faster testing turnaround times, a subset of the study site was extracted. It contains the same attribute categories as the larger dataset but for a smaller geographic area. You will use this much smaller shapefile throughout this exercise. (After a few exercises, you can switch back to the larger dataset and rerun the model.)

Exercise 7c

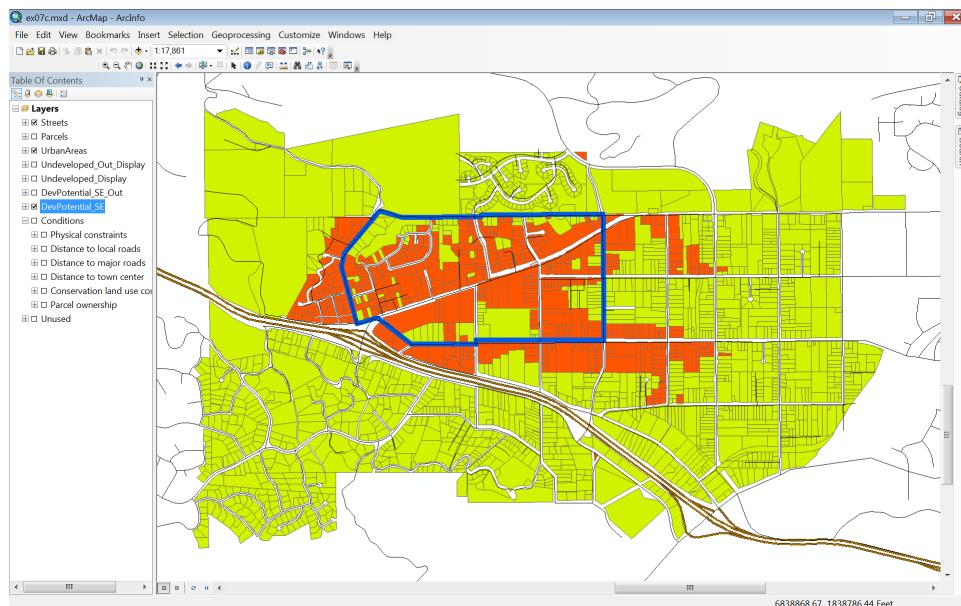
You will now work with the smaller dataset and make the necessary adjustment in the model as well.

Load the ArcMap document

- 1 Start ArcMap. Navigate to C:\ESRIPress\AgentAnalyst\Chapter07. Open ex07c.mxd.

The ArcMap display appears and both the parcels for the larger dataset and the parcels defining the smaller dataset are loaded in the table of contents. In ArcToolbox, the Chapter07Ex07c toolbox has been created for you. You might want to display your layers in list mode if they are not. To do so, click the List By Drawing Order button at the top of the table of contents (the left button just above Layers).

- 2 In the ArcMap Table Of Contents panel, clear the check box for the group layer Undeveloped_Display, then click and select the smaller dataset DevPotential_SE and zoom to this layer. This dataset contains a subset of parcels in DevPotential.shp, with some of the parcel’s development status altered for the purpose of this exercise. Again, you will not be considering different land-use types (commercial, residential, open space, and so on) in the exercise.



5
6
7
8
9

Note that the parcel boundaries are shown in this small area.

Load the Agent model

- 3** Right-click the Chapter07Ex07c toolbox, point to New, and select Agent Analyst Tool.
- 4** Once the Agent Analyst window appears, click File and select Import. Do not save the changes when prompted.
- 5** Navigate to C:\ESRIPress\AgentAnalyst\Chapter07\Models and open ex07c.sbp.

Note: If you receive an error, see the note after step 4 of exercise 7a.

Change the data source for the parcels in the urban growth model to this smaller dataset

- 6** In the Environment panel, click Parcel. In the Property panel, click the Edit button to the right of the Data Source property to open the Data Source Editor.
- 7** In the Data Source editor, click the Browse button, and select DevPotential_SE.shp in the Data folder. Click Open to select the new shapefile.

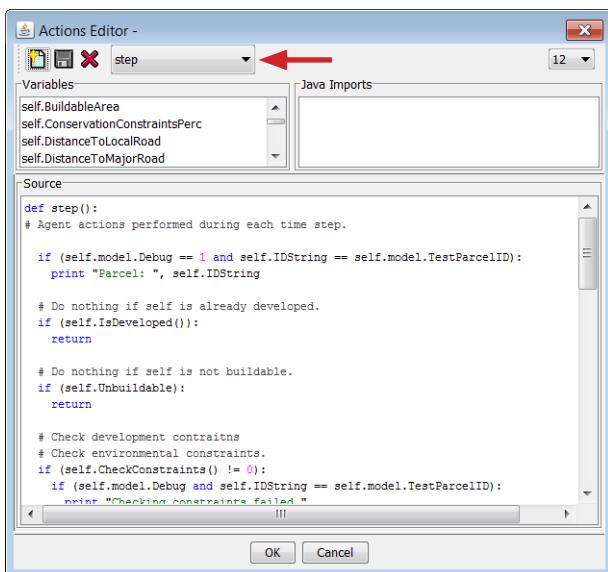
The small dataset contains the same parcel attribute information as the larger dataset, but for a smaller portion of the geographical area.

- 8** Click OK to accept the change and close the Data Source Editor.

The data source is now set to the dataset subset.

Examine the step action in the model

- 9** In the Environment panel, click Parcel. In the Property panel, click the Edit button to the right of the Actions property to open the Actions Editor.
- 10** By default, the step action will be displayed in the Source panel.



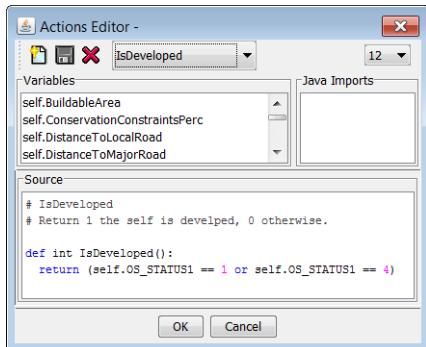
Note that in the step action, as well as in many other actions, you will see the code segment
`if (self.model.Debug == 1 and self.IDString == self.model.TestParcelID) :`, which is used to print out debugging information when the Debug field of the model is set to nonzero. You can set this field before the model run in the model's Settings window (you will see how in later exercises).

In the step action, the parcel agent considers various factors before it makes the decision whether to change its development status. This code first determines if the parcel is developed. If the parcel is already developed, then it does nothing during this time step, otherwise it considers several factors to determine whether it should be developed, including the parcel's physical constraints for development, the ownership of the parcel, the parcel's access to local roads, the parcel's development suitability in terms of its distance to town, its distance to major roads, and consideration of surrounding conservation land. It checks the percentage of the developed neighboring parcels. The check of these factors and constraints is often done by the step action's call to some other actions defined for the parcel agents.

Examine the code for determining whether to change development status

The step action first calls the IsDeveloped action.

- 11 With the Actions Editor for parcel agents still open, select the IsDeveloped action from the Actions drop-down list. You will see the following code in the Source panel. The action determines whether a parcel is currently developed.



5
6
7
8
9

This action returns 1 if the parcel is already developed or 0 if the parcel is not yet developed. Whether a parcel is already developed is determined by the value of OS_STATUS1. OS_STATUS1 is a field in the attribute table of the parcel shapefile, with the following values:

OS_STATUS1 (development status of a parcel):

- 1 = developed
- 2, 3 = undeveloped
- 4 = developed; high density

The code (`return (self.OS_STATUS1 == 1 or self.OS_STATUS1 == 4)`) returns 1 if OS_STATUS1 equals 1 or 4; it returns 0 if OS_STATUS1 equals 2 or 3. The IsDeveloped action is a simplification of what could happen. For example, you could modify the code to take into account the fact that some developed parcels may revert back to be undeveloped under certain conditions.

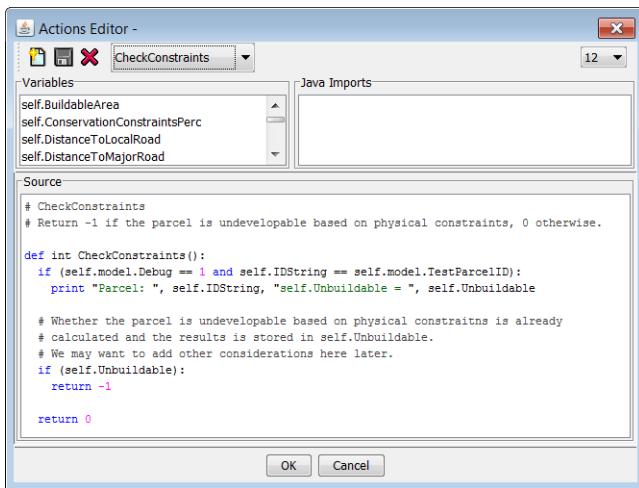
Go back to the step action to examine its code after the IsDeveloped action is called

- 12 In the Actions Editor, select the step action from the Actions drop-down list. The Source panel will display the step action's code again.

After checking its development status, the parcel agent calls the CheckConstraints action to see whether there are physical constraints that would prohibit its development.

Examine the CheckConstraints action

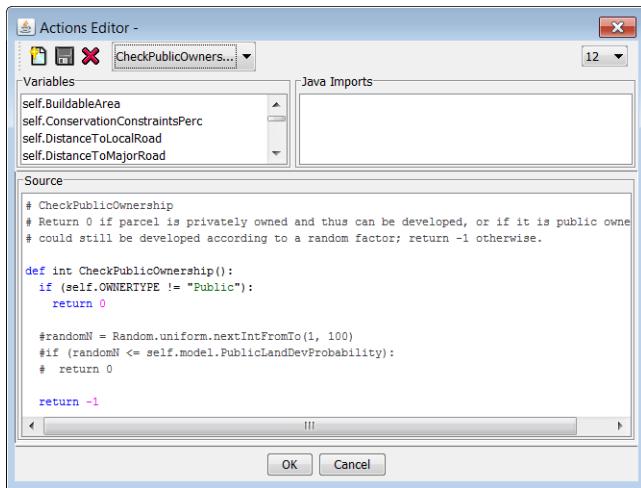
- 13** In the Actions Editor, select CheckConstraints from the Actions drop-down list. View the following code in the Source panel. This action determines whether a parcel can be developed or not based on the physical constraints contained within the parcel.



Physical constraints (e.g., steep slopes, fault line buffer zones, stream buffers, and so on) are assumed to be permanent conditions in this model, and, as we mentioned previously, we have precalculated the total nonbuildable area for the parcels ahead of time based on the parcels' physical constraints. You further precalculate the buildability of a parcel at the time parcel agents are initialized, and store the results in the agent's UnBuildable attribute. At this stage (with the code not being complete), the CheckConstraints action only checks whether the parcel agent's Unbuildable property (`if (self.Unbuildable) :`) is true. If it is true, the action will return -1, otherwise it will return 0.

The parcel agents then check the ownership of the parcel by calling the CheckPublicOwnership action from the step action. Open the CheckPublicOwnership action.

- 14** Select CheckPublicOwnership from the Actions drop-down list. You will see the following code in the Source panel. The action determines whether the parcel is publicly owned.



At this stage (with some of the code commented out), the CheckPublicOwnership action only checks whether the parcel is publicly owned (`if (self.OWNERTYPE != "Public")`). If the parcel is not publicly owned, the action returns 0 (`return 0`). If it is publicly owned, it will not be developed. Later, we will add more content to this action.

Next in the step action, the parcel agent verifies that it has access to local roads—a necessary condition for it to be developed. Explore the CheckDistanceToLocalRoads action.

- 15** In the Actions Editor, select the CheckDistanceToLocalRoads action from the Actions drop-down list. You will see the following code in the Source panel. The action determines whether the parcel agent is within an allowable distance to a road for development.

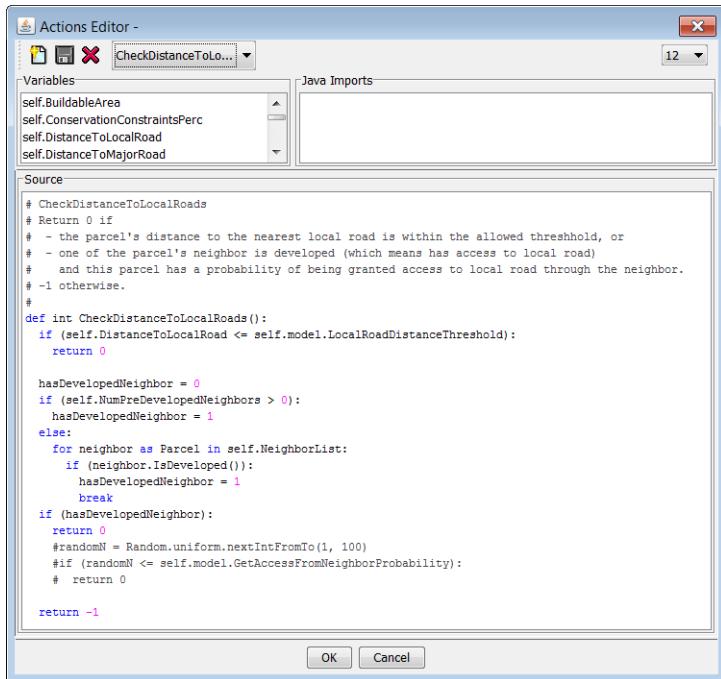
5

6

7

8

9



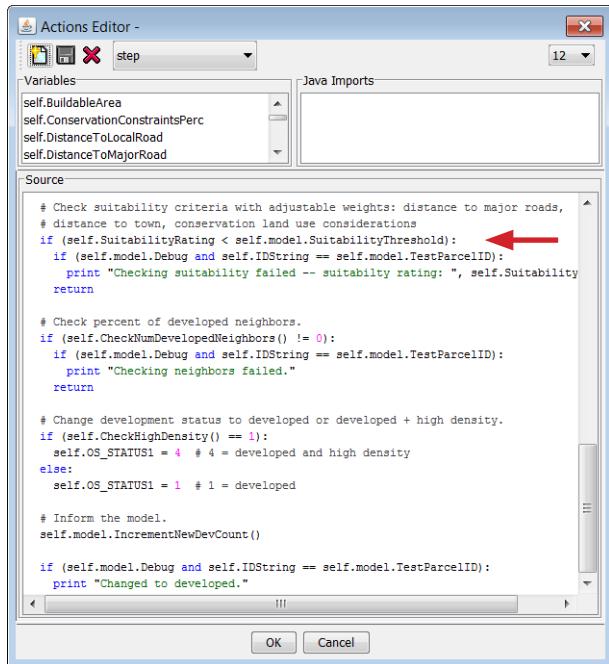
Currently, the algorithm looks only at the distance between this parcel and the nearest local road and determines whether this parcel has a neighboring parcel that is already developed. If the distance between this parcel and the nearest local road is lower than a threshold value (`if (self.DistanceToLocalRoad <= self.model.LocalRoadDistanceThreshold) :`), the parcel is considered to have direct access to local roads; otherwise, if the parcel has some direct neighboring parcel that's already developed (`if (self.NumPreDevelopedNeighbors > 0) :`), it is assumed that this parcel can gain access to a local road through this neighboring parcel. If this parcel does not have an adjacent neighboring parcel that is developed, the parcel checks its neighboring parcels (`for neighbor as Parcel in self.NeighborList:`) to see if any of them have a direct neighboring parcel that is developed (`if (neighbor.IsDeveloped()) :`). Note that this code calls the `IsDeveloped` action that you saw earlier to test the neighboring parcels. If it is determined that the parcel has direct or indirect access to local roads, the `CheckDistanceToLocalRoads` action returns 0. Otherwise, it will return a nonzero value, in which case the parcel is considered not suitable for development (for this time step). Ignore any commented code in the logic of the model. As you know, it will not be executed. In later exercises you will uncomment additional lines of code.

Both the distance (between the parcel and the nearest local road) and the threshold value are preset; however, the threshold value can be adjusted. The code that determines whether the parcel has local road access can also be enhanced. For example, you might assume that the parcel could obtain access to a local road through an undeveloped neighboring parcel under certain conditions.

Thus far, you have seen the parcel agent's actions checking constraints on development—such as constraints from the physical environment and constraints imposed by accessibility to local roads. In addition to constraints, you need to consider the development suitability of a parcel.

Return to the step action

- 16** In the Actions Editor, select the step action from the Actions drop-down list and examine the code that checks the parcel's development suitability rating.



In the step action code highlighted by the arrow, the parcel agent evaluates the development suitability (`if (self.SuitabilityRating < self.model.SuitabilityThreshold):`).

The parcel's development suitability rating, `self.SuitabilityRating`, is calculated in this sample model based on the following criteria:

- The parcel's distance to major roads (e.g., freeway, highway)
- The parcel's distance to town and village centers
- The parcel's location in relation to conservation considerations, such as proposed wild-life corridors

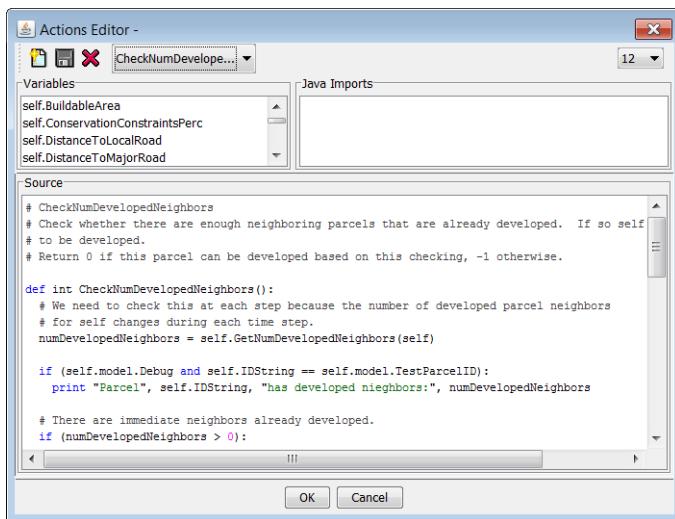
5
6
7
8
9

The value of self.SuitabilityRating for the parcel has already been calculated before the step action executes. This development suitability code checks its values against a threshold value that is also preset before this step action executes and stored in the model. If the development value does not meet the threshold value, the parcel will not be developed during this time step. You will look at how the parcel's development suitability rating and the model's suitability threshold value are calculated in the later exercises.

Next, the parcel examines the development status of its neighboring parcels. This model assumes that—everything else being equal—a parcel is more likely to be developed if more of its neighbors are already developed.

Examine the CheckNumDevelopedNeighbors action

- 17** In the Actions Editor, select CheckNumDevelopedNeighbors from the Actions drop-down list and examine the code in the Source panel, as shown in the following figure:

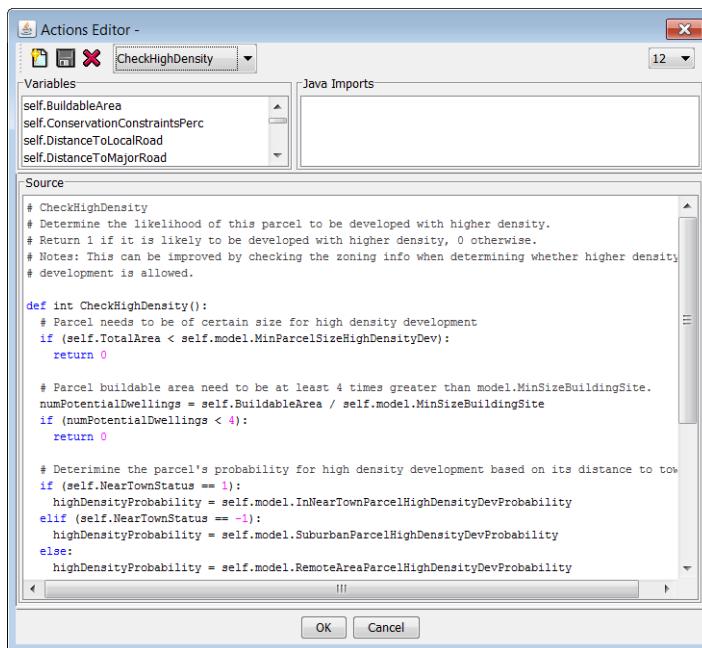


With some of the code commented out, the active code is very simple. The parcel agent first gets the number of developed neighboring parcels (`numDevelopedNeighbors = self.GetNumDevelopedNeighbors(self)`). If the parcel has a neighbor that is already developed (`if (numDevelopedNeighbors > 0):`), it passes the developed-neighbor test (the function returns 0). If none of the immediate neighboring parcels are developed, the function checks to see if any of these neighbors has a neighboring parcel that is developed (`if (numDevelopedDistantNeighbors > 0):`). If yes, the parcel (`self`) is also considered to have passed the developed-neighbor test. If the parcel does not pass the test, the action returns -1, and the parcel will be barred from being developed during this time step. You will see in the next exercise how to make this code a bit more sophisticated by looking at the percentage of developed neighboring parcels.

If the parcel has passed all the tests thus far, then you assume that this parcel will be developed during this time step. In the step action, the parcel now needs to decide whether it will be developed at a higher density.

Examine the CheckHighDensity action

- 18** In the Actions Editor, select CheckHighDensity from the Actions drop-down list and examine the code in the Source panel, shown in the following figure. The action determines whether the parcel agent, since it is going to be developed, will undergo higher-density (multifamily) development or not.



The CheckHighDensity action first verifies that the parcel has enough buildable area for higher density development by checking how many minimally sized potential dwellings the buildable area can accommodate (`numPotentialDwellings = self .BuildableArea / self.model.MinSizeBuildingSite`). The model then checks the probability of this parcel being developed at a higher density, which is based on where this parcel is located—whether it is in or near town (`if (self .NearTownStatus == 1)`), in a suburban area (`elif (self.NearTownStatus == -1)`), or in a rural area (`else`). It then checks this probability against a threshold value. In the end, if the parcel determines that it should be developed with higher density, the function returns 1, otherwise it returns 0. In either case, the parcel will be developed.

5

6

7

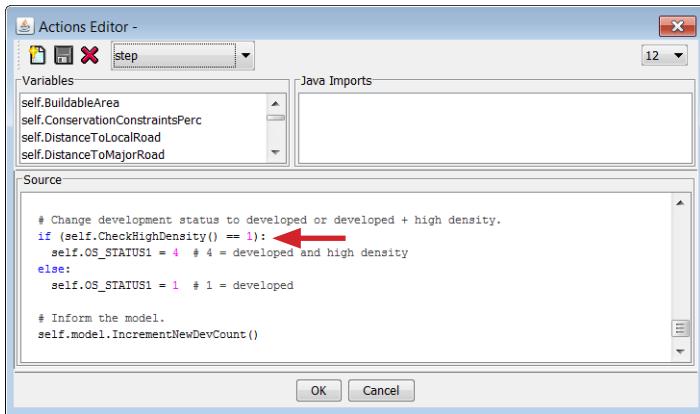
8

9

Return to the step action

The CheckHighDensity action that you just reviewed returns a status check determining whether the type of development should be higher density or not; the step action sets the agent's OS_STATUS property to the appropriate development type.

- 19 In the Actions Editor, select the step action from the Actions drop-down list and examine the code at the end of the action. This code sets the type of development that will occur in the parcel agent—normal or higher-density development.



After the CheckHighDensity action returns (see the red arrow in the preceding image), the parcel sets self.OS_STATUS1 to either 1 (developed) or 4 (developed with higher density), and communicates this status change to the model (self.model.IncrementNewDevCount();).

This code asks the model to increment its count for newly developed parcels. This enables the urban growth model to stay informed about the number of parcels being developed each year (each time step). The assignment uses the parcel agent's predefined attribute, self.model, which provides access to the model and its attributes and actions.

- 20 Close the model and ArcMap. Do not save any changes.

In this exercise, you have examined a series of deterministic actions that a parcel agent undertakes based on various constraints and criteria at each time step (each year) to decide whether it will become developed. In the next exercise, you will add probability and randomness to the parcel's decision making.

Building probability and randomness into the agent's behavior

In the previous exercise, the evaluation of the parcel agent's conditions was done in a deterministic way. For example, if a parcel is publicly owned, then it is immediately disqualified from being developed. However, in real-world situations, things often do not happen deterministically. Although rare, there is some probability that a publicly owned parcel would be developed. You therefore need to build such probability into agent behavioral rules.

5

6

7

8

9

Exercise 7d

Load the ArcMap document

- 1 Start ArcMap. Navigate to C:\ESRIPress\AgentAnalyst\Chapter07. Open ex07d.mxd.

In the ArcMap display, you will see the same layout as in the previous exercise. In ArcToolbox, the Chapter07Ex07d toolbox has been created for you.

Load the Agent Analyst model

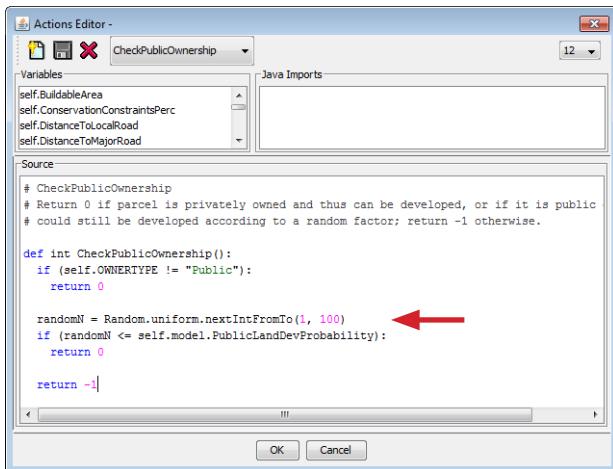
- 2 Right-click the Chapter07Ex07d toolbox, point to New, and select Agent Analyst Tool.
- 3 Once the Agent Analyst window appears, click File and select Import. Do not save the changes when prompted.
- 4 Navigate to C:\ESRIPress\AgentAnalyst\Chapter07\Models and open ex07d.sbp.

Note: If you receive an error, see the note after step 4 of exercise 7a.

There are several places in this model that have probability calculations in the parcel agent's actions.

Add probability to an action

- 5 In the Environment panel, click Parcel. In the Property panel, click the Edit button to the right of the Actions property to open the Actions Editor.
- 6 From the Actions drop-down list, select CheckPublicOwnership. In the Source panel, uncomment the three commented lines of code pointed to by the arrow so that it looks like the code in the following image.



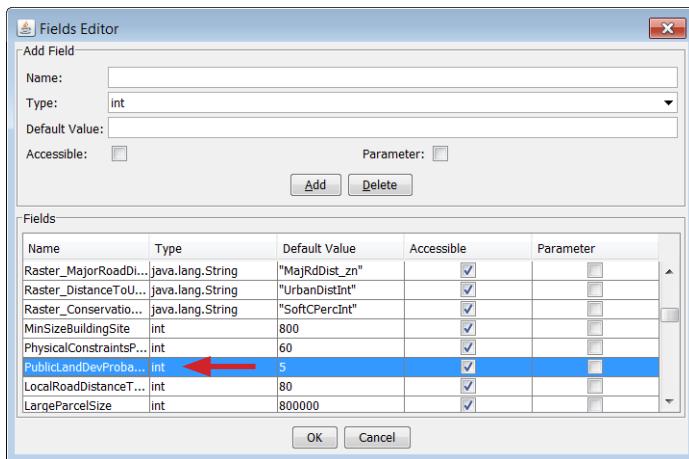
Randomness is added to determine whether the parcel can turn from public to developed. If the parcel is privately owned (`if (self.OWNERTYPE != "Public"):`), this action returns 0, meaning it passed the ownership test (it can be developed as far as the ownership issue is concerned). If it is publicly owned, then a random integer from 1 to 100 is created (`randomN = Random.uniform.nextIntFromTo(1, 100)`). The random number is compared with the value of `self.model.PublicLandDevProbability`, which is also a number from 1 to 100 indicating the probability for a publicly owned parcel to be developed; the smaller this value is, the less chance the random number will be below it (`if (randomN <= self.model.PublicLandDevProbability):`). If the random number (`randomN`) is greater than `PublicLandDevProbability`, the action returns -1, meaning the parcel failed to pass the probability test for public land to be developed (because it falls outside the probability range) and therefore cannot be developed, otherwise it returns 0 (meaning it may be developed).

If it is assumed publicly owned parcels are rarely developed, the value of `PublicLandDevProbability` can be set to be very small. As a result of adding the randomness to this action, there is a chance that a publicly owned parcel can be developed. How much chance will be determined in the next step.

- 7 Click OK to save your work and close the Actions Editor.

You will now adjust the `PublicLandDevProbability` value.

- 8 In the Environment panel, click Urban Growth Model. In the Property panel, click the Edit button to the right of the Fields property to open the Fields Editor.
- 9 Select the `PublicLandDevProbability` field, highlighted in the following figure. You can change the default value based on your current situation. Currently, the value is set to 5, meaning there is a 5 percent chance that a publicly owned parcel could be developed.



10 Close the Fields Editor.

Add probability to the large parcel's development

The size of a parcel is another factor that the parcel agent can consider when deciding whether it would develop during a time step.

- 11** In the Environment panel, click Parcel. In the Property panel, click the Edit button to the right of the Actions property to open the Actions Editor.
- 12** In the Actions drop-down list, select step. Comment out the line with the return statement and uncomment the five subsequent lines of code in the Source panel so that it looks like the code in the following image. The lines of code highlighted by the red arrow add randomness to the parcel size criteria. As a result, even though a parcel may be larger than the maximum size for development, there is a chance it still will be developed.

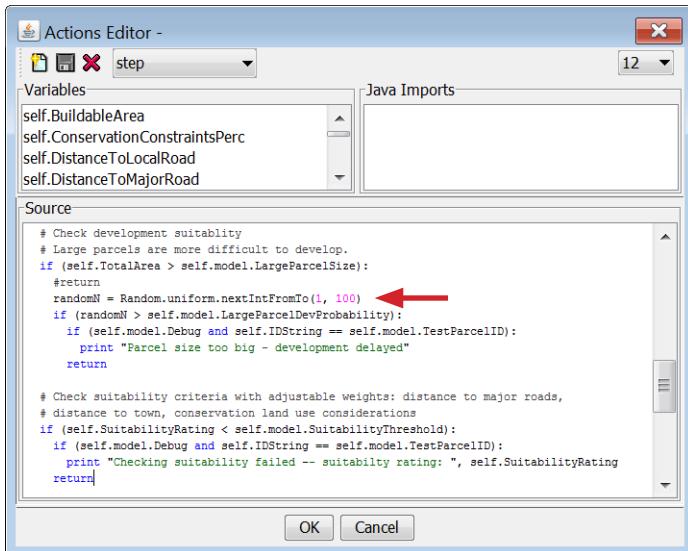
5

6

7

8

9



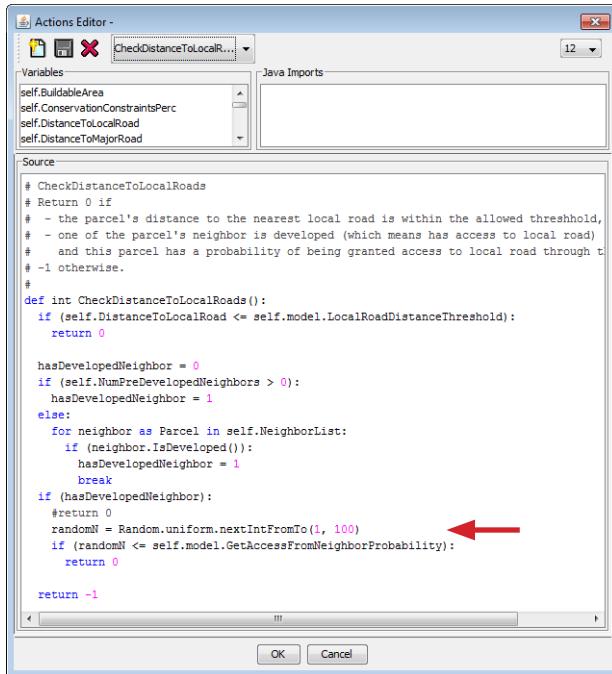
In the code, parcels that are very large are more difficult to develop (`if (self.TotalArea > self.model.LargeParcelSize) :`). If the parcel size (`self.TotalArea`) exceeds a specified area defining the size of a parcel that is too large to develop (`self.model.LargeParcelSize`), a random number is generated (`randomN = Random.uniform.nextIntFromTo(1, 100)`). The random number (`randomN`) is compared with the preset probability value for large parcel development (`if (randomN > self.model.LargeParcelDevProbability) :`). This value (`LargeParcelSize`) can be adjusted in the urban growth model's Fields Editor.

Add additional probability to the model in the `CheckDistanceToLocalRoads` action

The model assumes that as long as a parcel has a developed neighbor, it will obtain access to local roads through this neighbor. In a real-world situation, this is not guaranteed to happen (the neighbor may not be friendly and would not grant the access). We want to add the probability calculation in the `CheckDistanceToLocalRoads` action to simulate this situation. Again, you can decide how likely it is that a parcel can get access through a developed neighboring parcel by adjusting the value of `self.model.GetAccessFromNeighborProbability` in the model's Fields Editor, as you did in an earlier step. In the following step, you will add code to the `CheckDistanceToLocalRoads` action to add the probability calculation.

- 13** Select the `CheckDistanceToLocalRoads` action in the Actions Editor.
- 14** In the Source panel, comment out and uncomment the three lines of code indicated by the red arrow so that it looks like the code in the following image. This

code adds randomness to the distance to road criteria. As a result, a parcel agent can be far from a road and still be developed.



```

Actions Editor - CheckDistanceToLocalR...
Variables Java Imports
self.BrowsableArea
self.ConservationConstraintsPerc
self.DistanceToLocalRoad
self.DistanceToMajorRoad
Source
# CheckDistanceToLocalRoads
# Return 0 if
# - the parcel's distance to the nearest local road is within the allowed threshold,
# - one of the parcel's neighbor is developed (which means has access to local road)
# - and this parcel has a probability of being granted access to local road through t
# -1 otherwise.
#
def int CheckDistanceToLocalRoads():
    if (self.DistanceToLocalRoad <= self.model.LocalRoadDistanceThreshold):
        return 0

    hasDevelopedNeighbor = 0
    if (self.NumFreeDevelopedNeighbors > 0):
        hasDevelopedNeighbor = 1
    else:
        for neighbor as Parcel in self.NeighborList:
            if (neighbor.IsDeveloped()):
                hasDevelopedNeighbor = 1
                break
    if (hasDevelopedNeighbor):
        #return 0
        randomN = Random.uniform.nextIntFromTo(1, 100)
        if (randomN <= self.model.GetAccessFromNeighborProbability):
            return 0

    return -1

```

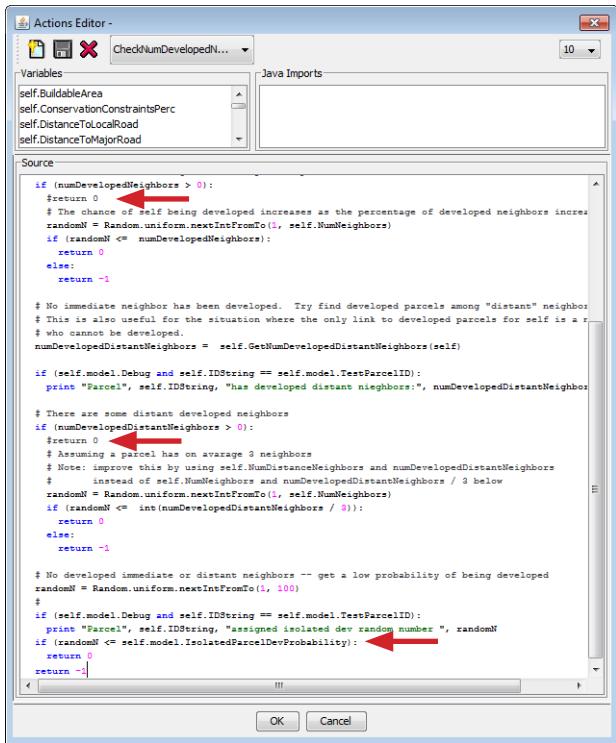
The screenshot shows the 'Actions Editor' dialog box with the title 'CheckDistanceToLocalR...'. The 'Source' tab contains the provided Python code. A red arrow points to the line `randomN = Random.uniform.nextIntFromTo(1, 100)`.

In this code, a random number from 1 to 100 is generated (`randomN = Random.uniform.nextIntFromTo(1, 100)`). This number is compared with a preset value for the probability of the developed neighbor's willingness to grant access to the parcel agent (`self.model.GetAccessFromNeighborProbability`). If the random number falls within the preset probability range, then the action returns 0, meaning the parcel can get access to local roads.

Add the probability calculations to the CheckNumDevelopedNeighbors action

- 15** Select CheckNumDevelopedNeighbors from the Actions drop-down list.
- 16** Comment out and uncomment the sections of code pointed to by the three red arrows until the code appears the same as in the following figure. The code highlighted by the red arrows adds randomness to the developed neighbors criteria.

5
6
7
8
9



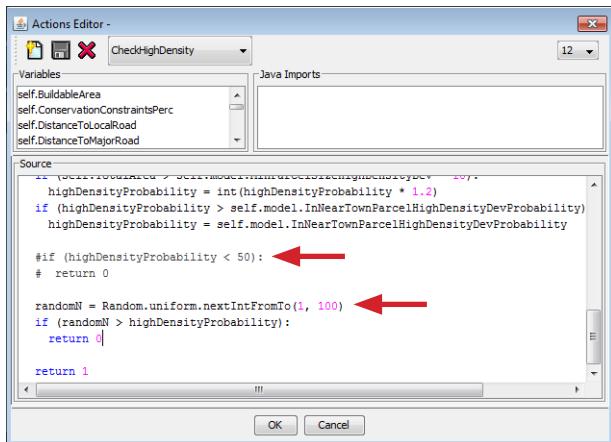
Random numbers are used in more than one place in this action. At the top arrow, one random number is used to give a development probability based on the percentage of immediate neighbors that have been developed (`randomN = Random.uniform.nextIntFromTo(1, self.NumNeighbors)`). Then, `randomN` is compared to the number of developed neighbors (`numDevelopedNeighbors`). Notice that in the code, the larger `numDevelopedNeighbors` is, the more likely the random number is to fall below it. If the random number is below `numDevelopedNeighbors`, the action returns a 0, which means the parcel can be considered to be developed.

At the second arrow, another random number is used in a similar way for determining whether the parcel will be developed based on how many of its second-degree neighbors (neighboring parcels of its immediate neighbors) have been developed. Here a random number is compared with the number of this parcel's distant neighbors that are developed (`if (randomN <= int(numDevelopedDistantNeighbors / 3)):`).

At the third arrow, a third random number is used when a parcel is isolated from all nearby development (`if (randomN <= self.model.IsolatedParcelDevProbability):`). In this case, no neighboring parcel or any second-degree neighboring parcel has been developed (for the “leap frog” pattern of development). `IsolatedParcelDevProbability` has a low probability value preset in the model, which, again, you can adjust.

Add code for the probability calculations for identifying whether the development will be normal or higher density

- 17** In the Actions Editor, select CheckHighDensity from the Actions drop-down list. In the Source panel, comment out and uncomment the lines of code by the red arrows so that it looks like the code in the following figure. The code highlighted by the second red arrow adds randomness to whether a parcel is to be developed with normal or higher density.



The purpose of adding probability to the CheckHighDensity action is to simulate the situation where a parcel may or may not develop with higher density even though its conditions qualify it for higher density development. After checking whether it meets the higher density development criteria, the parcel decides whether it will indeed develop with higher density by comparing a random number against the higher density probability value (`if (randomN > highDensityProbability) :`). The `highDensityProbability` value is based on where the parcel is located (see the previous exercise) and is set just prior to the code highlighted by the upper arrow. If the random number is greater than `highDensityProbability`, the parcel agent will not be developed with higher density; however, it will be developed.

There are other places where you may want to introduce a probability calculation as well—for example, when the development suitability is being checked. Even if a parcel is suitable for development, it may not get developed.

- 18** Click OK to save your changes and close the Actions Editor.
- 19** In the Agent Analyst window, click the Compile button to make sure the model compiles after your changes to the code. If the code compiles, you will not see any error message; otherwise, you probably mistyped some input, and you need to follow the compiler's instruction to fix the error.
- 20** Close the model and close ArcMap. Do not save any changes.

5
6
7
8
9

Based on many factors, the process of a parcel's development status change is not deterministic. For example, the owner may be short of funds and decide to develop his parcel with low density even though his parcel qualifies for higher-density development, and this is one of the factors whose value the model could not predict. To capture this stochasticity, or uncertainty, in this exercise, you added probability through random numbers in the decision making of the parcel agents.

In the next exercise, you will make certain conditions dynamic—that is, they are able to change with the decision making of the parcel agents—enabling you to more accurately capture the potential urban growth dynamics.

Initializing and keeping track of parcel agents' permanent and dynamic conditions

In the previous exercise, you determined that whether a parcel might develop in the current year depends on the evaluation of multiple factors related to a parcel's condition. Many of the parcel's conditions are treated as permanent (at least throughout the simulation period), such as the physical constraints on the parcel, which parcels are neighbors to a parcel, and so on. Therefore, the evaluation of these conditions can be a one-time task, during the initialization of the parcel agent at the beginning of the simulation run. Other parcel conditions may change over time, such as its development status, whether it has a high percentage of developed neighbors, whether it is in or near a town (since the town's boundary can expand over time), and so on. In this exercise, you will model permanent and dynamic conditions of the parcels.

The various properties and conditions of a parcel agent are maintained in the parcel's fields. You will see that dynamic properties and conditions are updated dynamically during each time step, and permanent properties and conditions are determined once at the beginning of the simulation.

Exercise 7e

Load the ArcMap document

- 1 Start ArcMap. Navigate to C:\ESRIPress\AgentAnalyst\Chapter07. Open ex07e.mxd.

In the ArcMap display you will see the same layout you have seen earlier. In ArcToolbox, the Chapter07Ex07e toolbox has been created for you.

Load the Agent Analyst model

- 2 Right-click the Chapter07Ex07e toolbox, point to New, and select Agent Analyst Tool.
- 3 In the Agent Analyst window, click File and select Import. Do not save the changes when prompted.
- 4 Navigate to C:\ESRIPress\AgentAnalyst\Chapter07\Models and select ex07e.sbp.

Note: If you receive an error, see the note after step 4 of exercise 7a.

Prepare the model to create dynamic properties

- 5 In the Environment panel, click Parcel. In the Property panel, click the Edit button to the right of the Fields property to open the Fields Editor.

5

6

7

8

9

In the Fields Editor, you will see the fields for storing the parcel's properties and conditions. These fields are listed in the data dictionary for parcel agents at the end of the chapter. Take a moment to review the use of these fields. Also, notice in the Fields Editor that some of the fields are specified as accessible. These fields will be accessible by the urban growth model.

During the model run, these fields are consulted for their values and, for those that represent the dynamic conditions of the parcel agent, have their values updated. In general, all the fields are first initialized when the simulation starts. You will learn where and how these fields are initialized and where, when, and how they are updated during the model run.

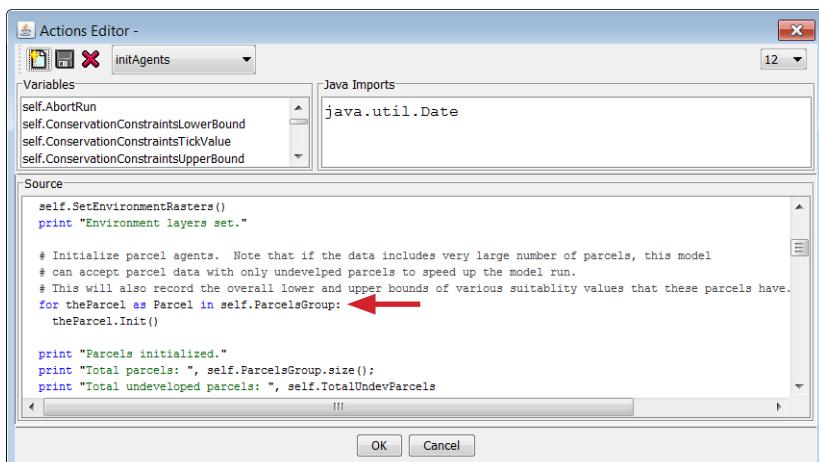
Notice that some of the parcel agents' properties are not included in this list of fields; rather, they are stored in the shapefile for the parcel agents. Remember that in exercise 7a, you learned how to specify a shapefile as the data source for parcel agents and saw that the shapefile contains the following properties about the parcels: development status, ownership, and total area. These properties are referred to, and some of them are updated, during the model run.

- 6 Close the Fields Editor if it is still open.

The following steps illustrate how the properties defined in the Fields Editor for parcel agents are initialized and updated.

View the initAgents action for the urban growth model

- 7 In the Environment panel, click Urban Growth Model. In the Property panel, click the Edit button to the right of the Actions property to open the Actions Editor.
- 8 In the Actions drop-down list, select initAgents. Examine the following code in the Source panel. The code highlighted by the red arrow calls the initialization action for setting the starting values for a parcel agent's fields.

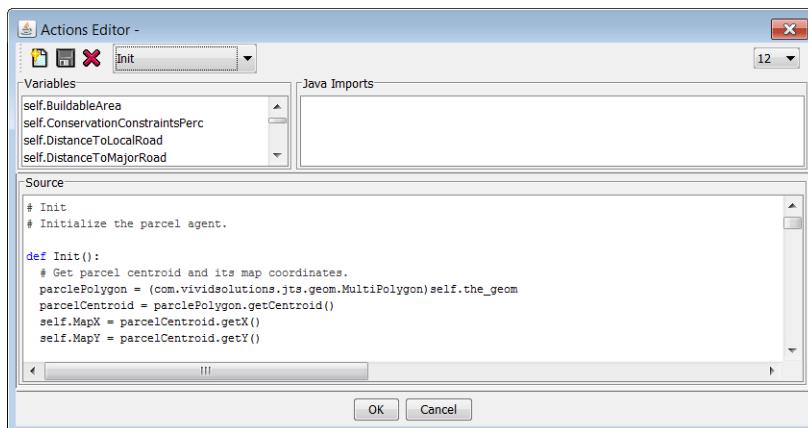


The model has access to the parcel agent group called `ParcelsGroup`. It can also access all the members in `ParcelsGroup` by using the “for ... in ...” coding syntax (`for theParcel as Parcel in self.ParcelsGroup:`). The model is iterating through all the parcel agents in `ParcelsGroup`, and for each parcel agent calls the `Init` action.

9 Close the Actions Editor for the urban growth model.

View the `Init` action for the parcel agents

- 10 In the Environment panel, click `Parcel`. In the Property panel, click the `Edit` button to the right of the `Actions` property to open the Actions Editor.
- 11 From the Actions drop-down list, select `Init`. The action sets the initial values for the fields for a parcel agent.



Examine the code and notice how a parcel agent initializes itself by setting the initial values for its properties defined in the Fields Editor. Note that many of these properties need to be calculated by accessing some environmental data layers. For example, the value to assign the parcel's `PhysicalConstraintsArea` field (the area of this parcel agent that is subject to physical environmental constraints) is retrieved from a raster data layer. The parcel agent first retrieves from the model the raster containing the hard constraints (`raster = self.model.getRaster(self.model.Raster_PhysicalConstraintsArea)`). Then the parcel agent finds the cell position of its centroid in the raster (`point = raster.mapToPixel(self.MapX, self.MapY, 0)`), retrieves from the raster its physical constraints area value from the cell (`val = raster.getValue(int(point.x), int(point.y), 0)`), and converts the value to the integer value it needs (`self.PhysicalConstraintsArea = int(val)`).

The parcel then calculates the percentage of this physically constrained area (`PhysicalConstraintsPerc`) based on `PhysicalConstraintsArea` (`self.PhysicalConstraintsPerc = self.PhysicalConstraintsArea * 100 / self.TotalArea`). It then calculates its buildable area based on

5

6

7

8

9

PhysicalConstraintsArea (self.BuildableArea = self.TotalArea - self.PhysicalConstraintsArea). It also determines, based on its buildable area, whether it is unbuildable (self.Unbuildable = self.IsUnbuildable()).

Similarly, the parcel agent retrieves the values for other fields from the relevant raster data layers: DistanceToLocalRoad from raster self.model.Raster_LocalRoadDistance, DistanceToMajorRoad from raster self.model.Raster_MajorRoadDistance, DistanceToTown from raster self.model.Raster_DistanceToUrban, ConservationConstraintsPerc (percentage of its area, which may be restricted for development due to conservation considerations, such as proposed wildlife corridors) from raster self.model.Raster_ConservationConstraintsPerc. It then further initializes the related fields.

The parcel agent also initializes its NeighborList field. The value will be filled in by the model after the Init action returns.

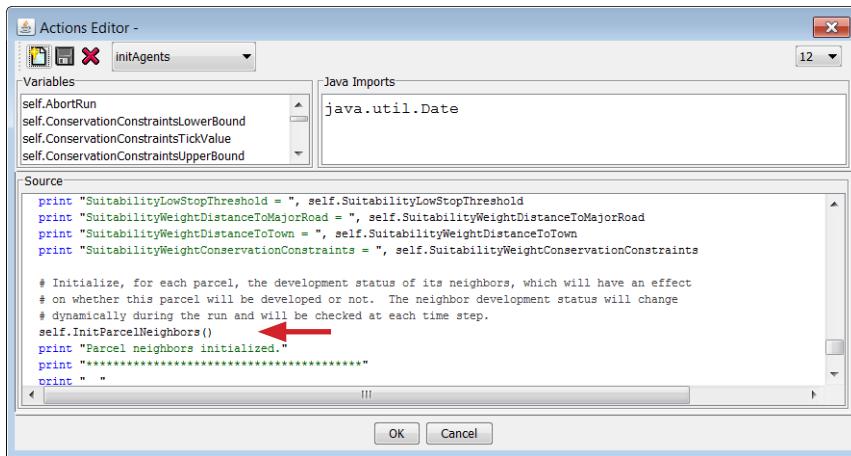
In addition to initializing the values of its various fields, the parcel also needs to report to the model certain information about itself that the model would need to know. For example, the model uses fields to keep track of the total number of undeveloped parcels (model.TotalUndevParcels), and the total number of unbuildable parcels (model.TotalUnbuildableParcels). To initialize these model fields, after initializing its development status, the parcel agent reports to the model if it is not developed by incrementing the count of undeveloped parcels (self.model.TotalUndevParcels = self.model.TotalUndevParcels + 1). Similarly, the parcel agent reports to the model if it has determined that it is not buildable (self.model.TotalUnbuildableParcels = self.model.TotalUnbuildableParcels + 1).

The parcel agents also help the model initialize threshold variables used for checking the parcels' development suitability in future steps. This is done by calling the action RecordSuitabilityValueBounds. The details of this segment of code are discussed in the next exercise.

12 Close the parcel agent's Actions Editor.

Return to the model's initAgents action to continue examining how the information about the parcels is initialized and used

13 In the Environment panel, click Urban Growth Model. In the Property panel, click the Edit button to the right of the Actions property to open the Actions Editor. You will see the model's initAgents action in the Source panel. Scroll down and examine how the information about the parcels' neighbors is initialized. You should scroll down until the Source panel appears the same as the following figure. The code highlighted by the red arrow calls the InitParcelNeighbors action, which creates a list of the undeveloped parcels and the number of developed parcels surrounding each undeveloped parcel agent.

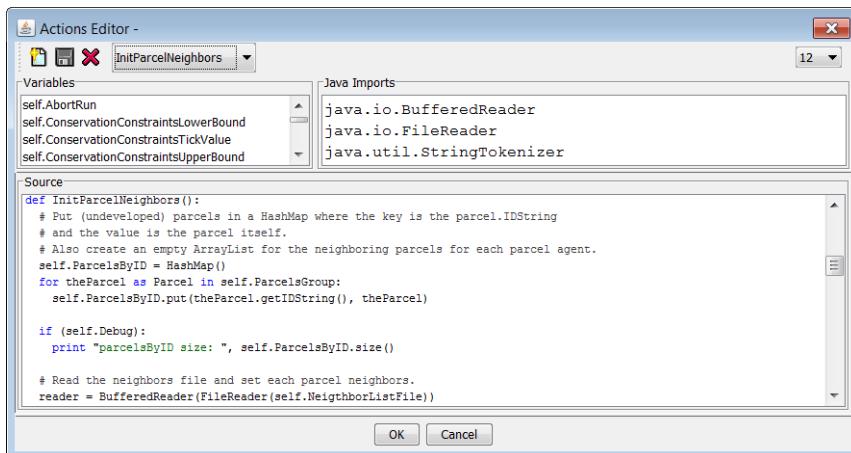


For a parcel agent, the development status of its neighboring parcels is a factor when it decides whether it should become developed. The number of developed parcels surrounding any particular parcel agent can also change during any time step.

For each parcel agent, the model calls an action to initialize the information about its neighbors (`self.InitParcelNeighbors()`).

Examine the neighbors initialization action

- 14** In the Actions Editor, select `InitParcelNeighbors` from the Actions drop-down list. The code for the action is displayed in the Source panel. The action identifies for each undeveloped parcel agent a list of surrounding parcels that are undeveloped and the number of developed parcels.



5
6
7
8
9

In this action, the model stores in each (undeveloped) parcel agent the following information about its neighbors to be used later:

- A list of neighboring undeveloped parcel agents
- The number of developed neighboring parcels (currently, they are not agents in this model for performance considerations)
- The total number of neighbors, including developed ones

For performance considerations, you are only storing, for each parcel agent, the neighboring parcels that are undeveloped at the start of the simulation. You also retain the number of neighboring parcels that have been developed before the simulation starts.

This action reads the parcel neighbor information from an input, tab-delimited text file. The path to this file is specified by the model's NeighborListFile field, which you can specify in the model's Fields Editor or modify at run time in the Urban Growth Model Settings window, on the Parameters tab. In this file, each line contains the following format:

LeftParcelID, RightParcelID

where ParcelIDs are the ID string in the IDString field of the parcel agents. The LeftParcelID is the ID of a parcel agent, the RightParcelID is the ID of a neighboring parcel.

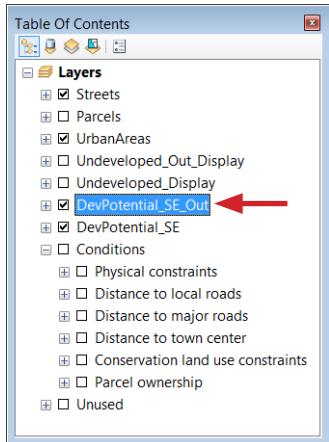
Each neighbor is represented by one line.

As mentioned earlier, some of the parcel agents' fields, after being initialized, will stay the same during the entire simulation. Some will change over time—the development status and the number of neighbors already developed. Some of the dynamic changes happen during parcel agents' actions—the development status change of the parcel happens during the parcel agent's step action. Some have to be calculated in a model action that is scheduled to execute at the end of a time step—the total number of developed neighbors for a parcel.

- 15 If the Actions Editor is still open, click OK to close it.

You will run this sample model with the current setting. Before running the model, you need to turn on the output shapefile layer in ArcMap.

- 16 In ArcMap, turn on the layer DevPotential_SE_Out if it is not already displayed.



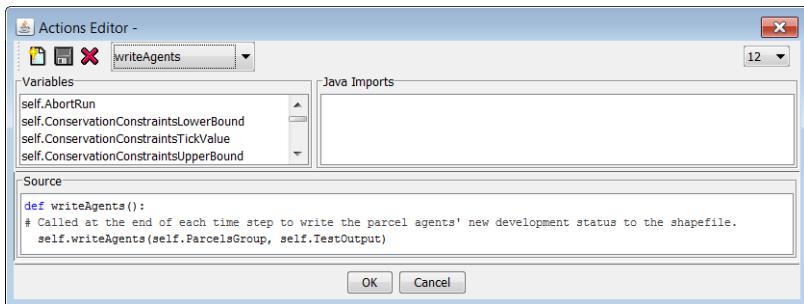
5
6
7
8
9

By default, the model's simulation results (in our case, the parcels' development status change) will be written to the data source shapefile (DevPotential_SE). But in this exercise, you will direct the output to a different shapefile, DevPotential_SE_Out, which starts out as an exact copy of DevPotential_SE the first time you run the simulation. Its content changes at each time step, reflecting the new development status for each parcel at the end of each time step. This way, the data source shapefile will not be overwritten every time you run the model, which is necessary since you will complete more than one test run. At the end of a simulation run, the output shapefile will contain the simulation results, which remain until the simulation runs again and overwrites the file during model initialization.

This output redirection is done in the model's writeAgents action, as shown in the following image.

View the code to write the parcel status for each time step to the shapefile

- 17** In the Environment panel, click Urban Growth Model. In the Property panel, click the Edit button to the right of the Actions property to open the Actions Editor.
- 18** In the Actions Editor, select the writeAgents action from the Actions drop-down list. View the code in the Source panel. The action writes the current status of the parcel agents to the specified output shapefile.



At the end of each time step this action specifies what information to write (the parcel agents) and where to write it (`self.TestOutput`). The value for `TestOutput` is specified on the Parameters tab of the Urban Growth Model Settings window.

- 19** Close the model's Actions Editor.

Run the model

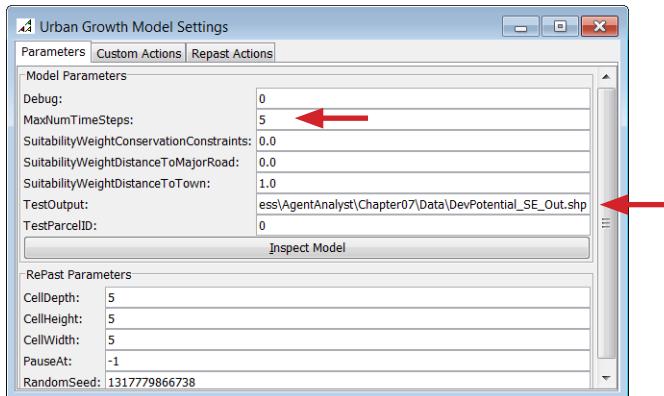
You will run the model with its default parameter setting (the values that have been set in the Fields Editor). To speed up the test run, the large parcel dataset has been replaced with the smaller one. You will also change, for the time being, the number of time steps from 20 to 5.

- 20** In the Agent Analyst window, click the Run button.

The Repast toolbar and the Urban Growth Model Settings window appear. The window controls the user-defined model parameters.

- 21** On the Parameters tab of the Urban Growth Model Settings window, modify the following:

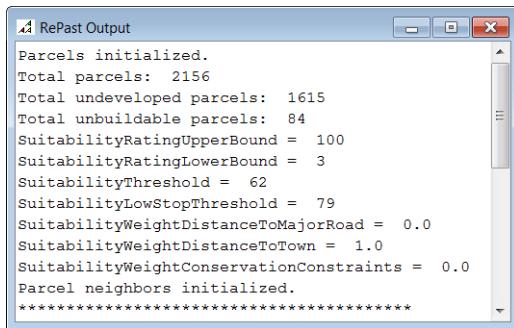
- Change the parameter for the maximum number of time steps, `MaxNumTimeSteps`, to 5, if it is not already.
- Change the parameter for the model output shapefile, `TestOutput`, to `C:\ESRIPress\AgentAnalyst\Chapter07\Data\DevPotential_SE_Out.shp`, if it is not already.



- 22** On the Repast toolbar, click the Step button to initialize the model with the current setting.

The Step button will run one step of the model. The first time it is clicked, the initialization process for the model is performed.

The initialization (execution of initAgents) will take a couple of seconds to minutes depending on your computer. (After initialization, the time steps will go more quickly.) When initialization finishes, you will see the output of the print statement in the model's initAgents action displayed in the RePast Output window, reporting about the number of total parcels, number of undeveloped parcels, and so on. (You may see the information displayed three times.)



- 23** Click the Step button again, and the model executes a time step.

Observe that ArcMap is updated to show the current status of the developed parcels after the time step. Also notice the RePast Output window is updated to show the number of parcels developed during this time step and the total number of parcels developed since the start of the model.

5

6

7

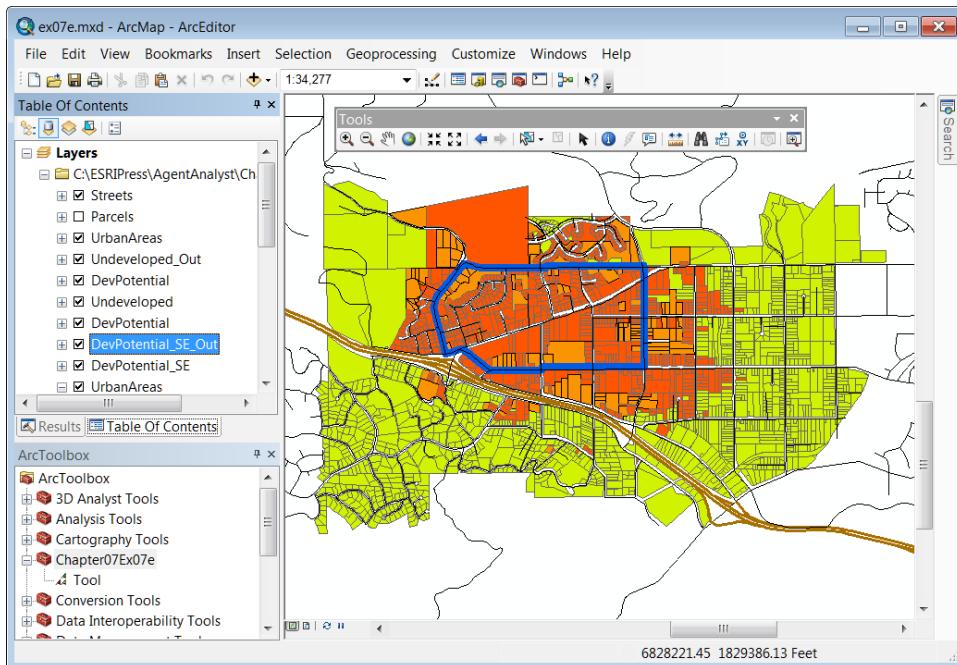
8

9

- 24** Click the Step button four more times.

This brings the total to five time steps, which is the maximum number of steps in the current model setting. The simulation ends. You will see the output in ArcMap similar to the following figure.

Note: Your simulation results will not be exactly like this because of the random factors built into the parcel agents' actions.



- 25** Click the Exit button on the Repast toolbar to exit the model run.

This development scenario favors parcels that are in and closer to town. These parcels are more likely to be developed first.

- 26** Close the model and close ArcMap. Do not save any changes.

Tracking changes at the model level due to the parcel agents' actions

In the previous exercises, you observed interactions between parcel agents and the model, how information is passed between parcel agents and the model, how the model initializes the agents, and how the parcel agents' permanent and dynamic conditions are recorded. In this exercise, you will see interactions between the parcels and the model again, and how certain information stored at the model level is dependent on the parcel agents' conditions or the results of their actions.

5

6

7

8

9

Exercise 7f

Load the ArcMap document

- 1 Start ArcMap. Navigate to C:\ESRIPress\AgentAnalyst\Chapter07. Open ex07f.mxd.

In ArcToolbox, the Chapter07Ex07f toolbox has been created for you.

Load the Agent Analyst model

- 2 Right-click the Chapter07Ex07f toolbox, point to New, and select Agent Analyst Tool.
- 3 In the Agent Analyst window, click File and select Import. Do not save the changes when prompted.
- 4 Navigate to C:\ESRIPress\AgentAnalyst\Chapter07\Models and open ex07f.sbp.

Note: If you receive an error, see the note after step 4 of exercise 7a.

Return to the initAgents action for the urban growth model that you were reviewing in exercise 7e

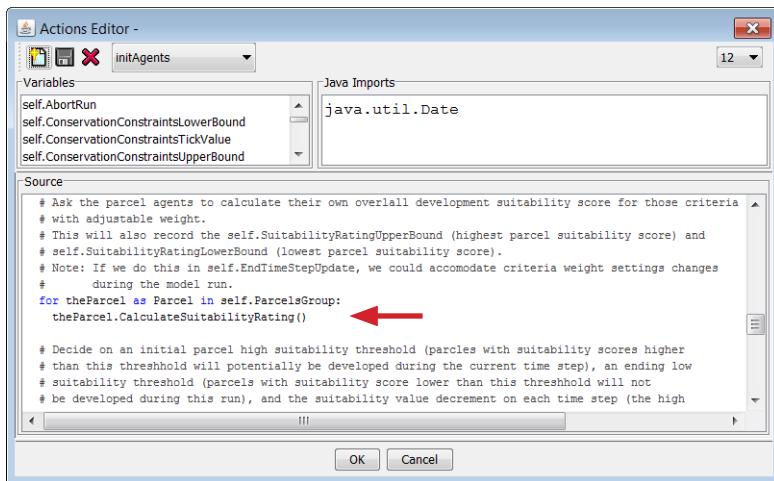
- 5 In the Environment panel, click Urban Growth Model. In the Property panel, click the Edit button to the right of the Actions property to bring up the model's Actions Editor window.

After the model has iterated through all the parcel agents and they have all performed their Init action (discussed in the previous exercise), the model asks the parcels to determine their development suitability ratings. Since there will be more pressure to develop and fewer locations to develop as the city grows, the model then decides on a reduction value by which to relax the suitability ratings thresholds each year. The model uses a placeholder algorithm to simulate the reality that parcels with higher suitability ratings will tend to develop sooner. It records the low and high bounds of the parcels' suitability ratings and

sets up a suitability threshold value for each time step. The suitability thresholds start at a high value and decrease each time step. As a result, the parcels with high suitability ratings will develop in earlier time steps, and those with lower ratings will develop in later time steps. You can replace this placeholder algorithm with a more realistic one, such as for setting up a development target (a maximum number of newly developed parcels) for each time step (based on population growth projections, economic growth trends, zoning regulations, and so on), so that during each time step, the more suitable parcels will develop until this development target is met. But for now, you will take a look at how the model currently works.

The model iterates over the parcels again, this time executing the CalculateSuitabilityRating action to calculate and normalize the parcels' own development suitability ratings.

- 6 Scroll down in the Source panel until you see the code segment calling the parcel agents' CalculateSuitabilityRating action.



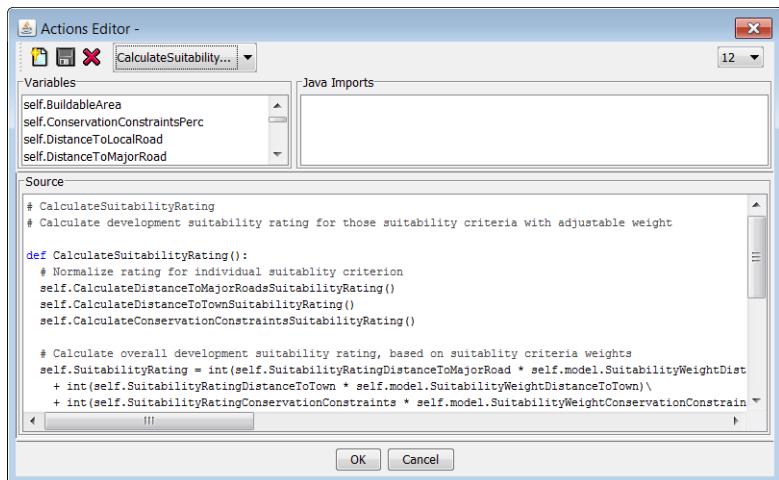
The highlighted code calls CalculateSuitabilityRating to determine the development suitability rating for each parcel.

The development suitability ratings range from 0 to 100. The parcels need to normalize the suitability ratings for each suitability criterion to the 0 to 100 scale by dividing the actual value for the criterion (for example, the actual distance to the nearest road for the distance to major road criterion) by an adjustment value that is based on the upper and lower bounds of the range of the data for the criterion (for example, the longest and shortest distance to the nearest major road among all the parcels). The upper and lower bounds were reported to the model earlier by the parcel agents from within their Init action.

Leave the Actions Editor window open for the remainder of this exercise.

Open the parcel's CalculateSuitabilityRating action

- 7 In the Environment panel, select the Parcel vector agent group. In the Property panel, click the Edit button to the right of the Actions property to open the Actions Editor.
- 8 In the Actions Editor, select CalculateSuitabilityRating from the Actions drop-down list. Uncomment the code in the Source panel so that it looks like the code in the following image. This action calculates the suitability rating on a 0 to 100 scale for each parcel agent for each suitability criterion.



In this action, the parcel calculates its own development suitability rating based on the three suitability criteria with adjustable weights: distance to major roads (`self.SuitabilityRatingDistanceToMajorRoad`), distance to town and village centers (`self.SuitabilityRatingDistanceToTown`), and percentage of area related to conservation land use considerations (`self.SuitabilityRatingConservationConstraints`). This action calls three subactions to calculate and normalize the parcel's suitability ratings in terms of each suitability criterion (`self.CalculateDistanceToMajorRoadsSuitabilityRating`, `self.CalculateDistanceToTownSuitabilityRating`, `self.CalculateConservationConstraintsSuitabilityRating`). The `CalculateSuitabilityRating` action then calculates a combined development suitability rating (`self.SuitabilityRating`) for the parcel agent based on the individual suitability criterion ratings (returned by the three actions cited above), as well as the weights on these suitability criteria. The values for these weights (`self.model.SuitabilityWeightDistanceToMajorRoad`, `self.model.SuitabilityWeightDistanceToTown`, `self.model.SuitabilityWeightConservationConstraints`) are stored in fields in the model and can be set in the Urban Growth Model Settings window during run time. You will explore these weights in exercise 7g.

5

6

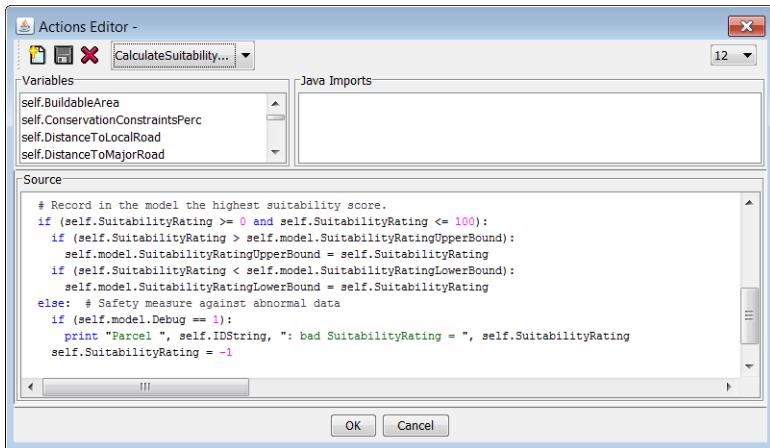
7

8

9

After calculating its development suitability rating, the parcel reports the related information to the model. The model tracks the upper and lower bounds for the parcel suitability ratings.

- 9 In the Actions Editor, in the same action, scroll down to see the following code:



In this code, the parcel reports to the model its suitability rating. If the rating for the parcel agent being processed is lower than the current low bound value or higher than the current high bound value, then a new low or high bound is set. The model will use the high and low bound values to calculate a development suitability threshold for each time step.

- 10 Return to the model's initAgents action. If the Actions Editor is not open, then in the Environment panel, click Urban Growth Model, and click the Edit button to the right of the Actions property. Scroll down the initAgents action until you see the parcels' CalculateSuitabilityRating action.

After the parcel agent finishes calculating its own development suitability ratings (and having reported to the model the low and high bounds of these ratings), the model decides on a suitability threshold value (`self.SuitabilityThreshold`) for the current time step. The undeveloped parcel agents that have a suitability rating greater than the threshold have a chance of being developed. As discussed earlier, the suitability threshold lowers each time step, allowing the less suitable parcels to become more desirable to develop as the more desirable undeveloped parcel agents get developed. The amount to lower the suitability threshold each time step is controlled by multiplying the suitability rating by a decrement factor (`self.SuitabilityDecrement`). The model also determines an absolute lower suitability threshold (`self.SuitabilityLowStopThreshold`) under which the parcel agent will not be developed under any circumstances.

In addition to reporting to the model information related to suitability scores, the parcel also reports its change in development status. You have reviewed how this reporting is done (in the parcel's step action) in some detail in exercise 7c. At the end of the parcel's step

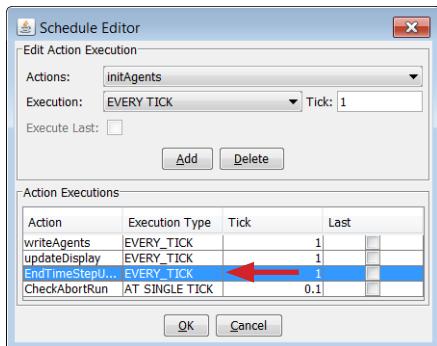
action, the parcel calls the model's IncrementNewDevCount action, which increments the model's total count of parcels developed during the current time step.

11 Close the parcel's Actions Editor.

You will now see how the model processes the information it received from the parcels during a time step.

Open the Schedule Editor

12 In the Environment panel, select Urban Growth Model. In the Property panel, click the Edit button to the right of the Schedule property to open the Schedule Editor.



Notice that the action EndTimeStepUpdate is scheduled for every tick (time step).

13 Close the Schedule Editor.

The model uses the action EndTimeStepUpdate to perform its year-end “review and planning.” The model performs the following types of tasks:

- Updates record keeping. For example, the model keeps track of the total number of newly developed parcels during the year, records the total number of newly developed parcels since the start of the simulation, and updates the total number of developed parcels surrounding each parcel agent (accomplished in the GetNumDevelopedNeighbors action).

Note: The GetNumDevelopedNeighbors action must be called at the end of a time step through the EndTimeStepUpdate action because all parcel agents must be processed before determining their current development status.

- Adjusts various parameters and threshold values that will be used in subsequent time steps. This is the time when the model decides on a new development suitability threshold value (`SuitabilityThreshold`) for each factor, above which a parcel agent has a chance of being developed.

5

6

7

8

9

Note: Not all thresholds are adjusted each time step. For example, this model adjusts the suitability threshold for InOrNearTownThreshold every five time steps to account for the expanding urban boundary over time.

- Produce a year-end report. See the print statements in this action. They produce the content that you see in the RePast Output window. You also can format the information you want to report in different ways, write it out to different destinations, and so on.
- 14** Close any Actions Editor windows that are open.
- 15** Close the model and close ArcMap. Do not save any changes.

In this exercise, you have seen how the model keeps track of various parameters that reflect the overall state of the development of the parcels. You have also explored how the weights of some of the development suitability criteria are initialized and used during the process. In the next exercise, you will see how these weights can be adjusted.

Building scenarios using different weights on suitability criteria

The previous exercise showed that you can adjust the weights on some development suitability criteria at run time. In this exercise, you will take a look at how you can adjust these weights as well as some other parameters to create different urban growth scenarios.

Exercise 7g

Load the ArcMap document

- 1 Start ArcMap. Navigate to C:\ESRIPress\AgentAnalyst\Chapter07. Open ex07g.mxd.

In the ArcMap display you will see the same layout as in previous exercises. In ArcToolbox, the Chapter07Ex07g toolbox has been created for you.

Load the Agent Analyst model

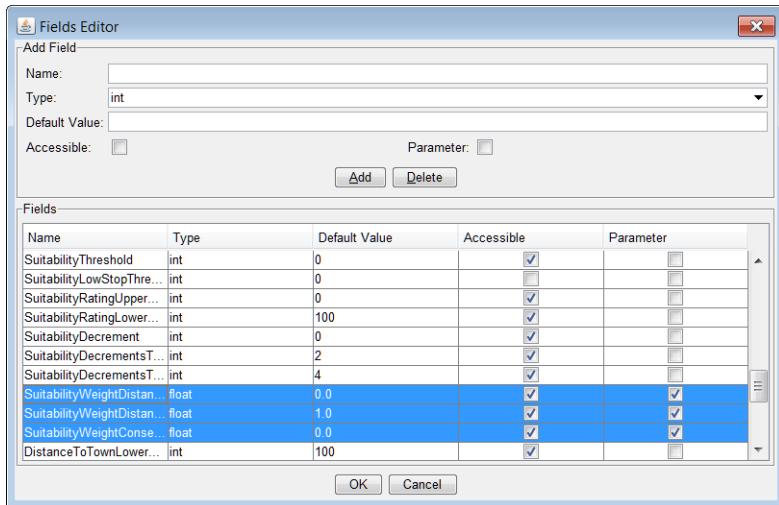
- 2 Right-click the Chapter07Ex07g toolbox, point to New, and select Agent Analyst Tool.
- 3 In the Agent Analyst window, click File and select Import. Do not save the changes when prompted.
- 4 Navigate to C:\ESRIPress\AgentAnalyst\Chapter07\Models and open ex07g.sbp.

Note: If you receive an error, see the note after step 4 of exercise 7a.

Explore the fields in the model

- 5 In the Environment panel, click Urban Growth Model. In the Property panel, click the Edit button to the right of the Fields property to open the Fields Editor. Scroll down the Fields list until you see the suitability criteria fields (highlighted in the following figure).

5
6
7
8
9



As previously mentioned, the user can adjust three weights in this model for the following development suitability criteria:

- Distance to major roads (SuitabilityWeightDistanceToMajorRoad)
- Distance to town and village centers (SuitabilityWeightDistanceToTown)
- Conservation land-use constraint areas (SuitabilityWeightConservationConstraints)

The weight on a development suitability criterion represents your decision on how important a suitability criterion is during urban growth. The values of these weights range from 0 to 1.0, and the sum of the three weights should equal 1.0. Currently, the weight for distance to town and village centers is set to 1.0, and the weights for the other two criteria are set to 0. This constitutes a development scenario that favors the distance to town and village centers criterion.

Notice that these suitability criterion weights are checked in the Parameter column, meaning that they will be exposed at run time in the Urban Growth Model Settings window and can be adjusted by the user before each model run (simulation).

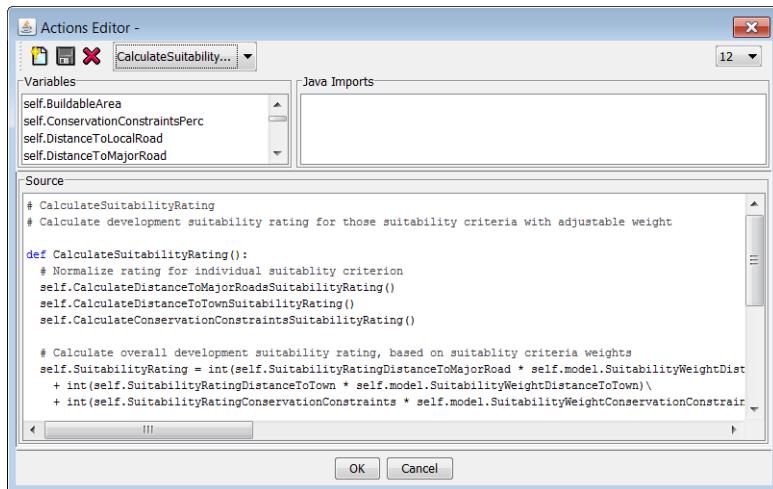
6 Close the Fields Editor.

Revisit the code where these suitability criterion weights are being used

- 7 In the Environment panel, click Parcel. In the Property panel, click the Edit button to the right of the Actions property to open the Actions Editor.
- 8 In the Actions drop-down list, select the CalculateSuitabilityRating action.

Exercise 7f demonstrated that the parcel calculates its own suitability score in this action. After it calculates its score for individual suitability criteria, it calculates a combined suitability score by multiplying the individual criteria scores by the appropriate weights and adding the adjusted scores together.

This action combines the scores for all the suitability criteria to determine the overall development suitability of a parcel agent.



The parcel agent uses the suitability criterion weights identified in the Urban Growth Model Settings window (that you set in the model's Fields Editor).

By varying the weights on different development suitability criteria (these must add up to 1.0, or 100 percent), you create various development scenarios. Changing weights alters the overall suitability scores of the parcels, and different parcels will be selected for development, thus different development patterns emerge.

9 Close the Actions Editor.

Run the model

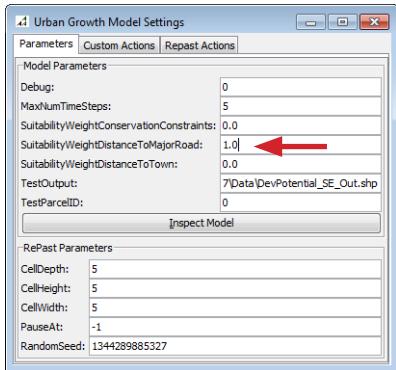
You will run the model with a high weight on the distance to major road criterion (being closer to a highway or freeway will make a parcel more suitable for development).

10 Click the Run button on the Agent Analyst toolbar.

The Repast toolbar and the Urban Growth Model Settings window appear.

11 On the Parameters tab of the Urban Growth Model Settings window, change SuitabilityWeightDistanceToTown to 0.0 and SuitabilityWeightDistanceToMajorRoad to 1.0.

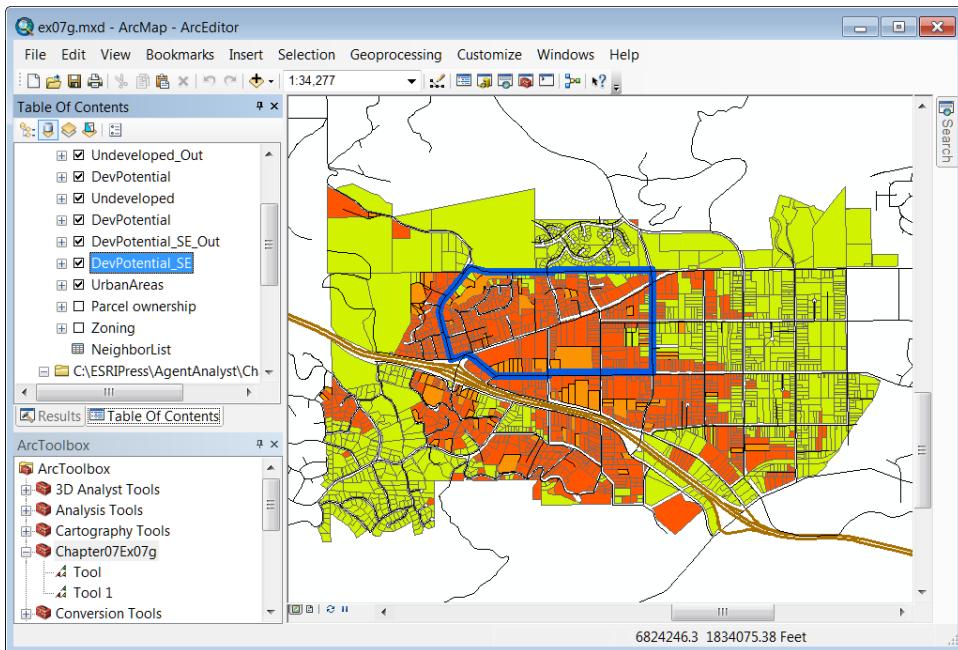
5
6
7
8
9



- 12** Click the Start button on the Repast toolbar.

This initiates the model run and executes it for five time steps without stopping.

At the end of the run, the results will look similar to the following figure. The distance to major roads was weighted as the most important suitability criterion.



Unlike the model run in exercise 7e, which favored parcels that are in or near town, this development scenario favors parcels that are closer to major roads (highways and freeways), as demonstrated in the development pattern in the preceding image. Again, your simulation results will not look exactly the same as this image because of the random factors built into parcel agents' actions.

You can run the model multiple times changing the input parameters to simulate various growth scenarios.

- 13** Close the model and close ArcMap. Do not save any changes.

Optional exercise: Running the model using the full dataset

So far, you have run the model on the small dataset, DevPotential_SE (about 2,000 parcels), representing a portion of the study area. If you want, you can run the model on a larger dataset. You can run the model on the Undeveloped dataset (which has about 12,000 parcels). This dataset contains all the undeveloped parcels in the entire study area (the model also works on just the undeveloped parcels). You can run the model on the complete parcel dataset for the study area, DevPotential (which contains 38,000 parcels), but it takes a long time to run. No matter which input dataset you choose, you will be using the same parcel neighbor information file because it was derived from the large dataset in a preprocessing step.

If you use the larger dataset (Undeveloped), the model initialization can take several minutes. The actual time to run the model (20 time steps) will vary from a few minutes up to a couple of hours depending on your computer.

To get started, open the C:\ESRIPress\AgentAnalyst\Chapter07\ ex07a map document.

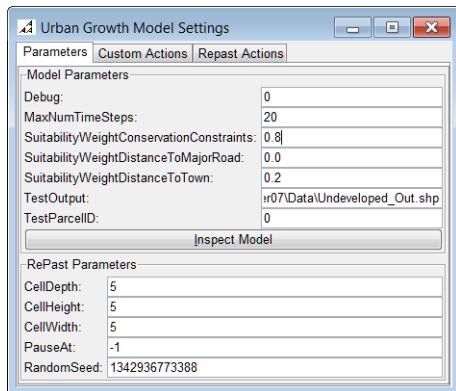
If necessary, set the Table Of Contents to List By Drawing Order. In ArcMap make sure that the check boxes of the two group layers, Undeveloped_Display and Undeveloped_Out_Display, are selected. All the layers under these two group layers should also be selected if they aren't already. Undeveloped_Out is the output layer; DevPotential is beneath the two group layers for display purposes. Right-click Undeveloped and zoom to this layer. You can also clear the check box for UrbanAreas and remove the blue outlines from the ArcMap display.

From the Chapter07Ex07a toolbox, add a new Agent Analyst model and import C:\ESRIPress\AgentAnalyst\Chapter07\Models\ex07a.sbp.

Now you will set the source of the parcel agents to the larger dataset. In the Environment panel, click Parcel and then edit the Data Source property. Click Browse, navigate to the C:\ESRIPress\AgentAnalyst\Chapter07\Data folder if it isn't already selected, and open Undeveloped.shp. Click OK to close the Fields Editor, and then click Run in Agent Analyst.

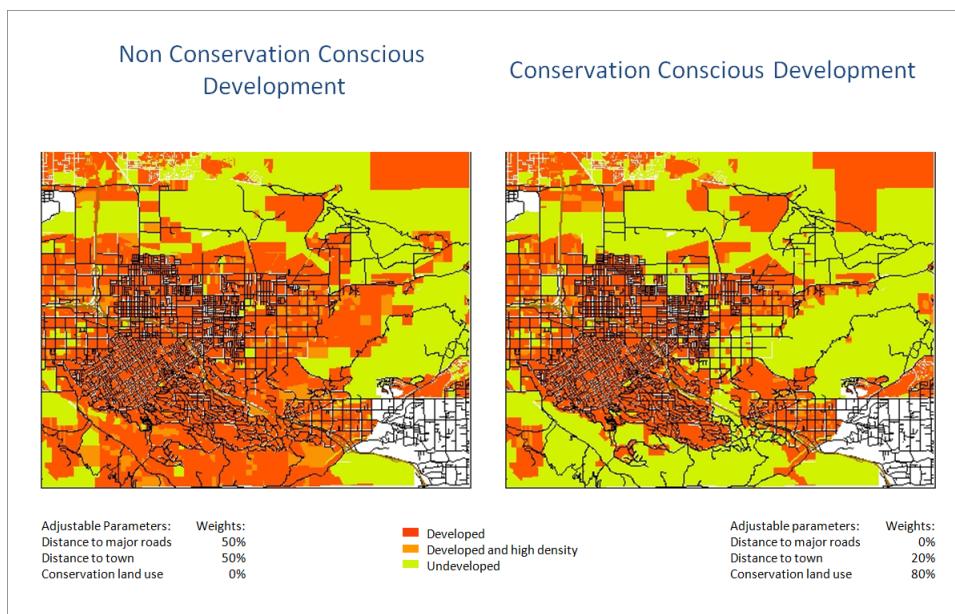
In the Urban Growth Model Settings window, change the output shapefile parameter TestOutput to C:\ESRIPress\AgentAnalyst\Chapter07\Data\ Undeveloped_Out.shp. Change MaxNumTimeSteps to **20**. Change the weights for the three suitability criteria (SuitabilityWeightDistanceToMajorRoad, SuitabilityWeightDistanceToTown, and SuitabilityWeightConservationConstraints) to what you would like, but make sure their sum is 1.0. The Settings window will look similar to the following figure, except for the weights you specify.

5
6
7
8
9



On the Repast toolbar, click the Start button to run the model or the Step button to run the model step by step.

The output of the model will depend on how you set up your suitability criterion weights. Following are the results of two separate runs with two different weight settings.



The results in the left image demonstrate the possible growth patterns when conservation is not prioritized, and in the scenario to the right the conservation criterion was highly weighted.

The urban growth model discussed in this chapter identified how to implement additional modeling techniques that can be applied to polygon agents. The model demonstrated how to add randomness to the model actions and how to expose parameters so that the user can easily

explore different possible growth scenarios. Through an understanding of the interactions of the various suitability criteria, town officials can make more informed decisions when planning for the potential growth in their town.

GIS data, fields, and actions dictionaries

Following is a data dictionary table listing some of the model-level attributes and parcel agents attributes, their data type, and a description. These attributes are selected for their relevance in the discussion of the exercises.

Table 7.2 Data dictionary of GIS datasets

| Dataset | Data type | Description |
|-------------------------|-----------|--|
| DevPotential.shp | polygon | Parcels in the entire study area. All the parcel data used in this sample model is real data but purposefully altered for the exercise. |
| Undeveloped.shp | polygon | Undeveloped parcels in the entire study area |
| Undeveloped_Out.shp | polygon | Same as Undeveloped but with parcels' new development status during the model run |
| DevPotential_SE.shp | polygon | Parcels in the southeast segment of the study area |
| DevPotential_SE_Out.shp | polygon | Same as DevPotential_SE.shp but with parcels' new development status during the model run |
| Streets.shp | polyline | Streets |
| UrbanAreas.shp | raster | Fictitious town/village boundaries |
| hardcareaint | raster | Raster whose cell values indicate a combined physical constraints value on building development, due to steep slope (> 20 percent), stream buffer area, fault line buffer area, flood zones, and so on |
| locrddist_zn | raster | Raster whose cell values indicate the distance between a cell and the nearest local road |
| majrddist_zn | raster | Raster whose cell values indicate the distance between a cell and the nearest major road (highway or freeway) |
| urbandistint | raster | Raster whose cell values indicate the distance between a cell and an urban/village area |
| softcpcent | raster | Raster whose cell values indicate the potential constraints for urban development due to conservation considerations such as wildlife corridors |

5

6

7

8

9

Table 7.3 Fields dictionary for the urban growth model

| Fields | Data type | Description |
|------------------------------------|-----------|--|
| Urban growth model | | |
| MaxNumTimeSteps | integer | The number of time steps specified for a model run |
| TimeStep | integer | The number of the current time step |
| TotalUndevParcels | integer | Total number of undeveloped parcels at a given time |
| TotalUnbuildableParcels | integer | Total number of unbuildable parcels |
| TotalNewDevParcels | integer | Total number of parcels that are newly developed during the entire model run |
| NumNewDevThisTimeStep | integer | Number of newly developed parcels during the current time step |
| NeighborListFile | string | Path to the text file that lists, for each parcel, all its immediate neighboring parcels |
| TestOutput | string | Path to the output parcel shapefile. This file is overwritten at the end of each time step, reflecting the current development state of the parcels. |
| EnvironmentRastersPath | string | The path to the workspace where the environmental data layers are stored |
| Raster_PhysicalConstraintsArea | string | Raster layer for combined physical constraints, with the information of total constraints area for each parcel |
| Raster_LocalRoadDistance | string | Raster layer for distance to local roads |
| Raster_MajorRoadDistance | string | Raster layer for distance to major roads |
| Raster_DistanceToUrban | string | Raster layer for distance to town center |
| Raster_ConservationConstraintsPerc | string | Raster layer for conservation land-use constraints |

Table 7.3 Fields dictionary for the urban growth model (*continued*)

| Fields | Data type | Description |
|---|-----------|--|
| MinSizeBuildingSite | integer | Minimum size of a building site |
| PhysicalConstraintsPercThreshold | integer | Threshold value on the percentage of unbuildable areas on a parcel due to physical constraints. A parcel exceeding this threshold will be too costly to develop, even though it may satisfy the requirement for the minimum size of a building site. |
| PublicLandDevProbability | integer | Probability for public owned parcels to be developed |
| LocalRoadDistanceThreshold | integer | Maximum distance between a parcel and the closest local road for the parcel to be considered as having access to local roads |
| LargeParcelSize | integer | Minimum size of a parcel to be considered large and more difficult to be developed |
| LargeParcelDevProbability | integer | Probability for a large parcel to be developed |
| IsolatedParcelDevProbability | integer | Probability for a parcel to be developed when its neighboring parcels (including distant neighbors) are not developed |
| GetAccessFromNeighborProbability | integer | Probability for a parcel to gain access to a local road through a neighboring parcel |
| MinParcelSizeHighDensityDev | integer | Minimum parcel size requirement for a parcel to undergo higher-density development |
| InNearTownParcelHighDensityDevProbability | integer | Probability of a parcel that is in or near town to undergo higher-density development |
| SuburbanParcelHighDensityDevProbability | integer | Probability of a parcel that is in a suburban area to undergo higher-density development |
| RemoteAreaParcelHighDensityDevProbability | integer | Probability of a parcel that is in a remote area to undergo higher-density development |

5

6

7

8

9

Table 7.3 Fields dictionary for the urban growth model (*continued*)

| Fields | Data type | Description |
|--------------------------------------|-----------|--|
| InOrNearTownThreshold | integer | Maximum distance to town for a parcel to be considered in or near town |
| SuburbanThreshold | integer | Maximum distance to town for a parcel to be considered in a suburban area |
| InNearTownThresholdIncrement | integer | The amount of distance to add to InOrNearTownThreshold periodically during the simulation, as a placeholder for simulating the expansion of town boundaries |
| SuitabilityThreshold | integer | Minimum development suitability score a parcel must have to be developed during the current time step |
| SuitabilityLowStopThreshold | integer | Low threshold of development suitability score. If a parcel's suitability score is lower than this threshold, it will not be developed during the model run. |
| SuitabilityRatingUpperBound | integer | Highest suitability score among all parcels |
| SuitabilityRatingLowerBound | integer | Lowest suitability score among all parcels |
| SuitabilityDecrement | integer | The amount of periodic relaxation on SuitabilityThreshold to allow parcels with lower suitability scores to develop in later time steps |
| SuitabilityWeightDistanceToMajorRoad | integer | Weight of the distance to major road suitability criterion |
| SuitabilityWeightDistanceToTown | integer | Weight of the distance to town suitability criterion |
| SuitabilityWeightConservationCons | integer | Weight of the conservation land-use suitability criterion |
| DistanceToTownLowerBound | integer | Lowest distance to town suitability criterion rating among all parcels |

Table 7.3 Fields dictionary for the urban growth model (*continued*)

| Fields | Data type | Description |
|-----------------------------------|-----------|---|
| DistanceToTownUpperBound | integer | Highest distance to town suitability criterion rating among all parcels |
| DistanceToTownTickCount | integer | The adjustment to use to normalize a parcel's distance to town suitability rating to 1 to 100 scale |
| DistanceToMajorRoadLowerBound | integer | Lowest distance to major road suitability criterion rating among all parcels |
| DistanceToMajorRoadUpperBound | integer | Highest distance to major road suitability criterion rating among all parcels |
| DistanceToMajorRoadTickCount | integer | The adjustment to use to normalize a parcel's distance to major road suitability rating to 1 to 100 scale |
| ConservationConstraintsLowerBound | integer | Lowest conservation land-use suitability criterion rating among all parcels |
| ConservationConstraintsUpperBound | integer | Highest conservation land-use suitability criterion rating among all parcels |
| ConservationConstraintsTickCount | integer | The adjustment to use to normalize a parcel's conservation land-use rating to 1 to 100 scale |
| Debug | integer | Whether debugging is turned on during the model run |
| TestParcel | integer | ID of the parcel that we want to know more detailed information about during debugging |
| AbortRun | integer | Whether to abort the model run because of some predefined error conditions |
| Parcel agent | | |
| MapX | double | Parcel's centroid x coordinate |
| MapY | double | Parcel's centroid y coordinate |

5

6

7

8

9

Table 7.3 Fields dictionary for the urban growth model (*continued*)

| Fields | Data type | Description |
|--|-----------|---|
| PhysicalConstraintsArea | integer | Area within a parcel that has physical constraints for development |
| PhysicalConstraintsPerc | integer | Percentage of a parcel's area that has physical constraints for development |
| BuildableArea | integer | Parcel's buildable area |
| UnBuildable | integer | Whether a parcel is unbuildable or not |
| ConservationConstraintsPerc | integer | Parcel's development constraints in terms of conservation land use |
| DistanceToTown | integer | Distance between the parcel and closest town |
| NearTownStatus | integer | Whether the parcel is in/near town or in suburban area or in remote area |
| DistanceToLocalRoad | integer | Distance between the parcel and closest local road |
| DistanceToMajorRoad | integer | Distance between the parcel and closest major road |
| NeighborList | ArrayList | List of neighboring parcels |
| NumNeighbors | integer | Number of neighboring parcels |
| NumPreDevelopedNeighbors | integer | Number of previously developed neighboring parcels |
| SuitabilityRating | integer | Parcel's development suitability score for those suitability criteria with adjustable weights |
| SuitabilityRatingDistanceToMajorRoad | integer | Parcel's development suitability score for the distance to major road criterion |
| SuitabilityRatingDistanceToTown | integer | Parcel's development suitability score for the distance to town criterion |
| SuitabilityRatingConservationConstraints | integer | Parcel's development suitability score for the conservation land use criterion |

Table 7.4 Actions dictionary for the urban growth model

| Declared actions | Description |
|---------------------------|--|
| Urban growth model | |
| initAgents | Initialize parcel agents |
| InitVars | Initialize model-level variables |
| SetEnvironmentRasters | Set the environment rasters |
| CheckSettings | Check the validity of the model-level parameter settings entered by the user |
| InitParcelNeighbors | Store in parcel agents information about their neighbors |
| IncrementNewDevCount | Called by parcel agents to increment the count of newly developed parcels during this time step |
| EndTimeStepUpdate | Executed at the end of each time step (year). Provide summary report on parcel's development; adjust some threshold values for the next time step (e.g., relax the development suitability score threshold). |
| updateDisplay | Update ArcMap display |
| writeAgents | Save the current state of the parcel agents to output shapefile |
| Parcel agents | |
| Init | Initialize self, record relevant environment conditions |
| step | Evaluate current conditions to decide whether self should change development status |
| IsDeveloped | Indicate whether self is already developed |
| IsUnbuildable | Indicate whether self is unbuildable |
| CheckConstraints | Evaluate whether self is undevelopable based on physical constraints |
| CheckDistanceToLocalRoads | Evaluate whether self has local road access |

5

6

7

8

9

Table 7.4 Actions dictionary for the urban growth model (*continued*)

| Declared actions | Description |
|---|---|
| CheckPublicOwnership | Check self's ownership type and evaluate the development possibility in terms of ownership type |
| CheckHighDensity | Determine the likelihood of self to be developed with higher density |
| CheckNumDevelopedNeighbors | Check whether there are enough neighboring parcels that are already developed |
| GetNumDevelopedNeighbors | Return the number of developed neighbor parcels |
| GetNumDevelopedDistantNeighbors | Return the number of parcels that are neighbors of neighbors and that are developed |
| DetermineNearTownStatus | Calculate self's status in terms of being in/near town, suburban, or remote |
| RecordSuitabilityValueBounds | Record in the model the highest and lowest values that are in the data for those suitability criteria with adjustable weights |
| CalculateSuitabilityRating | Calculate self's development suitability rating |
| CalculateDistanceToMajorRoadsSuitabilityRating | Check development suitability in terms of distance to major roads (highways or freeways) |
| CalculateDistanceToTownSuitabilityRating | Calculate suitability rating in terms of distance to town |
| CalculateConservationConstraintsSuitabilityRating | Calculate suitability rating based on conservation land-use considerations |

Section 3: Advanced techniques for points, polygons, and rasters

Chapter 8

Adding complexity to moving discrete point agents over continuous surfaces

by Kevin M. Johnston, Naicong Li, and Michelle Gudorf

- ◆ Geometric deterrents that influence decision making
- ◆ Adding memory to develop movement with intent
- ◆ Monitoring the status of each agent and using that status for agent assessment
- ◆ Creating probability distributions from the neighboring locations for the multiple criteria to create the decision space for the agent
- ◆ Moving the point agents
- ◆ Overriding the multicriteria decision-making process with momentary events

In chapter 2 you learned how to move a point agent without any real decision rules. In chapter 5 you learned how to weight the preference for the eight cells surrounding a point agent based on specified criteria. In this chapter you will learn how to build on the complexity of the decision-making process using point agents and how to account for multiple criteria when making a decision. You will learn how the energetic state of the point agent can affect what it does. To account for its energetic state, the point agent will appropriately adjust the weights assigned to each criterion relative to one another. Finally, in this chapter you will also see how the agent makes a decision trading off multiple criteria (in this case, determining where to move) and then implements the decision (in this case, makes the move).

As with all chapters, the intent of this chapter is to introduce you to agent-based modeling (ABM) principles, not how to create a cougar model. If the rules are not satisfactory they can be changed.

The modeling scenario

As discussed in chapter 5, during each time step a male cougar agent considers six criteria when making a decision:

- Security
- If it made a kill, remaining with the kill
- Pursuing a female cougar agent (for a male cougar agent)
- Staying within the home range
- Moving with intent toward good habitat/hunting
- Mating

The decision-making process utilizing multiple criteria is a two-step process:

- The model weights the eight cells surrounding the cougar agent for each criterion based on the attributes at the location or the position of the cell relative to a certain direction (e.g., it is closer to a female cougar agent).
- The model weights each criterion relative to one another.

For a full discussion of the background for this cougar model, see chapter 5.

In chapter 5, you established the weighting of the eight surrounding cells for security, remaining with the kill, and pursuing a female cougar agent (for a male cougar agent). In this chapter you will develop the weights of the eight surrounding cells for staying within the home range and moving with intent toward good habitat/hunting. You will learn how to weight each

criterion relative to the others based on the energetic state of the cougar agent during the time step and then combine the weights of the eight surrounding cells for each criterion with the weights of each criterion to make a decision. You will also learn how to override all the criteria to account for a special event—mating.

The energetic state of the cougar agent during each time step can affect the importance of each of the criteria relative to one another for that time step. That is, if the cougar agent is hungry, the model should weight moving with intent toward good habitat/hunting as more influential than other criteria. You will create an energetics model to establish and monitor the state of a cougar agent, and you will see how the energetic state can influence the decision making of the cougar agent. The energetics model can loosely be interpreted as monitoring how hungry the cougar agent is. Therefore, the decision the cougar agent makes each time step will depend on the amount of energy it has at that time step.

5

6

7

8

9

Background information

The cougar gains energy by eating and uses energy by performing all other activity (e.g., walking, stalking, and running). The amount of energy the cougar agent takes in and how much it uses defines its energetic state for a specific time step.

As the energetic state for the cougar agent increases or decreases, so will the weights for each criterion. For example, if the cougar has little energy (is very hungry), it must weight moving intently toward good habitat/hunting more importantly than the other criteria. As energy reserves increase, the cougar agent can increase the weights for other non-life-threatening activities such as looking for females.

The energetics model will do the following:

- Determine the energetic state of the cougar agent by identifying and monitoring how much energy will be gained and lost each time step.
- Establish the relative weights for each criterion each time step. The weights used each time step will depend on the cougar's energetic state.

Energy gained

Kilocalories (kcal) are the currency the cougar agent will use in this energetics model for decision making. This currency is mapped to fitness, which will influence behavior.

A cougar agent gains kilocalories when it is consuming a kill. The type of prey dictates the total kilocalories that are gained from consuming the prey (caloricAmount), how long it will take to eat the kill, and how many kilocalories the agent gains each hour while consuming the prey (hourlyConsumptionAmount). Each of these properties are set to fields associated with the prey agent, as demonstrated in chapter 5, exercise 5c.

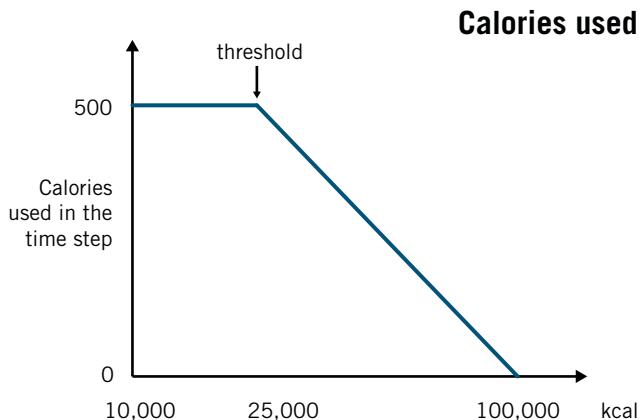
The kilocalories gained each time step the cougar agent is eating will be added to the cougar agent's energyReserve field. This field stores the current energy state of the cougar agent—essentially it determines how hungry the cougar agent is each time step.

Energetics—the calories used

A cougar agent loses energy based on the activity it is engaged in during a time step.

The activity the cougar agent performs in a step dictates how much energy (kcal) is lost in that time step. The cougar's activities can be divided into three classes (see table 8.4, "Field data from collared cougars"). The class numbers increase by the calories used and loosely correlate to the following activities: class I—walking, class II—stalking, and class III—running. Since the model does not directly take into account the specific caloric activities (e.g., whether the cougar agent is walking or running), the energetics model will generalize the behavioral classes. For example, generally, when the cougar agent's kcal status is high, it will become less active, and walk/wander over the hour time step. As the cougar's energetic level reduces, the cougar will on average for the hour time step perform more energy-intensive activities, such as stalking prey and running.

Following is the plot of the formula for determining how many kilocalories are reduced from the cougar agent's energetics level each time step. The kcal level in the plot ranges from the point of death to the maximum kcal the cougar agent can possess. The current energy level is defined on the x axis, and the number of kilocalories to subtract at each state on the y axis. For example, if the cougar's energetics level is 50,000 kcal (identified on the x axis), then the cougar should subtract 340 kcal (determined from the y axis) from its energetics level that time step.



Formulas:

If $\leq 25,000$ kcal : calories_used = 500

If $> 25,000$ kcal : calories_used = $-((1/156.25) * (\text{kcal} - 25,000)) + 500$

The amount of kilocalories used in a time step is subtracted from the cougar agent's energyReserve field.

5

6

7

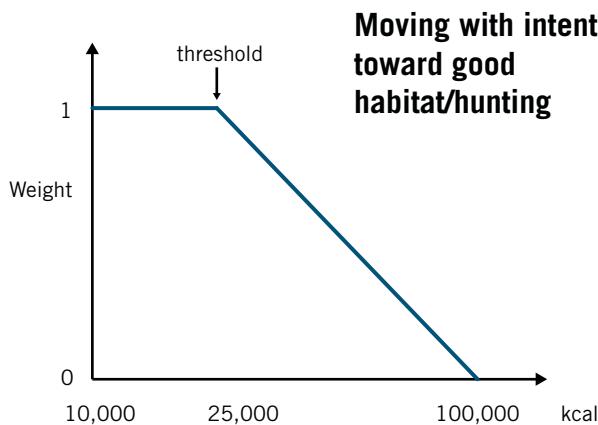
8

9

Weighting the criteria relative to one another using energetics

In addition to adding and subtracting energy to monitor the cougar agent's state for each time step, the energetics model also weights each criterion relative to one another. These weighted criteria are used by the cougar agents to decide which cell to move into.

Separate plots (formulas) were created for five of the six criteria: security, remaining with the kill, pursuing a female cougar agent, staying within the home range, and moving with intent toward good habitat/hunting. (See “Formulas for determining the relative weights for each criterion per time step” later in this chapter). The mating criterion is treated differently, as will be seen in exercise 8e of this chapter. The current state in kcal is listed on the x axis, and the weight to apply to the criterion for that time step on the y axis. For an example, review the moving with intent toward good habitat/hunting plot and formula presented in the following figure.



Formulas:

If $\text{kcal} \leq 25,000$: weight = 1

If $\text{kcal} > 25,000$: weight = $-((1/75,000) * (\text{kcal} - 25,000)) + 1$

Below a threshold (25,000 kcal), the cougar agent must put all its effort into finding food. There is a second threshold (10,000 kcal) at which the cougar dies (the y axis). Above the first threshold, the weight linearly decreases to the cougar's maximum kcal.

To use these criterion-weighting plots, each time step the cougar queries its current kcal state and then calculates the weight for the criterion based on the formula from the appropriate plot. The weights, once calculated for each criterion, are added together and normalized to a 0 to 1 scale. For example, if the energy reserve is 50,000 kcal, then the initial weight for the moving with intent toward good habitat/hunting criterion is .667.

The weights determined for each of the eight cells surrounding the cougar agent are then adjusted (multiplied) by the normalized weights for each criterion. As a result, for each criterion, each of the eight cells surrounding the cougar agent is weighted based on the attributes in the cell or the relative position of the cell toward a desired destination. Then the preferences for

each surrounding cell are adjusted based on how important the criterion (the weight determined from the energetics criterion plots) is to the cougar agent that time step.

Conceptually, to implement this two-step process for weighting each of the eight surrounding cells, for each criterion the model uses the marbles analogy discussed in chapter 5 (see “Creating a probability distribution for determining the movement of an agent into the eight surrounding cells.”) First, for each criterion, a number of marbles is placed in each cell’s bucket based on the preference for that cell for the criterion. Then the number of marbles in each bucket in each cell is multiplied by the weights derived from the criterion-weighting plots (formulas) identified later in this chapter in the section “Formulas for determining the relative weights for each criterion per time step.” The multiplication theoretically adjusts the number of marbles in each bucket in the eight surrounding cells based on the importance of that criterion.

The cells’ cumulative weights (the total number of marbles) create a distribution from which a random value is selected from the distribution to determine which cell the cougar agent should move into that time step. Conceptually, a marble is selected from the aggregation of all the marbles in the larger bucket. The bucket the selected marble originated from is the cell the cougar agent should move into.

Geometric deterrents that influence decision making

In this chapter you will complete the cougar agent model developed in chapters 2 and 5. You will weight the eight surrounding cells based on staying within the home range and moving with intent toward good habitat/hunting. You will create an energetics model and use it to add energy if the cougar agent is eating, subtract energy based on the activity being performed in the time step, and monitor the current status of the energy level for the cougar agent. Based on the current energy status, the model weights the five criteria relative to one another to adjust the weighting of the eight cells surrounding the cougar agent. The cougar agent uses the adjusted weights for each of the eight surrounding cells for each criterion to make a decision. You will also add rules to override this decision-making process by providing a momentary event, adding the mating criterion.

5

6

7

8

9

Exercise 8a

A female cougar's home range needs to be large enough to encompass an adequate food supply for the cougar and her young. A male cougar's home range needs to be large enough so that there is adequate food, and, on average, three females will be within it. In chapter 5, exercise 5e, you saw that when a male cougar agent detects a female cougar agent, that female becomes an attractor, and the male cougar will move toward her. To ensure that the cougar agent stays within its home range, the boundary of the home range will operate as a repellent. As the cougar agent nears the boundary, it will be less likely to continue moving toward it.

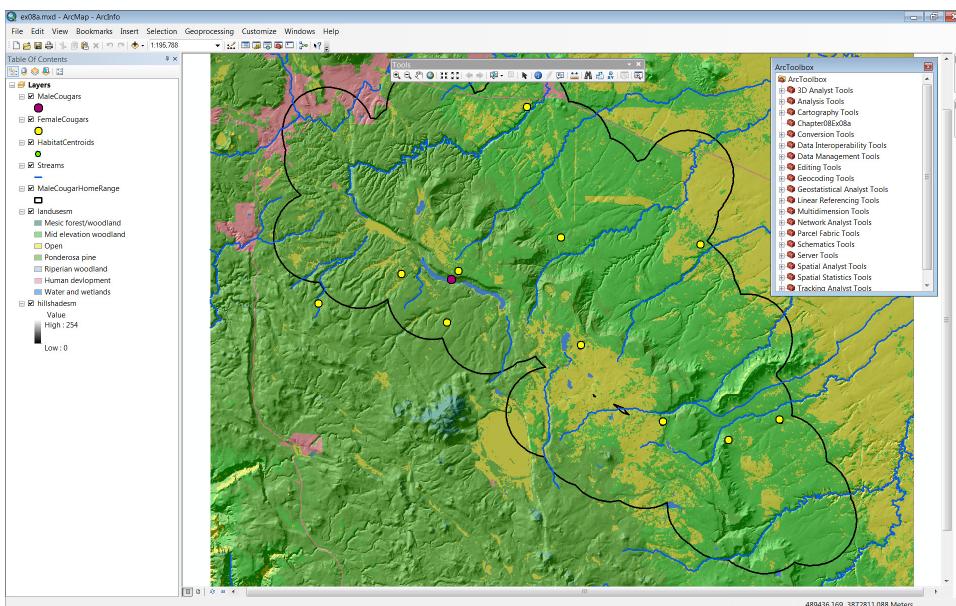
For simplicity, in this model the home range is modeled as a given polygon derived by field data.

In this exercise, you learn how to weight the eight surrounding cells based on the staying within the home range criterion.

Load the ArcMap document

- 1 Start ArcMap. Navigate to C:\ESRIPress\AgentAnalyst\Chapter08. Open ex08a.mxd.

The map document opens. In the ArcMap display, you see land use displayed transparently on top of a hillshade. The purple dot represents the male cougar, the yellow dots the female cougars, and the light green dots are the habitat centroids (the centroids are somewhat hidden by the female cougars). The black polygon outline represents a home range of a male cougar. In ArcToolbox, the Chapter08Ex08a toolbox has been created for you.



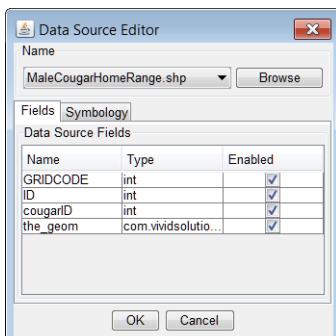
Exercise 8a

Load the Agent Analyst model

- 2 Right-click the Chapter08Ex08a toolbox, point to New, and select Agent Analyst Tool.
- 3 Once the Agent Analyst window appears, click File and select Import. Do not save the changes when prompted.
- 4 Navigate to C:\ESRIPress\AgentAnalyst\Chapter08\Models and open ex08a.sbp.

Identify the home range that belongs to each cougar agent

- 5 In the Environment panel, click HomeRange. In the Property panel, click the Edit button to the right of the Data Source property to open the Data Source Editor.

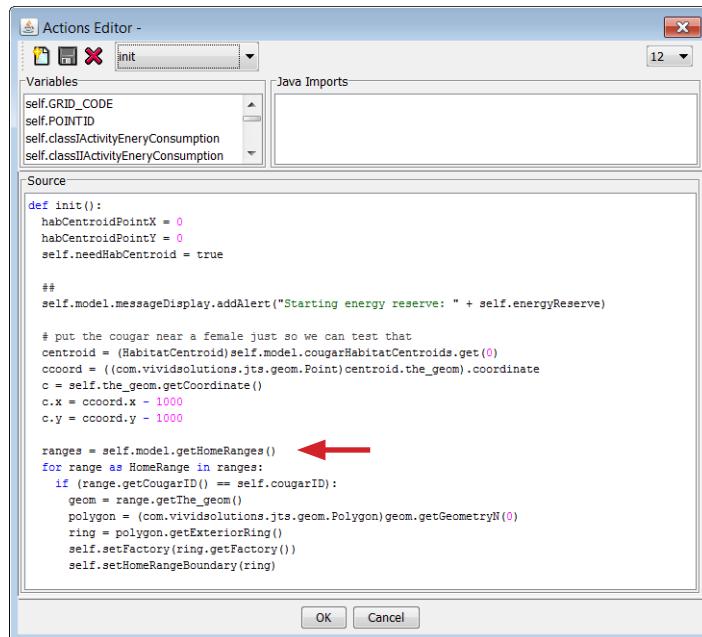


The data source for the home range agent is the MaleCougarHomeRange.shp shapefile. Notice in the Data Source Fields panel that the cougarID and the geometry of the home range are imported as fields. Each home range in the shapefile will become a home range agent. As a result you can identify which male cougar is associated with each home range polygon.

Set the associated home range boundary to the homeRangeBoundary field for each cougar agent

You will set the home range boundary only once, in the initialization of the model, by using the init action.

- 6 In the Environment panel, click MaleCougar. In the Property panel, click the Edit button to the right of the Actions property to open the Actions Editor.
- 7 From the Actions drop-down list, select init. View the following code in the Source panel. The code denoted by the red arrow associates the specific home range with the cougar agent being processed.



In the code highlighted with the red arrow, all the home range agents are identified (`ranges = self.model.getHomeRanges()`). For each home range (`for range as HomeRange in ranges:`) the model tests if the home range belongs to the cougar agent that is being processed (`if (range.getCougarID() == self.cougarID):`). If it does, the boundary polygon for the home range is identified

5

6

7

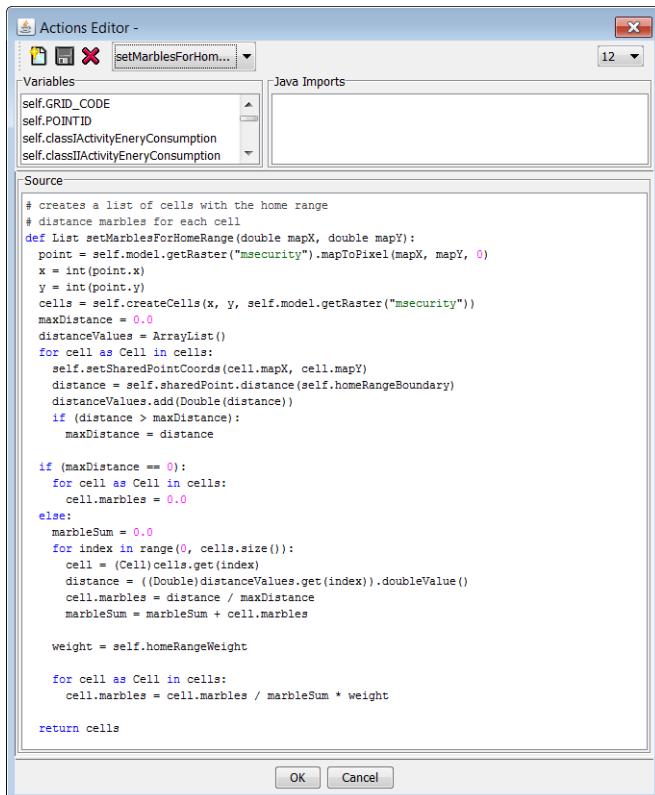
8

9

(geom = range.getThe_geom()) and is set to the homeRangeBoundary field for that cougar agent (`self.setHomeRangeBoundary(ring)`).

Each cougar agent now knows its home range boundary. However, during each time step, the eight cells surrounding the cougar agent need to be weighted based on the staying within the home range criterion.

- 8 From the Actions drop-down list, select setMarblesForHomeRange. View the following code in the Source panel. The eight cells surrounding the cougar agent are weighted based on their relative position to the home range boundary.



The `setMarblesForHomeRange` action is similar to the `setMarblesForSecurity`, `setMarblesForPrey`, and `setMarblesForFemale` actions that you developed in chapter 5. Each of the eight cells surrounding the cougar agent is identified. The relationship of the cell to the criterion is established. Based on the preference for the attributes in the cell or the position of the cell relative to factors concerning the criterion (how close is the home range boundary), each cell is weighted by placing the appropriate number of marbles in the associated bucket for that cell (see “Creating a probability distribution for determining the movement of an agent into surrounding cells,” in chapter 5).

In the setMarblesForHomeRange code identified in the previous figure, the map coordinates of the cougar agent's position are passed to the action (`def List setMarblesForHomeRange(double mapX, double mapY):`). The msecurity raster is used to identify the cell location of the current location of the cougar agent (`point = self.model.getRaster("msecurity").mapToPixel(mapX, mapY, 0)`). The eight cells surrounding the cougar agent are identified as they were for each of the criterion developed in chapter 5 (`cells = self.createCells(x, y, self.model.getRaster("msecurity"))`). For each of the eight surrounding cells (for `cell as Cell in cells:`) a point object is created and the `setSharedPointCoords` action assigns the coordinates of the cell that is being processed to the point object (`self.setSharedPointCoords(cell.mapX, cell.mapY)`). Using the distance function on the point object, the distance from the point (the cell center) to the home range polygon is calculated (`distance = self.sharedPoint.distance(self.homeRangeBoundary)`). As the distances are calculated for each of the eight surrounding cells, they are added to an array (`distanceValues.add(Double(distance))`). As each cell is processed, it is tested to see if it is the farthest from the home range boundary, and if it is, the value is stored in a variable (`if (distance > maxDistance): maxDistance = distance`).

For each of the eight surrounding cells (`for index in range(0, cells.size()):`), the number of marbles assigned to the cell will correspond to the distance that cell is from the home range boundary divided by the maximum distance for all the cells (`cell.marbles = distance / maxDistance`). The division by the maximum distance will ensure that the number of marbles assigned to each cell is on a 0 to 1 relative scale. The number of marbles in each cell's bucket is adjusted by multiplying the number of marbles by the weight for the criterion (see exercise 8c for identifying the weights for each criterion) and then normalizing them to the relative scale so that they will be consistent with the other criteria.

So far, in chapters 5 and 8, you have evaluated the eight surrounding cells for four criteria (security, remaining with the kill, pursuing a female cougar agent, and staying within the home range). There are two additional criteria for the cougar agent to evaluate before making a decision of where to move: moving with intent toward good habitat/hunting and mating.

Run the model

- 9** Click the Run button in the Agent Analyst window.
- 10** Prior to running the model, on the Parameters tab of the Cougar Model Settings window, clear the check boxes for the HuntingOn and MatingOn parameters (this will disable these two criteria).

- 11** Run the cougar model by clicking Start on the Repast toolbar.

If you let the model run for some time you will notice that as the male cougar agent (the purple dot) approaches the home range boundary, it moves away from it, thus keeping the male cougar agent within its home range.

Death has not been implemented in this model demonstration. If you let your model run long enough, you may see negative energy values associated with the cougar agent. It would be easy enough to add a condition that classifies cougar agent as dead when it goes below a specified survival threshold.

- 12** Stop the model.
- 13** Close the model. Do not save any changes.
- 14** Close ArcMap. Do not save any changes.

Adding memory to develop movement with intent

Cougars know where there is good habitat for hunting and security within their home range. To capture this biology in the model, the most ideal habitats within the home range become attractors for the agents. When moving through its territory, especially when hunting, the cougar agent will move toward the habitat attractors (the centroid within the most ideal habitat) within the home range. The habitat attractors keep the cougar agent moving with intent within its home range.

The optimum cougar habitat includes areas that offer security, areas that are rugged (providing the cougar with a hunting advantage), and areas that contain prey and water. The centroids within the most ideal cougar habitats are identified as points in a shape file.

5

6

7

8

9

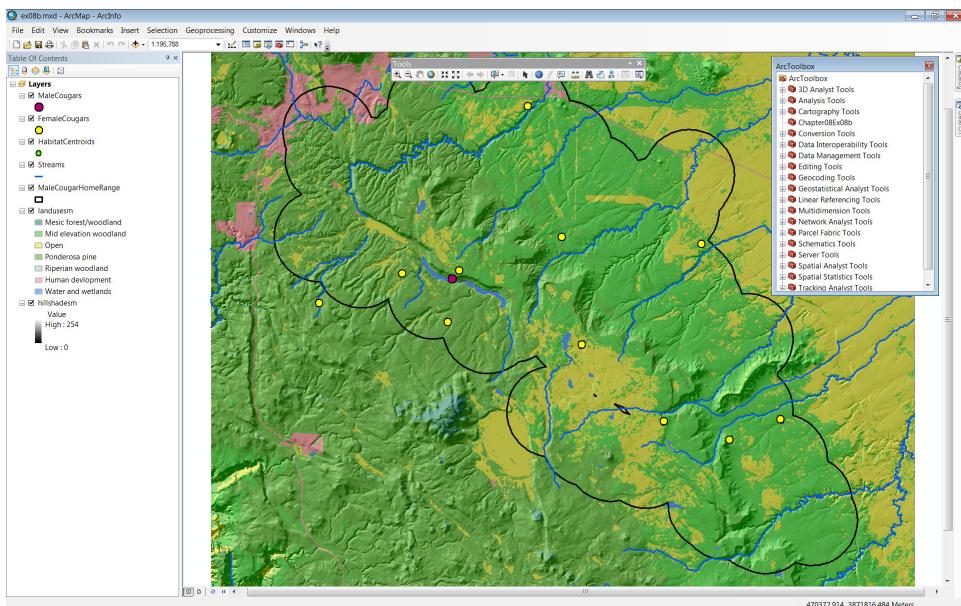
Exercise 8b

Moving with intent toward good habitat/hunting using the habitat attractors is the fifth criterion to consider prior to the cougar agent making a move. When hungry, the cougar agent will weight this criterion higher because it knows by moving toward these good habitat points that it is more likely to make a kill. The eight cells surrounding the cougar agent must be weighted relative to this criterion.

Load the ArcMap document

- 1 Start ArcMap. Navigate to C:\ESRIPress\AgentAnalyst\Chapter08. Open ex08b.mxd.

In the ArcMap display, you will see land use displayed transparently on top of a hillshade. The purple dot represents the male cougar, the yellow dots the female cougars, and the light green dots are the habitat centroids (the centroids are somewhat hidden by the female cougars). The black polygon outline represents the home range of a male cougar. In ArcToolbox, the Chapter08Ex08b toolbox has been created for you.

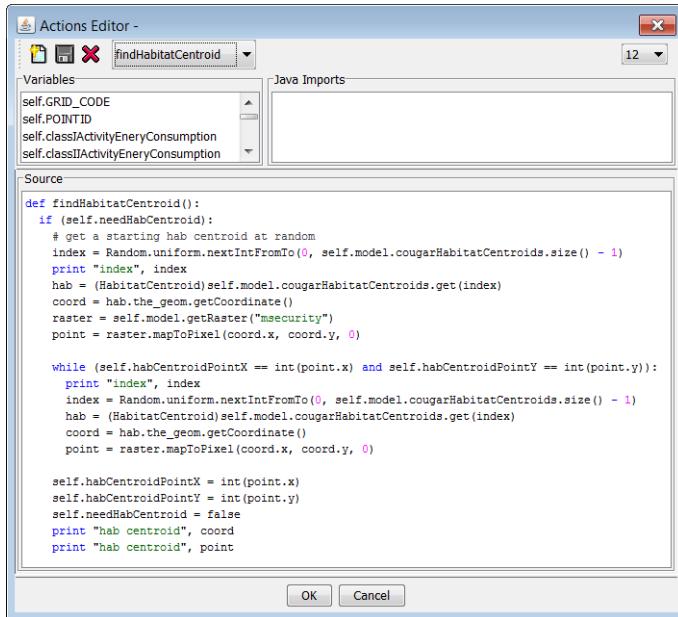


Load the Agent Analyst model

- 2 Right-click the Chapter08Ex08b toolbox, point to New, and select Agent Analyst Tool.
- 3 When the Agent Analyst window appears, click File and select Import. Do not save the changes when prompted.
- 4 Navigate to C:\ESRIPress\AgentAnalyst\Chapter08\Models and open ex08b.sbp.

Identify the habitat centroid the cougar agent will move toward

- 5 In the Environment panel, click MaleCougard. In the Property panel, click the Edit button to the right of the Actions property to open the Actions Editor.
- 6 From the Actions drop-down list, select findHabitatCentroid. Uncomment the following code in the Source panel. The action identifies the habitat centroid for the cougar agent to move toward.



In this code, the `findHabitatCentroid` action is called by the step action for the male cougar agent. The cougar agent's `needHabCentroid` field indicates if the cougar has already identified and is moving toward a habitat centroid. The `habCentroidPointX` and `habCentroidPointY` fields identify the map coordinates of the identified centroid.

If the cougar agent has not identified a habitat centroid to move toward or it has reached a centroid in the previous time step, it will need to identify a new centroid to move toward (`if (self.needHabCentroid) :`). One will be randomly selected (`index = Random.uniform.nextIntFromTo(0, self.model.cougarHabitatCentroids.size() - 1)`). The map coordinates of the centroid of the selected centroid are identified (`coord = hab.the_geom.getCoordinate()`) as well as the cell location (`point = raster.mapToPixel(coord.x, coord.y, 0)`).

If the randomly selected habitat centroid is the same as the one that was previously selected (`while (self.habCentroidPointX == int(point.x) and self.habCentroidPointY == int(point.y)) :`), another habitat centroid is selected.

The new habitat centroid's x position (`self.habCentroidPointX = int(point.x)`) and y position (`self.habCentroidPointY = int(point.y)`) fields are set as well as the field identifying whether the cougar agent has identified a habitat centroid to move toward (`self.needHabCentroid = false`).

5

6

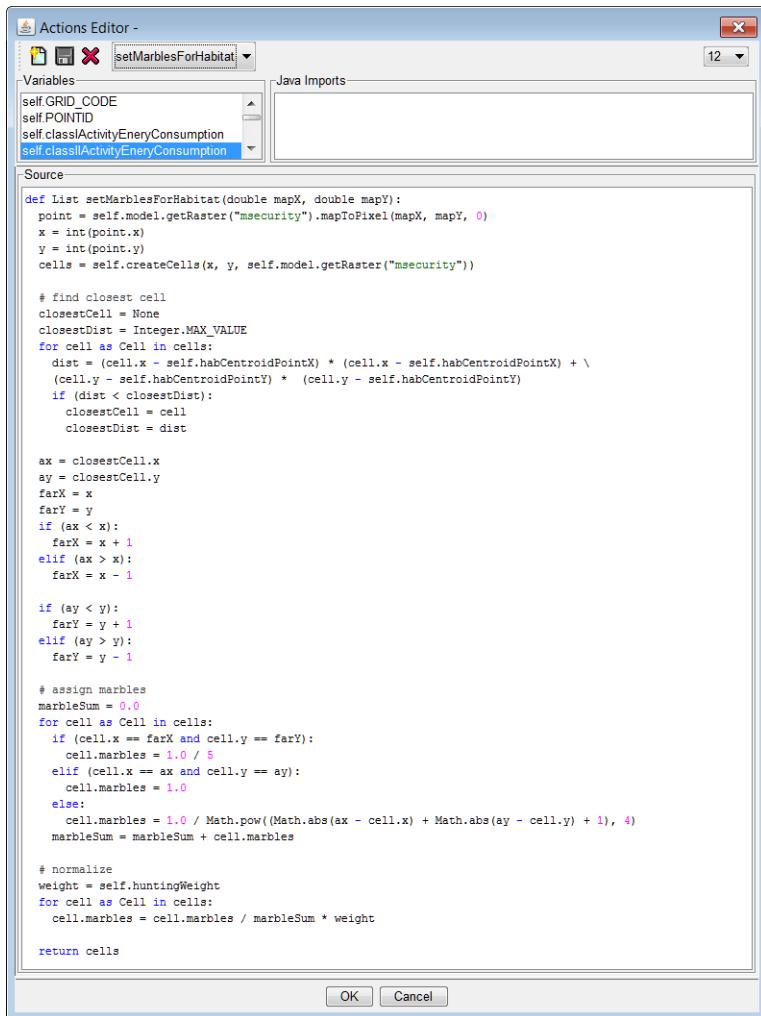
7

8

9

The eight cells surrounding the cougar agent must be weighted relative to their positions to the identified habitat attractor for evaluating the moving with intent toward good habitat/hunting criterion.

- 7 From the Actions drop-down list, select setMarblesForHabitat. View the following code in the Source panel. This action weights the eight cells surrounding the cougar agent relative to moving toward the good habitat/hunting criterion.



The code shown here is similar to the setMarbles actions for each criterion: setMarblesForSecurity (security), setMarblesForPrey (remaining with the kill), setMarblesForFemale (pursuing a female cougar agent), and setMarblesForHomeRange (staying within the home range) discussed in the exercises in chapter 5 and in exercise 8a in this chapter.

In this code, the current location of the cougar agent is passed to the action (`def List setMarblesForHabitat(double mapX, double mapY):`). The eight cells surrounding the cougar are identified (`cells = self.createCells(x, y, self.model.getRaster("msecurity"))`). For each cell (`for cell as Cell in cells:`) the distance to the habitat centroid is calculated (`dist = (cell.x - self.habCentroidPointX) * (cell.x - self.habCentroidPointX) + (cell.y - self.habCentroidPointY) * (cell.y - self.habCentroidPointY)`), the closest cell is determined (`if (dist < closestDist):`), and the cell and map distance of the closest cell are stored in a variable (`closestCell = cell; closestDist = dist`).

The boundary conditions (whether the surrounding cell is outside the study area) are monitored with the `farX` and `farY` variables. The marbles are assigned to each surrounding cell. The closest cell is assigned a value of 1 (`elif (cell.x == ax and cell.y == ay): cell.marbles = 1.0`), the farthest cell is assigned a value of 0.2 (`if (cell.x == farX and cell.y == farY): cell.marbles = 1.0 / 5`), and the remaining cells are assigned based on a decreasing power function as they move from the closest cell (`cell.marbles = 1.0 / Math.pow((Math.abs(ax - cell.x) + Math.abs(ay - cell.y) + 1), 4)`). The weight for the moving with intent toward good habitat/hunting criterion is set (`weight = self.huntingWeight`). The number of marbles is normalized to be compatible with the other criteria and multiplied by the assigned weight for the moving with intent toward good habitat/hunting criterion (`cell.marbles = cell.marbles / marbleSum * weight`).

8 Close the Agent Analyst model. Do not save any changes.

9 Close ArcMap. Do not save any changes.

The eight surrounding cells have now been weighted for five criteria for the cougar agent to consider before making a movement: security, remaining with the kill, pursuing a female cougar agent, staying within the home range, and moving with intent toward good habitat/hunting.

5

6

7

8

9

Monitoring the status of each agent and using that status for agent assessment

Which of the five criteria processed in the previous exercises (security, remaining with the kill, pursuing a female cougar agent, staying within the home range, and moving with intent toward good habitat/hunting) are most important to the cougar agent at any time step is dependent on the cougar's current energetic state at that time step; essentially, how hungry is the cougar? For example, the hungrier the cougar agent is, the more intently it should hunt; therefore, it should weight the moving with intent toward good habitat/hunting (moving toward a habitat centroid) criterion more importantly than the other criteria. Weighting the criterion more importantly adjusts the number of marbles in each of the eight buckets for that criterion, resulting in that criterion contributing more marbles to the aggregated sum of all the marbles (see "Creating a probability distribution for determining the movement of an agent into surrounding cells," in chapter 5).

The cougar can gain energy each time step it consumes its kill. The amount of kilocalories the cougar agent gains consuming the kill and how long it will take to consume it are dependent on the type of prey (see chapter 5, exercises 5b and 5c).

The cougar agent loses energy when performing an activity (e.g., walking or running).

Exercise 8c

To determine the energetic state of the cougar agent for any time step, a running total of the energy gained and used must be monitored each time step.

Identifying the cougar's energetic state at a time step will dictate the weights for each criterion (see "Background information" earlier in the chapter for a complete discussion of the energetics models and the weighting of the criteria).

Load the ArcMap document

- 1 Start ArcMap. Navigate to C:\ESRIPress\AgentAnalyst\Chapter08. Open ex08c.mxd.

In the ArcMap display you will see the land use transparently displayed over the hillshade. The purple dot is a male cougar, the yellow dots are female cougars.

Note: You might not see the male cougar agent in the display if you ran the model for many time steps in exercise 8a. The agent will reappear when you run the model.

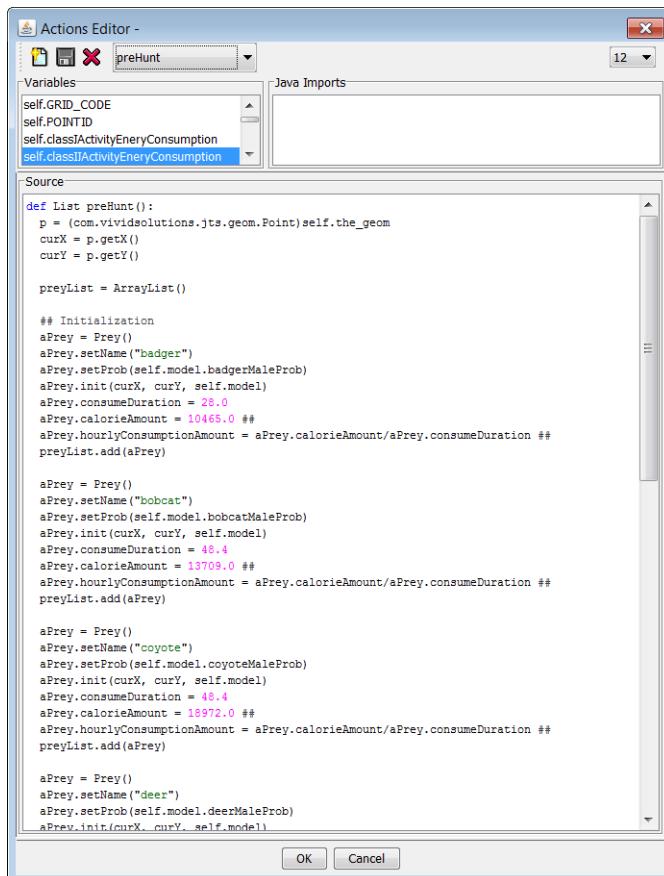
In ArcToolbox, the Chapter08Ex08c toolbox has been created for you.

Load the Agent Analyst model

- 2** Right-click the Chapter08Ex08c toolbox, point to New, and select Agent Analyst Tool.
- 3** Once the Agent Analyst window appears, click File and select Import. Do not save the changes when prompted.
- 4** Navigate to C:\ESRIPress\AgentAnalyst\Chapter08\Models and open ex08c.sbp.

Set the energy gained by eating a particular prey (in kcal) in the preHunt action

- 5** In the Environment panel, click MaleCougar. In the Property panel, click the Edit button to the right of the Actions property to open the Actions Editor.
- 6** From the Actions drop-down list, select preHunt. View the following code in the Source panel.



5
6
7
8
9

The preHunt action identifies how long a cougar agent will take to consume each kill (`aPrey.consumeDuration = 28.0`), the total calories gained (`aPrey.calorieAmount = 10465.0`), and the amount of calories gained each hour the cougar agent consumes the kill (`aPrey.hourlyConsumptionAmount = aPrey.calorieAmount/aPrey.consumeDuration`), and sets the values to the appropriate fields.

The cougar agent will add the appropriate kcal when it is eating and remove kcal when doing other activities. This is done in the step action.

- 7 In the Actions drop-down list, click step. View the following code in the Source panel.

```

Actions Editor - step
Variables: self.GRID_CODE, self.POINTID, self.classIActivityEnergyConsumption, self.classIIActivityEnergyConsumption
Java Imports: None

Source:
    self.model.pause()
    self.targetFemale = female

    if (self.prey != None): ←
        timeDiff = self.model.getTickCount() - self.preyTimeStart
        ## Stop eating if the prey is all gone
        if (timeDiff >= self.prey.consumeDuration):
            self.prey = None
            self.model.messageDisplay.addAlert("Finished eating")
            self.model.pause()
        ## Eating part of the prey - adding the prey's calorie amount / prey's consumeDuration, up
        ## cougars max energy reserve amount (when that happens, discard the rest of the prey)
        else:
            remainingRoom = self.maximumEnergyReserve - self.energyReserve
            self.model.messageDisplay.addAlert("Room left: " + remainingRoom)
            if (remainingRoom > self.prey.hourlyConsumptionAmount):
                self.energyReserve = self.energyReserve + self.prey.hourlyConsumptionAmount
                self.model.messageDisplay.addAlert("Energy added: " + self.prey.hourlyConsumptionAmount)
            else:
                self.energyReserve = self.energyReserve + remainingRoom
                self.model.messageDisplay.addAlert("Energy added: " + remainingRoom)
            self.prey = None
            self.model.messageDisplay.addAlert("Full - Finished eating")
            self.model.pause()

        self.model.messageDisplay.addAlert("Energy left: " + self.energyReserve)

        ## The cougar hunts when he has finished consuming the prey previously hunted and he is getting
        if (self.model.huntingOn and (self.prey == None and self.energyReserve < self.energyReserve)):
            self.hunt(curX, curY)

        ## Formula for energy consumed during this step
        # If eating prey, use Class I activity consumption
        # If <= 25,000 kcal : calories_used = 1/156.25(kcal - 25,000) + 500
        # If > 25,000 kcal : calories_used = 1/156.25(kcal - 25,000) + 500
        if (self.prey != None): ←
            energyConsumed = self.classIActivityEnergyConsumption
            elif (self.energyReserve <= 25000.0):
                energyConsumed = 500.0
            else:
                energyConsumed = 500 - (self.energyReserve - 25000)/156.25

            self.energyReserve = self.energyReserve - energyConsumed
            self.model.messageDisplay.addAlert("Energy consumed: " + energyConsumed)
            self.model.messageDisplay.addAlert("Energy left: " + self.energyReserve)

        self.model.messageDisplay.addAlert("=====")

```

The code by the upper arrow adds kilocalories when the cougar agent is eating, and the code by the lower arrow subtracts kilocalories based on the activity the cougar agent is performing during the time step.

The code highlighted by the first arrow determines if the cougar agent is consuming a kill (`if (self.prey != None) :)`). If it is, the time the cougar agent has been eating the kill is determined (`timeDiff = self.model.getTickCount() - self.preyTimeStart`). If the cougar is finished consuming the kill (`if (timeDiff >= self.prey.consumeDuration) :)`, the prey field will be set to None (`self.prey = None`). If the cougar is not done consuming the kill, the cougar agent's stomach capacity is checked (`remainingRoom = self.maximumEnergyReserve - self.energyReserve`). If the cougar agent still has room to eat (`if (remainingRoom > self.prey.hourlyConsumptionAmount) :)`, then the hourly calories gained from consuming that particular kill are added to its energy level (`self.energyReserve = self.energyReserve + self.prey.hourlyConsumptionAmount`).

If the cougar agent is not consuming a kill during the time step, the prey field would be set to None, and therefore it will not enter the earlier if condition (`if (self.prey != None) :)`.

Because each time step the cougar agent performs some action, the amount of kilocalories burned during the action must be subtracted from its energy reserve. The code highlighted by the second arrow performs the kcal reduction. If the cougar agent is eating (`if (self.prey != None) :)`, the energy used will be set to the classIActivityEnergyConsumption field (`energyConsumed = self.classIActivityEnergyConsumption`), which was defined in the Fields Editor for the male cougar agent. If the cougar agent is below the threshold of 25,000 kcal, it will be in a survival mode, reducing its reserve by 500 kcal (`energyConsumed = 500.0`). In all other cases, based on the agent's current energy state, an activity level will be defined using the plot (the formula) discussed in the section "Background information," and the appropriate kcal used performing the activity calculated (`energyConsumed = 500 - (self.energyReserve - 25000) / 156.25`). The energy used is then subtracted from the cougar's energy reserve (`self.energyReserve = self.energyReserve - energyConsumed`).

Run the model

- 8** Click Run in the Agent Analyst window. When the Repast toolbar appears, click the Start button.

A dialog box appears. Remember that the model will pause when the male cougar agent detects a female, makes a kill, finishes a kill, or is mating (identified in the dialog box). Click the Start button again to continue the model.

In the dialog box, notice that the energy level of the male cougar agent is identified. When the male cougar agent makes a kill, the type of prey that was killed is identified. As the

5

6

7

8

9

agent consumes the kill, his energy level goes up and he stays with the kill until he consumes it.

9 Close the model. Do not save any changes.

10 Close ArcMap. Do not save any changes.

The current state of the cougar agent's energy reserve will influence the weighting of each criterion and will affect the decision the cougar agent makes during that time step. In the next exercise, you will see how the cougar agent makes a decision using the multiple criteria.

Creating probability distributions from the neighboring locations for the multiple criteria to create the decision space for the agent

In the previous exercises you identified the relative preference for each of the eight cells surrounding a cougar agent based on the goals of the criteria and the attributes in the cell (e.g., the security level) or the cell's relative position (e.g., closest to a kill). You have weighted the surrounding cells for five criteria:

- Security
- Remaining with the kill
- Pursuing a female cougar agent (for a male cougar agent)
- Staying within the home range
- Moving with intent toward good habitat/hunting

The sixth criterion, mating, is accounted for as a momentary event and is implemented outside the multicriteria decision-making process. You will implement the mating criterion in exercise 8f.

The cougar agent examines the five criteria, and based on its current energetic state, it will make a decision to move to a surrounding cell (or not move at all).

Exercise 8d

You will now learn how to make trade-offs between the criteria.

Load the ArcMap document

- 1 Start ArcMap. Navigate to C:\ESRIPress\AgentAnalyst\Chapter08. Open ex08d.mxd.

In the ArcMap display you will see the land use displayed transparently on top of the hillshade. The purple dot represents a male cougar, and the yellow dots represent females. In ArcToolbox, the Chapter08Ex08d toolbox has been created for you.

Load the Agent Analyst model

- 2 Right-click the Chapter08Ex08d toolbox, point to New, and select Agent Analyst Tool.

- 3 Once the Agent Analyst window appears, click File and select Import. Do not save the changes when prompted.
- 4 Navigate to C:\ESRIPress\AgentAnalyst\Chapter08\Models and open ex08d.sbp.

In the previous exercises you focused on creating the individual components of the model, but to gain an overview of how the components fit together, you need to further examine the step action.

- 5 In the Environment panel, click MaleCougar. In the Property panel, click the Edit button to the right of the Actions property to open the Actions Editor.
- 6 From the Actions drop-down list, select step. View the following code in the Source panel. The step action controls what the male cougar agent does each time step.

Actions Editor -

Variables: self.GRID_CODE, self.POINTID, self.classIActivityEnergyConsumption, self.classIIActivityEnergyConsumption

Java Imports:

```

Variables
self.GRID_CODE
self.POINTID
self.classIActivityEnergyConsumption
self.classIIActivityEnergyConsumption

Source
def step():
    # check if need to new habitatCentroid
    self.findHabitatCentroid()

    # decide where to move to given the
    # current location
    p = (com.vividsolutions.jts.geom.Point)self.the_geom
    curX = p.getX()
    curY = p.getY()

    if (not self.matingOn):
        self.calcMove(curX, curY)

    # check if female is in range
    if (self.model.matingOn and self.targetFemale == None):
        for female in FemaleCougar in self.model.females:
            if (female.isAvailable):
                distanceToFemale = self.the_geom.distance(female.the_geom)
                # is female within 3000
                if (distanceToFemale < 3000):
                    self.model.messageDisplay.addAlert("FOUND FEMALE")
                    self.model.pause()
                    self.targetFemale = female

    if (self.prey != None):
        timeDiff = self.model.getTickCount() - self.preyTimeStart
        # Stop eating if the prey is all gone
        if (timeDiff >= self.prey.consumeDuration):
            self.prey = None
            self.model.messageDisplay.addAlert("Finished eating")
            self.model.pause()
        # Eating part of the prey - adding the prey's calorie amount / prey's consumeDuration, up to the
        # # cougars max energy reserve amount (when that happens, discard the rest of the prey)
        else:
            remainingRoom = self.maximumEnergyReserve - self.energyReserve
            self.model.messageDisplay.addAlert("Room left: " + remainingRoom)
            if (remainingRoom > self.prey.hourlyConsumptionAmount):
                self.energyReserve = self.energyReserve + self.prey.hourlyConsumptionAmount
                self.model.messageDisplay.addAlert("Energy added: " + self.prey.hourlyConsumptionAmount)
            else:
                self.energyReserve = self.energyReserve + remainingRoom
                self.model.messageDisplay.addAlert("Energy added: " + remainingRoom)
                self.prey = None
                self.model.messageDisplay.addAlert("Full - Finished eating")
                self.model.pause()

            self.model.messageDisplay.addAlert("Energy left: " + self.energyReserve)

        ## The cougar hunts when he has finished consuming the prey previously hunted and he is getting hungry
        if (self.model.huntingOn and (self.prey == None and self.energyReserve < self.energyReserveForHuntingInterest)):
            self.hunt(curX, curY)

    ## Formula for energy consumed during this step
    # If eating prey, use Class I activity consumption
    # If <= 25,000 kcal : calories_used = 500
    # If > 25,000 kcal : calories_used = - 1/156.25(kcal - 25,000) + 500
    if (self.prey != None):
        energyConsumed = self.classIActivityEnergyConsumption
    elif (self.energyReserve <= 25000.0):
        energyConsumed = 500.0
    else:
        energyConsumed = 500 - (self.energyReserve - 25000)/156.25

    self.energyReserve = self.energyReserve - energyConsumed
    self.model.messageDisplay.addAlert("Energy consumed: " + energyConsumed)
    self.model.messageDisplay.addAlert("Energy left: " + self.energyReserve)

    self.model.messageDisplay.addAlert("=====")

```

OK Cancel

The step action for the male cougar agent is called each time step the model runs. This action is the organizing action controlling the flow of the male cougar agent's decision

5

6

7

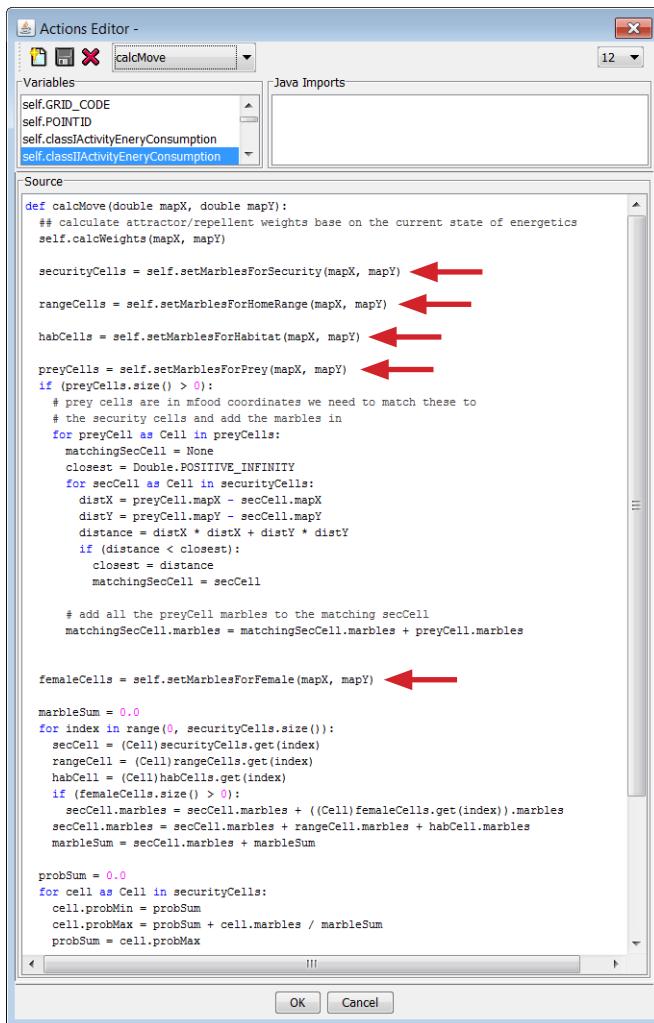
8

9

making. The code highlighted by the upper arrow identifies the current location of the cougar agent. At the second arrow the cougar agent checks to see if it is mating. If it is not mating, it calls the calcMove action. As you will see later, the calcMove action sets the weights for each of the cells surrounding the male cougar agent (calling the setMarbles actions) for each criterion, sets the weights for each criterion relative to one another, and then makes a move. At the third arrow, after the male cougar agent makes a move, if it has not identified a female cougar agent to move toward in a previous step, it checks to see if it can detect one this time step. At the fourth arrow, after the male cougar makes the move, it checks to see if it is consuming a kill during that time step, and if it is, it adds the appropriate energetics. If the cougar agent is not consuming a kill, then it should hunt in the cell it just moved into by calling the hunt action (at the fifth arrow). And finally, at the lowest arrow, the energy level for the cougar is reduced based on the activity it performs during the time step.

View the calcMove action to understand how the cougar agent weights each criterion relative to one another and how each of the eight surrounding cells is weighted

- 7 From the Actions drop-down list in the open Actions Editor, select calcMove. View the following code in the Source panel. The action identifies the weights for each criterion and the weights for each of the eight cells surrounding the cougar agent relative to each criterion.



```

Actions Editor - calcMove
Variables: self.GRID_CODE, self.POINTID, self.class!ActivityEnergyConsumption, self.class!ActivityEnergyConsumption
Java Imports: None
Source:
def calcMove(double mapX, double mapY):
    ## calculate attractor/repellent weights base on the current state of energetics
    self.calcWeights(mapX, mapY)

    securityCells = self.setMarblesForSecurity(mapX, mapY) ←
    rangeCells = self.setMarblesForHomeRange(mapX, mapY) ←
    habCells = self.setMarblesForHabitat(mapX, mapY) ←
    preyCells = self.setMarblesForPrey(mapX, mapY) ←
    if (preyCells.size() > 0):
        # prey cells are in mfood coordinates we need to match these to
        # the security cells and add the marbles in
        for preyCell as Cell in preyCells:
            matchingSecCell = None
            closest = Double.POSITIVE_INFINITY
            for secCell as Cell in securityCells:
                distX = preyCell.mapX - secCell.mapX
                distY = preyCell.mapY - secCell.mapY
                distance = distX * distX + distY * distY
                if (distance < closest):
                    closest = distance
                    matchingSecCell = secCell

            # add all the preyCell marbles to the matching secCell
            matchingSecCell.marbles = matchingSecCell.marbles + preyCell.marbles

    femaleCells = self.setMarblesForFemale(mapX, mapY) ←

    marbleSum = 0.0
    for index in range(0, securityCells.size()):
        secCell = (Cell)securityCells.get(index)
        rangeCell = (Cell)rangeCells.get(index)
        habCell = (Cell)habCells.get(index)
        if (femaleCells.size() > 0):
            secCell.marbles = secCell.marbles + ((Cell)femaleCells.get(index)).marbles
            secCell.marbles = secCell.marbles + rangeCell.marbles + habCell.marbles
        marbleSum = secCell.marbles + marbleSum

    probSum = 0.0
    for cell as Cell in securityCells:
        cell.probMin = probSum
        cell.probMax = probSum + cell.marbles / marbleSum
        probSum = cell.probMax

```

The code in the Actions Editor defines a `calcMove` action. It starts by calling `self.calcWeights`. Then, it calls five different `setMarbles` actions: `setMarblesForSecurity`, `setMarblesForHomeRange`, `setMarblesForHabitat`, `setMarblesForPrey`, and `setMarblesForFemale`. Each of these calls is highlighted with a red arrow pointing to the method name.

In this code, based on the current energetic state of the cougar agent, the weights for each criterion are determined by calling the `calcWeights` action (`self.calcWeights(mapX, mapY)`). In the `calcWeights` action, the weights are determined from the formulas for the plots discussed in “Background information” earlier and presented in “Formulas for determining the relative weights for each criterion per time step” later in this chapter. You will examine the `calcWeights` action in the next series of steps.

Each of the eight cells surrounding the cougar agent is weighted based on each criterion by calling the appropriate `setMarbles` action (see the five arrows). You have explored each of these actions in earlier exercises. The `setMarblesForSecurity`, `setMarblesForHomeRange`, `setMarblesForHabitat`, `setMarblesForPrey`, and `setMarblesForFemale` actions are called.

5

6

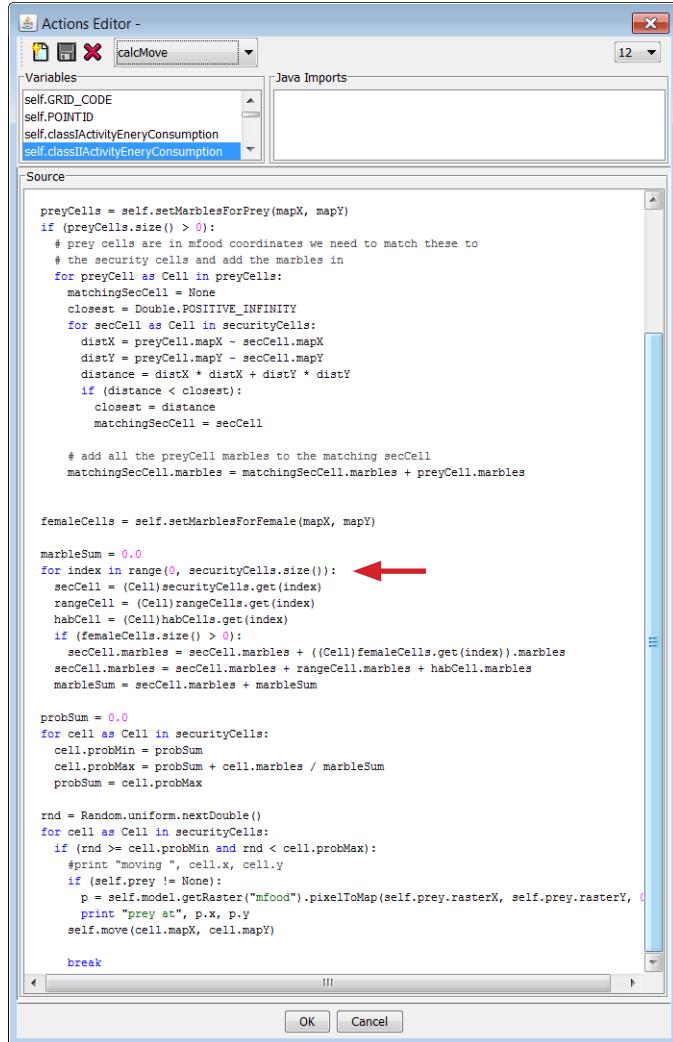
7

8

9

Each of the eight cells surrounding the cougar agent is now weighted for each criterion, and the weights for each criterion, based on the cougar agent's energetic state, are identified.

Scroll down to review the remaining code for the calcMove action.



The screenshot shows the Actions Editor window with the 'calcMove' action selected. The 'Source' tab displays the following Python code:

```

preyCells = self.setMarblesForPrey(mapX, mapY)
if (preyCells.size() > 0):
    # prey cells are in mfood coordinates we need to match these to
    # the security cells and add the marbles there
    for preyCell as Cell in preyCells:
        matchingSecCell = None
        closest = Double.POSITIVE_INFINITY
        for secCell as Cell in securityCells:
            distX = preyCell.mapX - secCell.mapX
            distY = preyCell.mapY - secCell.mapY
            distance = distX * distX + distY * distY
            if (distance < closest):
                closest = distance
                matchingSecCell = secCell

        # add all the preyCell marbles to the matching secCell
        matchingSecCell.marbles = matchingSecCell.marbles + preyCell.marbles

femaleCells = self.setMarblesForFemale(mapX, mapY)

marbleSum = 0.0
for index in range(0, securityCells.size()): ←
    secCell = (Cell)securityCells.get(index)
    rangeCell = (Cell)rangeCells.get(index)
    habCell = (Cell)habCells.get(index)
    if (femaleCells.size() > 0):
        secCell.marbles = secCell.marbles + ((Cell)femaleCells.get(index)).marbles
        secCell.marbles = secCell.marbles + rangeCell.marbles + habCell.marbles
        marbleSum = secCell.marbles + marbleSum

probSum = 0.0
for cell as Cell in securityCells:
    cell.probMin = probSum
    cell.probMax = probSum + cell.marbles / marbleSum
    probSum = cell.probMax

rnd = Random.uniform.nextDouble()
for cell as Cell in securityCells:
    if (rnd >= cell.probMin and rnd < cell.probMax):
        #print "moving ", cell.x, cell.y
        if (self.prey != None):
            p = self.model.getRaster("mfood").pixelToMap(self.prey.rasterX, self.prey.rasterY, 0)
            print "prey at", p.x, p.y
            self.move(cell.mapX, cell.mapY)

        break

```

A red arrow points to the line of code where the marbles from the femaleCells are added to the secCell. The code is part of a larger loop that iterates through the securityCells.

In the code highlighted with the red arrow, after the setMarbles actions have been called, the marbles in each cell for each criterion (`secCell = (Cell)securityCells.get(index)`) are added together (`secCell.marbles = secCell.marbles + rangeCell.marbles + habCell.marbles`). Notice in the previous line of code

that if the male cougar is pursuing a female cougar, the marbles for the pursuing a female criterion are added to the security marbles. The sum of all the marbles for each cell is tracked (`marbleSum = secCell.marbles + marbleSum`).

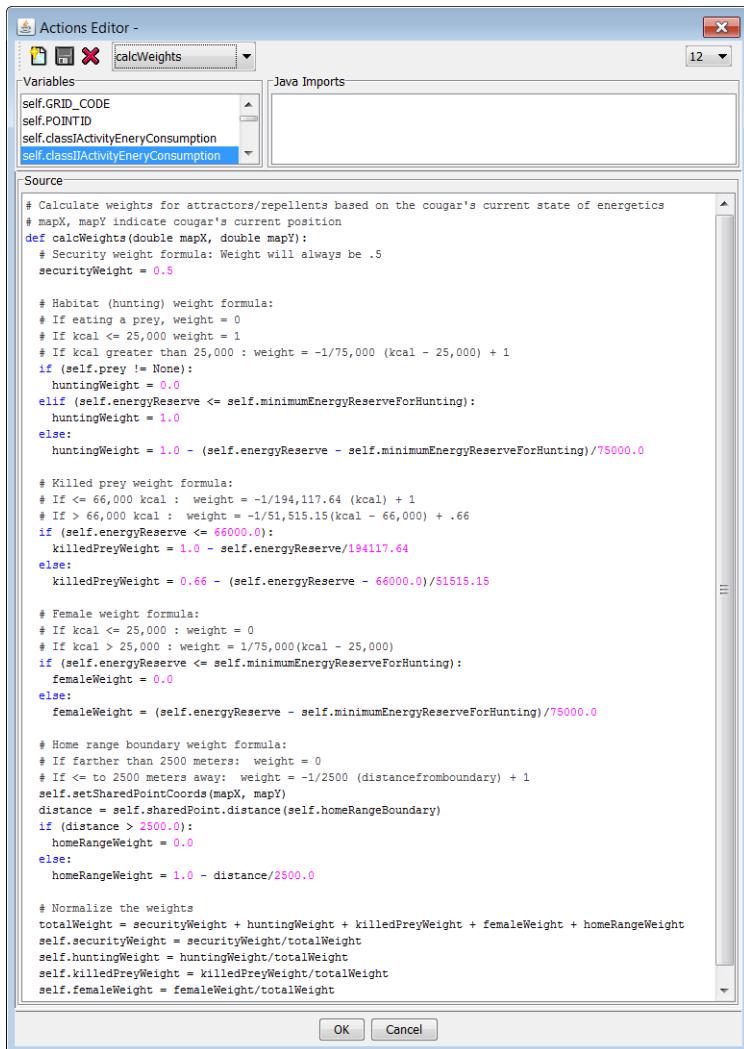
For each cell (`for cell as Cell in securityCells :`), in order to create a distribution generated from all the cells' weightings (the total number of marbles for all the cells for all the criteria), the probability ranges on a 0 to 1 scale are determined (`cell.probMax = probSum + cell.marbles / marbleSum`). For the first cell being processed, the starting range begins at 0 (`cell.probMin = probSum`). To define the end value of the probability range for the cell on the 0 to 1 scale, the number of marbles at the location (`probSum + cell.marbles` [for the first cell `probSum` equals 0]) is divided by the total sum of marbles (`marbleSum`). This process continues for each cell, with the starting range of the subsequent cell on the 0 to 1 scale beginning where the previous cell's range ended (`probSum = cell.probMax`). For example, the top right cell may range from 0 to .28, the top center .28 to .43, the top left .43 to .55, the immediate left .55 to .61, the lower left .61 to .76, the immediate bottom .76 to .87, the lower right .87 to .91, and the cell immediately to the right .91 to 1.

A random number is calculated (`rnd = Random.uniform.nextDouble()`) and tested relative to each cell's probability range defined above on the 0 to 1 scale (`for cell as Cell in securityCells :`). If the random number falls within the assigned probability range for the cell (`if (rnd >= cell.probMin and rnd < cell.probMax)`), then the cougar agent will move into that cell by calling the move action (`self.move(cell.mapX, cell.mapY)`). Continuing the example, if the random number is .89, then the cougar agent should move into the cell located to its lower right. You will examine the move action in the next exercise.

To see how the weights were calculated, examine the `calcWeights` action, which was called in the `calcMove` action.

- 8 From the Actions drop-down list in the open Actions Editor, select `calcWeights`. View the following code in the Source panel. This action sets the weights for each criterion relative to one another based on the current energetic state of the cougar agent.

5
6
7
8
9



In the code, the weights are assigned to each criterion based on the formulas for the plots discussed earlier in “Background information” and presented in “Formulas for determining the relative weights for each criterion per time step” later in this chapter. The current energetic state of the cougar agent is tested against the various thresholds for the criterion (`if (self.energyReserve <= 66000.0):`). If it is below the threshold, then one formula is applied (`killedPreyWeight = 1.0 - self.energyReserve / 194117.64`). If it is above the threshold, a different, increasing or decreasing formula may be applied (`killedPreyWeight = 0.66 - (self.energyReserve - 66000.0) / 51515.15`). In the formula for the remaining with the kill criterion, if the energy level just before making the kill is below 66,000 kcal, the cougar agent really needs

the energy, so it should stay with the kill, thus applying a higher weight and increasing that weight slowly with lower energetic states. If the energetic state is greater than 66,000 kcal, the cougar agent is not in as much need of the kill, thus the weight will reduce more dramatically as its energetics level increases. As a result of the reduction in the weight for the remaining with the kill criterion, the cougar agent will wander a little farther from the prey.

9 Close the model. Do not save any changes.

10 Close ArcMap. Do not save any changes.

The eight cells surrounding the cougar agent are now weighted per criterion and each criterion is weighted relative to one another based on the energetic state of the cougar agent for that time step. A distribution of the cumulative probabilities for each cell is calculated and tested against a random number. If a cell is more favorable relative to each criterion during that time step, that cell will have a larger probability range in the distribution, and there's a greater chance the random number will fall within the range. Thus, the chance of that cell being selected is greater. The probability range that the random number falls into identifies the cell the cougar agent should move into during the time step.

5

6

7

8

9

Moving the point agents

In the previous exercise, the cougar agent identified which of the eight surrounding cells to move into. In this exercise, you will finally have the cougar agent make a movement.

Exercise 8e

Load the ArcMap document

- 1 Start ArcMap. Navigate to C:\ESRIPress\AgentAnalyst\Chapter08. Open ex08e.mxd.

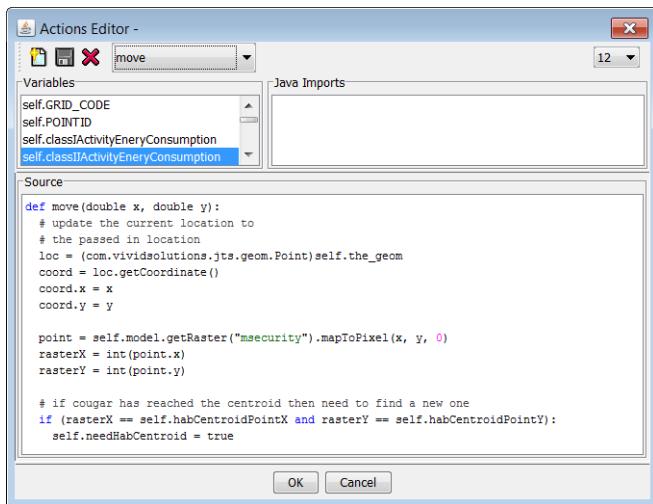
In the ArcMap display you will see the land-use raster transparently displayed over the hillshade. The purple dot represents a male cougar agent, and the yellow dots represent female cougar agents. In ArcToolbox, the Chapter08Ex08e toolbox has been created for you.

Load the Agent Analyst model

- 2 Right-click the Chapter08Ex08e toolbox, point to New, and select Agent Analyst Tool.
- 3 Once the Agent Analyst window appears, click File and select Import. Do not save the changes when prompted.
- 4 Navigate to C:\ESRIPress\AgentAnalyst\Chapter08\Models and open ex08e.sbp.

View the move action

- 5 In the Environment panel, click MaleCougar. In the Property panel, click the Edit button to the right of the Actions property to open the Actions Editor.
- 6 From the Actions drop-down list, select move. View the following code in the Source panel. This action moves the cougar agent.



The move action is called by the calcMove action after each of the eight cells surrounding the cougar agent is weighted relative to each criterion and each criterion is weighted relative to each other as discussed in exercise 8d. In the code in the calcMove action discussed earlier, the x and y map position of the cell the cougar agent will move into is passed to the action (def move (double x, double y) :). A geometry object is created for the location of the cougar agent (loc = (com.vividsolutions.jts .geom.Point)self.the_geom), and the map coordinates of the cougar agent are identified from the geometry object (coord = loc.getCoordinate ()). The current location of the cougar agent is changed through the geometry object associated with the cougar agent to the map coordinates of the centroid of the cell in which to move into (coord.x = x; coord.y = y). The cell value for the map coordinate is determined (point = self.model.getRaster ("msecurity") .mapToPixel (x, y, 0)) to verify if the cougar agent has reached a habitat centroid. If it has, a new habitat centroid needs to be identified (self.needHabCentroid = true). Setting needHabCentroid to true signals the cougar agent to look for a new habitat centroid the next time step and will be used as the attractor for the moving with intent toward good habitat/hunting criterion.

- 7** Close the model. Do not save any changes.
- 8** Close ArcMap. Do not save any changes.

The cougar has now made a move by trading off multiple criteria in its decision making.

5
6
7
8
9

Moving agents in x, y space, not GIS space

One way to move agents is to simply change the current coordinate values. This gives you the maximum amount of control when moving an agent in geographic space, but it may also require additional programming to best calculate target coordinates.

Agent Analyst uses the JTS Topology Suite for performing GIS-based operations such as feature manipulation. JTS is a Java-based application programming interface (API) that attempts to implement the OpenGIS Simple Features Specification (SFS).

The JTS Topology Suite is immediately evident in a vector agent as the “the_geom” field. This field holds all the geometry information for the agent. It is equivalent to the Shape field found in Esri products.

JTS should not be confused with Esri’s ArcObjects, which also provides an API for GIS-based spatial operations. Agent Analyst uses a Java version of ArcObjects that ships with ArcEngine that greatly enhances the spatial analytical power of Agent Analyst. More about Esri’s ArcObjects will be covered in chapter 10.

For more on the JTS Topology Suite, see <http://www.vividsolutions.com/jts>.

Overriding the multicriteria decision-making process with momentary events

Certain temporary behaviors can override the multicriteria trade-off approach discussed in earlier exercises. In the case of the cougar model, the male cougar agent will select mating over all other criteria. Because of the weighting determined by the energetics model employed earlier, it is highly unlikely that the cougar would be mating if it does not have adequate energy.

5

6

7

8

9

Exercise 8f

In this exercise, you explore how the model will override all criteria.

Load the ArcMap document

- 1 Start ArcMap. Navigate to C:\ESRIPress\AgentAnalyst\Chapter08. Open ex08f.mxd.

In the ArcMap display you will see the land use transparently displayed over the hillshade. The location of the male cougar agent is identified by a purple dot and the females by yellow dots. In ArcToolbox, the Chapter08Ex08f toolbox has been created for you.

Load the Agent Analyst model

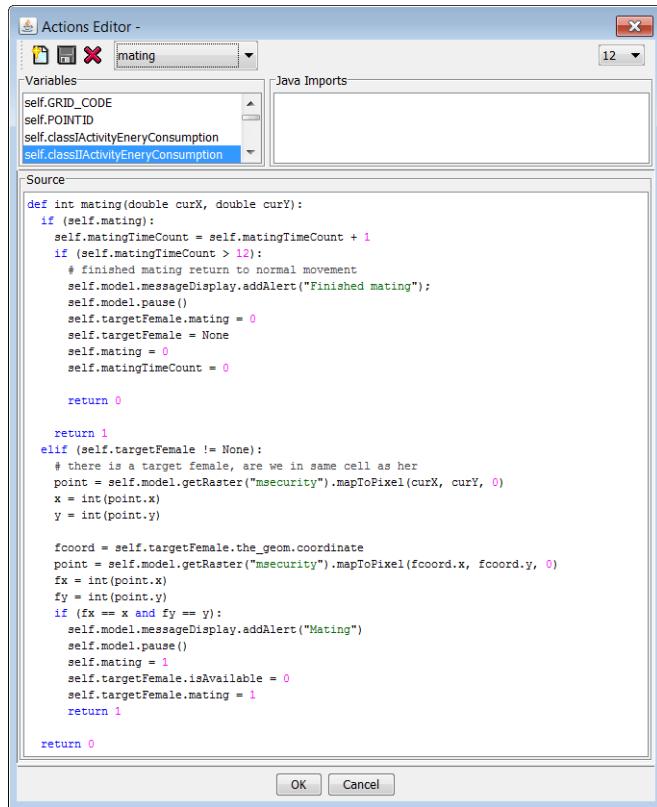
- 2 Right-click the Chapter08Ex08f toolbox, point to New, and select Agent Analyst Tool.
- 3 Once the Agent Analyst window appears, click File and select Import. Do not save the changes when prompted.
- 4 Navigate to C:\ESRIPress\AgentAnalyst\Chapter08\Models and open ex08f.sbp.

View the mating action

- 5 In the Environment panel, click MaleCougar. In the Property panel, click the Edit button to the right of the Actions property to open the Actions Editor.
- 6 From the Actions drop-down list, select mating. View the following code in the Source panel.

In the step action that you examined earlier (refer to exercise 8d), the male cougar agent, before making any move, checks to see if it is mating or not by calling the mating action (`if (not self.mating(curX, curY))`). If the mating action returns to the step action, the agent is not mating; then, in the step action, the agent will make a move (`self.calcMove(curX, curY)`). If the male cougar agent is mating, it should stay with the

female, so the calcMove action (which was discussed in exercise 8e) is not called, and all other criteria are ignored.



In the preceding code, the current location of the male cougar agent is passed to the mating action (`def int mating(double curX, double curY) :`). If the mating field is set to true, the cougar agent is mating (`if (self.mating) :`). When mating in the wild, the male cougar will generally stay with the female cougar for 12 hours. The time that the male cougar agent has been with the female cougar agent is stored in the `matingTimeCount` field. Every time step the male cougar agent is mating, one hour is added to `matingTimeCount` (`self.matingTimeCount = self.matingTimeCount + 1`). If the count is greater than 12 hours, the value of the mating field (which indicates whether the agent is mating) for the female cougar that the male cougar has been mating with is set to 0, which means not mating (`self.targetFemale.mating = 0`), the mating field for the male cougar that was mating is set to 0 (`self.mating = 0`), the identified target female the male cougar agent is pursuing is set to None (`self.targetFemale = None`), and the running time count that records the time the male cougar agent has been mating is set to 0 (`self.matingTimeCount = 0`).

If the male cougar agent is not mating, a target female has been detected in a prior time step, and the male cougar has been moving toward her (`elif (self.targetFemale != None) :`), then the male cougar determines whether it is in the same cell as the target female agent. To do so, the male cougar agent first identifies the cell it currently is in (`point = self.model.getRaster("msecurity").mapToPixel(curX, curY, 0)`) and then identifies where the target female agent is located, first in map coordinates (`fcoord = self.targetFemale.the_geom.coordinate`) and then the cell it is in (`point = self.model.getRaster("msecurity").mapToPixel(fcoord.x, fcoord.y, 0)`). If the male cougar agent and the female cougar agent are in the same cell (`if (fx == x and fy == y) :`), they begin to mate. The mating field for the male cougar agent is set to 1, indicating it is mating (`self.mating = 1`) as is the female cougar agent's mating field (`self.targetFemale.mating = 1`), and the female cougar agent is set to not being available for another male to pursue (`self.targetFemale.isAvailable = 0`).

When cougar agents are mating, this criterion takes priority over all other criteria—therefore, the two agents will stay together.

Run the model

- 7 Click Run in the Agent Analyst window, and then click Start on the Repast toolbar to run the cougar model.

Watch the movement of the male cougar agent. Notice how he will make the trade-offs between the criteria as his state and situation changes.

Note: Remember that the model pauses when a female is detected, a kill is made, the kill is consumed, and mating is under way. You must click Start again when these events occur to continue the model.

- 8 Close the model. Do not save any changes.
- 9 Close ArcMap. Do not save any changes.

In chapters 2, 5, and 8 you have gone through a complex series of exercises to explore how a point agent can move with intention by examining multiple criteria. Additional criteria can be added by examining the attributes contained in each of the eight surrounding cells or through attractors and repellants. The difficult part of adding a criterion is identifying the relevant criterion, defining the mechanism to accurately identify the weights for the eight surrounding cells, and determining the interaction of the criterion with the other criteria in order to appropriately weight each criterion in any given situation.

5

6

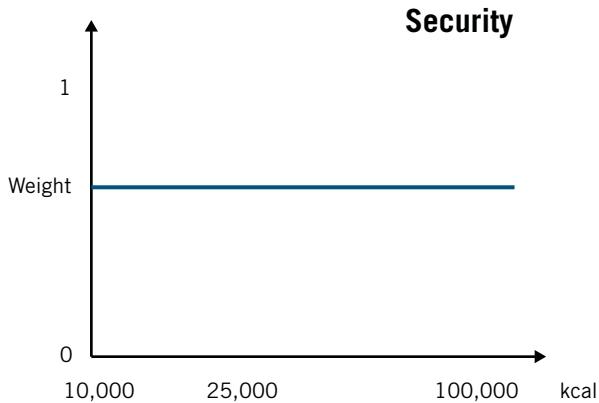
7

8

9

Formulas for determining the relative weights for each criterion per time step

The kcal amounts in the following plots range from the point of death to the maximum kcal the cougar agent can possess.

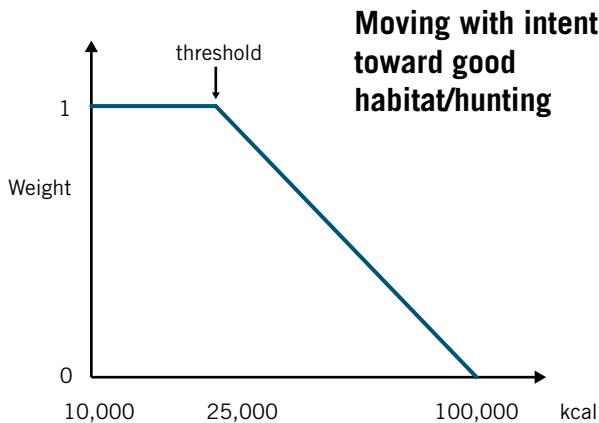


Formula:

The weight will always be .5

This plot illustrates that the weight assigned to the security criterion does not change with the male cougar's energy level.

The cougar agent will always want to be secure. The apparent reduction in the security criterion occurs only because the other criteria increase, not because the security criterion decreases.



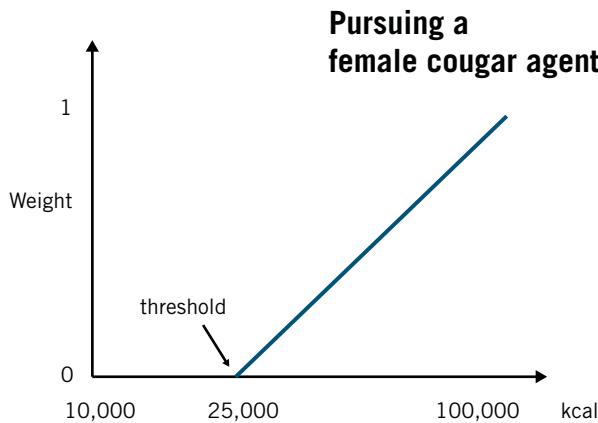
Formulas:

If kcal <= 25,000 : weight = 1;

If kcal > 25,000 : weight = $-(1/75,000) * (\text{kcal} - 25,000) + 1$

This plot illustrates how the weights for the criterion of moving with intent toward good habitat/hunting change according to the male cougar agent's energy level.

Below a threshold (25,000 kcal), the cougar agent must put all its effort into finding food. There is a second threshold (10,000 kcal) at which the cougar dies (the y axis). Above the first threshold, the weight linearly decreases to zero at the cougar's maximum possible kcal energy level.



This plot illustrates how the weights for a male cougar agent for pursuing a female cougar agent change according to the male's energy level.

Formulas:

If kcal <= 25,000 : weight = 0

If kcal > 25,000 : weight = $(1/75,000) * (\text{kcal} - 25,000)$

The male will strongly pursue the female when it has a high energetic state (when it is not hungry). As the male cougar agent loses kilocalories, its interest in the female is reduced, even to zero, as it becomes more important to eat and gain energy.

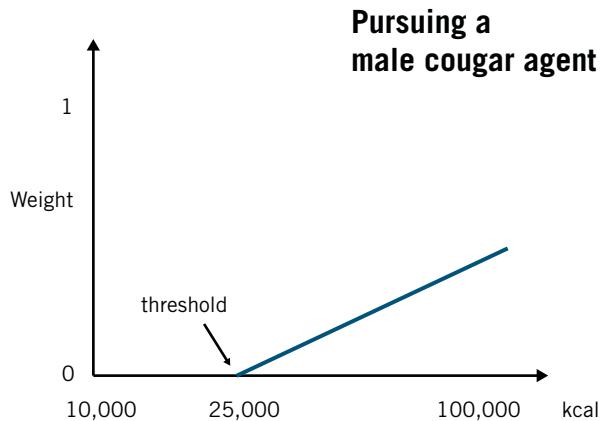
5

6

7

8

9

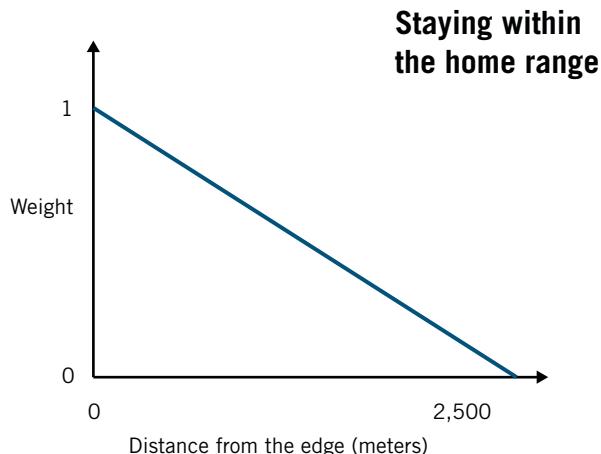
**Formulas:**

If $\text{kcal} \leq 25,000$: weight = 0

If $\text{kcal} > 25,000$: weight = $(1/150,000) * (\text{kcal} - 25,000)$

This plot shows how the weights for a female cougar agent pursuing a male cougar agent vary with the female's energy level.

The female has similar pursuing behavior as the male, except the female never weights pursuing a male cougar agent above .5.

**Formulas:**

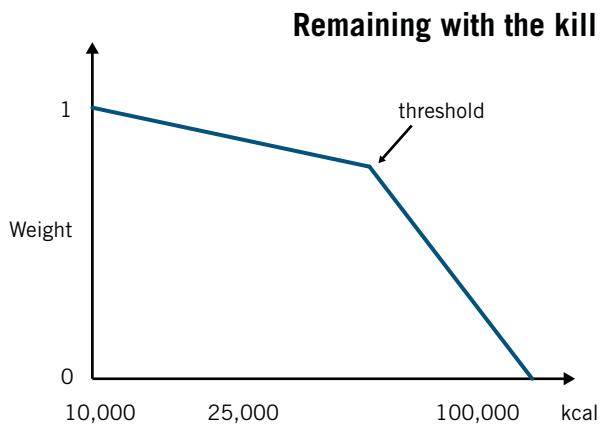
If farther than 2,500 meters from the boundary : weight = 0

If \leq to 2,500 meters from the boundary :

weight = $-(1/2,500) * (\text{distancefromboundary}) + 1$

This plot shows how the weights for staying within the home range increase as the cougar agents get closer to the home range boundary (a repellant).

The cougar agent establishes and generally remains within its home range. The formula assumes the home range boundary becomes a repellant only if the cougar agent is within a mile and a half of the home range boundary. As the male or female agents get closer to the edge of the home range, the weight of the repellant increases.



Formulas:

If $\text{kcal} \leq 66,000 \text{ kcal}$: weight = $-((1/194,117.64) * (\text{kcal})) + 1$

If $\text{kcal} > 66,000 \text{ kcal}$: weight = $-((1/51,515.15) * (\text{kcal} - 66,000)) + .66$

This plot shows how the weights for the remaining with the kill criterion (the kill attractor) change according to the cougar agent's energy level.

The cougar agent will stay with the kill (and the weight will slowly decrease) until the immediate need for energy is met (66,000 kcal). At that point, the weight for this criterion decreases dramatically because the cougar's interest in the kill diminishes as it becomes more full.

5

6

7

8

9

GIS data, fields, and actions dictionaries

Table 8.1 Data dictionary of GIS datasets

| Dataset | Data type | Description |
|----------------------|-------------------|--|
| badger | raster | Distribution of badgers |
| bobcat | raster | Distribution of bobcats |
| coyote | raster | Distribution of coyotes |
| deer | raster | Distribution of deer |
| elk | raster | Distribution of elk |
| FemaleCougars | shapefile—point | The locations of the female cougar agents each time step |
| fire | raster | The location of a burning fire. It is not actively used in the exercises. |
| firecostsm | raster | The susceptibility surface the fire moves across. It is not actively used in the exercises. |
| firestart | raster | The starting location of the fire. It is not actively used in the exercises. |
| HabitatCentroids | shapefile—point | The location of the habitat attractors. Derived from the probability of a kill dataset (mfood). |
| hillshade | raster | The hillshade of the study area |
| landusesm | raster | The land use types |
| MaleCougarHome-Range | shapefile—polygon | The home ranges of the male cougars |
| MaleCougars | shapefile—point | The locations of the male cougar agents each time step |
| mfood | raster | Probability of a kill—combines the hunting advantage for a cougar with prey density to determine for each location the probability of obtaining a kill. The roughness of the terrain is calculated by internal software from the USGS. The rougher terrain provides more hunting advantage to the cougars. |
| msecurity | raster | The level of security at each cell. The dataset defines the level of security for the cougar. The layer is mainly a combination of vegetation and distance from human activity. |
| porcupine | raster | Distribution of porcupines |
| Streams | shapefile—lines | The streams |
| StudyBoundary | shapefile—polygon | The boundary of the study site |

Table 8.2 Fields dictionary for the cougar model

| Field | Data type | Description |
|-------------------------|---|---|
| Cougar model | | |
| rasterPath | java.lang.String | The relative path to the animal distribution rasters |
| badgerMaleProb | double | The probability of badger in the male cougar diet |
| bobcatMaleProb | double | The probability of bobcat in the male cougar diet |
| coyoteMaleProb | double | The probability of coyote in the male cougar diet |
| porcupineMaleProb | double | The probability of porcupine in the male cougar diet |
| deerMaleProb | double | The probability of deer in the male cougar diet |
| yearlingElkMaleProb | double | The probability of a yearling elk in the male cougar diet |
| adultElkMaleProb | double | The probability of male elk in the cougar diet |
| scavengerElkMaleProb | double | The probability of a scavenger male elk in the cougar diet |
| matingOn | boolean | If the mating criteria is active |
| huntingOn | boolean | If the hunting criteria is active |
| messageDisplay | uchicago/src/simbuilder.util.MessageDisplay | A field for the message to display |
| MaleCougar agent | | |
| homeRangeBoundary | com.vividsolutions.jts.GeometryFactory | The polygon defining the home range boundary for the male cougar agent |
| factory | com.vividsolutions.jts.GeometryFactory | A geometry factory for storing the home range geometry |
| sharedPoint | com.vividsolutions.jts.GeometryFactory | The coordinates of a specified point geometry. Used in measuring the distance from the current cougar agent position to the home range polygon. |

5

6

7

8

9

Table 8.2 Fields dictionary for the cougar model (*continued*)

| Field | Data type | Description |
|-----------------------------------|--------------|---|
| needHabCentroid | boolean | Identifies if the male cougar agent needs to identify a habitat centroid to move toward |
| habCentroidPointX | integer | The x coordinate of the habitat centroid for the male cougar agent to move toward |
| habCentroidPointY | integer | The y coordinate of the habitat centroid for the male cougar agent to move toward |
| prey | Prey | The type of prey that was killed |
| preyTimeStart | double | Identifies the starting time for consuming a kill |
| targetFemale | FemaleCougar | The female cougar agent the male cougar agent has detected and will move toward |
| mating | integer | Indicates if the cougar agent is mating |
| matingTimeCount | integer | Identifies the starting time the cougar agent starts mating |
| energyReserve | double | Monitors how much energy the cougar agent has at each time step |
| minimumEnergyReserveForHunting | double | The minimum kcal threshold below which the cougar agent must focus all its effort on hunting to survive |
| classIActivityEnergyConsumption | double | The amount of calories used each time step when performing class I activity |
| classIIActivityEnergyConsumption | double | The amount of calories used each time step when performing class II activity |
| classIIIActivityEnergyConsumption | double | The amount of calories used each time step when performing class III activity |

Table 8.2 Fields dictionary for the cougar model (*continued*)

| Field | Data type | Description |
|---------------------------------|-----------|---|
| maximumEnergyReserve | double | Identifies the maximum amount of energy the cougar can have |
| securityweight | double | The weight determined for the security criterion for a given time step |
| homeRangeweight | double | The weight determined for the staying within the home range criterion for a given time step |
| huntingWeight | double | The weight determined for the hunting criterion for a given time step |
| femaleWeight | double | The weight determined for the pursuing a female criterion for a given time step |
| killedPreyWeight | double | The weight determined for the remaining with the kill criterion for a given time step |
| energyReserveForHuntingInterest | double | It is the minimum kcal threshold below which the cougar agent loses interest in hunting |
| Cell agent | | |
| x | integer | The x raster position of the cell agent |
| y | integer | The y raster position of the cell agent |
| mapX | double | The x map position of the cell agent |
| mapY | double | The y map position of the cell agent |
| Marbles | double | The number of marbles assigned to the cell agent |
| probMin | double | The minimum probability for a surrounding cell agent when calculating a distribution for determining a move |

5

6

7

8

9

Table 8.2 Fields dictionary for the cougar model (*continued*)

| Field | Data type | Description |
|-------------------------|------------------|---|
| probMax | double | The maximum probability for a surrounding cell agent when calculating a distribution for determining a move |
| Prey agent | | |
| prob | double | The probability of the prey in the male cougar diet |
| name | java.lang.String | The name of the prey agent |
| enabled | integer | Determines if the prey is available when a kill is made |
| rasterX | integer | The x location of a current kill |
| rasterY | integer | The y location of a current kill |
| rasterName | java.lang.String | The name of the raster containing the distribution of the prey |
| probMin | double | The minimum probability for the available prey agent when calculating a distribution for determining which prey is killed |
| probMax | double | The maximum probability for the available prey agent when calculating a distribution for determining which prey is killed |
| consumeDuration | double | How long it takes to consume the prey |
| calorieAmount | double | The total amount of calories gained from consuming the prey |
| hourlyConsumptionAmount | double | The amount of calories gained per hour consuming the prey |

Table 8.2 Fields dictionary for the cougar model (*continued*)

| Field | Data type | Description |
|---------------------------|-----------|---|
| FemaleCougar agent | | |
| homePointX | double | The x coordinate location of the habitat centroid the female cougar will move relative to |
| homePointY | double | The y coordinate location of the habitat centroid the female cougar will move relative to |
| isAvailable | integer | Indicates if the female cougar agent is available for mating |
| mating | integer | Identifies if the female cougar agent is mating |

5
6
7
8
9**Table 8.3** Actions dictionary for the cougar model

| Declared actions | Description |
|------------------------|--|
| Cougar model | |
| initAgents | Initializes the cougar model |
| updateDisplay | Updates the ArcMap display |
| writeAgents | Writes the location of the agents to the identified shapefile |
| setRasters | Identifies the location of a series of species-distribution rasters that will be used for the hunting criteria |
| showMessages | Displays messages to the user |
| setupHook | Deletes previous messages and establishes the “hook” into the mechanism that allows user code to be run |
| fire | Creates the fire model using ArcGIS Spatial Analyst |
| HomeRange agent | |
| step | The action to perform on the home range agent each time step |
| Cougar agent | |
| step | The action that is performed on each male cougar agent each time step |
| setFactory | Sets a geometry factory and is necessary to create geometry |

Table 8.3 Actions dictionary for the cougar model (*continued*)

| Declared actions | Description |
|------------------------------|---|
| setSharedPointCoords | Sets the coordinates of self.sharedPoint. When self.sharedPoint is used after calling this action, self.sharedPoint will then have those coordinates. |
| move | Moves the male cougar agent |
| init | Initializes the male cougar agent |
| createCells | Identifies each of the eight surrounding cells |
| setMarblesForSecurity | Sets the weights for the eight surrounding cells relative to the staying secure criterion |
| setMarblesForHomeRange | Sets the weights for the eight surrounding cells relative to the staying within the home range criterion |
| findHabitatCentroid | Locates the habitat centroid that will act as the habitat attractor for the male cougar agent |
| setMarblesForHabitat | Sets the weights for the eight surrounding cells relative to the moving toward good habitat/hunting criterion |
| calcMove | Performs the calculations for determining the move for the male cougar agent |
| hunt | Determines if a prey is killed and the type of prey that is killed |
| preHunt | Initializes the prey characteristics and determines if the prey is available at the site of a kill |
| setMarblesForPrey | Sets the weights for the eight surrounding cells relative to the hunting criterion |
| setMarblesForFemale | Sets the weights for the eight surrounding cells relative to pursuing a female criterion |
| mating | If the male agent is mating, the number of hours that the male agent is with the female agent is incremented. If the male agent is pursuing a female agent, the action determines if the male agent moved into position to mate with a female cougar agent. |
| calcWeights | Calculates the weights for each criterion based on the state of the male cougar agent |
| HabitatCentroid agent | |
| step | The action that is performed each time step on the HabitatCentroid agents |

Table 8.3 Actions dictionary for the cougar model (*continued*)

| Declared actions | Description |
|---------------------------|---|
| FemaleCougar agent | |
| step | The action that is performed each time step on the female cougar agents |
| setMarblesForHabitat | Sets the weights for the eight surrounding cells relative to the moving toward good habitat/hunting criterion. For simplicity, this is the only criterion used for determining the female cougar agent movement |
| createCells | Identifies each of the eight surrounding cells |
| move | Moves the female cougar agent |

5

6

7

8

9

Table 8.4 Field data from collared cougars: predicted energy expenditure and proportion of time spent by activity class

| Cougar agent | Avg. weight (kg) | Basal Metabolic Rate (BMR) | Class I activity | Class II activity | Class III activity |
|--|------------------|----------------------------|------------------|-------------------|--------------------|
| Energy expenditure (kcal/4 hrs) | | | | | |
| Female cougar | 45.5 | 204.4 | 61.3 | 715.4 | 1839.5 |
| Male cougar | 68 | 276.3 | 82.9 | 966.9 | 2486.4 |
| Total energy expenditure (kcal) | | | | | |
| Female cougar | 45.5 | 204.4 | 265.7 | 919.8 | 2043.9 |
| Male cougar | 68 | 276.3 | 359.2 | 1243.2 | 2752.7 |
| Proportion of time by activity class | | | | | |
| Female cougar | 45.5 | 204.4 | 0.5605 | 0.3865 | 0.053 |
| Male cougar | 68 | 276.3 | 0.487 | 0.3420 | 0.171 |
| Average speed of movement (meters per hour) | | | | | |
| Female cougar | 45.5 | 204.4 | 42 | 510.9 | 1321.7 |
| Male cougar | 68 | 276.3 | 46.7 | 556 | 1494.7 |

Section 3: Advanced techniques for points, polygons, and rasters

Chapter 9

Adding complexity to agent movement on representative networks

by Elizabeth R. Groff and Mary Jo Fraley

- ◆ Building an advanced vector movement model
- ◆ Creating routes consisting of street nodes
- ◆ Defining and using activity spaces in Agent Analyst
- ◆ Moving agents around predefined activity spaces
- ◆ Using the cartographic capabilities of the ArcMap interface to represent agent movement
- ◆ Incorporating variety into activity spaces

The ability to move agents randomly on a network provides the foundation for representing spatially realistic movement. However, modeling human activity requires more than simple random movement. Other factors must be integrated into the model to achieve a more realistic representation. For example, human activity is typically goal-directed; we travel to get to specific locations. Research also suggests that human activity is patterned and consists of the locations/nodes people visit and the paths/routes they take between those places (Hägerstrand, 1973; Horton and Reynolds, 1971; Lynch, 1960). Typically, some locations/nodes are visited routinely; for example, home or a workplace. Other locations are visited often but not daily; for example, grocery stores, gyms, coffee shops, and the like. The particular locations that we visit are shaped by the distribution of opportunities to undertake the desired activity and by individual preferences. The distribution of available opportunities is one type of constraint on human travel, and individual preferences represent another.

In this chapter, you will first explore the constraint related to the distribution of locations across a landscape. The overall distribution of locations for human activity is influenced by the patterns of land use in a city. Some areas are predominantly residential, other areas are office or warehouse districts, and still other areas are predominantly commercial. Specific opportunities for housing, employment, and shopping are influenced by where housing units, employers, and retail stores are located within these areas. For example, when deciding where to buy milk, we must find a store that sells milk, and this limits our options to a smaller set of retail stores such as groceries.

At the individual level, each person chooses a specific home, obtains a particular job or chooses a college to attend, and picks specific recreational and shopping locations that fit both proximity and preference criteria. So, in the case of a grocery store, the particular grocery store we choose is a reflection of our preferences about store proximity, prices, cleanliness, and so on. Of course, these are not the only constraints on human activity. Another important constraint involves timing. Each person has a unique set of obligations that have temporal components to them. This set of time constraints influences when a person leaves one place in order to arrive at another as well as how long they stay there. Time constraints also influence and are influenced by the mode of transportation taken. For a more complete discussion regarding time, geography, and the constraints on human activity, please see Hägerstrand (1970, 1973) and Pred (1967, 1969, 1977) as well as others (Carlstein, Parkes, and Thrift, 1978; Thrift and Pred, 1981).

The current example of moving agents along a representative vector network extends the simple movement model discussed in chapter 6. This chapter demonstrates how data about the physical environment of a city can be used to create more realistic travel behavior in an agent-based model. Specifically, you'll learn how to develop and implement activity spaces for agents in the model.

The modeling scenario

To simulate travel that is more realistic than the random movement introduced in chapter 6, this extension to the model has agents move according to a predefined path. However, the inability of Agent Analyst to connect to network databases makes dynamic routing of origins and destinations selected during the model run impossible. The method outlined here is an alternative that can be used until this functionality is included in Agent Analyst. Once again, the focus is on modeling human travel patterns, but the same functionality could be applied in transportation, utilities, and medical research, as well as with people.

Background information

We know that people have activity spaces, which consist of the places they visit frequently and the routes they take between those places. Important places are called anchor points and are visited the most frequently of all places. An individual's activities tend to be centered on their anchor points. More specifically, research reveals that human activity is associated with the distribution of opportunities for housing, jobs, shopping, recreation, and so on. Locations with a high concentration of jobs tend to be anchor points related to employment for more people. In the same way, locations with high-density residential land use tend to be home anchor points for more people. Areas with concentrations of shopping, recreation, and other services that are frequently used also tend to be part of many activity spaces because of the regularity with which they are visited.

Unfortunately, data describing the activity spaces of individuals is not typically collected. Thus, agent-based modeling provides an alternative in that we can model human behavior from general rules, also called heuristics. These general rules about human travel are used to develop representative activity spaces. In this example, each citizen agent in the model has four anchor points (i.e., locations they visit regularly): a home, a work/school location, and two recreation/shopping locations. Since there is no way of knowing which residents of a neighborhood are patronizing individual stores, the relative concentrations of those locations in a place are used to represent the relative concentrations in the model.

The following methodology uses empirical data describing the physical and social environment of a city to realistically distribute the activities of agents in the city. In the model presented here, human agents in the model have four nodes that represent their activity spaces. An example using the assignment of home nodes is illustrative of the process that was applied for homes, jobs, and recreation. For example, if 5 percent of the people in a city live in a particular block group, then 50 of the 1,000 agents in a model are assigned to live in that block group. For a more detailed explanation of how agents are assigned activity nodes, please see previous work by Groff (2006, 2007). First, the population of agents is distributed across all block groups as just described. Second, all agents are assigned a specific home node from their block group. Said another way, the 50 agents just mentioned who live in a particular block group are randomly assigned home street nodes within that block group. Each block group is then processed until

5

6

7

8

9

all agents have been assigned a home node. The same procedure is repeated for jobs and for recreation. Since this is not directly related to Agent Analyst functionality, a preprocessed file is included that contains an agent name and the four nodes they are to visit.

Once each agent has four node locations assigned, the route between those nodes is calculated. All agents start the day at home, go to work/school, and then go to their assigned recreation nodes. The amount of time each agent spends traveling is dependent on the distance that must be covered to make it to all four places before the end of the day.

To include some variation in how agents travel, the number of blocks they travel each model tick can be varied. For example, in one tick an agent may travel four blocks, and the next tick they may travel only two blocks. This allows the model to reflect the agent “getting caught at a street light” or “hitting all green lights.”

Building an advanced vector movement model

The following exercises expand on the network movement principles that you learned in chapter 6. Instead of moving the agents randomly along a street network, you will learn how to have agents move along a street network purposefully (i.e., along routes that connect a set of places and represent the route that each agent travels in a day). This will provide you with the ability to develop agent activity spaces in ArcGIS and then have the travel behavior of the agents in your model reflect those activity spaces. This allows a much more realistic depiction of agent travel behavior than random movement. We are again using a street centerline file for this exercise, but any set of connected line segments can be used.

As mentioned earlier, these steps are necessary because Agent Analyst cannot read network datasets. This is an alternative work flow to achieve a similar result. The activity spaces used here were defined using the following steps. First, each agent was randomly assigned a set of places to visit. Second, ArcGIS Network Analyst is used to find the best path between places. Third, the route is written to a file and followed by the agent during the course of a model day.

The next section starts with an introduction to a completed model that enables the use of activity spaces on representative networks. During the course of doing these exercises, you will learn how to create an activity space from a list of nodes using ArcGIS Network Analyst, how to read the list of nodes representing the activity space into Agent Analyst, how to have agents use the activity space for movement decisions, and how to display the results of agent movement in ArcMap.

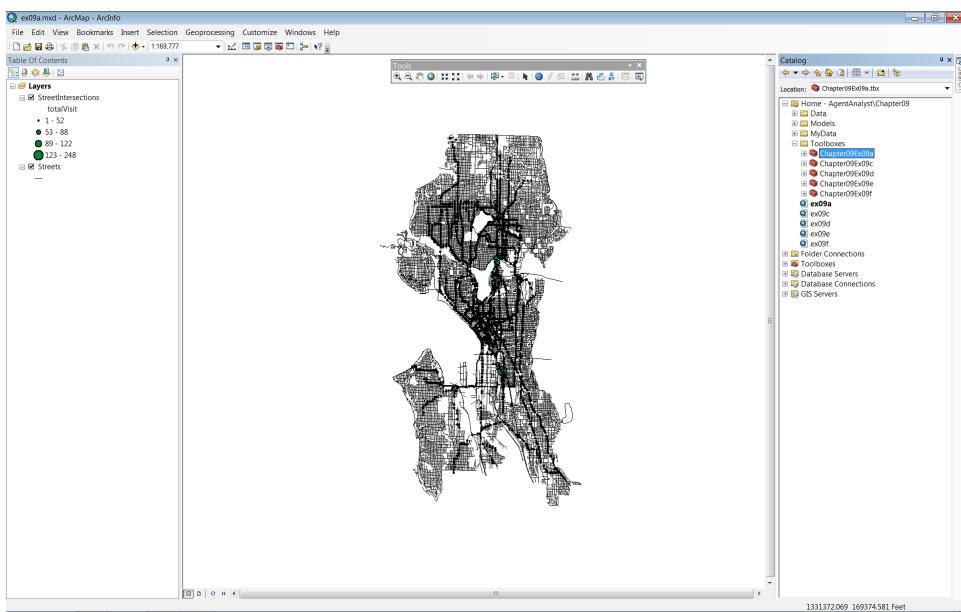
Exercise 9a

This exercise introduces you to the basics of a more advanced model that enables movement along a vector network. You will learn how to create this model during the course of completing exercises 9a–9f.

Load the ArcMap document

- 1 Start ArcMap. Navigate to C:\ESRIPress\AgentAnalyst\Chapter09. Open ex09a.mxd.

The map document opens. In the ArcMap display, you will see the Streets line shapefile and a StreetIntersections point shapefile for Seattle. In ArcToolbox, the Chapter09Ex09a toolbox has been created for you.



Exercise 9a

The StreetIntersections shapefile contains the points that represent street intersections in the model. The activity spaces of agents were created using ArcGIS Network Analyst and then converted to a list of intersections that represent the agents' daily paths between sets of activity nodes. Agent movement follows the list of nodes in the agent's activity space (i.e., the list of nodes that make up the agent's daily route). Each time an agent moves, the agent selects the next node from the list of nodes in its activity space. This design reduces processing time because the routing of agents occurs outside Agent Analyst (i.e., as mentioned before, the shortest path is produced in an earlier step using ArcGIS Network Analyst).

Explore the attributes

- Right-click the StreetIntersections layer, and click Open Attribute Table.

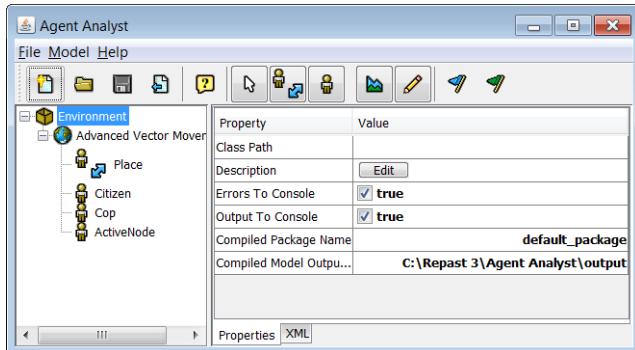
Each row is a street intersection, and each column or field contains information about the intersection. The field you will be working with in this exercise is totalVisit.

StreetIntersections is also the shapefile to which the model writes output information related to agent movement between places. A cumulative total of citizen agent visits to each node is kept during the model run and written out periodically to the totalVisit field in the shapefile.

Load the Agent Analyst model

- Right-click the Chapter09Ex09a toolbox, point to New, and select Agent Analyst Tool.

- 4 Once the Agent Analyst window appears, click File and select Import. Do not save the changes when prompted.
- 5 Navigate to C:\ESRIPress\AgentAnalyst\Chapter09\Models and open ex09a.sbp. Now the model for exercise 9a is loaded into Agent Analyst and ready for use.



There are four agent types in the Advanced Vector Movement model. The place agent is all the nodes that are part of our network. Citizen agents are the agents that you will move based on their routines. Cop agents move randomly throughout the places in the model. The ActiveNode agent is the current active node for the cop and citizen agents.

Explore the place agent

- 6 In the Environment panel, click Place. In the Property panel, click the Edit button to the right of the Data Source property to open the Data Source Editor.

Since this is a stripped down version of a robbery model, the only functionality implemented is movement. The place agent is created from the StreetIntersections shapefile. These are all the locations that the agents can move to within the model. Also notice the totalVisit field, which will be used to symbolize how many agents have visited a particular place.

Run the model

- 7 To execute the model, click the Run button in Agent Analyst and then click Start on the Repast toolbar.

The graduated symbol classification scheme was created using data from a day's worth of agent movement. The classification scheme persists and can be used to examine the results of agent movement in subsequent runs. Observe how the graduated symbols representing agent movement on the map begin to appear and grow larger as the number of visits accumulates across the nodes in Seattle.

This pattern of aggregate agent travel is very different from the pattern generated by random movement in chapter 6. Although there are only 20 agents in this sample model, it is clear that agents here have a travel routine they follow. The same roads tend to have the majority of visits. In addition, as expected, arterial roads also have more travel than residential roads because they tend to be more direct routes than residential roads and are attached to bridges that allow travel over water bodies. Recall that each tick represents one minute of model time. An entire day is represented by 1,440 ticks.

- 8** To further explore the model results, reopen the attribute table for StreetIntersections.
- 9** Right-click the totalVisit field and select Sort Descending. This allows you to see the actual number of times each node was visited by agents during the model day.

Close Agent Analyst and ArcMap

- 10** Close the model and close ArcMap. Do not save any changes.

Creating routes consisting of street nodes

In this exercise you will create a network dataset and use this network dataset to create a route for an agent. A route consists of the four nodes/places an agent visits in a day and the set of streets (represented here as nodes) the agent must traverse to travel between the nodes. The places visited and the routes taken make up the agent's routine activity space. For this model, the places represent a home node, a work/school node, and two recreation/shopping nodes. Typically, when creating routes you focus on the lines (representing streets) that make up the routes. Since Agent Analyst does not support routes, you need to convert the routes to the list of nodes that make up the route instead. Nodes are located at intersections in the street network.

5
6
7
8
9

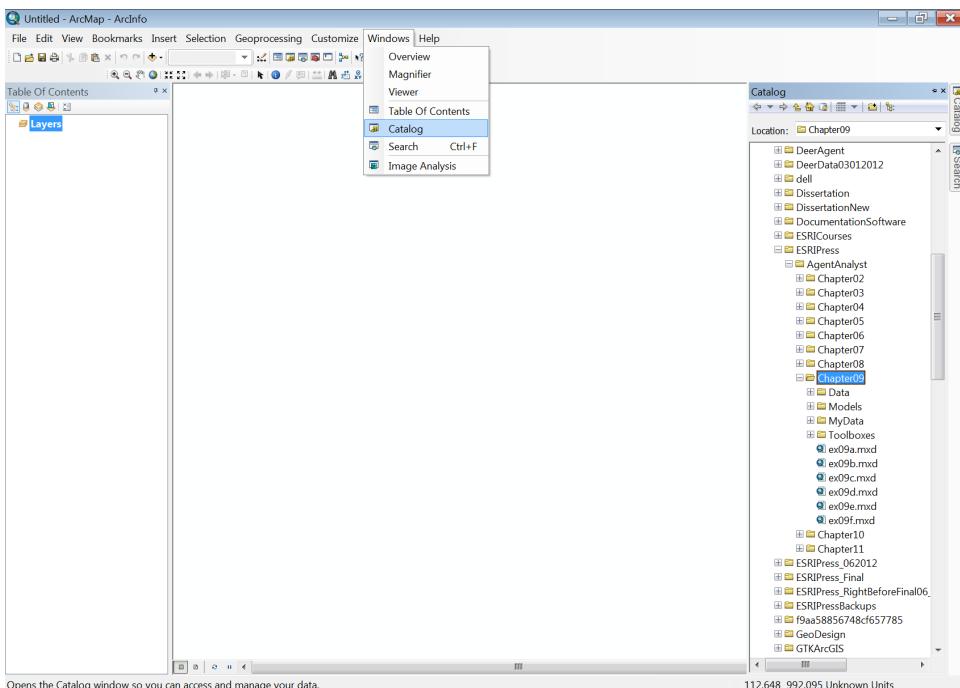
Exercise 9b

First you need to create a network dataset that will consist of the Street Nodes and Streets shapefiles that you created and used in chapter 6. You need to turn on the Network Analyst Extension if it is not already available.

Create a network dataset

- 1 Start ArcMap.

When ArcMap opens, if ArcCatalog is not already open, click Windows and then click Catalog.



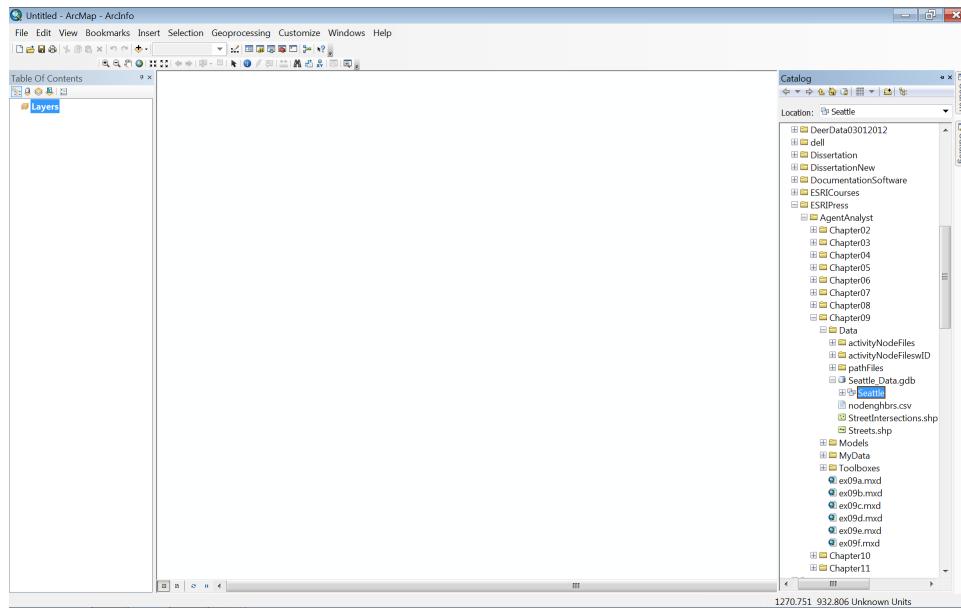
- 2 In ArcCatalog, navigate to C:\ESRI\Press\AgentAnalyst\Chapter09\Data. Expand the Data folder, and you will see a file-based geodatabase called Seattle_Data.

Note: If not already connected, you might need to connect to the C drive using the Connect To Folder button and selecting C:.

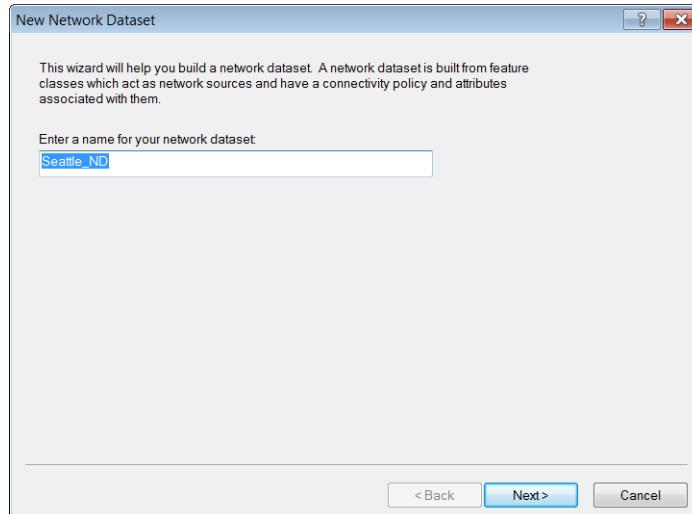
You need to create a network dataset from the streets so that you can generate routes for your agents.

- 3 Expand Seattle_Data, right-click Seattle, point to New, and then click Network Dataset, as shown in the following figure.

Note: If you do not have Network Analyst installed, the Network Dataset option will not be available. To check whether you have Network Analyst licensed and installed but not enabled, click Customize on the ArcMap menu and then click Extensions. Select the check box for Network Analyst. If you cannot enable Network Analyst, you will not be able to complete this exercise. In that case, read through the remaining steps and then continue with exercise 9c.

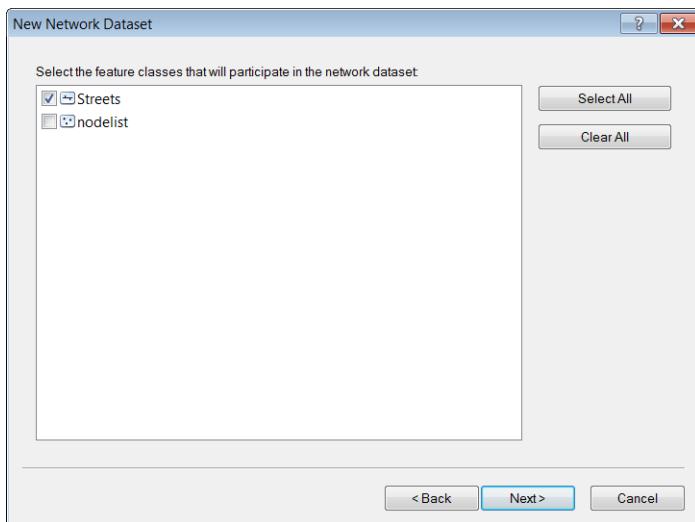


- 4 When the New Network Dataset dialog box appears, name the network dataset **Seattle_ND**, and then click Next.

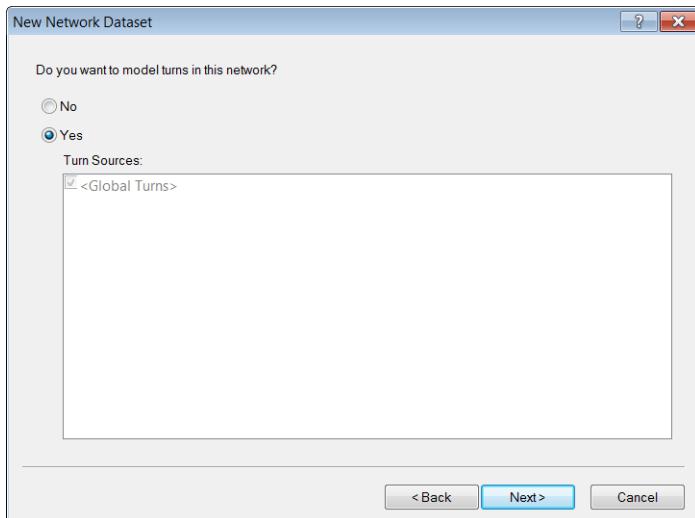


5
6
7
8
9

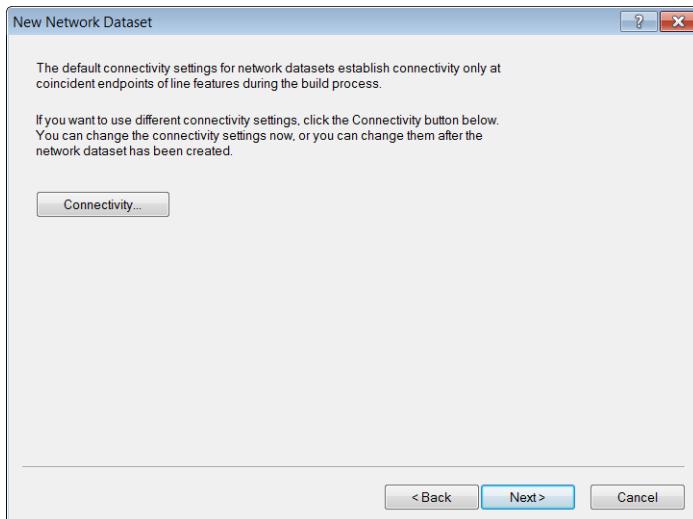
- 5 Select the check box for Streets to select the feature class that will participate in the network dataset, and then click Next.



- 6 On the “Do you want to model turns in this network?” page, select Yes and click Next.



The dialog box for defining the connectivity settings appears.

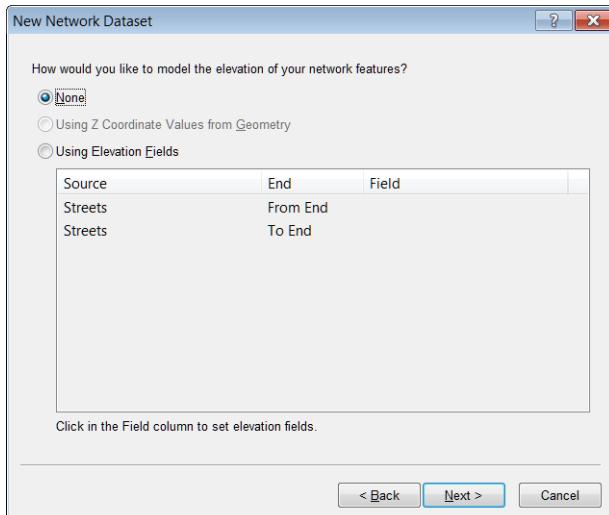


5
6
7
8
9

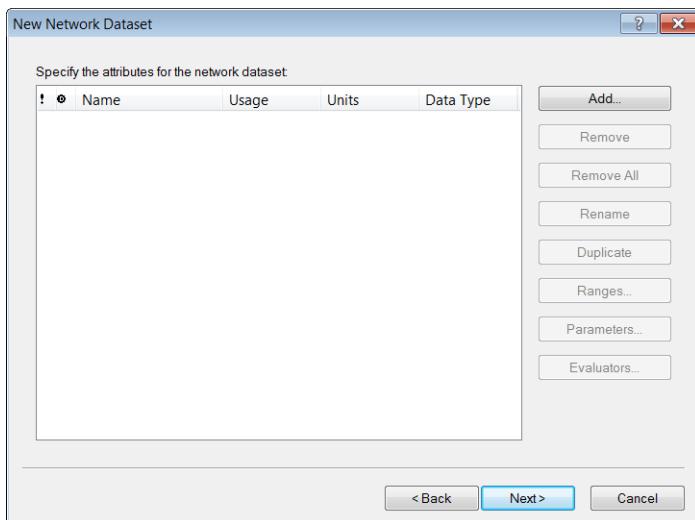
- 7 Accept the default connectivity settings by clicking Next.

The dialog box for defining the elevation of your network features appears.

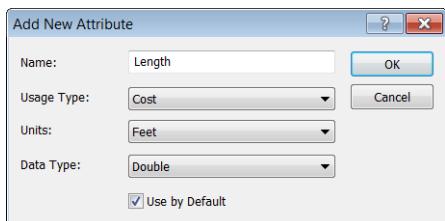
- 8 In the following dialog box, select None and then click Next.



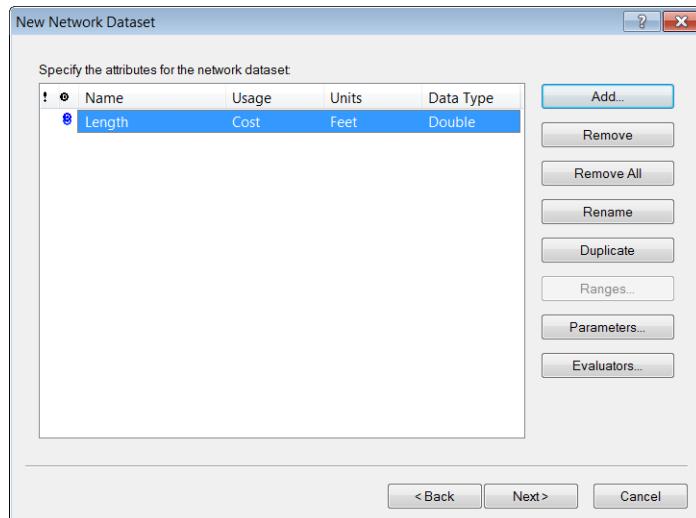
- 9 When the “Specify the attributes for the network dataset” page appears, click the Add button to add an attribute.



- 10 When the Add New Attribute dialog box appears, type **Length** for the name and select Feet from the Units list. Click OK.

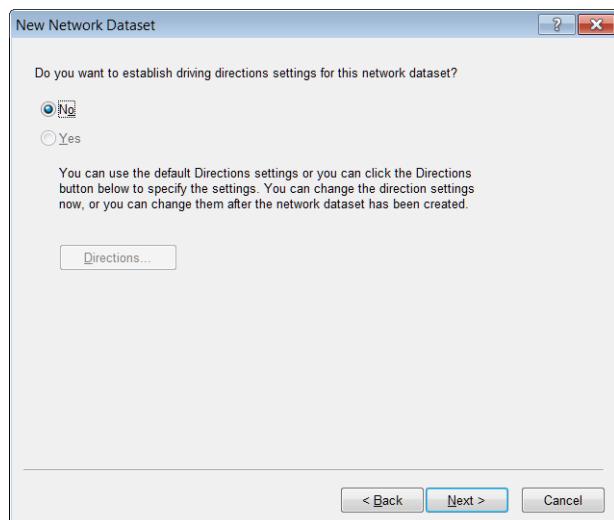


- 11 In the New Network Dataset dialog box, you will see the attribute you just defined, as shown in the following figure. Click Next.

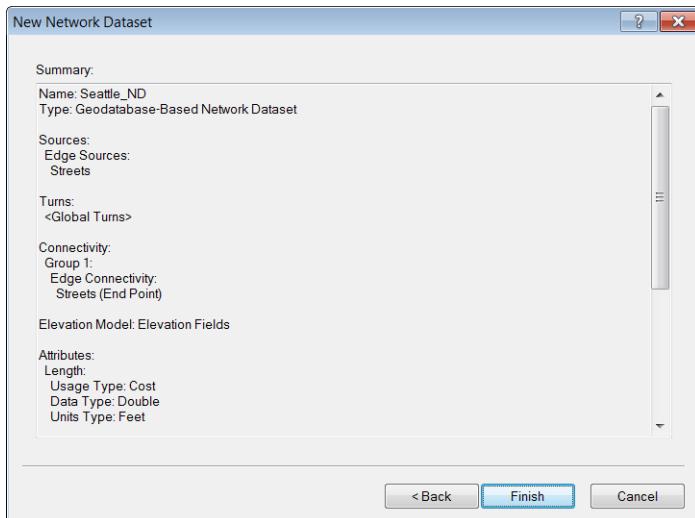


5
6
7
8
9

- 12 When the dialog box shown in the following figure appears and asks whether you want to establish driving directions for the network dataset, select No, and then click Next.

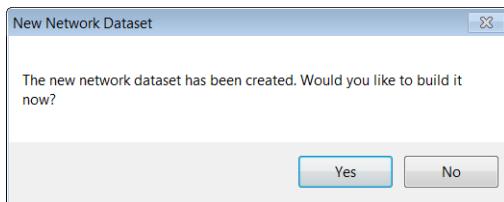


- 13** You are presented with a summary of what you have entered throughout the wizard, as shown in the following figure. Click Finish.



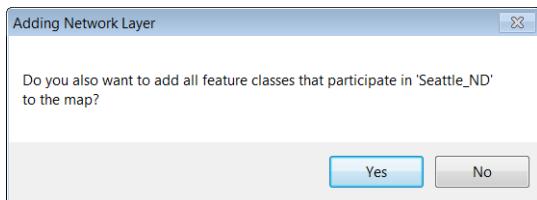
Your new network dataset has been created.

- 14** When asked if you want to build your network dataset now, click Yes.



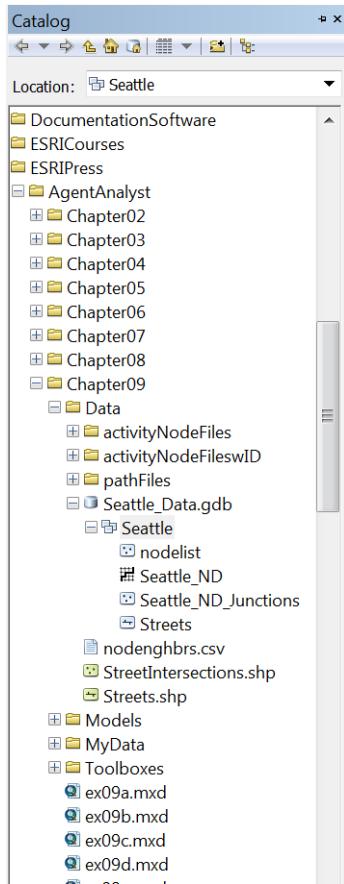
You will see a message box showing the progress of the build.

- 15** After the network dataset is built, you are presented with the dialog box shown in the following figure. Click No.



You have now created your network dataset, Seattle_ND, as shown in ArcCatalog in the following figure.

Note: These are the same steps you would follow to create a network dataset for use outside Agent Analyst. You are ready to create paths for the agents using this network dataset.



Now you will create the routine activity spaces for the agents using the places each agent visits and the street network dataset you just created.

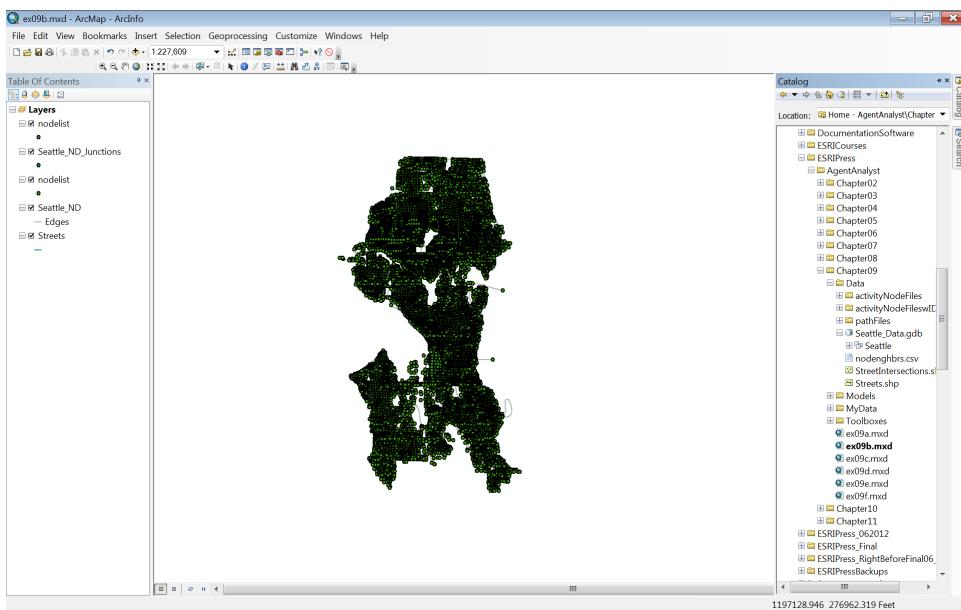
Open the ArcMap document

- 16** Navigate to C:\ESRIPress\AgentAnalyst\Chapter09 and open ex09b. Answer No if you're prompted to save your changes to Untitled.

Note: If you do not have Visual Basic for Applications (VBA) installed on your computer, you may receive a warning. Ignore the warning, and click OK. Continue with the exercise.

In the ArcMap Table Of Contents, you will see the nodelist layer.

- 17** Add the network dataset that you just created to ArcMap by selecting the Add Data button () and navigating to C:\ESRIPress\AgentAnalyst\Chapter09\Data. Select Seattle_Data.gdb, and then click Add.
- 18** In the Add Data dialog box, select Seattle and click Add.

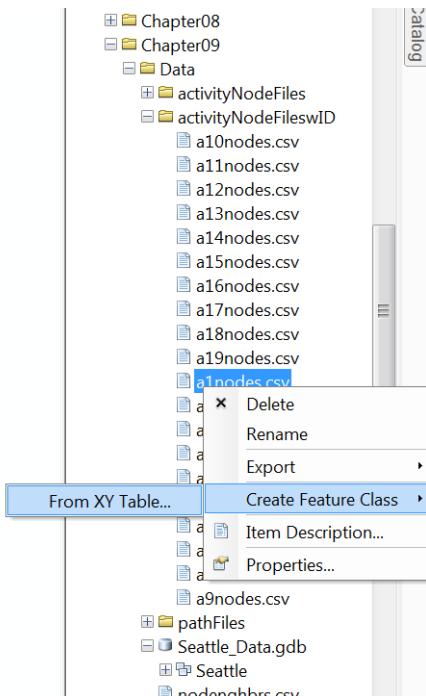


Now that the network dataset is added to ArcMap, you can generate a path that our agent will travel between activity locations. You can do this by using the Network Analyst Extension to ArcGIS. First you need to add the comma-delimited (.csv) text file (supplied in your Data folder) to the ArcMap document as XY Data. Each file belongs to a different agent (e.g., a1nodes.csv has the nodes for agent a1) and contains a list of the nodes the agent visits in a day.

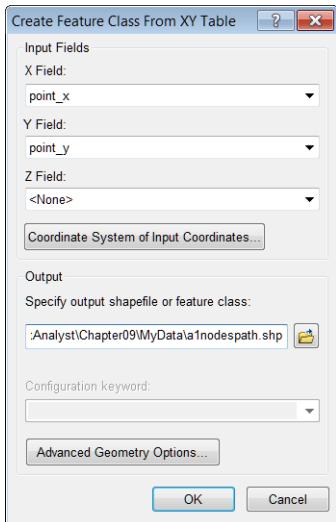
Note: In this exercise you will be creating the paths between the activity locations for one agent. This process would need to be repeated for each agent. To save time, we have already created the paths for the other agents (a total of 20) for you.

Add the .csv file that contains the activity nodes that will be used to create the network paths

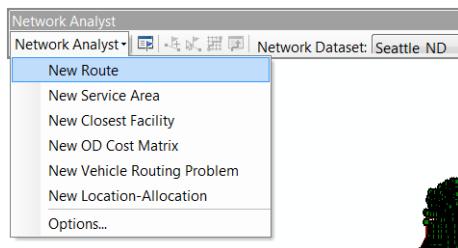
- 19 In ArcMap, go to ArcCatalog and navigate to C:\ESRIPress\AgentAnalyst\Chapter09\ Data. Expand the Data folder, and then expand activityNodeFileswID. Right-click a1nodes, point to Create Feature Class, and then click From XY Table, as shown in the following figure.



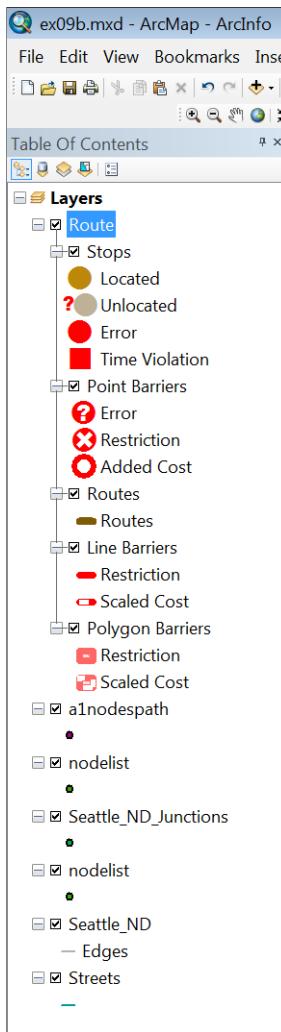
- 20 In the Create Feature Class From XY Table dialog box, you need to select the field that contains the x value (point_x) and the field that contains the y value (point_y). Also set the path to the output file C:\ESRIPress\AgentAnalyst\Chapter09\MyData\ a1nodespath.shp, as shown in the following figure. Click OK.



- 21** In the ArcMap display, click the Add Data button and browse to the MyData directory. Add the newly created shapefile, and you should see your alnodespath added to ArcMap as a layer.
- 22** Enable the Network Analyst Extension if it isn't already turned on. You can double check that the extension is turned on by going to Customize, Extensions and selecting the check box for Network Analyst. To display the Network Analyst toolbar, on the Customize menu, point to Toolbars, and then select Network Analyst.
- 23** On the Network Analyst toolbar, open the Network Analyst drop-down list and select New Route.

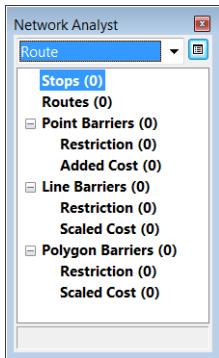


This will add a route analysis layer to the ArcMap Table Of Contents.



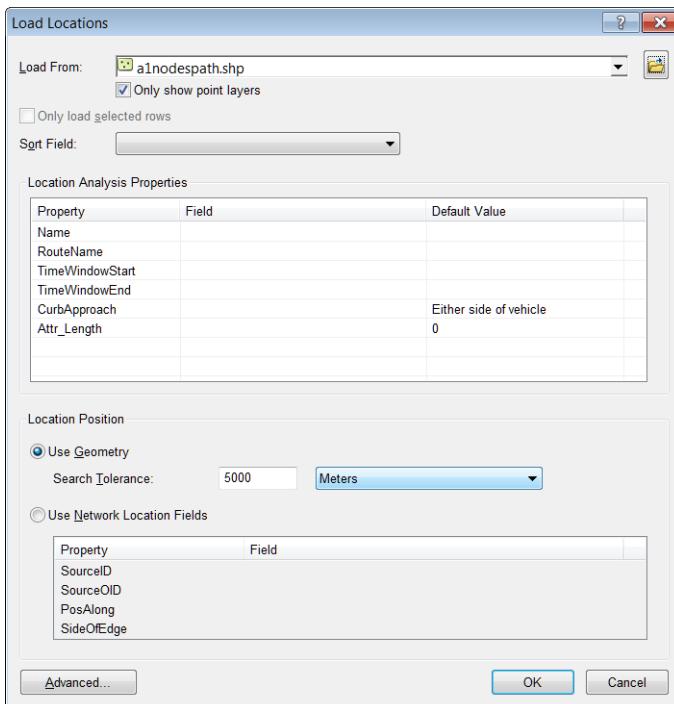
5
6
7
8
9

- 24 On the Network Analyst toolbar, click the first button (Show/Hide Network Analyst Window), to open the Network Analyst window shown in the following figure. You will see the stops, routes, and barriers that are part of this analysis layer.

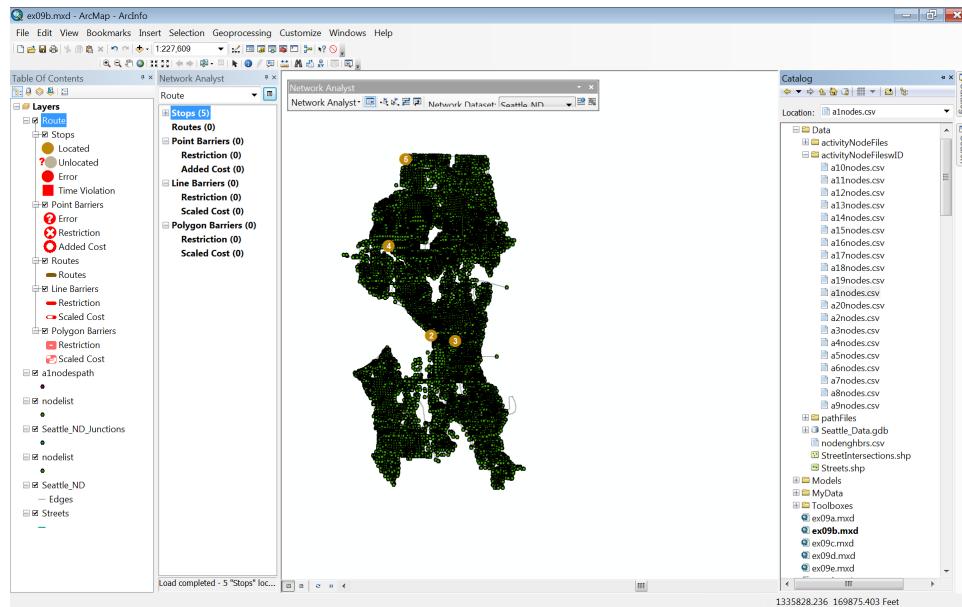


Identify the stops for the network route

- 25 In the Network Analyst window, right-click Stops and then click Load Locations....
- 26 In the Load Locations dialog box, click the Browse button and navigate to C:\ESRIPress\AgentAnalyst\Chapter09\MyData. Select the a1nodespath.shp shapefile and click Add to return to the Load Locations dialog box, as shown in the following figure. Click OK.



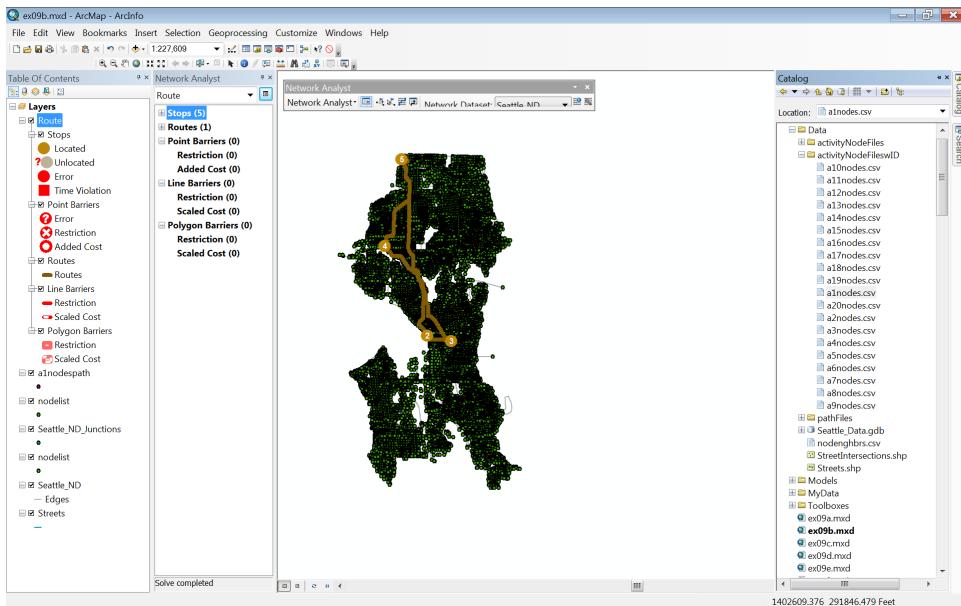
You should now see five stops (identified from the .csv file) loaded for your route analysis layer within the Network Analyst window and on your map, as shown in the following figure. (The first stop is not visible as it is under the fifth stop; remember the agent starts from home and returns home in this model, so stops 1 and 5 are always the same location.)



Once the stops have been added you are ready to solve the route. These are the places in an agent's routine activity space. The next step will find the shortest path between them.

- 27** Click the Solve button () on the Network Analyst toolbar. This creates a route for the provided stops.

5
6
7
8
9



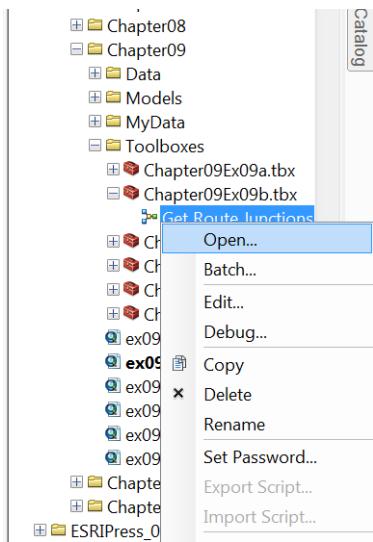
You now have a route. However, because Agent Analyst cannot read this route for our agents to travel, you need to create a list of nodes in the order they are traversed from this route.

Create the list of nodes for this route

To create a list of nodes, you need to add to the map the junctions that make up the route. These junctions are the nodes representing street intersections. To obtain the junctions, you need to run a geoprocessing model to generate your final output of ordered junctions. Since the agents in the model visit their places in a particular order, it is important that the junctions within the route are in the correct order.

- 28** In ArcToolbox, the Chapter09Ex9b toolbox has been created for you, as shown in the following figure. The toolbox contains the Get Route Junctions ArcGIS ModelBuilder model to select the junctions that make up the route. Open the model.

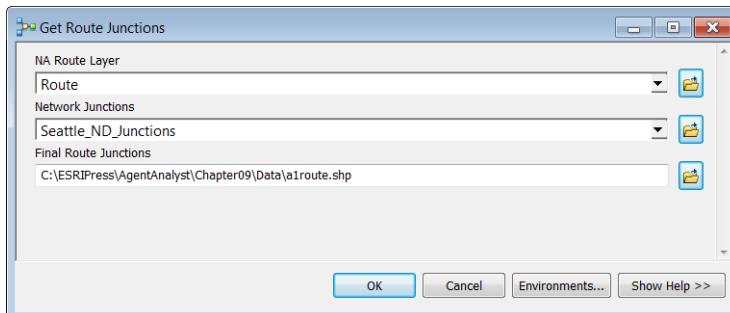
Note: This tool requires the ArcInfo license of ArcGIS. If you do not have a license, read through the remaining sections of this exercise and then proceed to exercise 9c.



5
6
7
8
9

The model takes three parameters, which should be populated as shown in the following figure:

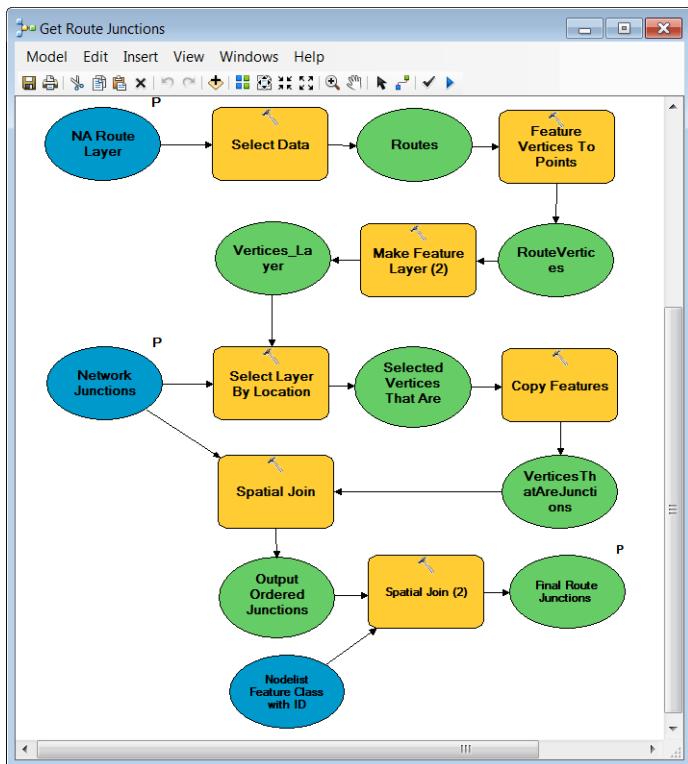
- NA Route Layer is the route that you just created when solving the new route.
- Network Junctions are the Seattle_ND_Junctions that were created when you built the network dataset and are also on the map.
- Final Route Junctions is the shapefile that will be written out containing the list of junctions that make up the route.



- 29** Click OK to run the model and generate the output junctions shapefile.

To save time, the other 19 agents' paths are already generated and available to you under C:\ESRIPress\AgentAnalyst\Chapter09\Data\pathFiles. Use these files for the rest of this chapter's exercises.

The next section explains the steps in the model that you just ran, which are illustrated in the following figure.



The geoprocessing model takes the Network Analyst route layer that you created when you solved the route and uses Select Data to select only the route that was created. Next, Feature Vertices To Points converts the route from a line to its vertices. Make Feature Layer is used to pass this layer to Select Layer By Location, which uses the Network Junctions from the network dataset and selects only the junctions that make up the route that was created. These selected junctions are written to an in-memory feature class and a Spatial Join is done to get the proper order of the junctions. Another Spatial Join is done to get the unique ID from the nodelist that was created in chapter 6 onto these junctions so that the IDs for the nodes that are read into Agent Analyst match the node IDs of our routes.

30 Close ArcMap without saving your changes.

In this exercise you created a network dataset to allow the routing of an agent's activity space within ArcMap. Using Network Analyst you created a route analysis layer and loaded the locations of stops for your first agent and created a route. Then you ran a geoprocessing model to take the route that was created and write out the junctions with a unique ID for the route that you will use later in Agent Analyst to simulate agent movement.

Defining and using activity spaces in Agent Analyst

Activity spaces of agents have spatial and temporal components. The spatial component consists of the places visited and the routes between those places and is defined using two files. One file lists the home, work, and recreation nodes. The other file contains a list of street nodes that have to be traversed to visit those places during the course of a day. Together these files define the spatial component of an activity space and have to be read into the model before they can be used by the agents to travel anywhere. The temporal component has to do with the length of time an agent spends at each place and in transit. While we are not discussing the temporal aspects of agent activity spaces in depth here, they can be addressed through the creation of a text file that lists the time to spend at each node and in transit (see Groff, 2006).

Each file of places and street nodes comprising the route are read in separately. This enables more than one potential activity space to be read into Agent Analyst for each agent. Using multiple activity spaces per agent allows differences in weekday versus weekend activities to be reflected in the model. The ability to randomly choose which activity space the agents will be using each day is a way to incorporate more dynamic randomness into a model to better reflect real-life travel. You will not be implementing multiple activity spaces for each agent in this example, but keep it in mind as you build your own models. In this exercise you learn how to move an agent along the shortest paths between each set of activity nodes. This simulates the routine travel paths of an agent during the course of a day.

Exercise 9c

Load the ArcMap document

- 1 Start ArcMap. Navigate to C:\ESRIPress\AgentAnalyst\Chapter09. Click ex09c.mxd.

Note: If you do not have VBA installed on your computer you may receive a warning. Ignore the warning and click OK. Continue with the exercise.

In ArcMap you will see the street intersections. In ArcToolbox, the Chapter09Ex09c toolbox has been created for you.

Load the Agent Analyst model

- 2 Right-click the Chapter09Ex09c toolbox, point to New, and select Agent Analyst Tool.
- 3 When the Agent Analyst window appears, click File and select Import. Do not save the changes when prompted.
- 4 Navigate to C:\ESRIPress\AgentAnalyst\Chapter09\Models and open ex09c.sbp.

Set up fields for directed movement

To implement directed movement along a path, the citizen agents need additional fields for handling the anchor places (home, work, rec1, rec2) as well as their paths of travel.

- 5 In the Environment panel, click Citizen. In the Property panel, click the Edit button to the right of the Fields property to open the Fields Editor.
- 6 In the Fields Editor, define the fields shown in table 9.1. The table has the field names and settings necessary for the fields to hold information about the citizen agents' activity spaces. Click OK when you are done.

Note: When adding fields, be sure you click the Add button after each entry. For the last two fields, you need to enter **java.util.ArrayList** in the Type box; it is not in the predefined list of types.

Table 9.1 Fields Editor data

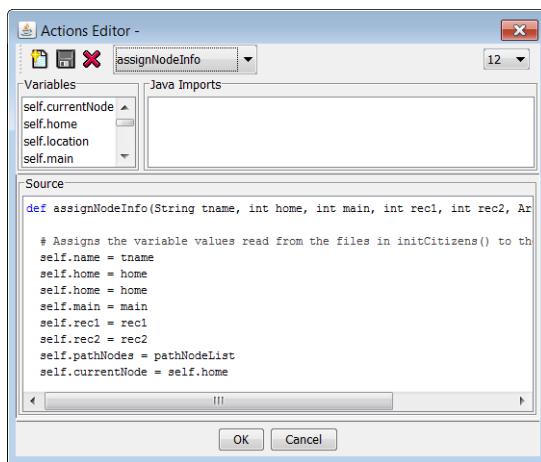
| Name | Type | Default value | Accessible | Parameter |
|-------------|---------------------|---------------|------------|-----------|
| home | integer | | Yes | |
| main | integer | | Yes | |
| rec1 | integer | | Yes | |
| rec2 | integer | | Yes | |
| position | integer | | Yes | |
| currentNode | integer | | Yes | |
| pathNodes | java.util.ArrayList | | Yes | |
| nodeList | java.util.ArrayList | | Yes | |

You should have a total of eight fields when you are done.

Now you have attributes (at this point empty fields) created to hold the information about activity nodes and the path nodes between them. You also need a way to bundle all the bits of data about each in order to populate the fields. Because activity spaces are complex, you need to build an array of objects with each object having its own values. The way to accomplish this is with an action.

Define an action to populate fields

- 7 In the Environment panel, click Citizen. In the Property panel, click the Edit button to the right of the Actions property to open the Actions Editor.
- 8 In the Actions Editor, select assignNodeInfo from the Actions drop-down list. Uncomment the code in the Source panel so that it matches the code in the following image. Click OK.



Adding the assignNodeInfo action is important because to achieve directed movement, the citizen agent needs to know more than just its current node (as in random movement in chapter 6). In advanced movement, the citizen agent needs to maintain its activity nodes (home, main, rec1, and rec2) and be able to identify the places that correspond to these locations. Citizen agents also need to maintain the set of nodes (path node list) that they traverse in the course of visiting their activity nodes as well as their current position in their list of path nodes. To accomplish this you utilize the fields that you added earlier for the citizen agent and use the assignNodeInfo action to populate these values for each individual citizen. The assignNodeInfo action populates the fields by reading in the files for each citizen agent.

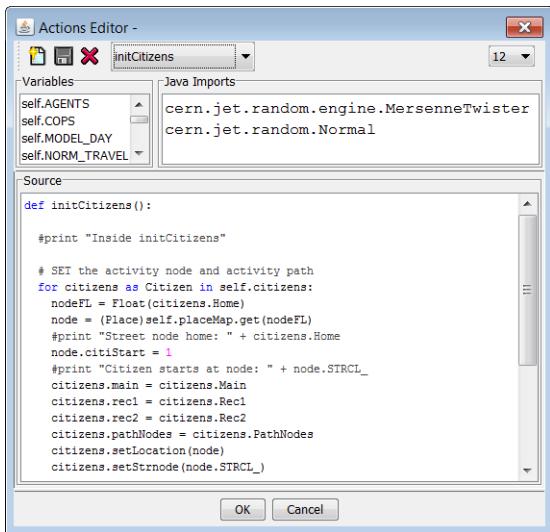
To use the information collected in the assignNodeInfo action, you have to make changes in the initCitizens action. This exercise also uses random number distributions. To use random number distributions, you need to specify the required Java classes in the Java Imports pane. Two classes are needed, one to specify the type of pseudo-random number generator (`cern.jet.random.engine.MersenneTwister`) and another to specify the type of random number distribution (`cern.jet.random.Normal`). The Mersenne Twister

5
6
7
8
9

pseudo-random number generator is one of the most robust and also one of the quickest. It is often used in scientific research.

Initialize your citizen agents

- 9 In the Environment panel, click Advanced Vector Movement to get to the model level. In the Property panel, click the Edit button to the right of the Actions property to open the Actions Editor.
- 10 In the Actions Editor, select initCitizens from the Actions drop-down list and examine the code in the Source panel. The action populates each agent's attributes for the home, main, rec1, and rec2 nodes as the agent is created.



In this code, the activity node and the activity path are being set for each citizen within the loop (node = (Place) self.placeMap.get (nodeFL))

Then the main, rec1, rec2, and pathNodes variables are set:

```

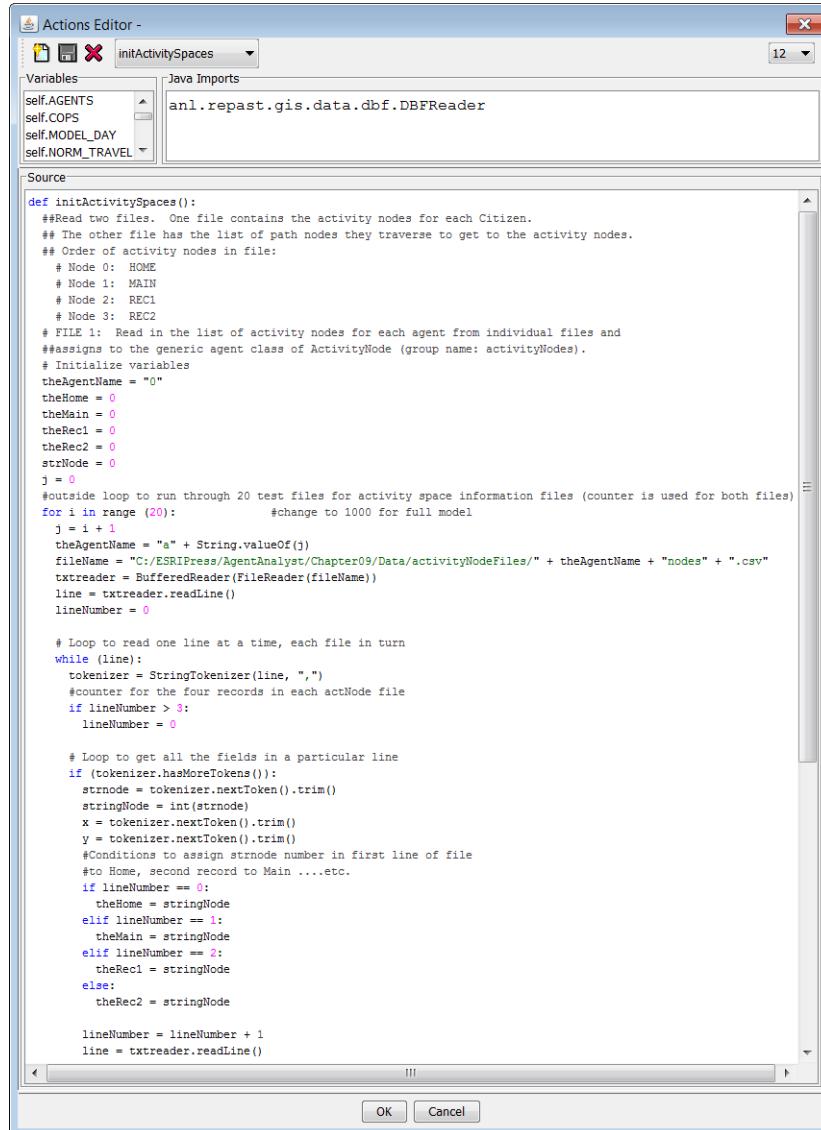
citizens.main = citizens.Main
citizens.rec1 = citizens.Rec1
citizens.rec2 = citizens.Rec2
citizens.pathNodes = citizens.PathNodes
citizens.setLocation(node)
citizens.setStrnnode(node.STRCL_)

```

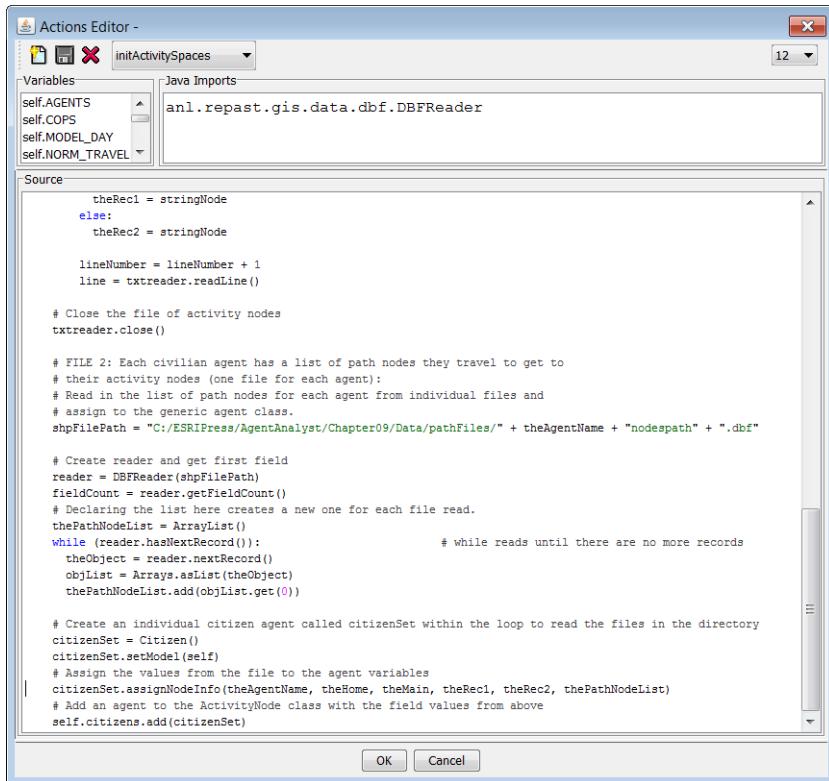
Next, you need to read in the lists of places and nodes for each activity space for each agent.

Create the activity spaces for each citizen agent

- 11** In the Actions Editor, select initActivitySpaces from the Actions drop-down list. Uncomment the code in the Source panel so that it matches the code in the following two figures. The action initializes the activity spaces for each agent.



5
6
7
8
9



In this code, the first section, where you are reading the activity nodes, is very similar to chapter 6, exercise 6c, step 12, where you set up the places. You create a buffered reader to read in the .csv file and then parse the file and set the values for the activity nodes. Then you call `assignNodeInfo(theAgentName, theHome, theMain, theRec1, theRec2, thePathNodeList)`, and this assigns the values to the fields for each agent.

For each agent you also need to read in the list of nodes for the route to the agent's activity nodes. You get the shapefile for each agent (`shpFilePath = "C:/ESRIPress/AgentAnalyst/Chapter09/Data/pathFiles/" + theAgentName + "nodespath" + ".dbf"`), and then create an instance of a DBFReader (`reader = DBFReader(shpFilePath)`), which enables the opening of a .dbf file. An array list is created to hold the list of nodes that make up the route (`thePathNodeList = ArrayList()`). A while loop is used to populate the array for this agent:

```

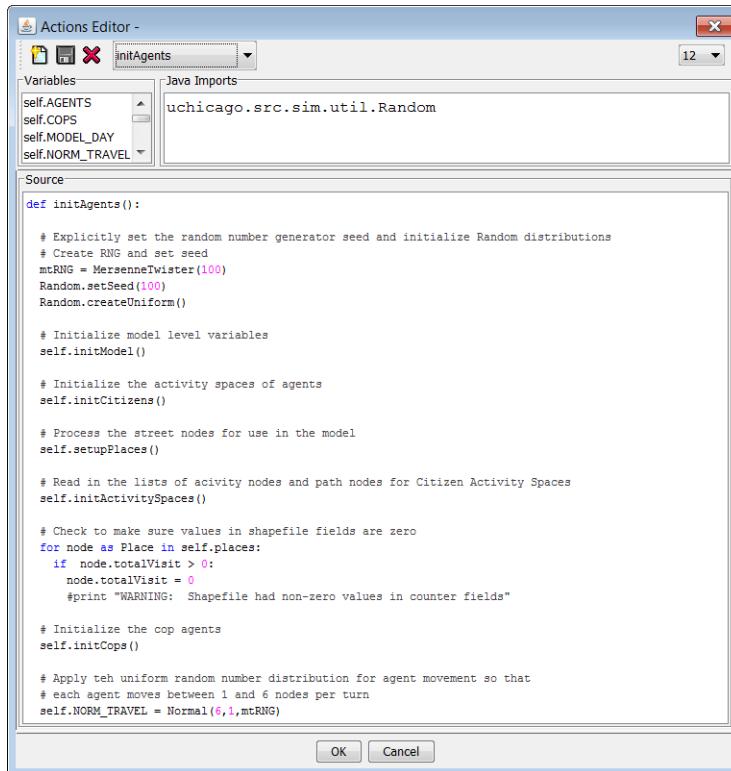
while (reader.hasNextRecord()):
    theObject = reader.nextRecord()
    objList = Arrays.asList(theObject)
    thePathNodeList.add(objList.get(0))

```

A citizenSet object is created, which is equivalent to a citizen (`citizenSet = Citizen()`) `citizenSet.setModel(self)`). To fill the citizenSet, the `assignNodeInfo` action is called to read the data from the two files and populate the agent variables (`citizenSet.assignNodeInfo(theAgentName, theHome, theMain, theRec1, theRec2, thePathNodeList)`). Last, an agent is added to the `ActivityNode` class with the field values from earlier (`self.citizens.add(citizenSet)`).

Remember, an action does not do anything until it is called.

- 12** While still at the model level, select the `initAgents` action and add a call to the `initActivitySpaces` action. Uncomment the call to `self.setupPlaces`. Also uncomment the remaining lines of code so it matches the following figure.



Print statements can be used to check the contents of the citizen agent's fields. You will add these statements to the `initAgents` action to ensure your agent's activity nodes are being populated correctly.

5

6

7

8

9

- 13** Add the following lines after your call to self.initActivitySpaces, and then click OK.

```
for citizens as Citizen in self.citizens:  
    print "Name: " + citizens.getName()  
    print "Home: " + citizens.getHome()  
    print "Main: " + citizens.getMain()  
    print "Rec1: " + citizens.getRec1()  
    print "Rec2: " + citizens.getRec2()
```

At this point, the code for the model is incomplete, so if you try to run the model you will receive an error.

- 14** Close ArcMap without saving your changes.

Now you are ready to use the information about the agents' activity spaces to move them along their paths to visit their activity nodes.

Moving agents around predefined activity spaces

To move the agents at every time step, incorporate the code into the step action. At the start of each model step, each agent evaluates its status to see if it is supposed to stay at a place or move somewhere else. The code tells the agent what to do under each condition.

The simplest way to move agents is to have them traverse their activity space one node (i.e., street intersection) at a time. To accomplish this, the code is added in the step action of the Citizen class. You want to put movement at the level of the Citizen class because it iterates through each of the agents in the model at each tick of the model.

Just as with random travel, directed travel uses an ActiveNode class to keep track of which street nodes have an agent on them. In the robbery model, this information is used to determine which of the following agents are present at the node: a motivated offender, a citizen agent meeting the criteria to be considered a suitable target, and the lack of a capable guardian (i.e., absence of enough citizens or a cop at the places).

Exercise 9d

To move the agents, each agent's current position in its path node list is used to assign the agent to a node (a place agent). Next, the nodes are added to the ActiveNode class so that you know which nodes to examine for the convergence of offenders, targets, and guardians. In this "stripped down version," you use the ActiveNode class to avoid having to iterate through all 16,035 intersection nodes. The next node in an agent's path node list is found by adding 1 to the agent's current position (`target = self.position + 1`). Finally, the agent's current node is changed to reflect its movement to a new node.

Load the ArcMap document

- 1 Start ArcMap. Navigate to C:\ESRIPress\AgentAnalyst\Chapter09. Open ex09d.mxd.

In the ArcMap display you will see the street center lines for the City of Seattle, the intersection nodes, and the Routes layer. In ArcToolbox, the Chapter09Ex09d toolbox has been created for you.

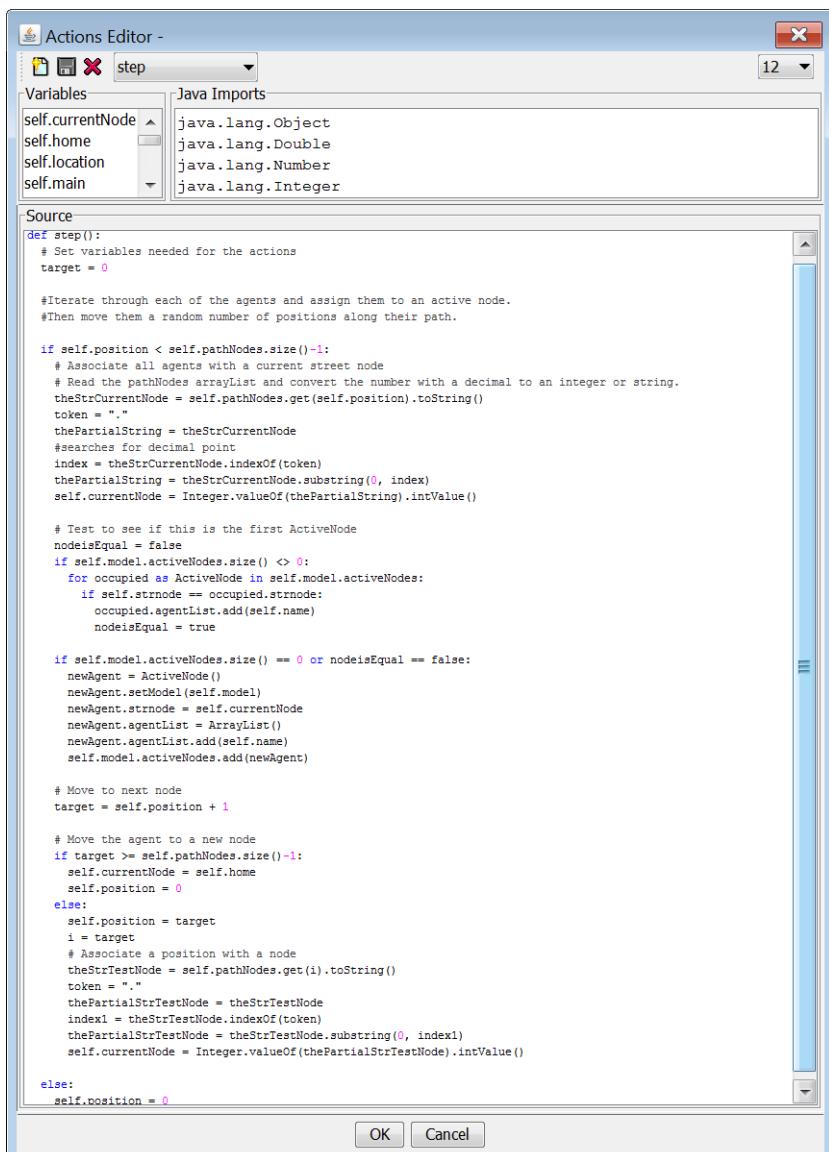
Load the Agent Analyst model

- 2 Right-click the Chapter09Ex09d toolbox, point to New, and select Agent Analyst Tool.
- 3 Once the Agent Analyst window appears, click File and select Import. Do not save the changes when prompted.
- 4 Navigate to C:\ESRIPress\AgentAnalyst\Chapter09\Models and open ex09d.sbp.

5
6
7
8
9

Move the agents

- 5 In the Environment panel, click Citizen. In the Property panel, click the Edit button to the right of the Actions property to open the Actions Editor.
- 6 In the Actions Editor, select step from the Actions drop-down list. Uncomment the code in the Source panel so that it looks like the code partially shown in the following image. When you're finished, all the comment markers in the first column should have been removed. The code moves the citizens along the network. Read through the documentation of the code to find the specifics of the actions described earlier.



In this code, you iterate through each of the agents and assign them to an active node. The first statement checks to see if the agent is at the end of its path (`if self.position < self.pathNodes.size() - 1:`). If it is, the else condition executes (`else:`), and the agent is assigned back to its home (`self.position = 0`). If an agent is not at the end of its path, then it is associated with a current street node. To do this, the code parses out the decimal place from the string and converts it to an integer:

```

theStrCurrentNode = self.pathNodes.get(self.position).toString()
token = "."
thePartialString = theStrCurrentNode
index = theStrCurrentNode.indexOf(token)
thePartialString = theStrCurrentNode.substring(0, index)
self.currentNode = Integer.valueOf(thePartialString).intValue()
nodeisEqual = false

```

The next section of code adds the node to a list of active nodes. It identifies each node occupied by an active agent and then adds the node to the ActiveNode class. In other words, if the array has values in it, there is an ActiveNode agent that exists with a particular strnode value, so the code adds the name of the citizen agent to the agentList (an arrayList). If there is no ActiveNode with the same value as the currentNode, the code adds a new ActiveNode agent, populates the strnode number with the currentNode, and adds the name of the citizen agent to the agentList (an arrayList).

```

if self.model.activeNodes.size() <> 0:
    for occupied as ActiveNode in self.model.activeNodes:
        if self.strnode == occupied.strnode:
            occupied.agentList.add(self.name)
            nodeisEqual = true

if self.model.activeNodes.size() == 0 or nodeisEqual == false:
    newAgent = ActiveNode()
    newAgent.setModel(self.model)
    newAgent.strnode = self.currentNode
    newAgent.agentList = ArrayList()
    newAgent.agentList.add(self.name)
    self.model.activeNodes.add(newAgent)

```

Now that you know where the agents are located, you need to move them one node from their current location. The code sets the position equal to the next node by adding 1 to the current position (`target = self.position + 1`). Before actually moving the agent to a new node, you need to make sure you aren't at the end of the path based on the new position (i.e., that the target position would not be at or beyond the end of the agent's path). This is accomplished by using an if conditional statement that tests to make sure the agent is not at the end of its path (`if target >= self.pathNodes.size() - 1:`). If the agent is at the end of its node list, then its current node is set to the home node (`self.currentNode = self.home`) and their current position number is reset to

5

6

7

8

9

zero (`self.position = 0`). If the agent is not at the end of the list, then set the current position to the target position (`self.position = target`).

To actually move the agent to a new node, it is necessary to use the position of the target node to get the value of the node object. This takes a few steps similar to those used to assign the node originally.

```
i = target
# Associate a position with a node
theStrTestNode = self.pathNodes.get(i).toString()
token = "."
thePartialStrTestNode = theStrTestNode
index1 = theStrTestNode.indexOf(token)
thePartialStrTestNode = theStrTestNode.substring(0, index1)
self.currentNode = Integer.valueOf(thePartialStrTestNode).intValue()
```

7 Close ArcMap without saving your changes.

In the step action, each active agent has been moved by one node along its route. In addition, the agent's new node has been added to the list of active nodes in preparation for the next tick of the model.

Using the cartographic capabilities of the ArcMap interface to represent agent movement

The current model is set up to update the display every 100 ticks. But let's say you are interested in seeing your agents as they move along their paths. You can accomplish this through the Schedule Editor.

Exercise 9e

Load the ArcMap document

- 1 Start ArcMap. Navigate to C:\ESRIPress\AgentAnalyst\Chapter09. Open ex09e.mxd.

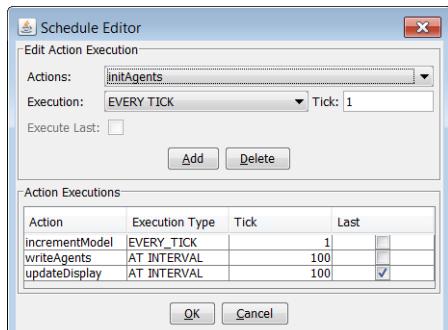
In the ArcMap display you will see the street center lines for the City of Seattle, the intersection nodes, the Routes layer, and street intersections. In ArcToolbox, the Chapter09Ex09e toolbox has been created for you.

Load the Agent Analyst model

- 2 Right-click the Chapter09Ex09e toolbox, point to New, and select Agent Analyst Tool.
- 3 Once the Agent Analyst window appears, click File and select Import. Do not save the changes when prompted.
- 4 Navigate to C:\ESRIPress\AgentAnalyst\Chapter09\Models and open ex09e.sbp.

Set the schedule

- 5 In the Environment panel, click Advanced Vector Movement. In the Property panel, click the Edit button to the right of the Schedule property.



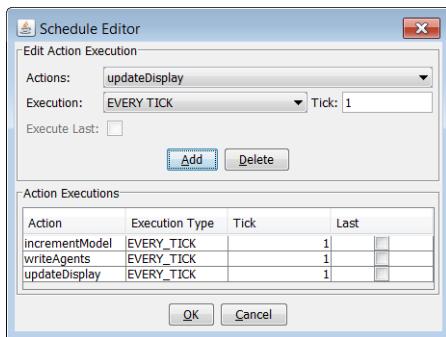
5
6
7
8
9

In the Schedule Editor, three schedules are already defined. You need to highlight two of these and delete them from the model since they update the model every 100 ticks.

- Under Action Executions, select writeAgents and click Delete. Also delete the updateDisplay action.

Change when to execute each scheduled action

- In the Schedule Editor, select writeAgents in the Actions drop-down list. Under Execution, choose EVERY TICK so that the display is updated at every tick of the model. Set Tick to 1. Make sure you click the Add button. Also add an action for updateDisplay to have the same values. Your Schedule Editor should look the same as the following image. Click OK.



Prepare the symbology for the model results

- Right-click StreetIntersections.shp in the ArcMap Table Of Contents, and then click Properties.... In the Layer Properties dialog box, click the Symbology tab. In the Show box, under Quantities, select Graduated Colors. In the Fields pane, set Value to totalVisit and set Normalization to none.
- Click the Classify button. In the Classification dialog box, click the Exclusion button, and in the lower text box type "**totalVisit**" =0.

This setting will keep any nodes with zero total visits from appearing on your map and make visualizing the agents' steps much easier. Alternatively, you could allow the zero nodes to be drawn but symbolize them so that they fade into the background.

- Click OK to close each dialog box.

Run your model to see your agents moving with every tick of the model

- 11** In the Agent Analyst window, click the Run button. Click the Start button on the Repast toolbar.

Observe the movements of the agents on the map.

End the session

- 12** Stop the model by clicking the Stop button. Close the Repast toolbar by clicking Exit.

- 13** Close Agent Analyst and ArcMap without saving changes.

5

6

7

8

9

Incorporating variety into activity spaces

In this exercise we want to add some randomness to the model. You do this to take into account the differences in speed for different modes of travel. Agents walking might travel 10 to 12 blocks per model step on average, while those in cars might travel twice as far (depending on the length of time represented by a model step). Randomness can also be used to reflect variations in travel time for two cars along the same route because one encountered a different combination of red and green lights than the other.

Exercise 9f

Because you are simulating travel along roads, it would be unrealistic to have two agents travel exactly the same distance each tick of the model. Rather, you would expect them to get stopped at different intersections and to walk or drive different distances in the same time period depending on whether they make it through traffic lights. You can use a random number generator to simulate the ability to travel ranges of distances during each tick of the model. For example, in this model you estimate agents will travel between one and six blocks per tick of the model. If you believe the agents are far less likely to travel only one street block or to make it six blocks, but are more likely to make it about three blocks, you may want to use a normal distribution to simulate the probability of each outcome. To set up the use of a random number generator, you need to do two things. First, you need to add a global variable at the model level. Second, you need to create and add randomness to the movement.

Load the ArcMap document

- 1 Start ArcMap. Navigate to C:\ESRIPress\AgentAnalyst\Chapter09. Open ex09f.mxd.

In the ArcMap display you will see two layers. One depicts StreetIntersections, and the other Streets. In ArcToolbox, the Chapter09Ex09f toolbox has been created for you.

Load the Agent Analyst model

- 2 Right-click the Chapter09Ex09f toolbox, point to New, and select Agent Analyst Tool.
- 3 When the Agent Analyst window appears, click File and select Import. Do not save the changes when prompted.
- 4 Navigate to C:\ESRIPress\AgentAnalyst\Chapter09\Models and open ex09f.sbp.

Add the global variable to the model level

- 5 In the Environment panel, click Advanced Vector Movement. In the Property panel, click the Edit button to the right of the Fields property.

- 6** Add the field defined in table 9.2. You have to enter the field's type manually because it is not in the predefined list of types. Click Add, and then click OK.

Table 9.2 Fields Editor data

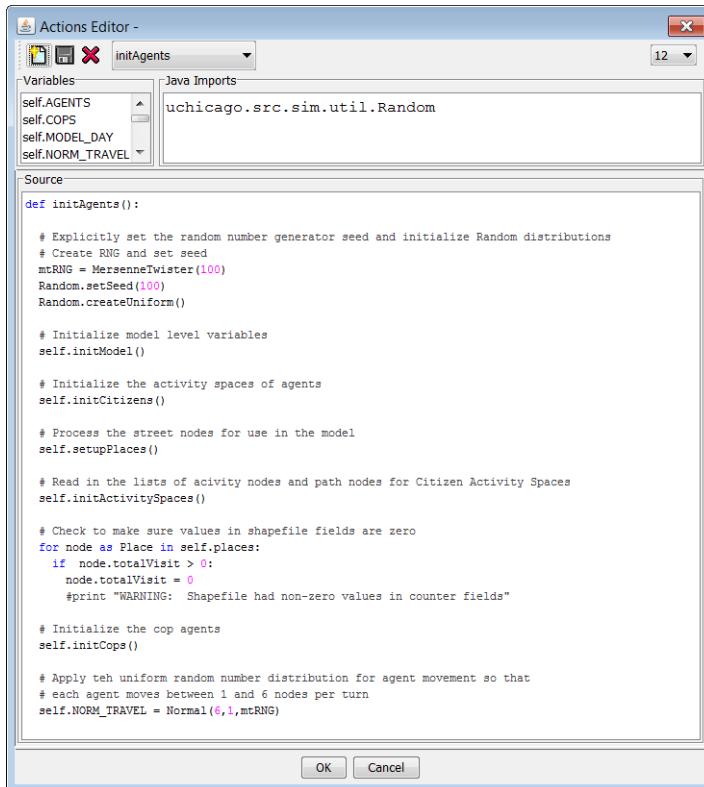
| Name | Type | Default value | Accessible |
|-------------|------------------------|---------------|------------|
| NORM_TRAVEL | cern.jet.random.Normal | | Yes |

Add randomness to the movement in the initAgents action

- 7** In the Environment panel, click Advanced Vector Movement. In the Property panel, click the Edit button to the right of the Actions property to open the Actions Editor.
- 8** In the Actions drop-down list, select initAgents. Add the following code to the bottom of the action, and then click OK:

```
self.NORM_TRAVEL = Normal(6,1,mtRNG)
```

Your initAgents action should look the same as the action in the following image.



5
6
7
8
9

The line you added sets the `self.NORM_TRAVEL` variable equal to a normal distribution random number generator (`self.NORM_TRAVEL = Normal(6, 1, mtRNG)`). This will generate a number between 1 and 6 (for more information on random number generators, see chapter 6).

You will now want to define movement at the level of the Citizen class because it iterates through all the agents each tick of the model.

Just as with random travel, directed travel uses an ActiveNode class to keep track of which street nodes have an agent on them. In the robbery model, this information is used to determine which of the following agents are present at the node: a motivated offender, a citizen agent meeting the criteria to be considered a suitable target, and the lack of a capable guardian (i.e., absence of enough citizens or a cop at the places).

To move the agents, each agent's current position in its path node list is used to assign the agent to a node (a place agent). Next the nodes are added to the ActiveNode class so that we know which nodes to examine for the convergence of offenders, targets, and guardians. In this “stripped down version,” you use the ActiveNode class to avoid having to iterate through all 16,035 intersection nodes. The next node in an agent's path node list is found by adding 1 to its current position (`target = self.position + 1`). Finally, the agent's current node is changed to reflect its movement to a new node.

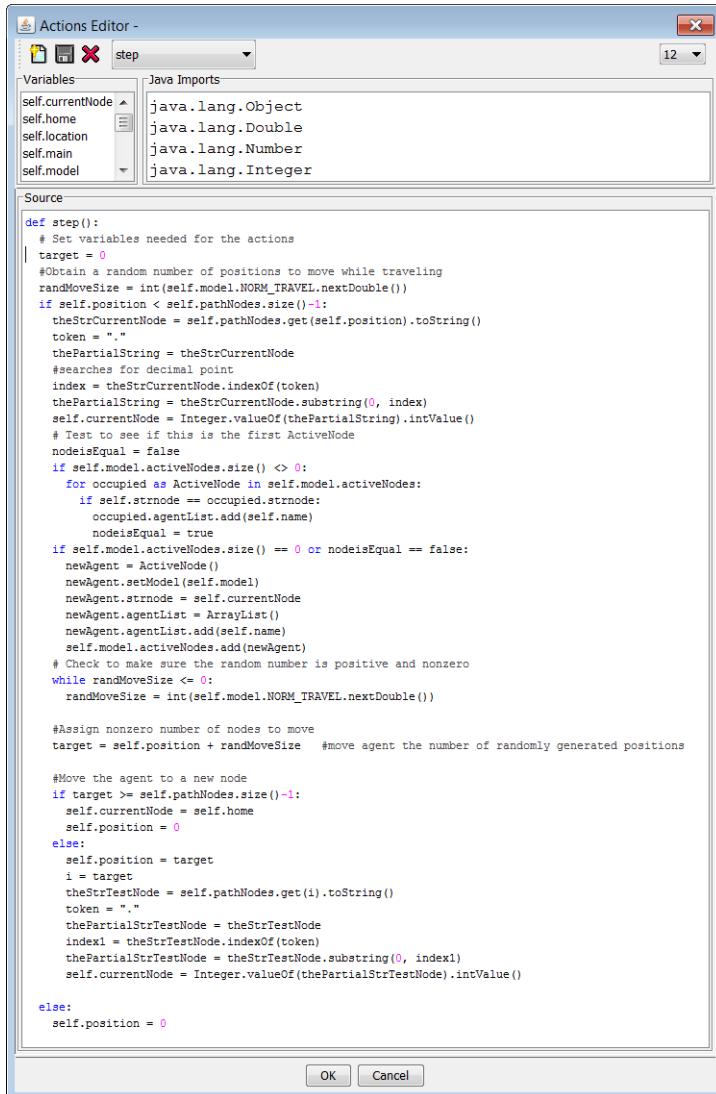
Open the step action for the citizen agent

- 9 In the Environment panel, click Citizen. In the Property panel, click the Edit button to the right of the Actions property to open the Actions Editor.

In the Actions drop-down list, select step. Read through the documentation of the following code to find the specifics of the actions described earlier.

- 10 Uncomment the code in the Source panel for the step action of the citizen agent so that it matches the code in the following image.

The step action moves the citizen agent using the ActiveNode list.



The screenshot shows the 'Actions Editor' window with the title bar 'Actions Editor - step'. The window has two main sections: 'Variables' and 'Source'. The 'Variables' section lists several variables under 'self': currentNode, home, location, main, and model. The 'Java Imports' section includes java.lang.Object, java.lang.Double, java.lang.Number, and java.lang.Integer. The 'Source' section contains the following Python code:

```

def step():
    # Set variables needed for the actions
    target = 0
    #Obtain a random number of positions to move while traveling
    randMoveSize = int(self.model.NORM_TRAVEL.nextDouble())
    if self.position < self.pathNodes.size()-1:
        theStrCurrentNode = self.pathNodes.get(self.position).toString()
        token = "."
        thePartialString = theStrCurrentNode
        #searches for decimal point
        index = theStrCurrentNode.indexOf(token)
        thePartialString = theStrCurrentNode.substring(0, index)
        self.currentNode = Integer.valueOf(thePartialString.intValue())
        # Test to see if this is the first ActiveNode
        nodeisEqual = false
        if self.model.activeNodes.size() > 0:
            for occupied as ActiveNode in self.model.activeNodes:
                if self.strnode == occupied.strnode:
                    occupied.agentList.add(self.name)
                    nodeisEqual = true
        if self.model.activeNodes.size() == 0 or nodeisEqual == false:
            newAgent = ActiveNode()
            newAgent.setModel(self.model)
            newAgent.strnode = self.currentNode
            newAgent.agentList = ArrayList()
            newAgent.agentList.add(self.name)
            self.model.activeNodes.add(newAgent)
            # Check to make sure the random number is positive and nonzero
            while randMoveSize <= 0:
                randMoveSize = int(self.model.NORM_TRAVEL.nextDouble())

            #Assign nonzero number of nodes to move
            target = self.position + randMoveSize #move agent the number of randomly generated positions

            #Move the agent to a new node
            if target >= self.pathNodes.size()-1:
                self.currentNode = self.home
                self.position = 0
            else:
                self.position = target
                i = target
                theStrTestNode = self.pathNodes.get(i).toString()
                token = "."
                thePartialStrTestNode = theStrTestNode
                index1 = theStrTestNode.indexOf(token)
                thePartialStrTestNode = theStrTestNode.substring(0, index1)
                self.currentNode = Integer.valueOf(thePartialStrTestNode.intValue())
        else:
            self.position = 0

```

At the bottom of the window are 'OK' and 'Cancel' buttons.

Since the agents are going to move a random number of nodes each tick, it is necessary to obtain the number of nodes prior to assigning each agent a new position. This is accomplished by setting a random move size variable and setting it equal to the random number from a random number generator (`randMoveSize = int(self.model.NORM_TRAVEL.nextDouble())`).

From this point, the procedure to move agents along a directed path is very similar to what was explained in exercise 9d, but it is repeated here for convenience. As before, this code iterates through each of the agents and assigns them to an active node. The first statement

checks to see if the agent is at the end of its path (`if self.position < self.pathNodes.size() - 1:`). If the agent is, the `else` condition executes (`else:`), and the agent is assigned back to its home (`self.position = 0`). If an agent is not at the end of its path, then it is associated with a current street node. To do this, the code parses out the decimal place from the string and converts it to an integer:

```
theStrCurrentNode = self.pathNodes.get(self.position).toString()
token = "."
thePartialString = theStrCurrentNode
index = theStrCurrentNode.indexOf(token)
thePartialString = theStrCurrentNode.substring(0, index)
self.currentNode = Integer.valueOf(thePartialString).intValue()
nodeisEqual = false
```

As introduced earlier, the next section of code adds the node to a list of active nodes. First, it identifies each node occupied by an active agent and then adds the node to the `ActiveNode` class. In other words, if the array has values in it, there is an `ActiveNode` agent that exists with a particular `strnode` value, so the code adds the name of the citizen agent to the `agentList` (an `arrayList`). If there is no `ActiveNode` with the same value as the `currentNode`, then the code adds a new `ActiveNode` agent, populates the `strnode` number with the `currentNode`, and adds the name of the citizen agent to the `agentList` (an `arrayList`).

```
if self.model.activeNodes.size() <> 0:
    for occupied as ActiveNode in self.model.activeNodes:
        if self.strnode == occupied.strnode:
            occupied.agentList.add(self.name)
            nodeisEqual = true

if self.model.activeNodes.size() == 0 or nodeisEqual == false:
    newAgent = ActiveNode()
    newAgent.setModel(self.model)
    newAgent.strnode = self.currentNode
    newAgent.agentList = ArrayList()
    newAgent.agentList.add(self.name)
    self.model.activeNodes.add(newAgent)
```

Now that you know where the agents are located, you need to move them. Instead of moving the agent one node at a time (as you did in exercise 9d), you are moving them a random number of nodes each tick, using a random number to do this. Before attempting to move the agents, you need to check to ensure that the randomly generated number is positive and nonzero. A while statement is used to accomplish this:

```
while randMoveSize <= 0:
    randMoveSize = int(self.model.NORM_TRAVEL.nextDouble())
```

Since the number of nodes moved each tick varies based on the random number generated, once again you need to check to make sure the new position is not at or past the end of the

route. The code sets the target position equal to the current position plus the number generated by the random number generator (`target = self.position + randMoveSize`). Before actually moving the agent to a new node, you need to make sure you aren't at the end of the path based on the new position (i.e., that the target position would not be at or beyond the end of the agent's path). This is accomplished by using an if conditional statement that tests to make sure the agent is not at the end of its path (`if target >= self.pathNodes.size() - 1:`). If the agent is at the end of its node list, then its current node is set to the home node (`self.currentNode = self.home`) and its current position number is reset to zero (`self.position = 0`). If the agent is not at the end of the list, then set the current position to the target position (`self.position = target`).

To actually move the agent to a new node, it is necessary to use the position of the target node to get the value of the node object. This takes a few steps similar to those used to assign the node originally (i.e., when you initially assigned the agent to a node).

```
i = target
# Associate a position with a node
theStrTestNode = self.pathNodes.get(i).toString()
token = "."
thePartialStrTestNode = theStrTestNode
index1 = theStrTestNode.indexOf(token)
thePartialStrTestNode = theStrTestNode.substring(0, index1)
self.currentNode = Integer.valueOf(thePartialStrTestNode).intValue()
```

In the step action, each active agent has been moved by a random number of nodes along its route. In addition, the agent's new node has been added to the list of active nodes in preparation for the next tick of the model.

- 11** Close the Actions Editor.
- 12** Close the Agent Analyst model. Do not save any changes.
- 13** Close ArcMap. Do not save any changes.

You have completed the exercises to move agents along a defined network according to a pre-defined set of trips they make. Because Agent Analyst currently does not read network datasets directly, you learned an alternative method for creating and using those paths in Agent Analyst. The alternative method involves creating the paths using ArcGIS Network Analyst and then extracting the unique ID from the route features to generate text files. The text files contain the list of nodes the agent must traverse to visit a set of destinations in one day. You also learned how to add randomness to the traversal of the route by varying the number of nodes traveled with each model tick. Additional complexity could be achieved by creating multiple activity spaces that are randomly assigned each model day. These techniques could be applied to many other types of substantive areas such as modeling disease spread and rainfall impacts on river levels downstream.

5

6

7

8

9

GIS data, fields, and actions dictionaries

Table 9.3 Data dictionary of GIS datasets

| Dataset | Data type | Description |
|-------------------------|-----------|---|
| Streets.shp | Line | Street centerline of Seattle |
| StreetIntersections.shp | Point | Intersections in street centerline of Seattle |

Table 9.4 Fields dictionary for the crime model

| Attribute | Data type | Description |
|--|------------------------|--|
| Advanced vector model agent classes | | |
| places | Vector agent | Vector agents representing street intersections |
| citizen | Generic agent | Citizens |
| cops | Generic agent | Police officers |
| ActiveNode | Generic agent | Nodes with agents on them |
| Advanced vector model fields | | |
| placeMap | java.util.HashMap | A hash map used to store the list of all the places with strnode-id as the key |
| modelStep | integer | Variable to hold the incremented model step |
| MODEL_DAY | integer | The number of ticks in a model day |
| NUM_PLACES | integer | The total number of intersection nodes |
| COPS | integer | The total number of cop agents in the model |
| NORM_TRAVEL | cern.jet.random.Normal | Holds the normal random number distribution |
| Place data source and fields | | |
| Data Source | string | Path to the street nodes/intersections shapefile |
| strcl_ | string | The unique ID of the street node |
| myNeighbors | java.util.ArrayList | Each street node and its neighbors |
| Cop fields | | |
| strnode | integer | The node where a police officer is located as an integer |
| location | place | The place object that corresponds to that integer |

Table 9.4 Fields dictionary for the crime model (*continued*)

| Attribute | Data type | Description |
|--------------------------|---------------------|---|
| Citizen fields | | |
| strnode | integer | The node where a citizen is located as an integer |
| location | place | The place object that corresponds to that integer |
| name | java.lang.String | The unique name of the agent ($a_1 \dots a_n$) |
| home | integer | The home node number |
| main | integer | The work node number |
| rec1 | integer | The first recreation node number |
| rec2 | integer | The second recreation node number |
| position | integer | The current position in the path node list |
| currentNode | integer | The node at which an agent is currently located |
| pathNodes | java.util.ArrayList | The array of path nodes that make up an activity space for each agent |
| nodeList | java.util.ArrayList | The array of activity nodes for each agent |
| ActiveNode fields | | |
| strnode | integer | The node where a visitor is located as an integer |
| agentList | java.util.ArrayList | The array of agents who are currently occupying the place object |

5
6
7
8
9**Table 9.5** Actions dictionary for the crime model

| Declared actions | Description |
|----------------------------|---|
| Simple vector model | |
| initAgents | Sets the random number seed generator and initializes the random number distribution. Calls the initModel, initCitizens, initCops, and setupPlaces actions. |
| setupPlaces | Initializes a hash map and identifies the neighbors of each place for random movement. First, a hash map is created to store a list of all the places with strnode-id as the key. Next, the action reads the nodeNeighbors.csv file and associates the set of neighbors with the correct place (i.e., it populates the field myNeighbors in the Place class). |

Table 9.5 Actions dictionary for the crime model (*continued*)

| Declared actions | Description |
|-------------------------|--|
| initModel | Sets values for constants and static variables in the model. Some of these values are parameters and can be changed through the Repast user interface at model run time. |
| initCitizens | Creates citizen agents, names them, and assigns them to a strnode (number) and a location (place). The action uses a uniform distribution to select the nodes on which to place the citizens at the start of the model. |
| initCops | Creates cop agents and assigns them to a strnode (number) and a location (place). The action uses a uniform distribution to select the nodes on which to place the cops at the start of the model. |
| updateDisplay | Changes the display in ArcMap. |
| incrementModel | Counts the cumulative number of agents at each node and stores the information. Keeps count of the model step number and resets totalVisit field values to zero at the start of each model run. |
| writeAgents | Writes the values of the counter for each place node to the totalVisit field in the strnodes2 shapefile. The action is currently scheduled to be called every 100 ticks. |
| initActivitySpaces | Reads the files of activity nodes and paths and creates a new citizen agent with an activity space as specified by the appropriate activity nodes and path nodes. Uses the assignNodeInfo action in the Citizen class to set the field values in citizen agents. |
| Citizens | |
| step | First the action gets the list of all the places in the Places class. For each node that has a citizen, the list of neighbor nodes is shuffled and then the citizen is assigned to the first position in the node list of possible moves. The strnode and the location fields are changed to reflect the citizen's new position. |
| assignNodeInfo | Assigns the field values from the activity node and path files to each of the citizen agents. It is called by the model action, initActivitySpaces. |
| Cops | |
| step | First the action gets the list of all the places in the Places class. For each node that has a cop, the list of neighbor nodes is shuffled and then the cop is assigned to the first position in the node list. The strnode and the location fields are changed to reflect the cop's new position. |

References

- Carlstein, T., D. Parkes, and N. J. Thrift. 1978. *Human Activity and Time Geography*. New York: Halsted Press.
- Groff, E. R. 2006. "Exploring the Geography of Routine Activity Theory: A Spatio-temporal Test Using Street Robbery." Unpublished diss., University of Maryland, College Park.
- . 2007. "'Situating' Simulation to Model Human Spatio-Temporal Interactions: An Example Using Crime Events." *Transactions in GIS* 11(4): 507–30.
- Hägerstrand, T. 1970. "What About People in Regional Science?" *Papers of the Regional Science Association* 24: 7–21.
- . 1973. "The Domain of Human Geography." In R. J. Chorley, ed., *Directions in Geography*, 67–87. London: Methuen.
- Horton, F. E., and D. R. Reynolds. 1971. "Action Space Differentials in Cities." In H. McConnell and D. Yaseen, eds., *Perspectives in Geography: Models of Spatial Interaction*, 83–102. DeKalb: Northern Illinois University Press.
- Lynch, K. 1960. *The Image of the City*. Cambridge, MA: M.I.T. Press.
- Pred, A. 1967. *Behavior and Location: Foundations for a Geographic and Dynamic Location Theory*, Part I. Gleerup: Lund.
- . 1969. *Behavior and Location: Foundations for a Geographic and Dynamic Location Theory*, Part II. Gleerup: Lund.
- . 1977. "The Choreography of Existence: Comments on Hägerstrand's Time-Geography and Its Usefulness." *Economic Geography* 53: 207–21.
- Thrift, N. J., and A. Pred. 1981. "Time-Geography: A New Beginning." *Progress in Human Geography* 5(2): 277–86.

5

6

7

8

9

Section 4: Agent-based modeling accentuated with GIS

Chapter 10

Building synergy between GIS and agent-based modeling

by Nathan Strout and Michelle Gudorf

Case studies

- ◆ Case study 1: Using GIS analysis tools for data generation and organization in the model process
- ◆ Case study 2: Using ArcGIS spatial modeling capability to call ArcObjects from Agent Analyst—developing a fire model
- ◆ Case study 3: Adding temporal related viewing capability to the ABM output using ArcGIS Tracking Analyst and the bird migration model
- ◆ Case study 4: Integrating third-party movie-making software with agent-based models to facilitate dynamic visualization

Exercises

- ◆ An overview of the bird migration model
- ◆ Creating a tracking log using ArcObjects to save agents at each tick
- ◆ Demonstrating the 2D ArcMap animation tools using the bird migration model
- ◆ Demonstrating the 3D ArcGlobe animation tools using the bird migration model

Most of you are already familiar with the benefits of a geographic information system (GIS) and what it can do. In the previous chapters you saw how agent-based modeling (ABM) can accentuate GIS capabilities. This chapter focuses on the specific GIS capabilities that can accentuate an agent-based model, highlighting the bidirectional relationship between ABM and GIS. Specifically, the following are capabilities that a GIS can provide to enhance an agent-based model:

- Data preparation
- Spatial modeling tools
- Display tools
- Spatial modeling validation tools

This chapter walks you through various case studies and exercises that illustrate how you can use GIS to facilitate the ABM process.

Background information

Agent Analyst allows the user to take advantage of the strengths of the GIS when creating, running, visualizing, and analyzing the results of their agent-based model. With this bidirectional synergy, the user is able to more effectively capture the interactions of the phenomena that they are interested in.

Data preparation

A GIS provides tools for spatial data preparation prior to and during modeling, thus facilitating the integrated ABM/GIS environment. GIS can be used to organize and derive the base data used in an agent-based spatial model:

Organize data: GIS allows you to assemble and manage various types of spatial data from multiple sources using the appropriate spatial extents, resolutions, and scales. Additionally, it allows you to align or transform the data into compatible spatial reference systems.

Derive data: When applying an agent-based model over a landscape, GIS tools can be used to derive additional base data such as slope, aspect, and distance from features from spatial data. Slope and aspect, identifying the locations of specific features (buildings, roads, barriers, and so on), buffering of features, and determining Euclidean distance from features are all examples of derivations of spatial data.

Spatial modeling

GIS also provides the spatial modeling tools necessary to facilitate the spatial aspects of the ABM process. These GIS tools create surfaces that are static for the entire model, or the tools can create multiple surfaces that change every time step within the agent-based model. Generally, there are three ways that these hundreds of spatial modeling tools can accentuate an agent-based model:

- By modeling changes within the landscape such as fire and flood. These fire and flood models can be built and run in the GIS for every time step of the agent-based model, and the agents can query the results of the growing fire or flood.
- By performing quantifiable analysis of the geometry of the features modeled, such as determining the extent of an area or the perimeter of polygons (e.g., forest stands), which may elicit a response from the agent, or using overlay tools to create new geometry from various data.
- By calculating statistics that will provide information about any number of questions that might influence an agent's decisions (e.g., identifying the minimum, maximum, or mean elevation value, or calculating average rainfall for each cell from 10 years of rainfall data).

Additionally, GIS allows for the integration of other third-party spatial models such as MODFLOW (which simulates a hydrologic system such as flow from rivers or flow into drains and provides a specific method of solving linear equations that describe the flow system) and timber-cutting plans. These third-party spatial models can, at each time step, create surfaces or produce other data that the agent may consider.

Display tools

GIS provides an array of options for model development and visual validation. Often we use GIS to display the data we are working with to understand the data and derived rules and to view the results using various display systems, such as 2D and 2D or 3D animation.

GIS is an effective environment in which to display and visually understand data. Viewing the spatial data allows you to view the distribution of phenomena and their relative positions to one another, which may influence rule development. These display tools can also allow the user to visually explore the results from an agent-based model. They provide another perspective of the model analysis, likely a qualitative interpretation of the model. Essentially a cursory analysis from a visual analysis can be expected. You can observe visually how agents are interrelating or reacting to the rules provided in the model. There are several ways to display the results from the Agent Analyst model.

Two-dimensional maps can be generated from a GIS in conjunction with Agent Analyst. These maps are static, representing a snapshot in time rather than a continuum of time. Two-dimensional mapping assumes that the phenomenon doesn't change until the data input changes and then another snapshot is created. GIS 2D animation tools allow you to visually

represent the results of a dynamic model like one generated in Agent Analyst, where agents represent something real, are dynamic, and move themselves according to programmed rules during time steps. Your Agent Analyst model can produce and capture an agent's change in movement or changes in any of its other properties during each time step and then store that movement in separate layers. The layers can be displayed sequentially using the ArcGIS animation tools. ArcGIS Tracking Analyst enables you to animate the series in 2D by offering sophisticated display tools. The same series of layers can be animated in 3D by using the 3D Analyst capability. In each of these animations, the moving agent is generally a static symbol.

Third-party movie-making software can accept input from moving agents and project the moving agents over variable components such as surfaces and features. Realistic effects can be added to characterize the terrain, vegetation, 3D structures, and lighting to represent changes in time of day and seasons. Agents have the capacity to move in six dimensions and change size in three dimensions. The frames of data can be assembled into a QuickTime or AVI movie.

Spatial validation tools

Once you have developed and run a model, there are many analytical post-processing tools you can use to validate the model results. These tools are often the same spatial tools used for ABM mentioned earlier. You might use buffering or Euclidean distance in the GIS to verify some measurable feature of the phenomenon. You may want to see whether your model is capturing the relationship of the agent with the features in a similar manner to the way the actual phenomenon interacts with the features. For example, in the cougar model discussed in chapters 2, 5, and 8, you can use the spatial validation tools to determine whether the cougar agents disperse themselves in secure areas, away from humans and near prey in a manner similar to actual cougars. Are the cougars clustering as viewed through hotspots, or is the movement distribution random? Additionally, you might use a GIS to summarize statistics about the agent and about the model output series.

Case studies and exercises

This chapter integrates a mixture of case studies and exercises to expose you to areas where GIS can accentuate ABM. First, four case studies will be presented to provide you with examples of how GIS tools facilitate spatial data preparation and spatial modeling and how to use display tools in animation and in spatial model validation. The four case studies demonstrate four different models.

Following the case studies, four exercises have been provided to walk you through the process of utilizing a GIS in spatial modeling and how to use ArcGIS 2D and 3D animation tools for the spatial modeling visualization process. The four exercises have been developed around a bird migration model.

Case studies

Case study 1: Using GIS analysis tools for data generation and organization in the model process

A land-use change model was selected to illustrate how GIS tools can be used to create base data for an agent-based model. By reviewing this case study, you will see how the following base data layers were derived and used as input:

- Parcel centroids
- Cost distance surfaces
- Distance to market derived from center and cost distance
- Assigned land use from raster to parcels

10

You will then observe how this GIS data can be used in ABM for modeling land-use change.

This land-use change model is an adaptation written by Robert Najlis and Nick Collier and was based on a SLUDGE model created by Dawn Parker from George Mason University. In this land-use change model, housing competes with agriculture and forest land-use types. The model is run on data from Stowe, Vermont. Stowe is a small New England town popular throughout the eastern United States for its outdoor recreation opportunities (cross country and downhill skiing, snowboarding, hiking, and water sports). The landscape at one time was dominated by agriculture and forest but now, because of a ski area and associated development, housing competes for the limited resources.

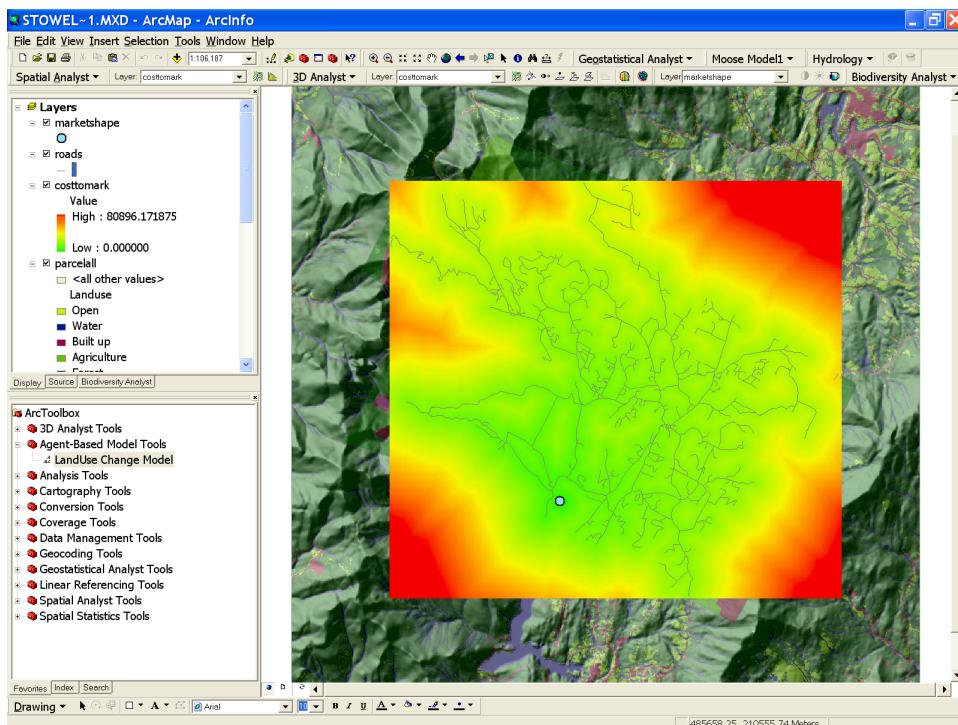
Developing a land-use change model

This model is a land-use change model in which residential housing competes with agriculture and forest land-use types. Land-use change will be based on economic value. Each parcel is a polygon agent representing a land-use type. Every parcel agent has an economic value. A parcel agent can change its land-use type from forest/agriculture to residential housing based on the economic value that parcel contributes as that land-use type. Each parcel agent obtains a value for itself. The agriculture and forest land uses do so from economic models based on the productivity of their land use minus distance to market, which is derived from a GIS layer.

Generating data using GIS

In the case of agriculture and forest agents, “commodity value minus cost to market” determines the value of the parcel agent. To establish value, you need to determine the distance from the parcel agent to the distribution center. To do this, you must select some point within each parcel from which to measure. In this case, GIS was used to identify and select the center of the parcel. A landscape cost surface was created that determines the cost (not Euclidean distance) of moving through each cell: every cell in the landscape was assigned a cost for a vehicle to carry the product through the cell. Cost values were determined by slope and land-use type. All road types were given a low cost value. The distribution center (source) and the cost surface were used as the input to the ArcGIS Spatial Analyst Cost Distance tool. The output surface resulting from the Cost Distance tool identifies, for every cell, the least accumulative cost to the source while traveling over the cost surface. Because we know each parcel agent’s center, we can then correlate it to a cell on the accumulative cost surface to derive the cost from the parcel to the distribution center. Thus a parcel that is far from a road and far from the distribution center will generally be the most costly for getting to market. However, the parcel might be closer to the distribution center but farther from a road and in rough terrain; therefore, the cost might be greater to get to the distribution center.

The following image shows the cost distance surface that was created in ArcGIS Spatial Analyst and was used as input for the agent model for the distance-to-market calculations. Green is less costly, and the market is symbolized as a blue circle. This is not a Euclidean distance surface but represents actual cost to market; note the roads are less costly to travel across.



Running the agent-based model using the GIS data

The agent-based model is used to create and evaluate the rules for determining whether a parcel agent should change its land-use type. A parcel can change its land-use type from forest/agriculture to residential housing based on the economic value that parcel contributes as that land-use type.

In this agent-based model, the agriculture/forest parcel uses as input the economic values for agriculture minus cost to market as defined in the cost surface data generated using GIS—“commodity value minus cost to market.” If the parcel has a low cost to the distribution center and the price for agriculture products is high, economically it is better for the parcel to remain as agricultural. If the parcel has a high cost to the distribution center (high cost to get the product to market) and the price of agriculture products is low, economically that parcel will be worth more if it becomes residential property. In an agrarian society where the price of agriculture products is high, the cost to market is lower, and the demand for housing is low, it is more economic for more parcels to be in agriculture. This would be the case for the United States in the 1800s. Note that the value of agricultural parcels in this model does not consider the economic value of aesthetics or the often public desire to preserve and maintain a working landscape.

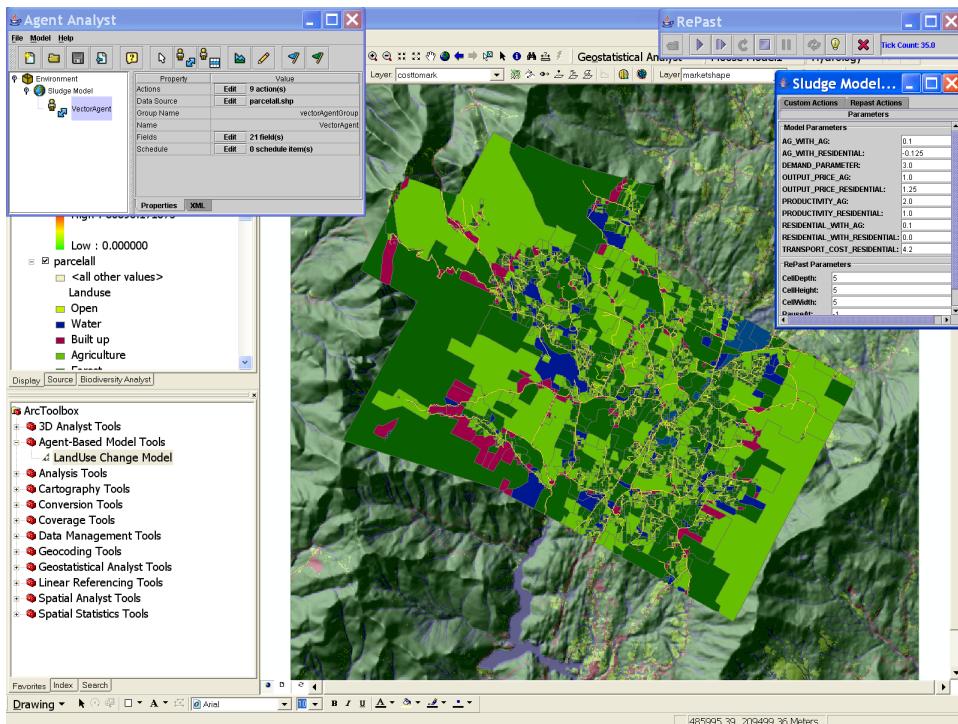
10

In the case of residential parcels, value is determined by fair market value. Each parcel adjusts its value based on the surrounding land use. Residential value increases if the parcel borders other residential parcels or parcels identified as agriculture or forest. The demand for residential housing also influences the parcel’s worth. Surrounding land use can also have a negative influence on its neighbors. For example, if a parcel is next to a landfill, its economic value if zoned residential is reduced. Note that for many of these parameters, the agent-based model needs to query GIS to identify neighboring parcels’ land-use type during any given time step. The model might run 100 time steps, so the composition of the neighboring land-use types can change over time, thus changing the adjustments to their value.

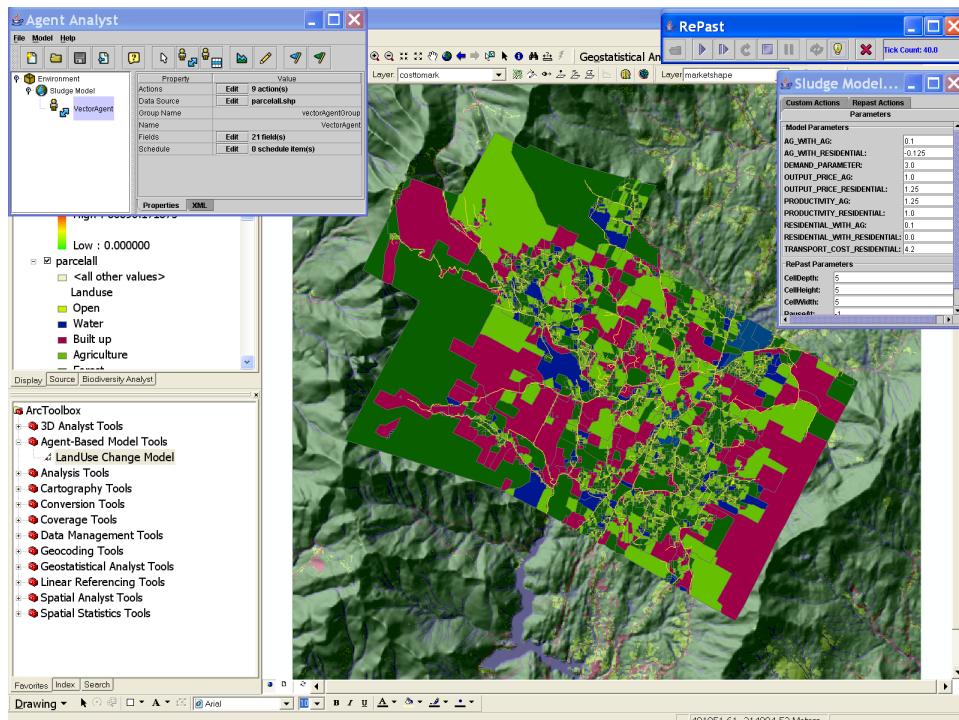
The model parameters can be changed to explore their influence. The type of each parcel will change based on the economic value of each parcel.

Applying model variations

You can control the relative importance of shifting from agriculture to residential land use by adjusting the value of agricultural use. In the model shown in the following figure, agriculture land use was given a higher relative value than housing, and then the model was run. Notice that agriculture (symbolized in light green) is dominating the landscape. This is similar to the situation in Stowe, Vermont, in the 1800s.



In the following image, the assigned agriculture land-use value was decreased, thus the relative housing value increased, and then the simulation was run. Notice how housing (symbolized in magenta) is encroaching into the area. This is what is happening in present-day Stowe. The land planners of Stowe can use this information to intervene if they so desire to keep the rural character of Stowe intact.



In this land-use model, the GIS provided the base spatial data (e.g., land-use and roads layers) and derived additional base data (e.g., polygon centroids and accumulative cost surfaces) that were integral to the decision making of the parcel agents in the model.

Case study 2: Using ArcGIS spatial modeling capability to call ArcObjects from Agent Analyst—developing a fire model

This case study provides an example of how the spatial modeling tools in a GIS can be fully integrated into the decision-making process of an agent in Agent Analyst. Both ArcGIS and Agent Analyst are Component Object Model (COM) compliant. ArcGIS is composed of objects through which you can access all the ArcGIS functionality from other applications. Both ArcGIS and Agent Analyst support Java object models. As a result, all ArcGIS functionality can be called directly within Agent Analyst. With this capability, Agent Analyst, using ArcGIS functionality, can create spatial models that can dynamically alter the landscape each time step, and the agents within the model can at each time step incorporate the changing landscape into their decision making, which can affect their actions.

The model highlighted in this case study is an example of this functionality and demonstrates how GIS can accentuate the Agent Analyst model by growing a fire on a landscape. GIS tools can be synchronized with each time step of the decision-making process of the Agent Analyst agents. The landscape and data used to grow the fire build on the data provided from the cougar model presented in chapters 5 and 8. In each time step the cougar agents react to the locations of the burning fire. The location of the fire will become a strong repellent that the cougar agents avoid.

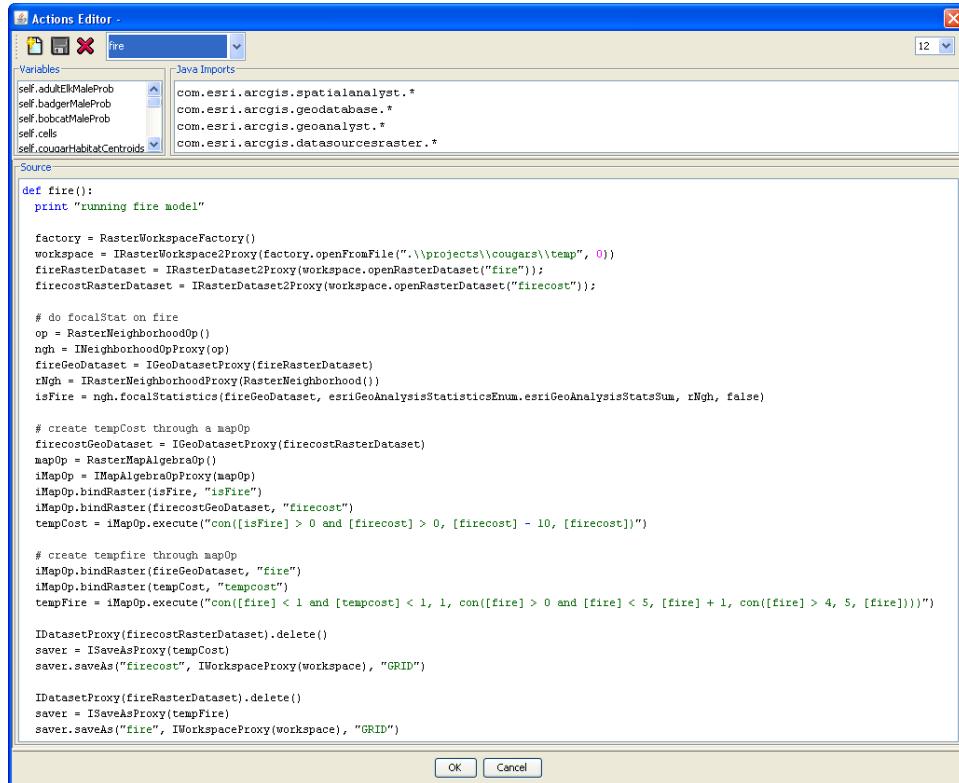
The fire growth rules defining where the fire moves each time step are determined through a series of ArcGIS Spatial Analyst tools. These tools are called by the Agent Analyst model through Java objects. The fire criterion will be weighted like all other criteria in the cougar model (e.g., remaining with a kill, pursuing a female cougar agent, staying within their home range, and so on) in Agent Analyst and integrated in the cougar's decision making.

Developing a fire model

In the fire model, the fire will grow over a raster. When developing the rules for growing a fire, you must specify the rules from a cell's perspective, and not from the mapping point of view (i.e., from the worm's-eye not the bird's-eye perspective). Often in modeling, when you think about qualities or requirements of a phenomenon, you generally think about it from the phenomenon's perspective—in this case, the fire's perspective. However, in ArcGIS Spatial Analyst, a cell can change its own value but not the values of other cells. As a result, the fire model cannot take the perspective of the fire by saying, "I am a burning fire and you are directly in my neighborhood path, so I should burn you." Based on the logic of the model, the fire model must take the perspective of the cells that are *not* burning during that time step: "I am a cell, and I am not burning. However, is there currently a fire in my neighborhood? If so, will I possibly burn this time step?" To determine "Will I burn?" a cell must query a fire resistance (or susceptibility) layer, which is the sum of all the factors (wetness value, fuel availability value, and so on) that determine the susceptibility of each cell for burning (the higher the value the more resistant the

cell is to the fire). A cell can then test its susceptibility (resistance) value. If it is below an identified value, the cell will burn. If not, it will reduce its resistance by a certain value (because of the drying effect caused by the burning fire) and perhaps in the next time step the cell will burn.

The fire action, shown in the following figure, contains the Java calls to spread the fire. For every time step, the action makes calls to ArcGIS tools through the Java objects to spread the fire.



The screenshot shows the 'Actions Editor' dialog box with the tab 'Fire' selected. The 'Java Imports' section lists several ArcGIS classes. The 'Source' section contains the following Java code:

```

def fire():
    print "running fire model"

factory = RasterWorkspaceFactory()
workspace = IRasterWorkspace2Proxy(factory.openFromFile(".\\projects\\cougars\\temp", 0))
fireRasterDataset = IRasterDataset2Proxy(workspace.openRasterDataset("fire"));
firecostRasterDataset = IRasterDataset2Proxy(workspace.openRasterDataset("firecost"));

# do focalStat on fire
op = RasterNeighborhoodOp()
ngh = INeighborhoodOpProxy(op)
fireGeoDataset = IGeoDatasetProxy(fireRasterDataset)
rNgh = IRasterNeighborhoodProxy(RasterNeighborhood())
isFire = ngh.focalStatistics(fireGeoDataset, esriGeoAnalysisStatisticsEnum.esriGeoAnalysisStatisticsSum, rNgh, false)

# create tempCost through a mapOp
firecostGeoDataset = IGeoDatasetProxy(firecostRasterDataset)
mapOp = RasterMapAlgebraOp()
iMapOp = IMapAlgebraOpProxy(mapOp)
iMapOp.bindRaster(isFire, "isFire")
iMapOp.bindRaster(firecostGeoDataset, "firecost")
tempCost = iMapOp.execute("con([isFire] > 0 and [firecost] > 0, [firecost] - 10, [firecost])")

# create tempfire through mapOp
iMapOp.bindRaster(fireGeoDataset, "fire")
iMapOp.bindRaster(tempCost, "tempcost")
tempFire = iMapOp.execute("con([fire] < 1 and [tempcost] < 1, 1, con([fire] > 0 and [fire] < 5, [fire] + 1, con([fire] > 4, 5, [fire])))")

IDatasetProxy(firecostRasterDataset).delete()
saver = ISaveAsProxy(tempCost)
saver.saveAs("firecost", IWorkspaceProxy(workspace), "GRID")

IDatasetProxy(fireRasterDataset).delete()
saver = ISaveAsProxy(tempFire)
saver.saveAs("fire", IWorkspaceProxy(workspace), "GRID")

```

The dialog box has 'OK' and 'Cancel' buttons at the bottom.

The fire model presented here is a simplified model to demonstrate how to use Java objects. It is not intended to be presented as a model to capture fire movement.

Generally, the code in the fire action tells the cell to do the following:

1. Evaluate whether there is a fire in the cell's neighborhood.
2. If yes, check the cell's resistance to the fire by evaluating its susceptibility value.
- 3a. If the cell is not resistant to the fire at this time step, it will burn.
- 3b. If the cell is resistant to the fire, reduce its resistance (susceptibility) value. The next time a fire approaches this cell, it may not be resistant to the fire.

Two rasters are imported (or called) into the fire model. The fire raster (`fireRasterDataset = IRasterDataset2Proxy(workspace.openRasterDataset("fire"));`) stores the location of the fire for that time step. Cells where the fire is present are assigned a 1; a 0 is assigned to cells where fire is not present. The firecost raster (`firecostRasterDataset = IRasterDataset2Proxy(workspace.openRasterDataset("firecost"));`) for each cell stores the resistance (or susceptibility) of the cell to the fire. The model identifies whether there are any fires in a nonburning cell's neighborhood (`isFire = ngh.focalStatistics(fireGeoDataset EsriGeoAnalysisStatisticsEnum .EsriGeoAnalysisStatsSum, rNgh, false)`). Since the sum statistic is used in the focalStatistics tool, if a cell on the resulting `isFire` raster has a value greater than 0, then the cell can possibly burn in that time step.

You have now identified the cells that are available to burn in that time step. You must then identify how resistant each of these cells is to fire. To do so, 10 resistance points are subtracted from the resistance raster, `firecost (tempCost = iMapOp.execute("con([isFire] > 0 and [firecost] > 0, [firecost] - 10, [firecost])"))`. If there is a fire in a cell's neighborhood (a value greater than 0 for `isFire`) and the cell is not resistant to the fire (`tempCost` is less than 0), then the cell should return a value of 1 for its burn status (`tempFire = iMapOp.execute("con([fire] < 1 and [tempcost] < 1, 1, con([fire] > 0 and [fire] < 5, [fire] + 1, con([fire] > 4, 5, [fire])))")`). The new locations of the current fire are used as the input to the next time step (`saver.saveAs("fire", IWorkspaceProxy(workspace), "GRID")`). In this way, the fire grows or diminishes with each run of each time step.

More complexity can be added to develop the fire movement, such as calculating for wind direction, spotting, slope, and aspect, and additional rules can be created to capture the movement of the fire.

This fire movement model is implemented in ArcGIS Spatial Analyst and can be accessed directly from the Agent Analyst model through Java objects. As a result of this interaction of modeling tools, the cougar agents modeled in chapters 2, 5, and 8 can directly react to the growing fire, treating it as a repellent, by integrating the results received from this fire model at each time step into the cougar agent's decision making within the Agent Analyst model. It should be noted that the modeler cannot view the fire's growth dynamically, because Agent Analyst is an out-of-process application. However, the results of the fire model can be replayed using any of the animation tools discussed in this chapter.

Case study 3: Adding temporal related viewing capability to the ABM output using ArcGIS Tracking Analyst and the bird migration model

Similar to the animation tools discussed later in this chapter's exercises, Esri ArcGIS Tracking Analyst is an extension for mapping objects that move or change status through time. Some of the features of Tracking Analyst are now available within ArcGIS Desktop 10 as "time-enabled layers," and most of the concepts presented in this case study should be easily translatable to this feature. This case study uses some of the additional features of Tracking Analyst for symbolizing temporal data and displaying temporal attributes. These features are not available with the ArcGIS time-enabled layers or the standard animation tools, but they are helpful when analyzing and communicating change-based models. These features are especially useful when visualizing point-based movement because of the option to display linear tracks to visualize the path of the moving points—in our case, of the moving agents.

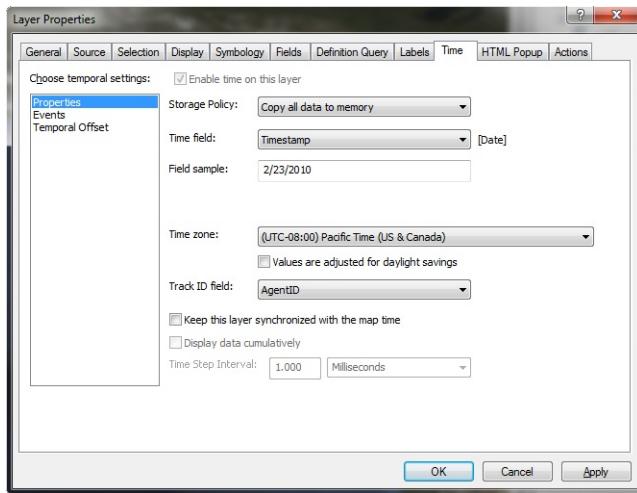
Tracking Analyst provides the capability to visualize and analyze time-series data, allowing for the visualization of real-time data representing moving phenomena (e.g., collared animals or vehicles). Of specific interest to Agent Analyst, Tracking Analyst provides tools to symbolize time-based data (e.g., moving agents), play back their movement, and create summary charts of the movement data, and also provides the ability to create AVI movies from the animation.

The symbolizing capability lets you symbolize point, line, and polygon data based on an event and then on the time stamp associated with the event (e.g., the color, size, and symbol), render the time interval, create track lines for the moving object, and uniquely symbolize the current position of the object. The Tracking Analyst Playback Manager allows you to view, rewind, pause, fast-forward, or play the time data backward or forward.

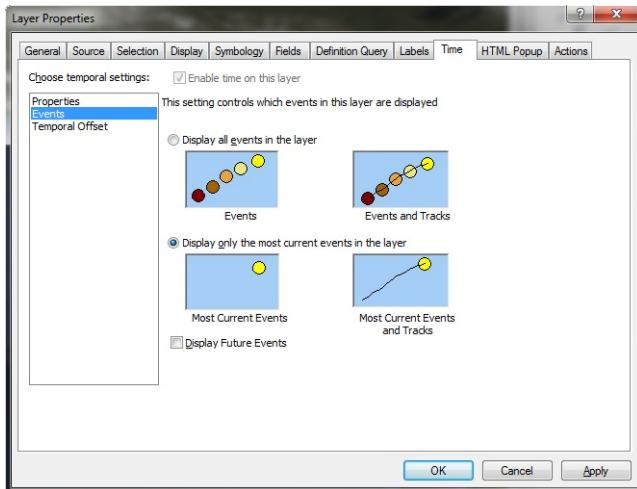
The Tracking Analyst temporal symbolization capability

The results from the bird migration model described and displayed in the exercises presented in this chapter will be used in this case study to demonstrate the capabilities of Tracking Analyst. The temporal layer created in the bird migration model (named Birds-Temporal) is used as the input data for Tracking Analyst.

The Tracking Analyst symbolization capabilities are integrated with and added to the ArcMap layer properties. Once the Tracking Analyst is enabled, the Layer Properties dialog box for Birds-Temporal can be opened and the Time tab selected. The following three images illustrate a few changes made in the dialog box to the Time tab and Symbology tab for setting up Tracking Analyst-configured layers.

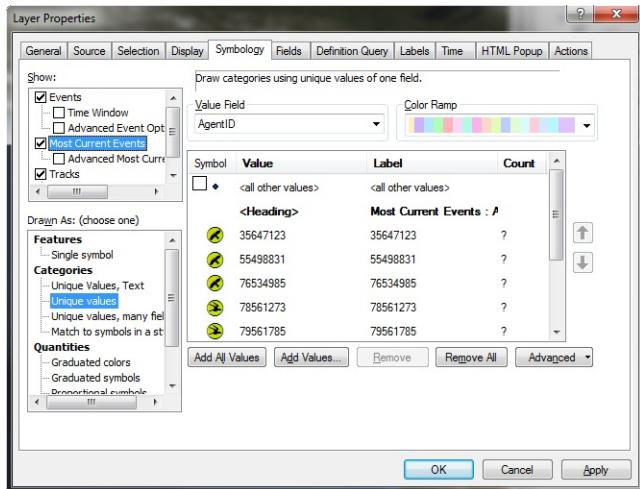


The Properties options on the Time tab are available for most layer types in ArcMap, ArcScene, and ArcGlobe without using Tracking Analyst. Notice in the figure that Time field is set to Timestamp and Track ID field is set to AgentID. These two options specify the two required fields to create a time-enabled layer. By configuring these options, ArcMap considers the layer “time-enabled” and provides some tools for displaying changes in both ArcGIS Desktop and ArcGIS Server products. For more on time-enabled layers and visualization tools, please see the ArcGIS Desktop Help.



Notice in the preceding image that the option to “Display only the most current events in the layer” is selected. This option lets you choose to display all of the points in the layer as “breadcrumbs” or just the current event during playback.

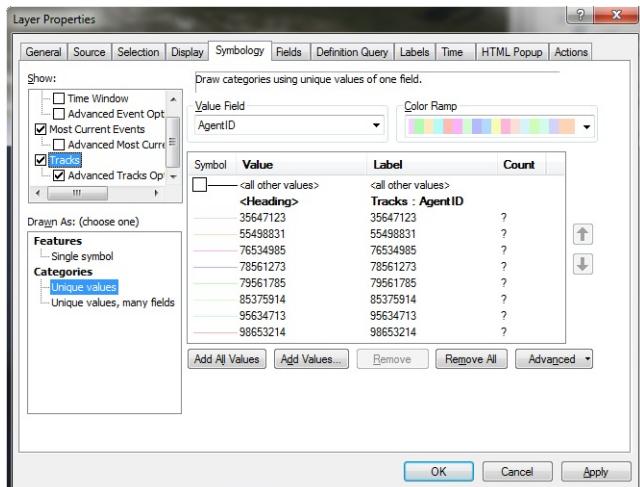
The Symbology tab allows you to customize the appearance of the layer's features based on its temporal attributes. In the following image, the Symbology tab of the Layer Properties dialog box for the Bird Temporal layer displays the bird agents.



10

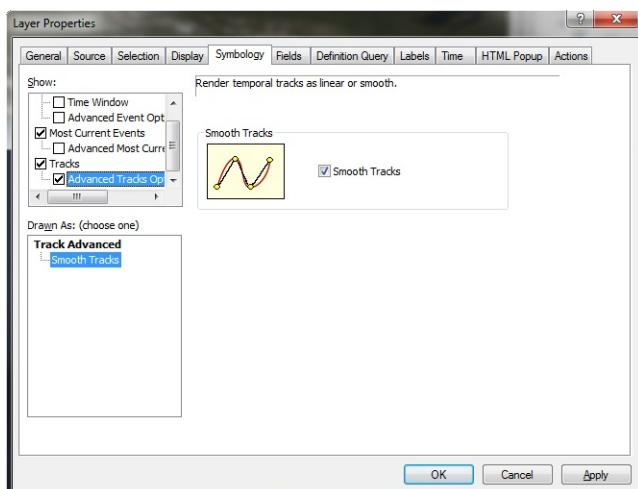
To examine the properties for the most current events, select Most Current Events in the Show list in the top-left panel of the Symbology tab. This property defines how the current point is displayed during playback. Also note in the figure that the icons for the points are set to display by category using unique values and represent the species of bird (goose or pintail).

The symbology for the flight paths (tracks) can also be controlled on the Symbology tab. In the following figure, the Layer Properties dialog box for the Bird Temporal layer displays the tracks for the bird agents.



The track properties can be set by selecting Tracks in the Show list in the top-left panel of the Symbology tab. When enabled, this allows you to insert a line behind the point to represent the agents' paths. The symbology options allow you to define how these path line features will be displayed. Note that the lines for this layer are set to display by categories using unique values and represent the AgentID.

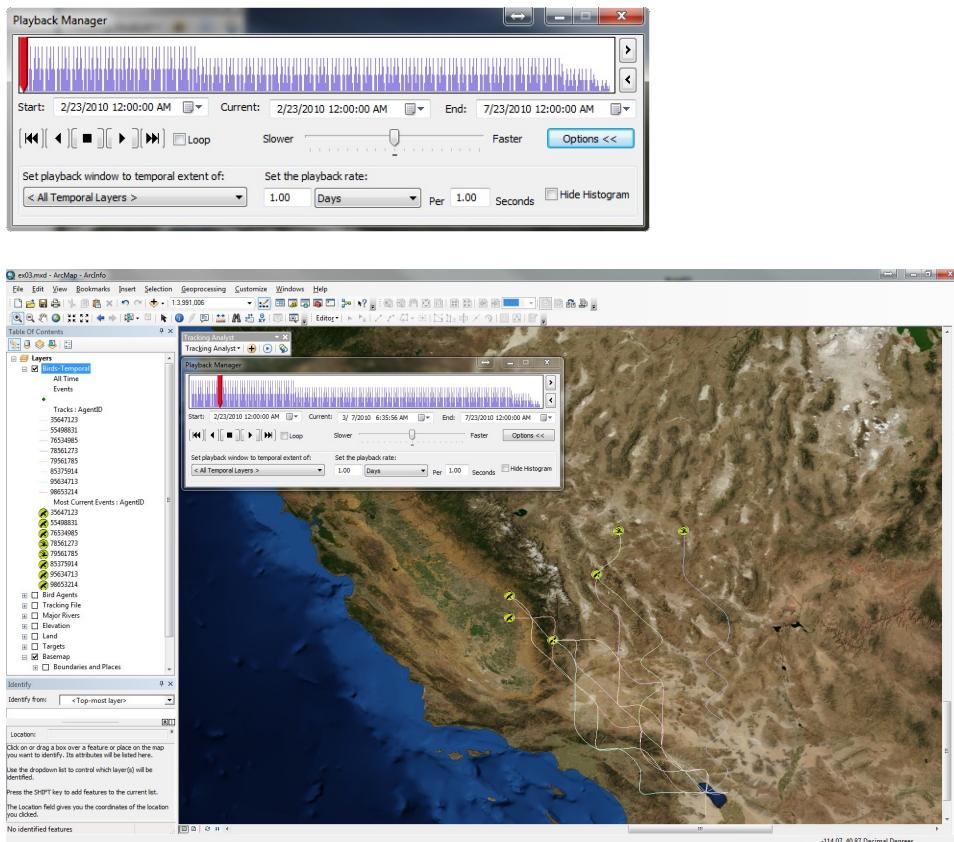
You can toggle the Smooth Tracks option by selecting Advanced Tracks Options in the Show list, as shown next. When on, this option "smoothes" the path line features to make the display of these features more realistic. This is done for stylistic reasons to represent the paths the agents may have taken between ticks.



Using Tracking Analyst Playback Manager to analyze and communicate the model output

The Tracking Analyst Playback Manager provides the controls to review the temporal layer as an animation. The Playback Manager can be accessed from the Tracking Analyst toolbar. The Playback Manager dialog box allows you to replay the model run using the temporal attributes and properties you defined in the temporal data layer.

In the following figure, the playback rate is set to one day per second. This rate may be adjusted to change how often the existing points are drawn on the map. When the Play button is clicked, playback begins from the currently displayed date in the Current field while also displaying the current position of the agents in the ArcGIS display, as shown in the following figure.



10

Notice in the preceding image that because the options for smoothing and displaying the tracks were selected in the layer properties, a curved line is drawn behind the point features. The histogram of feature movement displayed in the Playback Manager acts as a timeline view of the agents' movements. The red pointer identifies the current time for the playback. You can drag the pointer or use the Start, Current, and End date drop-down lists to select a date range or display a specific date.

Creating a video file of the bird migration model

You may want to create a video file of the playback. Once you are satisfied with your model and your temporal data layer settings, you can easily export the animation to an AVI file.

A video file has been created for the bird migration model. To play the video file, navigate to C:\ESRIPress\AgentAnalyst\Chapter10\Movies and play CaseStudy3.avi. Notice the Symbology and PlayBack Manager capabilities discussed earlier.

Case study 4: Integrating third-party movie-making software with agent-based models to facilitate dynamic visualization

Two approaches will be presented for integrating movie-making software with ABM. The first example (courtesy of Tony Turner) illustrates an animation of movement as determined in the agent-based model, and the second example (courtesy of Paul M. Torrens) illustrates how the simulated characteristics of an agent's movement are integrated into the decision making of the agents.

Animating the movement of the agents

There is synergy between ABM software and movie-making software. In this first example, the ABM software models the interaction of agents with their surroundings and with other agents. In addition, it allows the agents to query spatial data, it defines the decision rules, and it provides the mechanism to make a decision that results in a decision. These decisions can be animated with the movie-making software to produce realistic simulations of varying phenomena interacting to provide the visual realism that our eyes are accustomed to viewing when assessing patterns.

Normally, viewing the results of agent-based models in a GIS allows the modeler to see the static positional movement of agents captured from hundreds of time steps in one continuous stream and sped up to simulate movement. However, third-party movie-making software can be integrated with GIS and agent-based models to add definition and realism to the agents and their surroundings as they move. Use of the movie-making software gives the modeler the ability to increase the realism in illustrating the agents' characteristics and corresponding changing behaviors (e.g., movement of limbs, turning of the head, variation in speed, or a realistic-looking landscape).

Third-party movie-making software can accept input in the form of agent movement and then project the moving agents over variable components such as surfaces and features. Realistic effects can be added to characterize the terrain, vegetation, 3D structures, and lighting to represent changes in time of day and seasons, or even lights projected from an agent that illuminate the landscape.

Viewing the spatial data with characteristics simulating realism allows the modeler to view the distribution of phenomena in their positions relative to one another and may influence rule development. Although only providing a visual display of results, these movies can illustrate another perspective of the model analysis, likely a qualitative interpretation of the model. The user can observe visually how agents are interrelating across the spatial landscape or reacting to the rules provided in the model.

Agents in movie making

In Agent Analyst, agents establish the relationships and interactions with their surroundings (the landscape) and with other agents. From these interactions, the agents make a decision that can result in a move and a behavior (e.g., stand, walk, or run). The location and the behaviors of an agent (e.g., a human agent) can be mapped to a movie object (e.g., a human object). For example, when a human agent is walking through a crowded market, the appropriate properties for the movie human object can be set to move the movie human object over that same landscape. Similarly, when the Agent Analyst human agent begins to walk or move in a time step, the appropriate properties for the movie human object can be set to make the movie human object imitate that movement but with greater realism. Perhaps the agent will flee from an explosion in the market. The market can be given animated persona as well in the movie-making software.

You can run the Agent Analyst model in a batch and save the results (the location and associated behaviors) as a track and import the track and the terrain into the movie-making software to create the movie. The frames of data can be assembled into a QuickTime or an AVI movie.

Landscapes and environment in movie making

You can import the GIS layers used in the agent model into the movie-making software and thus the agents will be moving across the same landscape over which they were modeled. These GIS landscapes closely simulate the real terrain being modeled, allowing you to add lighting or other realistic objects to the appropriate locations in the landscape, such as vegetation, killed prey, human development, and animated objects (such as random moving vehicles along the roads).

In this first example, a vehicular line-of-sight model was selected to illustrate the movement of agents (a tank and a reconnaissance vehicle) across a GIS spatial landscape. These movements were incorporated into third-party movie-making software.

Model background

This vehicular line-of-sight model is a synergy of the movie-making software with the agents defined in the Agent Analyst software while moving across a GIS spatial landscape. In this example, a single reconnaissance vehicle is conducting a surveillance mission while avoiding enemy contact (the enemy tank). This movie allows you to observe how the vehicles move over the terrain and turn to track the line of sight. The line of sight can be displayed as an extension of the vehicle. Because line of sight is typically static, incorporating it with moving agents in the movie software allows a view of behavior change in the agents when they are in the line of sight of the other vehicle.

The terrain in this movie was created using ArcGIS, the behaviors of the vehicular agents by using Agent Analyst, and the 3D objects and effects by “movie” software. There was no attempt to be complete, accurate, or precise.

Rules for agents in a vehicular line-of-sight model

The agents interact with each other according to the following specified rules:

- Reconnaissance vehicle
 - Find the closest tank
 - Get close to the tank, but not “real” close
 - Move toward the tank over the easiest terrain
 - Maintain line of sight at a safe distance
- Tank
 - Follow terrain to high ground if contact is made with surveillance vehicle to better view that vehicle
 - Move away from reconnaissance vehicle if too close, but maintain line of sight

Agent interactions during each time step were recorded as JPEG frames. The terrain displayed in this movie is the landscape over which the agents were modeled. To more accurately develop the model, a vegetation layer can be incorporated to capture movement in forested areas.

A video file has been created using third-party movie-making software. To play the video file, navigate to C:\ESRIPress\AgentAnalyst\Chapter10\Movies and play Chapter10CaseStudy4a.wmv.

Integrating movement into the decision making of the agents

In the previous example from Tony Turner, the agent’s physical movement is determined by a series of decisions made within Agent Analyst and then imported into animation software. In the following examples developed by Paul M. Torrens, an agent’s movement and its physical characteristics are used as part of the agent’s decision making. Torrens’s goal is to place the agents in the right place at the right time, doing the appropriate actions while interacting with other agents.

Torrens’s model does not generalize movement within the landscape; instead, the physical characteristics of the bodily movement dictate the movement. In his models, pedestrian agents are pre-coded with an environmental awareness, and he wraps these agents in a geographic exoskeleton. Then, the pedestrian agents in these models consider various higher-, medium-, and lower-level behaviors when making movement decisions.

The integration of high-, medium-, and low-level behavior simulates realistic dynamic movement. The high-level behavior accounts for global considerations for the path planning of the movement. Medium-level behavior deals with vision and steering to simulate the gross move-

ment of the agents. Low-level behavior accounts for movement decisions that are imposed by physical bodily movement at the level of the agent's skeleton.

Moving a pedestrian agent

The pedestrian agent's higher-level behaviors include considerations that occur prior to any movement. Conceptually, Torrens's agents plan their trips prior to moving. This is accomplished within the models with a path-planning heuristic that can accept GIS data that the agents use to guide their overall movement.

Agents consider medium-level behaviors by visually identifying their surroundings. The agents will move between way points on the shortest paths but will do so using vision and steering. Vision is simulated using two filters. The first filter identifies the zone of interaction. In this zone, the agent gathers immediate information, which accounts for a behavior to avoid other agents—to alleviate the “breathing down their neck” effect. The second vision filter creates a cone allowing the agent to account for things in the forward direction of the agent. The agent will tally the nearest neighboring agents—their distance—and then calculate vectors for potential collisions. This second filter for vision allows three strides of reaction time for the agent. The agents develop a “mental map” of their immediate surroundings. The mental map includes the location, position, and direction and speed of other agents and objects that are important to them.

10

Steering a pedestrian agent

The pedestrian agents use the mental map to identify the visual cues for the agents to react to using steering to change their relative vectors of motion. This allows for agents to react, interact, and proact to their surroundings. The agents may move toward or away from the position of other stationary agents and objects. When pedestrian agents seek and flee other agents, they create curved paths because of adjustments the agent must make to the target. Pedestrians generally slow down as they reach their target.

A second component affecting an agent's steering is pursuing and evading other moving agents. Chasing or avoiding a moving target is accomplished through steering. Not only must the pedestrian agent move toward an agent when pursuing or away when fleeing, the agent must also anticipate the future position of the agent or object it is pursuing or avoiding.

A third component that affects an agent's steering involves the agent remaining on its shortest path. An agent will deviate from its path only to avoid a collision. Once an agent deviates, it must try to return to the path by the use of steering.

A fourth component affecting an agent's steering is a separation from other agents that must be created to maintain personal space and to avoid collisions. In this model, an agent will scan its immediate surroundings to identify potential collisions, calculate the collision vector, and then change its vector.

A fifth component influencing an agent's steering is alignment. Alignment allows agents to move with and join groups for collective movement. The agents can align with the other agents' vectors and match their vectors and velocity. Alignment allows the agent to keep pace with a friend.

Low-level agent behavior accounts for the agent's locomotion. Low-level behavior is based on the kinesiology/biomechanics of the body's locomotion. Torrens uses agent rigs to simulate the joints and bones of the agent's body. He uses root nodes for hips and creates linear chains to simulate movement. Motion is created by starting at the root nodes and then moves out through the chain of nodes by extrapolation and interpolation. Torrens has empirically tested the simulated movement with live humans.

Three examples of Torrens's model

Torrens's first example models a series of agents programmed with varying agendas, behaviors, and characteristics: passive, pursuing, hurried, paced, tired, drunk, seniors, middle-aged, and young agents. By letting the agents interact with high-, medium-, and low-level decision making, realistic crowd dynamics are created. The agents do not collide; they follow general paths, create fast moving lanes, and aggregate into subgroups, and people with ambulatory problems create bottlenecks. To view the simulation of the model, navigate to C:\ESRIPress\AgentAnalyst\Chapter10\Movies and play Chapter10CaseStudy4b.avi.

A second model simulates a protesting crowd and demonstrates a closer look at some of the low-level behavior. To view this simulation, open C:\ESRIPress\AgentAnalyst\Chapter10\Movies and play Chapter10CaseStudy4c.avi.

A third model simulates a pedestrian crowd fleeing from an explosion. The agents attempt to follow the shortest paths while avoiding collisions. Bottlenecks of irregular, organic-shaped groupings of agents evolve in this orderly, nonpanicked fleeing crowd. Notice that the crowd creates its own gridlock while avoiding potential collisions within limited space. A stop-and-go behavior is created as the crowd moves as a unit. To view the simulation, open C:\ESRIPress\AgentAnalyst\Chapter10\Movies and play Chapter10CaseStudy4d.avi.

These realistic pedestrian movement models provide an excellent study of the interactions and dynamics of movement within crowds.

Exercises

The following four exercises have been provided to walk you through the process of utilizing the GIS spatial modeling capabilities from an agent-based model and how to use ArcGIS 2D and 3D animation tools for the spatial modeling visualization process. These exercises have been developed around a bird migration model.

The modeling scenario

Bird migration is an incredibly complex activity to model. Because this is a learning exercise, the model presented here uses a few simple rules to simulate bird navigation to illustrate how to call ArcObjects using Agent Analyst and how to use the animation tools included with ArcGIS Desktop to visualize the model results. The bird migration model builds on much of the same action code and many of the same concepts you have studied in previous chapters, so the exercises will go into detail only on topics that have not been covered previously.

It is important to note that the bird migration model and the rules developed for it were developed as a learning example using existing data, and they are in no way meant to be a definitive model to represent bird migration patterns.

10

Background information

The Salton Sea, located in Southern California, is the largest lake in the state, at about 35 miles long and 15 miles wide. The Sea is an important biological resource, playing host to more than 400 species of birds with some species at the Sea numbering in the millions. The Sea is an especially critical resource for migratory birds as a stopover on the Pacific Flyway, with more than two-thirds of all migratory bird species being observed through bird-banding records.

The bird migration model presented in this chapter uses a small subset of eight bird-banding records from the Patuxent Wildlife Research Center Bird Banding Laboratory. These records consist of two species (snow and Ross's geese and northern pintail), and each are for birds that were banded at the Salton Sea. These species were used because they are two of the most frequent species banded at the Sea and because they have very different locations in their migrations to and from the Sea. While the entire record includes where the banded bird was encountered or recovered, the model only uses the banding location as the agents' starting point.

This model uses a few simple rules to simulate bird navigation:

- Birds avoid flying over large mountain ranges; they prefer to fly through valleys. This is modeled by simply setting a maximum elevation parameter so that agents will avoid higher elevations when navigating over a Digital Elevation Model (DEM).

- Birds use major rivers as landmarks for navigation. This is modeled by creating a distance grid from a major rivers polyline feature class and programming the agents to prefer to stay close to them.
- Birds don't fly long distances over open water; they prefer to fly over land but use the coastlines as landmarks. This is modeled by creating a distance grid from a continents polygon feature class and programming the agents to prefer to stay close to land.
- Birds fly to a target region. This model uses two different target regions for the two species: the geese fly to the Arctic North Slope, and the pintails fly to the Prairie Pothole Region. This behavior is modeled by creating a distance grid to each of these polygon feature classes and programming each agent to move closer on each model tick until it reaches its region.

An overview of the bird migration model

This exercise will provide an overview of the bird migration model. In subsequent exercises you will learn how specific portions of the model were developed, but more importantly, you will see how to use the ArcGIS tools to more effectively display the model results.

Exercise 10a

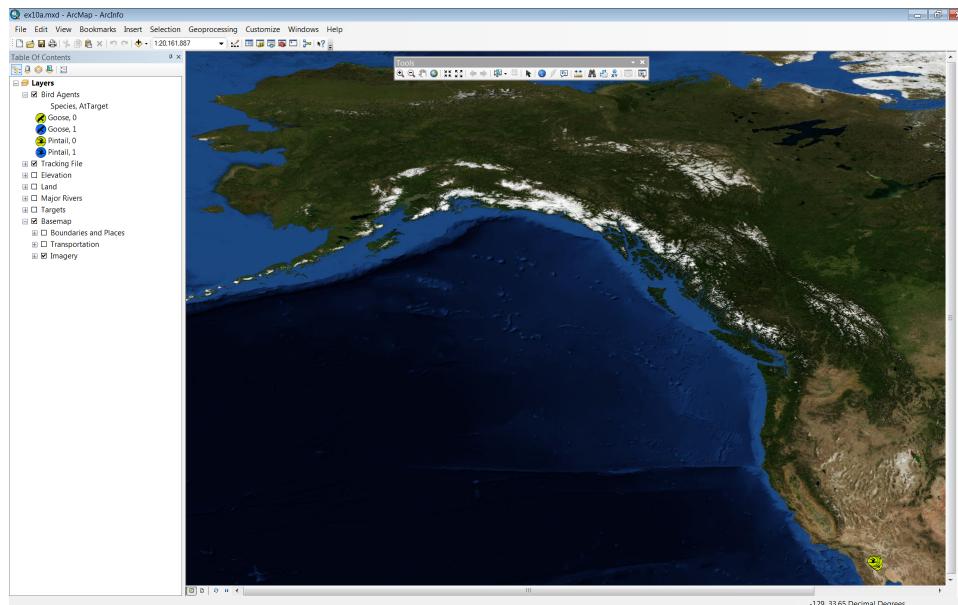
Load the ArcMap document

- 1 Start ArcMap. Navigate to C:\ESRIPress\AgentAnalyst\Chapter10. Open ex10a.mxd.

Note: Depending on your system, you might see a message asking if you want to enable hardware acceleration to improve drawing performance. You can answer Yes to try the accelerated mode.

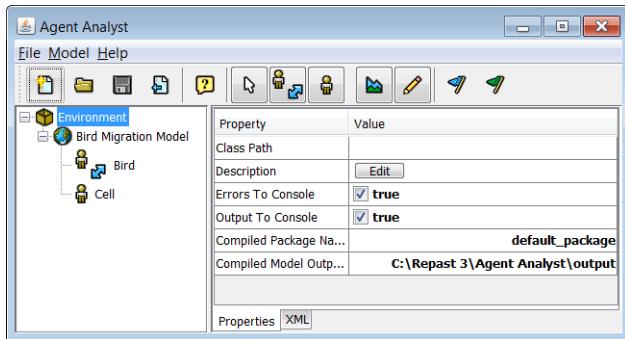
The map document opens. In the ArcMap display, you will see points representing the bird agents displayed on top of an imagery map service. (Internet access is required to view this layer.) In ArcToolbox, the Chapter10Ex10a toolbox has been created for you.

Note: If you do not see the imagery in the ArcMap display window, you may need to clear and then select the Imagery layer under the Basemap group layer in the Table Of Contents.



Load the Agent Analyst model

- 2** Right-click the Chapter10Ex10a toolbox, point to New, and select Agent Analyst Tool.
- 3** Once the Agent Analyst window appears, click File and select Import. Do not save the changes when prompted.
- 4** Navigate to C:\ESRI\Press\AgentAnalyst\Chapter10\Models and open ex10a.sbp.



There are two agents in the bird migration model: the bird agent that includes the two bird species, and a cell agent that is used in defining the neighboring cell locations immediately around the bird agent and when making a movement decision.

Explore the bird agents

- 5** In the Environment panel, click Bird. In the Property panel, click the Edit button to the right of the Data Source property to open the Data Source Editor.

The bird agents are created from information in the Birds shapefile. Each bird has a band number that acts as its ID, the bird's starting location (OrigX and OrigY), the species, a time stamp that we will increment on each model tick, and a Boolean AtTarget field that will denote whether the bird has reached its target.

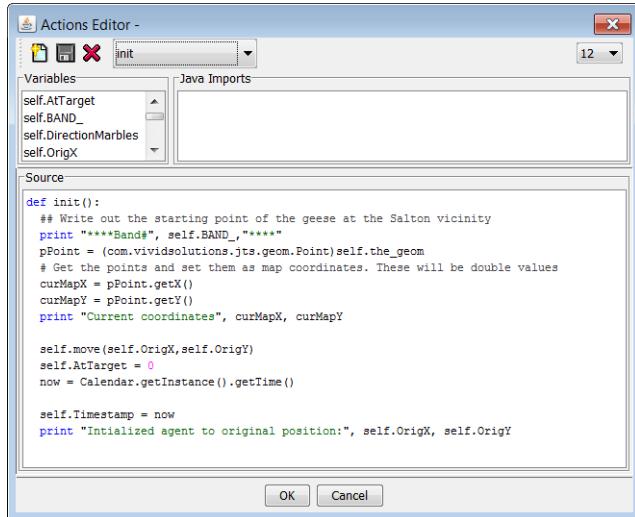
Explore the actions for the bird agent

- 6** In the Property panel, click the Edit button to the right of the Actions property to open the Actions Editor. Open the Actions drop-down list, and then scroll down the list to review the actions available.

You have seen variations of many of these actions in previous chapters. The actions work in conjunction to aid the bird agents in establishing the relationship for each criterion influencing their decision making, weighting each appropriately, and then making a move. There are a few actions specifically related to calculating time stamps and saving agents to a tracking file that you will explore in more detail in exercise 10b.

Explore the bird agent's init action

- 7 With the Actions Editor for the bird agent open, select the init action. Review the code shown in the following figure.



10

When running multiple iterations of your model or when you are simply debugging, it is often useful to simply return the agents to their starting location and reset some initial field values instead of specifying a new shapefile. The code in the bird agent's init action reads two fields with the original coordinates and returns the agent to that location and resets the AtTarget field to 0. This code also resets the agent's starting time to the system time.

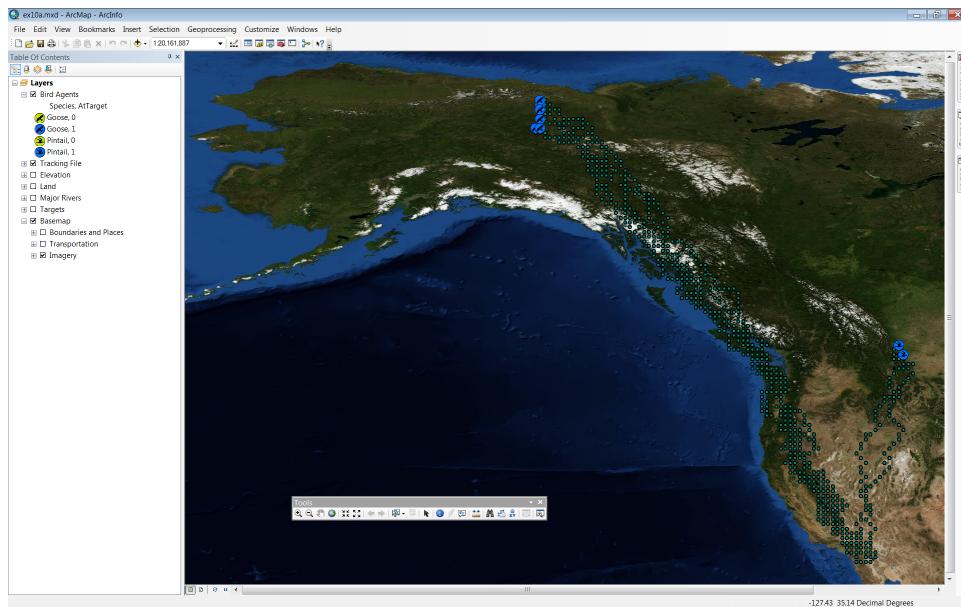
Run the model

Note: This chapter has you run the model once. Each time the model is run, the model writes to the same track shapefile. If you plan to run the model multiple times you should make a copy of the empty track shapefile. Using ArcCatalog, navigate to C:\ESRIPress\AgentAnalyst\Chapter10\Data, select BirdTracking.shp, right-click, and select Copy. Select the Data directory the shapefile is in, right-click, and select Paste. A copy of the BirdTracking shapefile is created (BirdTrackingCopy.shp). Before running the model again, navigate to C:\ESRIPress\AgentAnalyst\Chapter10\Data and delete BirdTracking.shp. Copy BirdTrackingCopy.shp and paste it into the Data directory the shapefile is in. A copy of BirdTrackingCopy.shp is created (BirdTrackingCopy2.shp). Select BirdTrackingCopy2.shp, right-click, and click Rename. Rename BirdTrackingCopy2.shp to BirdTracking.shp. You might need to add BirdTracking.shp to ArcMap.

- 8 In the Agent Analyst window, click Run. The Repast toolbar and the Bird Migrations Model Settings window appear.

- 9 On the Repast toolbar, click the Start button and observe the birds move.

The birds will continue to move toward their targets using the rules applied to them until they reach the target. When they reach the target, the AtTarget field value will change from 0 to 1 and a different point symbology will be applied. (Note that you might experience some bird agents having trouble finding their targets and flying past them. Additional rules can be added to ensure that all the bird agents locate their targets.) You will notice that a track of points is drawn behind the agents as well. This is accomplished using ArcObjects to write out a point along with a time stamp and some agent state values. You will be examining how to do this in exercise 10b.



Exercise 10a

- 10 Close Agent Analyst and ArcMap without saving your changes.

Creating a tracking log using ArcObjects to save agents at each tick

Most of the models you have created so far have changed the state of the underlying features without maintaining any kind of reproducible log or tracks of the movement or state change. This makes it very difficult to run multiple iterations to create visualizations of the model run or compare the details of multiple model runs. One way to do this is to use ArcObjects, which is integrated with the Agent Analyst toolkit. In this exercise we will look at how to maintain these tracks by creating a new point in a shapefile for each tick in the model.

Exercise 10b

Load the ArcMap document

- 1 Start ArcMap. Navigate to C:\ESRIPress\AgentAnalyst\Chapter10. Open ex10b.mxd.

In the ArcMap display, you will see points representing the bird agents displayed on top of an imagery map service. (Internet access is required to view this layer.) In ArcToolbox, the Chapter10Ex10b toolbox has been created for you.

Examine the attribute table for the Bird Agents layer

- 2 Right-click the Bird Agents layer in the ArcMap Table Of Contents, and then click Open Attribute Table.

Each bird has a band number that acts as its ID, the bird's starting location (OrigX and OrigY), the species, a time stamp that we will increment on each model tick, and a Boolean AtTarget field that is used to denote whether the bird has reached its target. (Note that the time stamp represents when the model was stopped or when the birds have reached their destination. Due to the randomness in the decision making of the bird agents, your time stamps may be slightly different from those in the following image.)

| FID | Shape | BAND_ | OrigX | OrigY | Species | Timestamp | AtTarget |
|-----|-------|----------|--------|-------|---------|-----------|----------|
| 0 | Point | 55498831 | -115.5 | 32.83 | Goose | 2/26/2012 | 1 |
| 1 | Point | 76534985 | -115.5 | 32.83 | Goose | 3/4/2012 | 1 |
| 2 | Point | 95634713 | -115.6 | 33 | Goose | 3/1/2012 | 1 |
| 3 | Point | 85375914 | -115.6 | 33 | Goose | 3/1/2012 | 1 |
| 4 | Point | 35647123 | -115.5 | 33.16 | Goose | 2/25/2012 | 1 |
| 5 | Point | 98653214 | -115.6 | 33.16 | Goose | 3/12/2012 | 1 |
| 6 | Point | 78561273 | -115.6 | 33.16 | Pintail | 11/27/201 | 1 |
| 7 | Point | 79561785 | -115.8 | 33.16 | Pintail | 11/25/201 | 1 |

(0 out of 8 Selected)

Bird Agents

Examine the attribute table for the Tracking File layer

- 3 Right-click the Tracking File layer in the ArcMap Table Of Contents, and then click Open Attribute Table.

This point shapefile is populated with a point for each agent at each tick in the model. It also captures the time stamp of the tick by using an offset parameter that the user can change. The Timestamp field can be used to create animations in ArcMap, ArcScene, and ArcGlobe when the model is complete.

| FID | Shape | Id | Timestamp | AgentID | AtTarget |
|-----|-------|----|------------|----------|----------|
| 0 | Point | 0 | 10/8/2011 | 55498831 | 0 |
| 1 | Point | 0 | 10/8/2011 | 76534985 | 0 |
| 2 | Point | 0 | 10/8/2011 | 95634713 | 0 |
| 3 | Point | 0 | 10/8/2011 | 85375914 | 0 |
| 4 | Point | 0 | 10/8/2011 | 35647123 | 0 |
| 5 | Point | 0 | 10/8/2011 | 98653214 | 0 |
| 6 | Point | 0 | 10/8/2011 | 78561273 | 0 |
| 7 | Point | 0 | 10/8/2011 | 79561785 | 0 |
| 8 | Point | 0 | 10/9/2011 | 55498831 | 0 |
| 9 | Point | 0 | 10/9/2011 | 76534985 | 0 |
| 10 | Point | 0 | 10/9/2011 | 95634713 | 0 |
| 11 | Point | 0 | 10/9/2011 | 85375914 | 0 |
| 12 | Point | 0 | 10/9/2011 | 35647123 | 0 |
| 13 | Point | 0 | 10/9/2011 | 98653214 | 0 |
| 14 | Point | 0 | 10/9/2011 | 78561273 | 0 |
| 15 | Point | 0 | 10/9/2011 | 79561785 | 0 |
| 16 | Point | 0 | 10/10/2011 | 55498831 | 0 |

(0 out of 991 Selected)

Tracking File

- 4 Close the Attribute Table.

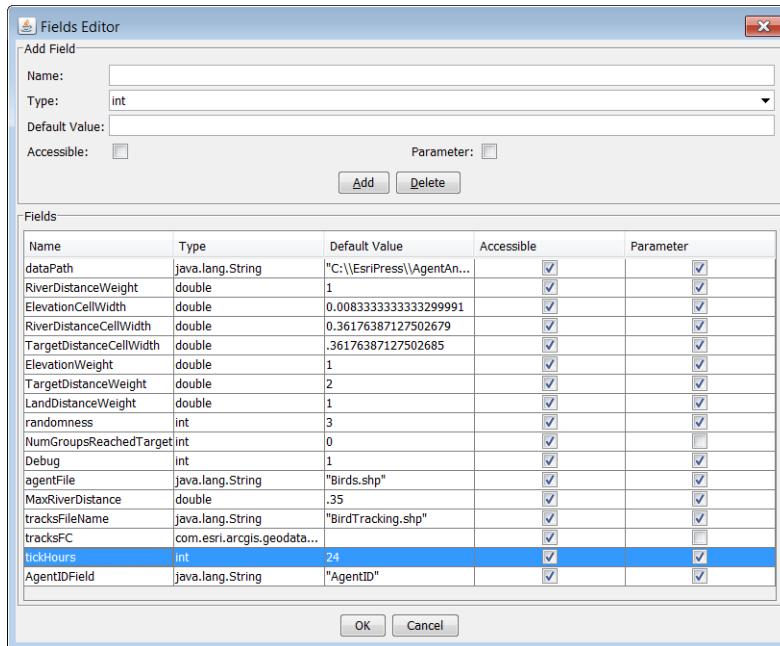
Load the Agent Analyst model

- 5 Right-click the Chapter10Ex10b toolbox, point to New, and select Agent Analyst Tool.

- 6 When the Agent Analyst window appears, click File and select Import. Do not save the changes when prompted.
- 7 Navigate to C:\ESRIPress\AgentAnalyst\Chapter5\Models and open ex10b.sbp.

Examine the model fields

- 8 In the Environment panel, click Bird Migration Model. In the Property panel, click the Edit button to the right of the Fields property to open the Fields Editor.
- 9 If necessary, scroll down or resize the dialog box to view the tickHours field.



This field captures the number of hours that each tick represents. This is how you will track actual time in the model. This field will be used as an offset to add to the time stamp. It is currently set to 24 so that each tick will represent one day.

- 10 View the tracksFileName and tracksFC fields.

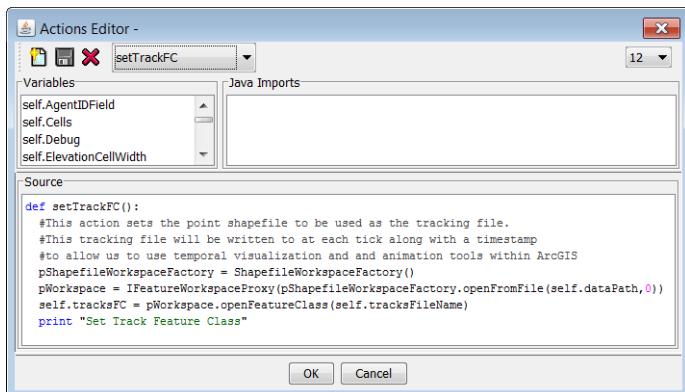
Notice that the Type column lists the tracksFC field as com.esri.arcgis.geodatabase.IFeatureClass. This is a reference to the ArcObjects library IFeatureClass interface, which is the main interface for getting and setting the properties of a feature class. (A shapefile is a type of feature class, along with feature classes found within a geodatabase.) Because this is an object reference, you cannot explicitly define the value using the Default Value column in the Fields Editor, as you can for numbers and string types. You need to set this value in an action that runs before the first tick (Tick 0.01).

To set the value of this shapefile object reference, tracksFileName (the name of the shapefile that will store the tracks) needs to be associated with the tracksFC object. The tracksFileName string is exposed as a parameter so that you can modify it at model run time. For more on the IFeatureClass interface, see the Esri Developer Help.

Examine the setTrackFC model action

- 11** In the Environment panel, click Bird Migration Model. In the Property panel, click the Edit button to the right of the Actions property to open the Actions Editor.
- 12** From the Actions drop-down list, select setTrackFC and review the code shown in the following figure.

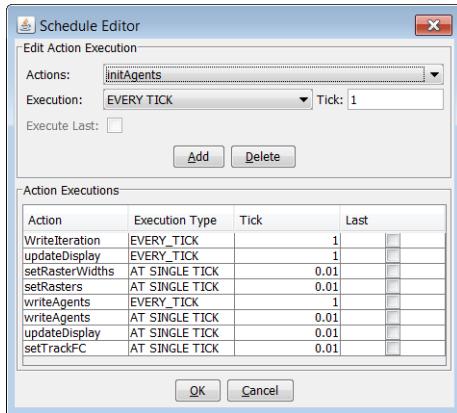
This action creates an IFeatureClass object from the shapefile name and workspace (folder) location.



Note that you first create a reference to the IWorkspace or folder (`pWorkspace = IFeatureWorkspaceProxy(pShapefileWorkspaceFactory.openFromFile(self.dataPath,0))`). Then, using the file name of the shapefile in that folder that you stored in the model-level field, you create the IFeatureClass reference (`self.tracksFC = pWorkspace.openFeatureClass(self.tracksFileName)`). Once this action executes, the model can manipulate the shapefile as an IFeatureClass using standard ArcObjects calls using the variable `self.tracksFC`.

Examine the model schedule

- 13 In the Environment panel, click Bird Migration Model. In the Property panel, click the Edit button to the right of the Schedule property to open the Schedule Editor.



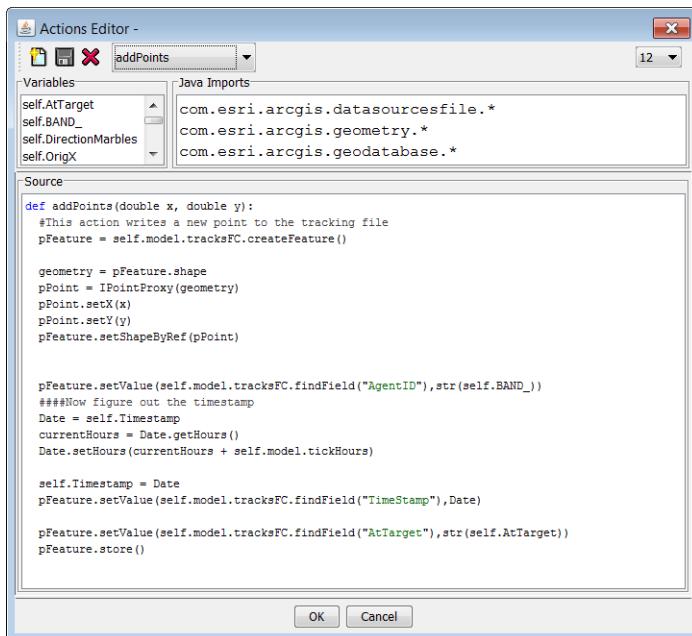
Note in the preceding figure that the setTrackFC action is scheduled to execute on Tick 0.01. This means that it will execute before the first tick, while the model is initializing. This action needs to execute before the first tick so that you can use the reference variable tracksFC while the model is running to create new points for the agents' locations at each tick.

10

Examine the addPoints action of the bird agent

- 14 In the Environment panel, click Bird. In the Property panel, click the Edit button to the right of the Actions property to open the Actions Editor.

- 15** From the Actions drop-down list, select addPoints and review the code shown in the following figure, noting the Java Imports statements in the top-right panel below the Actions list, which reference the Esri ArcObjects Java libraries.

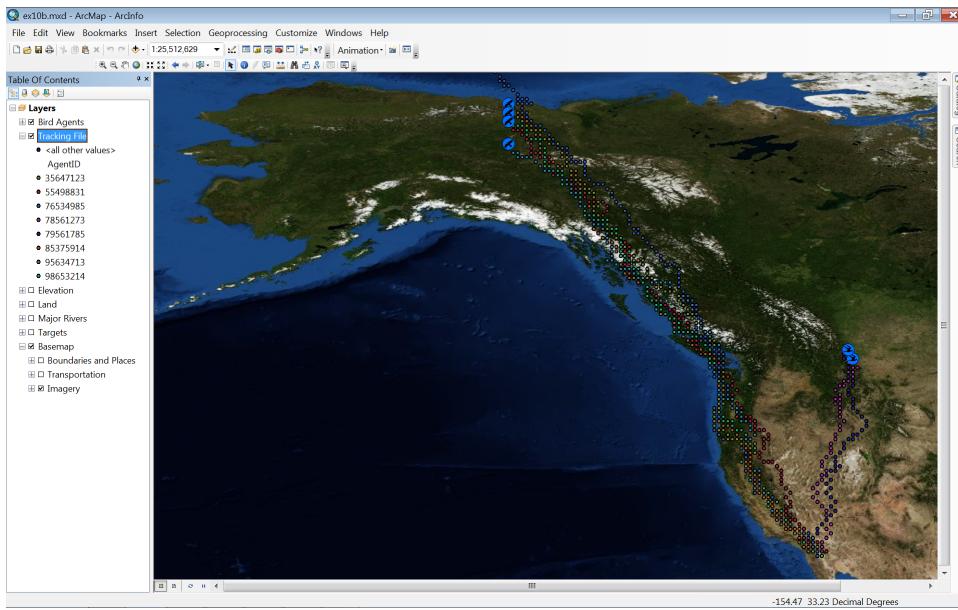


Without going into detail about the ArcObjects functions, this action accepts x and y variables and adds a new point to the tracking file along with the agent's ID, new time stamp, and new AtTarget value.

This action is called by the CalculateMove action, which is called in the agent's step action. The step action executes on each tick.

Since you have already run the model in exercise 9a, if displayed the Tracking File layer should look similar to the following image. In the image, the symbology for the track points have been set to display in a different color for each unique value in the AgentID field. As you can see, this gives you a much better product for analyzing the paths of the agents than the “live” model run.

Note: To symbolize your tracks in a similar manner, right-click Tracking File in the Table Of Contents and select Properties... Select the Symbology tab. In the Show: list, select Categories, and then select Unique Values. In the Value Field drop-down list, select AgentID. In the Color Ramp drop-down list, select any of the discrete (not gradient) color ramps. Then click the Add All Values button near the bottom of the window. Click OK.



16 Close Agent Analyst and ArcMap without saving your changes.

Demonstrating the 2D ArcMap animation tools using the bird migration model

In this exercise, you will explore how to use the tracking file generated from the bird migration model in exercises 10a and 10b to create an animation. Once a tracking file has been created, the standard ArcGIS animation tools may be used to animate the results. Using the animation tools in ArcMap, ArcScene, or ArcGlobe allows you to better understand and communicate how data changes over time and space. Some examples of using the animation tools include:

- Move features—animate the point locations of animal populations to understand patterns in their movement.
- Change the color of features—learn how fatalities from a disease are increasing.
- Change the size or shape of features—understand population increase in a city or changes in parcel boundaries.
- Examine changes using raster catalogs or netCDF data—view ocean temperature change or weather patterns.
- Plot change over time in a graph—examine changes in ozone levels or water pressure at different stations.

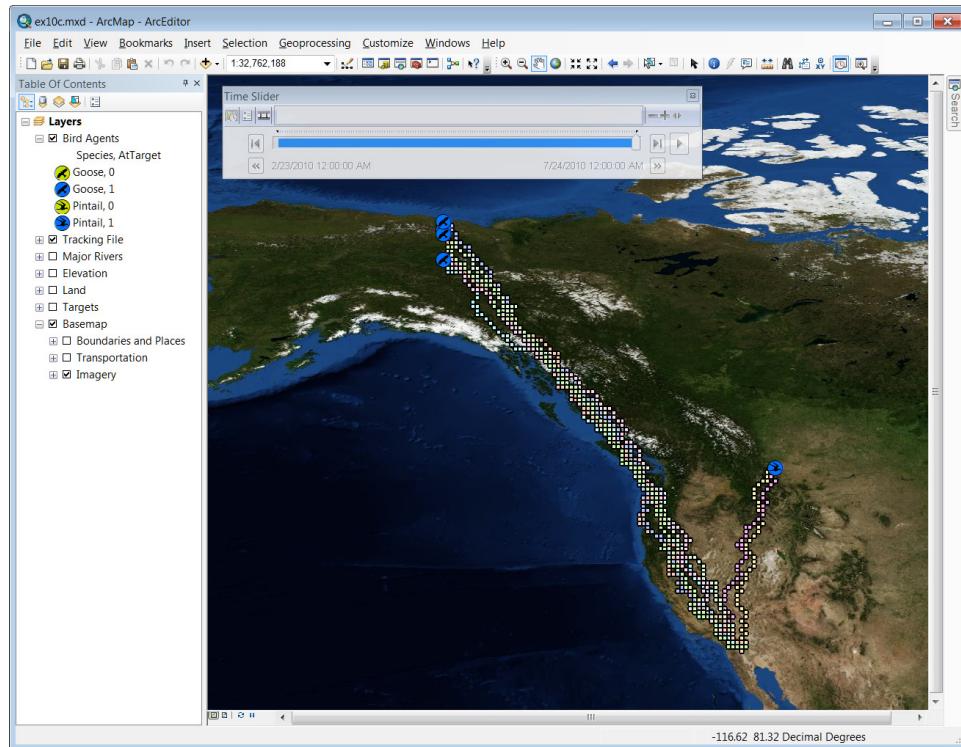
Exercise 10c

This exercise will walk you through a preconfigured animation file to expose you to some of the features and options available in the standard ArcGIS animation tools. The animation tools provide more features than will be discussed in this chapter, such as fading layers in and out and panning and zooming the map. To see how to start a new animation or learn more about the ArcGIS animation tools, along with some sample videos, please see the ArcGIS Desktop Help.

Load the ArcMap document

- 1 Start ArcMap. Navigate to C:\ESRIPress\AgentAnalyst\Chapter10. Open ex10c.mxd.

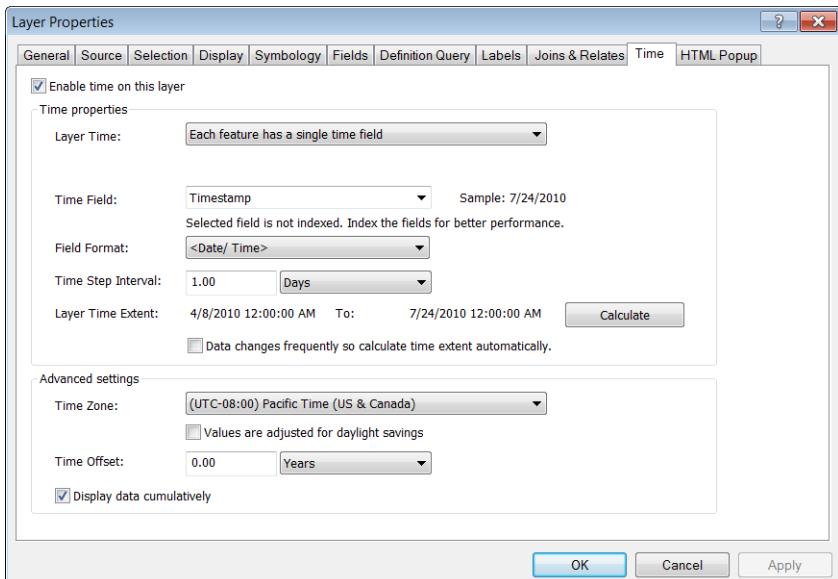
In the ArcMap display, you will see a tracking file created by running the bird migration model displayed on top of an imagery map service. (Internet access is required to view this layer.) The points are symbolized by the bird agent's band number.



Examine the time properties of this layer

- 2 Right-click the Bird Agents layer, and then click Properties... on the menu.
- 3 Click the Time tab to examine the configured properties.

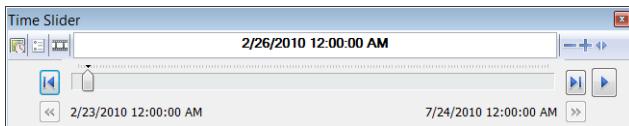
Note that “Enable time on this layer” is selected, that Time Field is set to Timestamp, and that Time Step Interval is set to 1 day.



4 Close the Layer Properties dialog box.

Display the time slider

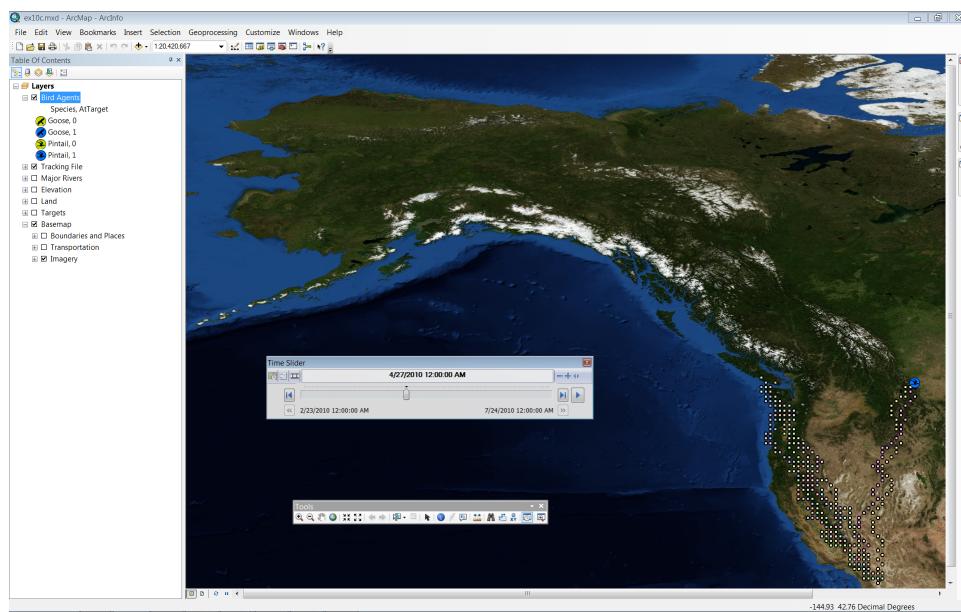
5 If the Time Slider window is not already open, on the Tools toolbar, click the Open Time Slider Window button ().



Run the time-enabled layer

6 If the Play button in the Time Slider window is enabled, click it to run the model. Otherwise, click the Enable Time On Map button (the leftmost button in the Time Slider window) and drag the slider back to the left. Then click the Play button. You will notice the track points start to change until the agents reach their target areas.

You can also drag the slider or click the Next or Back button to change the displayed point in time.



10

Display the Animation toolbar

- 7 Close the Time Slider window.
- 8 Click the Customize menu, point to Toolbars, and select Animation.

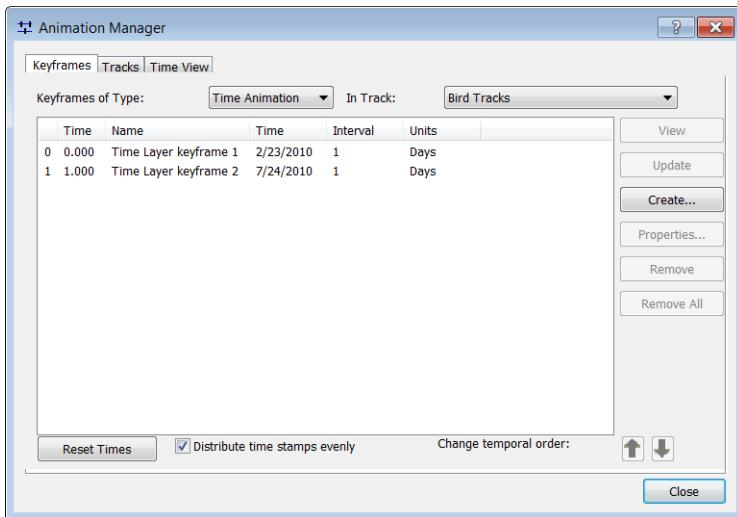
Examine the Animation Manager

- 9 From the Animation toolbar, open the Animation drop-down list and select Animation Manager....

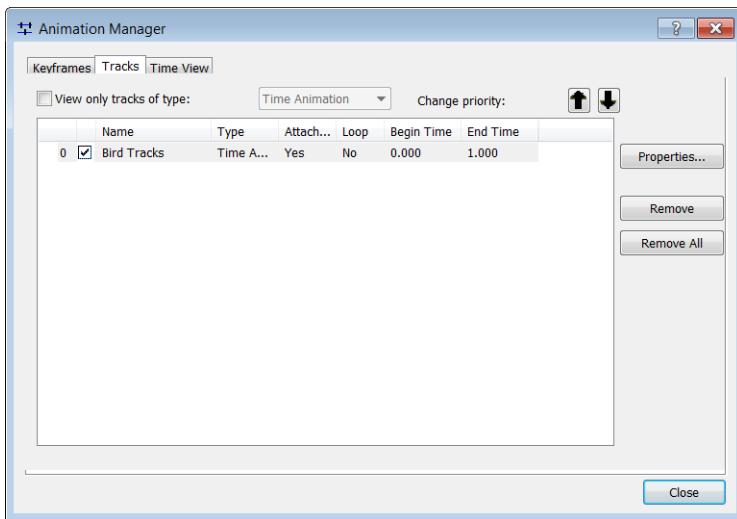
This will display the Animation Manager, which allows you to create, modify, and rearrange animation tracks.

- 10 Click the Keyframes tab (it should be selected by default).

Note the start and end dates in the Time column and that Units is set to Days.



- 11** Click the Tracks tab. This tab allows you to view and modify the properties of the available tracks.

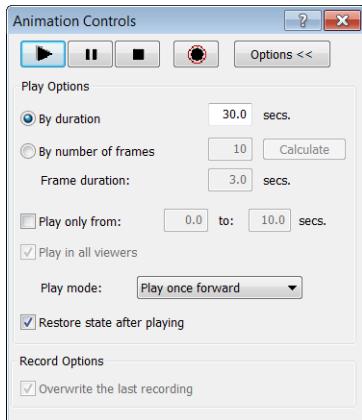


- 12** Close the Animation Manager.

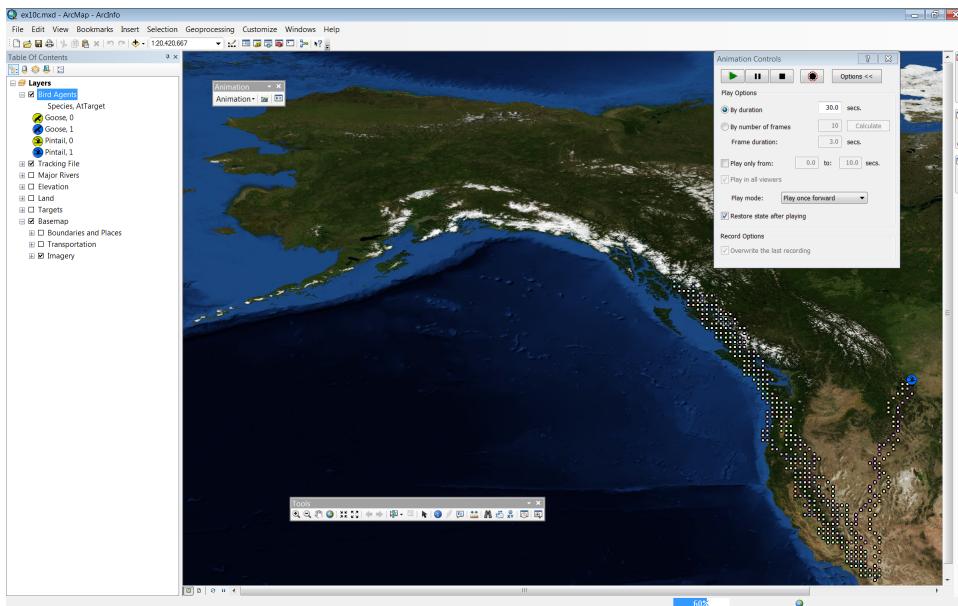
Play the animation

- 13** With the Animation toolbar on, click the Open Animation Controls button () to open the animation controls. This will display a standard play, pause, stop, and record toolbar.

- 14** Click the Options button to expand the Animation Controls dialog box. From here you can change the playback duration, pick a clip to play, and specify a few other basic settings.



- 15** Click the Play button to begin playback.



Export the animation as a video file

You will often want to create a video file of the animation output to share and communicate your model. Once you are satisfied with your model and your animation setup, you can easily export the animation to a shareable AVI or MOV video file.

- 16** To export the animation to a video, simply click the Animation drop-down list on the Animation toolbar and then click Export Animation.... A series of dialog boxes will guide you to save the video file.

For more information on exporting animations to video and the export settings available, please see the ArcGIS Desktop Help.

- 17** Close ArcMap without saving your changes.

Demonstrating the 3D ArcGlobe animation tools using the bird migration model

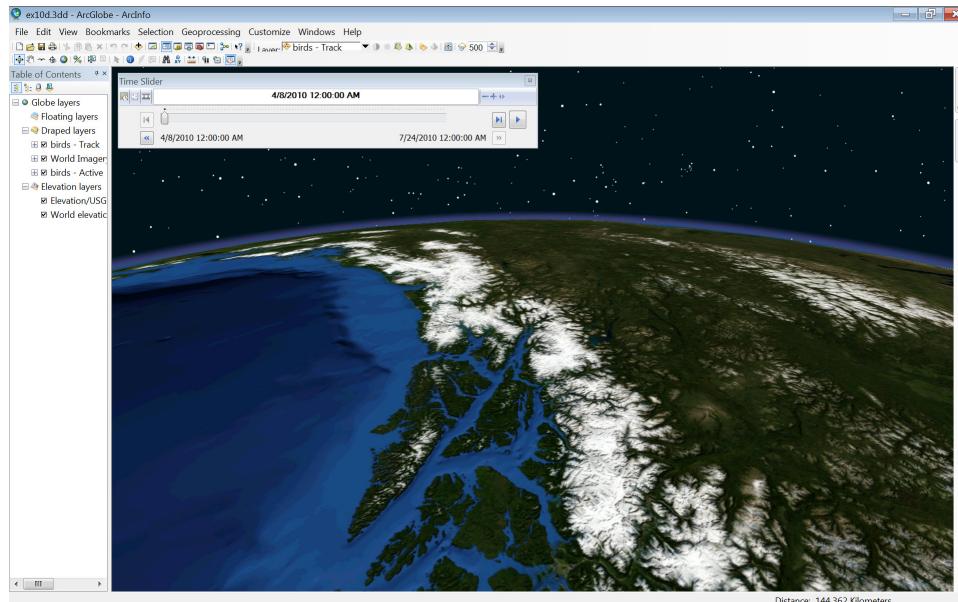
The steps for creating 3D animations in ArcGlobe are nearly identical to creating 2D animations in ArcMap. This exercise walks you through a preconfigured animation file to expose you to some of the features and options available in the standard ArcGIS 3D animation tools. This animation contains a bit more sophisticated animation than provided in exercise 10c, with some map movement during the point movement. This exercise doesn't provide a detailed description of the execution, but the ArcGlobe document is provided so that you can review how this was done. To learn more about creating animations with the ArcGIS animation tools, and to view some sample videos, please see the ArcGIS Desktop Help.

Exercise 10d

Load the ArcGlobe document

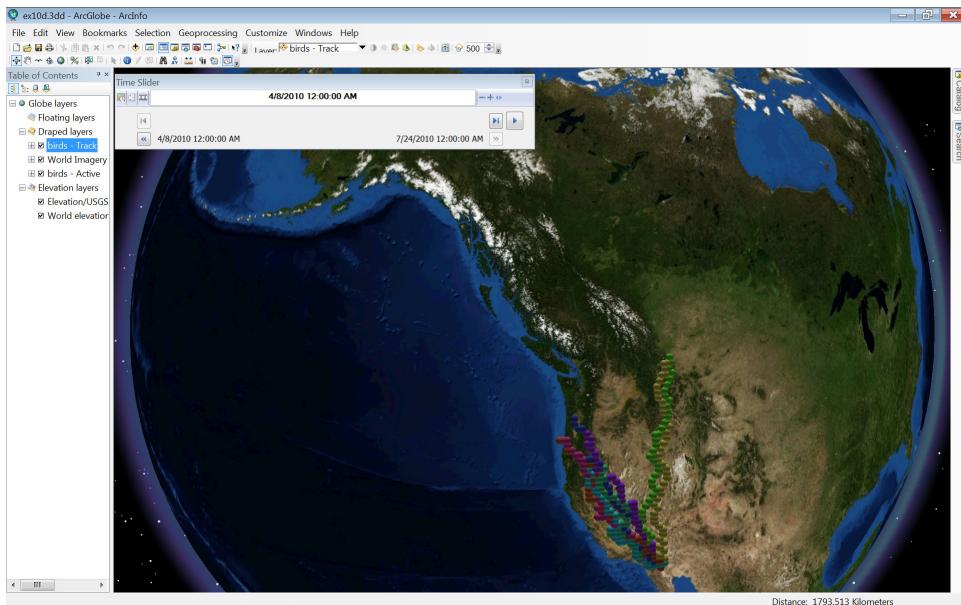
- 1 Start ArcGlobe. Navigate to C:\ESRIPress\AgentAnalyst\Chapter10. Open ex10d.3dd

In the ArcGlobe display, you will see an imagery map service zoomed into the northwestern portion of the United States and southwestern portion of Canada. (Internet access is required to view this layer and the 3D Analyst extension is necessary to run this exercise.)



As this may be your first time using ArcGlobe, pan around and zoom in and out of the image.

When you are done exploring the data, right-click on the birds-Track layer and select Zoom To Layer. Your extent should now show the curve of the planet, focused on the western coast of North America, with the agents' tracks. Each sphere color represents a different agent. Note that this is not the same track file used in exercises 10a–10c, so the agent positions will look different from what you have seen before.

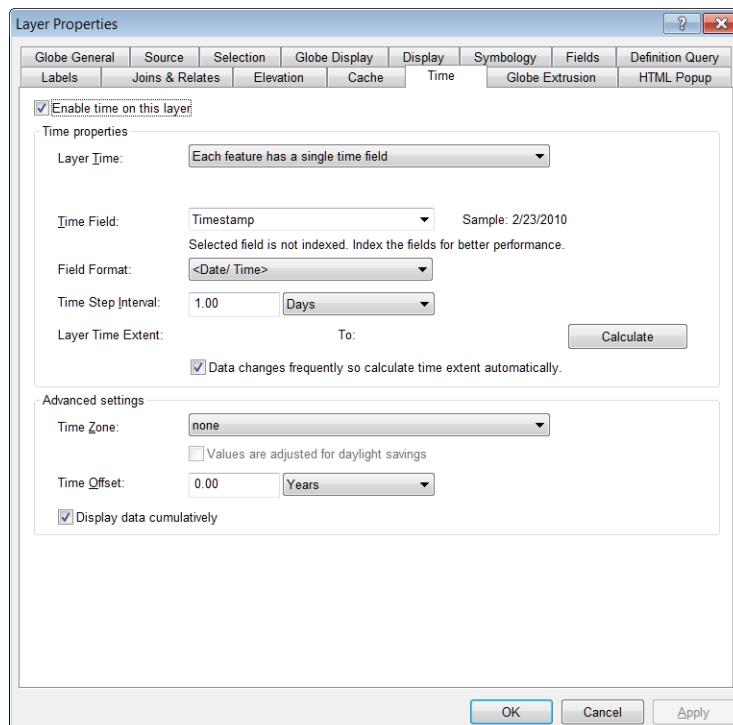


Exercise 10d

Examine the time properties of the birds-Track layer

- 2 Right-click the birds-Track layer, and then click Properties... on the menu.
- 3 Click the Time tab to examine the configured properties.

Note that “Enable time on this layer” is selected, that Time Field is set to Timestamp, and that Time Step Interval is set to 1 day. Also note that “Display data cumulatively” is selected. This option leaves previous points visible to give a breadcrumb effect.



10

- 4 Close the Layer Properties dialog box.

Display the Animation toolbar

- 5 Click the Customize menu, point to Toolbars, and select Animation.

Examine the Animation Manager

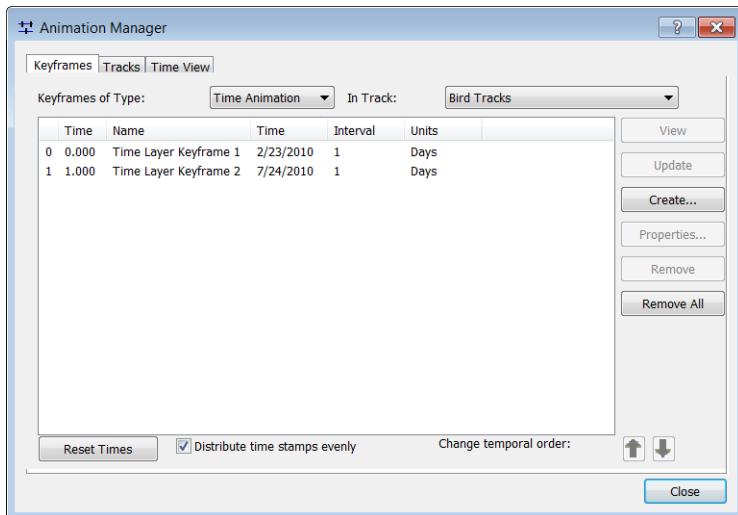
Note: If the Time Slider window is still open, close it.

- 6 From the Animation toolbar, open the Animation drop-down list and select Animation Manager.... The Animation Manager allows you to create, modify, and rearrange animation tracks.
- 7 Click the Keyframes tab (it should be selected by default).

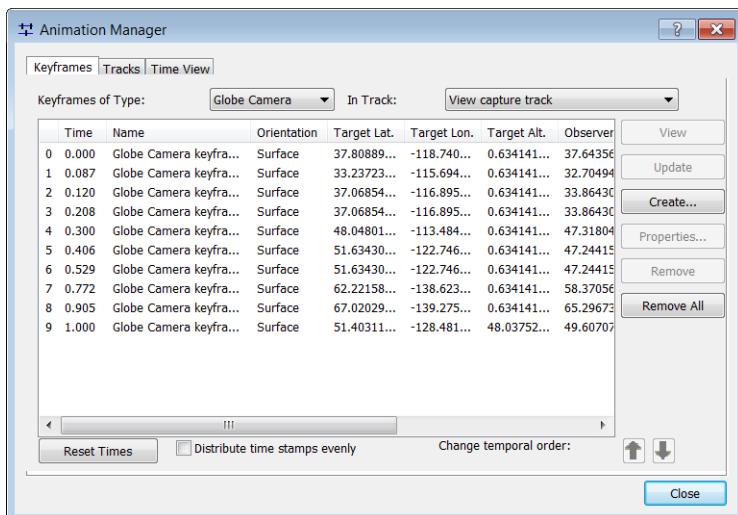
- 8** From the Keyframes of Type drop-down list, select Time Animation.

This will display the time-layer track for the symbols representing the current location of the birds.

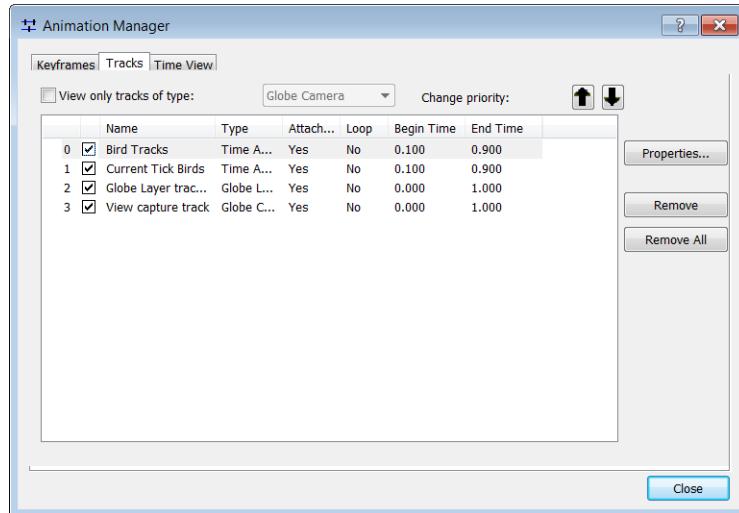
- 9** From the In Track drop-down list, select Bird Tracks, but Current Tick Birds is shown in the display. This track is used to animate the tracks of the birds.



- 10** From the Keyframes of Type list, select Globe Camera. This will display the keyframes that are being used for map movement (panning and zooming) during the animation, as shown in the following figure.



- 11 Click the Tracks tab. This tab lets you view and modify the properties of the available tracks.



10

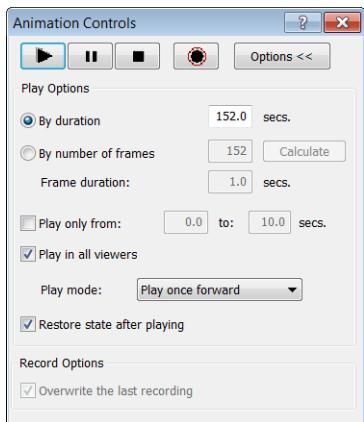
This animation uses two bird layers and two bird time-layer tracks to create an effect allowing you to display the current agents in a different color than the track so that there is no confusion among the points. The birds-Active layer is attached to the Current Tick Birds track and is set not to draw cumulatively, meaning that no points are left behind it. The birds-Track layer is attached to the Bird Tracks track and is set to draw cumulatively, meaning that the points are left behind.

- 12 Click Close to close the Animation Manager.

Play the animation

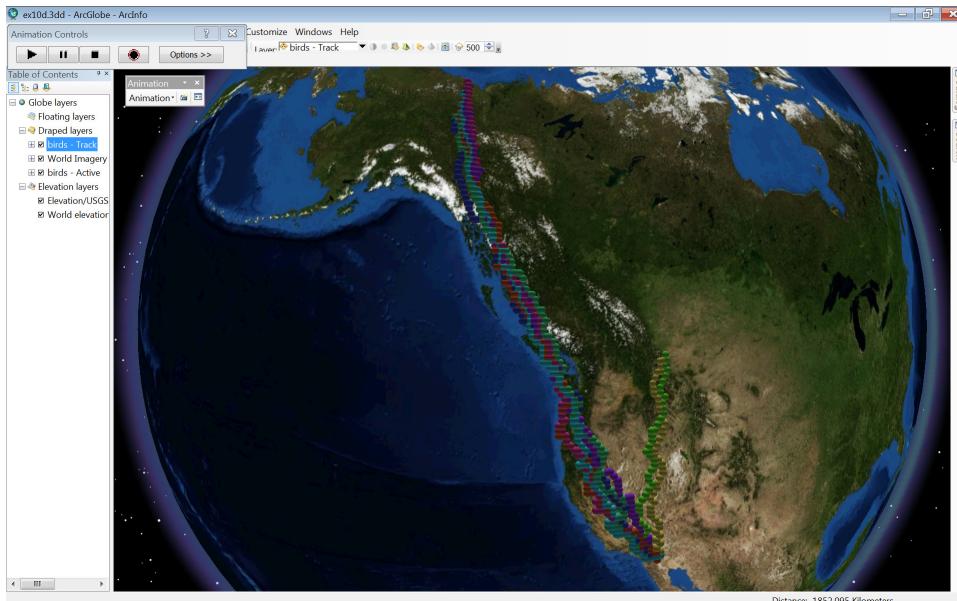
- 13 With the Animation toolbar on, click the Open Animation Controls button to open the animation controls. This displays a standard play, pause, stop, and record toolbar.

- 14 Click the Options button to expand the Animation Controls dialog box. From here you can change the playback duration, pick a clip to play, and specify a few other basic settings.



- 15 Click the Play button to begin playback.

The performance of the animation is highly dependent on your computer hardware, and depending on your Internet speed, the animation may be choppy.



Export the animation as a video file

Once you are satisfied with your model and your animation setup, you can easily export the animation to a shareable AVI or MOV video file.

- 16** Open the Animation drop-down list on the Animation toolbar and click Export Animation.... This will guide you through a series of dialog boxes to save the video file.

For more information on exporting animations to video and the export settings available, please see the ArcGIS Desktop Help.

- 17** Close ArcGlobe without saving your changes.

In this chapter you saw how GIS capabilities can accentuate an agent-based model. Since Agent Analyst is bidirectional, you are able to utilize the strengths of both the GIS and ABM capabilities in a single environment. The GIS capabilities that are of specific interest to the ABM environment include data preparation and creation, spatial modeling, analysis, and enhanced 2D and 3D display capabilities. With this synergistic environment, you are able to gain a deeper understanding of the phenomenon you are modeling.

GIS data, fields, and actions dictionaries

Table 10.1 Data dictionary of GIS datasets

| Dataset | Data type | Description |
|----------------|-----------|---|
| Birds | point | Fictionalized data representing bird banding locations around the Salton Sea. This shapefile is used as an agent data source. |
| Birds-postRun | point | Sample bird agent locations after a single model run |
| BirdTracking | point | Tracking file schema (empty) |
| distLand | raster | Surface representing the distance in meters from land |
| distRivers | raster | Surface representing the distance in meters from a major river |
| goosedist | raster | Surface representing the distance in meters from the goose target area (fictionalized) |
| pintaildist | raster | Surface representing the distance in meters from the pintail target area (fictionalized) |
| worlddem | raster | Surface representing continuous elevation |
| Countries | polygon | Country boundaries |
| NA_Rivers | line | Major rivers in North America |
| target_Goose | polygon | Fictionalized target for geese bird agents |
| target_Pintail | polygon | Fictionalized target for pintail bird agents |
| tracks-postRun | point | Sample tracking file after a single model run |

Table 10.2 Fields dictionary for the bird migration model

| Field | Data type | Description |
|-----------------------------|-----------|--|
| Bird migration model | | |
| dataPath | string | Path to data used in the model. Specifically used for finding raster datasets. |
| RiverDistanceWeight | double | Weight applied to river distance |
| ElevationCellWidth | double | Cell width of elevation raster |
| RiverDistanceCellWidth | double | Cell width of river distance raster |
| TargetDistanceCellWidth | double | Cell width of target distance raster |
| ElevationWeight | double | Weight applied to Elevation |

Table 10.2 Fields dictionary for the bird migration model (*continued*)

| Field | Data type | Description |
|------------------------|---|--|
| TargetDistanceWeight | double | Weight applied to Target Distance |
| LandDistanceWeight | double | Weight applied to Land Distance |
| randomness | integer | Randomness factor |
| NumGroupsReachedTarget | integer | Number of agents (bird groups) that have reached the destination |
| Debug | integer | Debug level (1 or 0) |
| agentFile | string | File name of the bird agents |
| MaxRiverDistance | double | Maximum distance to use river distance as a factor |
| tracksFileName | string | File name of tracking file |
| tracksFC | com.esri.arcgis.geodatabase.IFeatureClass | ArcObjects reference to the tracking shapefile |
| tickHours | integer | Number of hours represented by a single tick |
| AgentIDField | string | Unique idea of each agent |
| Bird agent | | |
| maxElevation | integer | The maximum elevation to consider elevation as a factor |
| maxRiverDistance | integer | The maximum distance to consider river distance as a factor |
| Cell agent | | |
| X | integer | The cell location (column) in the raster |
| Y | integer | The cell location (row) in the raster |
| mapX | double | The longitude of the origin of the cell |
| mapY | double | The latitude of the origin of the cell |
| Marbles | double | The marble count of the cell |
| probMin | double | The minimum probability value of the cell |
| probMax | double | The maximum probability value of the cell |

Table 10.3 Actions dictionary for the bird migration model

| Declared actions | Description |
|-----------------------------|---|
| Bird migration model | |
| initAgents | Initializes agent fields |
| updateDisplay | Refreshes ArcMap |
| writeAgents | Writes the agent points to the agent file |
| setRasters | Sets all raster variables |
| WriteIteration | Writes a simple status line between each tick of the model |
| setRasterWidths | Sets the raster cell sizes to force the agent points to the middle of the cells rather than the origin (corner) |
| setTrackFC | Sets an ArcObjects feature class variable to write to allow the model to write to the tracking file |
| Bird | |
| step | Gets the geometry of the agent and begins the move calculations |
| move | Moves the agent point to the defined coordinates |
| init | Initializes the agent points by moving them back to their starting locations and sets the start time to the computer's time (for multiple model runs) |
| getSurroundingCells | Gets the neighboring cells of a given cell |
| CalculateMove | Calculates where to move the agent |
| getCells | Gets the neighboring cells in the correct direction (the bird agents will not turn around and fly in the other direction) |
| setMarblesForRiverDistance | Gets a probability value for the river distance by evaluating surrounding cells |
| setMarblesForElevation | Gets a probability value for elevation by evaluating surrounding cells |
| setMarblesForTargetDistance | Gets a probability value for the target distance by evaluating surrounding cells |
| setMarblesForLandDistance | Gets a probability value for the distance from land by evaluating surrounding cells |
| getIndexForSortedList | Calculates the index for adding a new item to the sortedCellList |
| addPoints | Adds a new point to the tracking file. This includes the agent's ID, new coordinates, and time stamp. |
| Cell | |
| step | Empty |

Section 5: Advanced techniques for points, polygons, and rasters

Chapter 11

Patterns of movement in agent-based modeling

by Hamid R. Ekbia

- ◆ Performing simple movements for a point agent
- ◆ Moving a point agent toward a stationary point agent
- ◆ Moving a point agent toward a moving point target
- ◆ Having a point agent evade a moving point agent target
- ◆ Moving a polygon agent toward a fixed polygon agent
- ◆ Having a polygon agent follow a moving polygon agent
- ◆ Having a polygon agent evade a moving polygon agent
- ◆ Moving a point agent relative to a raster surface

In the previous chapters, you saw examples of different types of movement and how they are modeled in Agent Analyst—from the movement of cougars in the wild to the migration of people in developed environments, and from the rezoning of land parcels to the spread of crime in urban areas. As disparate as these movements might seem, there are basic patterns and similarities among them. This chapter is intended to outline these patterns, organize them according to types of agents, and provide examples of code that can be used for modeling similar types of movement. The chapter also describes scenarios to illustrate these ideas. As a result, the chapter synthesizes what you have learned, generalizes it, and provides code snippets that you can readily use in new models that you might be creating.

Background information

Things change in various ways—in shape, location, size, orientation, color, age, and numerous other ways. Some of these are changes in the properties and attributes of objects, and other changes are in the *spatial* relationship of one object to other objects. Implicit in both cases is a notion of time—any change happens in time. Despite the great variation, however, there are certain patterns that repeat themselves. In this chapter, we will treat spatial change separately, treating it as movement, and we will use the term “change” for all other kinds of transformation. Our focus will be on movement.

Objects also move in all sorts of ways—sometimes they move randomly without any destination, other times they move toward a destination or follow another object. Sometimes the destination can be treated as a point feature (an object without dimensions), other times it is a polygon feature (an enclosure such as a lake or a building), and so on. Some of these movements are more basic than others, in the sense that we can understand them as “primitives” out of which more complex movements can be created (in the same sense that molecules can be created from atoms). Rather than having just one, however, we might need a few primitives. In short, this is going to be our approach in this chapter: we begin with very simple, primitive movements of point agents, and make them more and more complex as we go along. Metaphorically, the picture that we have in mind is one of a primitive agent that is learning how to move around in its surrounding environment.

This chapter will model movement in a general, application-independent manner. The same type of movement is modeled differently in different applications. General movement is described generically but differently depending on whether point or polygon agents are involved or whether the environment is a vector or raster.

Exercises

The following exercises explore the basic principles of movement of point and polygon agents over vector and raster surfaces. The point and polygon agents used in these exercises are generic agents with no real-world correlation. The actions described can be applied to any point or polygon agent. The exercises are shorter than exercises in previous chapters and are designed to demonstrate a specific movement principle in a nonspecific application domain. Our use of specific agents (e.g., cougars or census tracks) in this chapter is just illustrative; the purpose is to model a particular movement in a generic, domain-independent manner.

All the actions are already created for you so that you can uncomment the code and view it in the models that you are creating when applicable.

Point agents

The first four exercises will look at the movements of point agents in various ways. You will start with simple movements, such as moving the agent in different directions. You'll then have them change their direction and then combine these basic movements to perform more complex movements, such as turning around. You will then see how to move the point agents toward stationary and moving targets and evade other point agents.

Performing simple movements for a point agent

This exercise illustrates a series of basic movements by a point agent. The most basic of all movements are moving forward and moving in different directions. You will then see how to combine these basic primitives to perform more sophisticated movements.

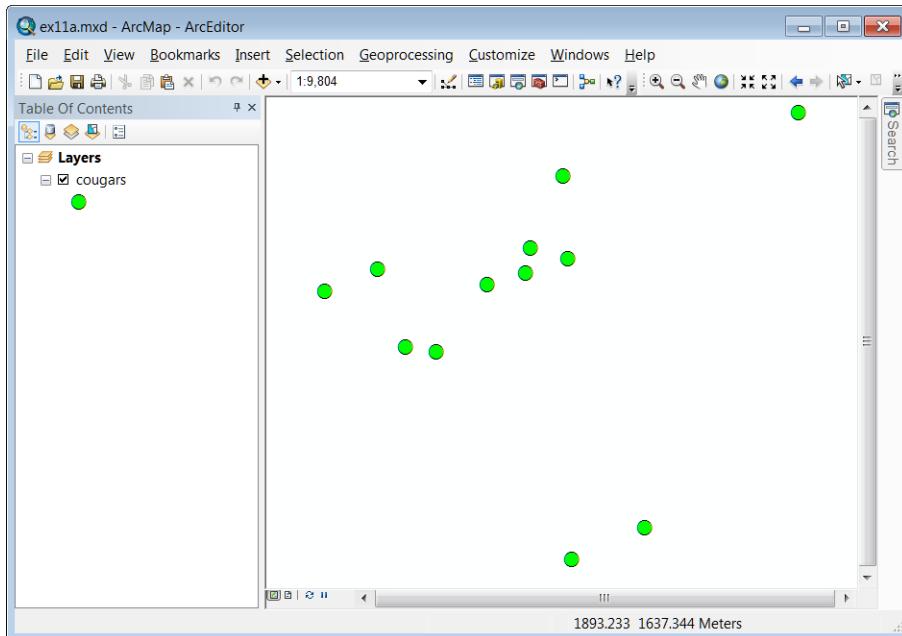
Exercise 11a

You will begin with the simplest type of agent—namely, a point agent that wants to freely move in different directions.

To perform different types of movements, the agent must be able to perform some basic movements—for example, moving north or moving south. Based on these basic movements, the agent can perform other complex movements.

Load the ArcMap document

- 1 Start ArcMap. Navigate to C:\ESRIPress\AgentAnalyst\Chapter11. Open ex11a.mxd. If you do not see any cougar agents in the display, right-click on the cougars layer in the Table Of Contents and click Zoom To Layer.



When the map document opens, in the ArcMap display you will see a series of points representing the locations of your agents (e.g., cougars). In ArcToolbox, the Chapter11Ex11a toolbox has been created for you.

Load the Agent Analyst model

- 2 Right-click the Chapter11Ex11a toolbox, point to New, and select Agent Analyst Tool.
- 3 Once the Agent Analyst window appears, click File and select Import. Do not save the changes when prompted.
- 4 Navigate to C:\ESRIPress\AgentAnalyst\Chapter11\Models and open ex11a.sbp.

Add and call the moveToNorth action

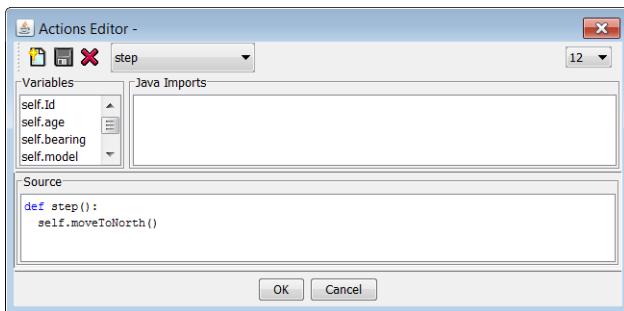
- 5 In the Environment panel, click Cougar. In the Property panel, click the Edit button to the right of the Actions property to open the Actions Editor.
- 6 In the Actions Editor window, select moveToNorth from the Actions drop-down list and uncomment the code shown in the following figure.



In this code, you first get the location of the cougar (`coordinate = self.the_geom.coordinate`), and then move the agent to the north (`coordinate.y = coordinate.y + 1`). By adding one to the y coordinate of the cougar agent, the cougar agent will move one map unit to the north. Any distance can be specified. For example, as you have seen in previous chapters, the cell size has been used to move the agent into the neighboring cell.

Now you want your agent to start moving to the north. To do this, you need to add the step action.

- 7 In the Actions Editor, select step from the Actions drop-down list and uncomment the code.

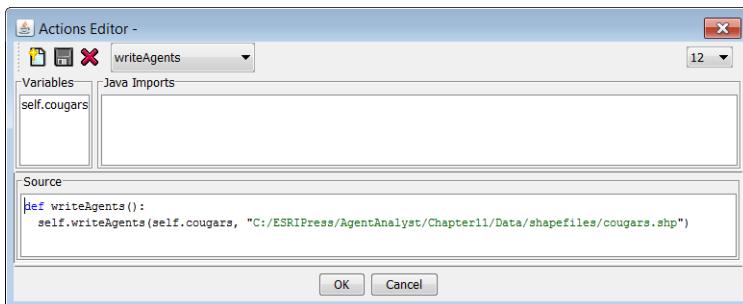


Here the code calls the `moveToNorth` action that makes the agents move to the north (`self.moveToNorth`).

So far, a number of point agents (cougars) are created based on the input information from the cougars shapefile. You have created one action, `moveToNorth`, to move each cougar to the north, and you call that action each time step from the `step` action. To see the cougar agents move in ArcMap you need to create a few more actions. After the cougars move, you need to first write the new locations of the cougars back to the `cougars.shp` shapefile.

Update and display the cougar location

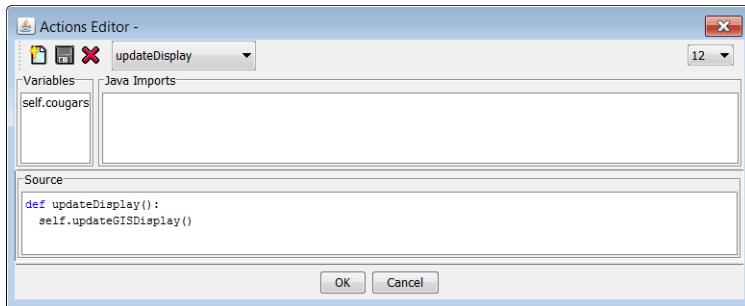
- 8 In the Environment panel, click Simple Model. In the Property panel, click the Edit button to the right of the Actions property to open the Actions Editor.
- 9 In the Actions Editor, select `writeAgents` from the Actions drop-down list and uncomment the code as shown in the following figure.



This code writes the agents' locations into the shapefile (`self.writeAgents(self.cougars, "C:/ESRIPress/AgentAnalyst/Chapter11/Data/shapefiles/cougars.shp")`). ArcMap will later use the updated information to display the agents.

To update the display, the `updateDisplay` action is needed to reflect the latest changes in the agents' locations.

- 10 In the Actions Editor, select `updateDisplay` from the Actions drop-down list and uncomment the code as shown in the following figure.



This code calls the `updateGISDisplay` action to update the display (`self.updateGISDisplay`).

To see the agents' movements, schedule `writeAgents` and `updateDisplay` to update every tick.

- 11 Click OK to close the Actions Editor and save your changes.
- 12 In the Environment panel, click Simple Model. In the Property panel, click the Edit button to the right of the Schedule property to open the Schedule Editor. First schedule `writeAgents` and then schedule `updateDisplay` so that both are performed every tick.
- 13 Click OK to save your edits and close the Schedule Editor.

11

Updating a GIS dataset with the model results

All the models in this book update point, polygon, or raster ArcGIS datasets and then those changed datasets are viewed in ArcMap. Some forethought must be given to how you want this updating to occur. In several of the chapters (such as this one), the model updates the input base dataset, overwriting the starting conditions. In the cougar model in chapters 5 and 8, the starting positions of the cougar agents are reset before each model run. In chapter 4, a copy of the base input is created before the model run and with each specified time step the model creates and writes to a uniquely named raster (based on the time step). In chapter 7, a copy of the base input is created with a different name and the output is written to this dataset. Each of these approaches has strengths and weaknesses, and it is up to you and your application to determine which approach is the best to implement.

No matter which approach you take, a copy of your original dataset should be made so that it can be used in subsequent model runs, during model development, or in case something happens in a model run that corrupts the original data.

Make a copy of the cougars shapefile

Exercises 11a, 11b, 11c, 11d, and 11h of this chapter all write to the same input shapefile. The altered shapefile is then used as the starting conditions for the next model run (or next exercise). If you do not run the model for many time steps, you will see little difference in the initial starting points for each exercise. However, if you let a model run for some time, the model can greatly change the base shapefile that will be used in the next exercise. Depending on the model run, the agents may not even be in the extent as specified in subsequent ArcMap (.mxd) documents. It is common practice in agent-based modeling to make copies of your base data and, in some cases, with each model run, reinstate the copy and use it as the input into the next model run to preserve the original starting locations and states of your agents.

Therefore, before running the model, you will learn how to make a copy of the base input data. If a model run greatly affects the input shapefile, you can replace it with the copy to reinstate the starting conditions before running subsequent exercises.

- 14** Using Catalog, navigate to and expand C:\ESRIPress\AgentAnalyst\Chapter11\Data\shapefiles, right-click cougars, and then click Copy. Then right-click shapefiles in ArcCatalog and select Paste. A copy of the cougars.shp shapefile is created in the same location, with the name cougarsCopy.shp.

You will use this copy of the shapefile in subsequent exercises to reinstate the starting conditions.

Observe the results of the moveToNorth action

- 15** Run the model and observe the moveToNorth action, then stop the run and continue with the exercise steps.

To move the cougar agent in other directions, to the east, south, and west, the moveToEast, moveToSouth, and moveToWest actions have been created.

To make your cougar agents behave more realistically, you can augment them to have a “bearing” property. Then you use the agent’s bearing to define new kinds of move actions.

Define the bearing field

- 16** In the Environment panel, click Cougar. In the Property panel, click the Edit button to the right of the Fields property to open the Fields Editor.
- 17** Define a new integer field named bearing. Set the default value of this field to 0. Here, you use a simple convention for representing the bearing of the agent: 0 for

north, 90 for east, 180 for south, and 270 for west. Therefore, your agents are heading north by default. Define this field as an accessible field.

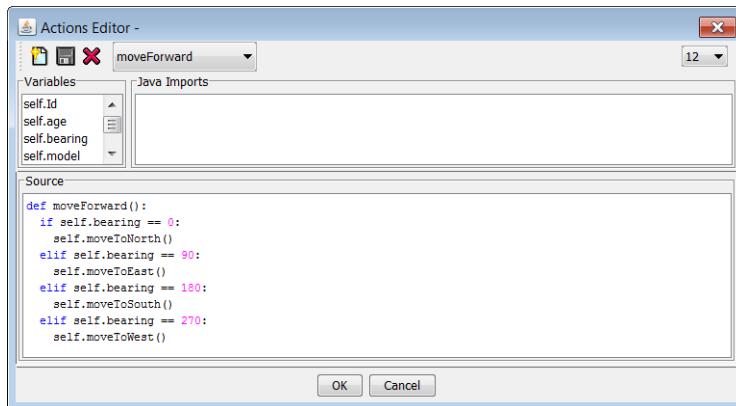
- 18** Click OK to close the Fields Editor and save your changes.

Using this simple field, the agent is able to perform simple movements.

For instance, for an agent to move forward, it must check its own bearing, and based on this value, it must change its position—that is, increase or decrease its x or y value. The moveForward action is already implemented (see the figure below).

Change the bearing of the agent

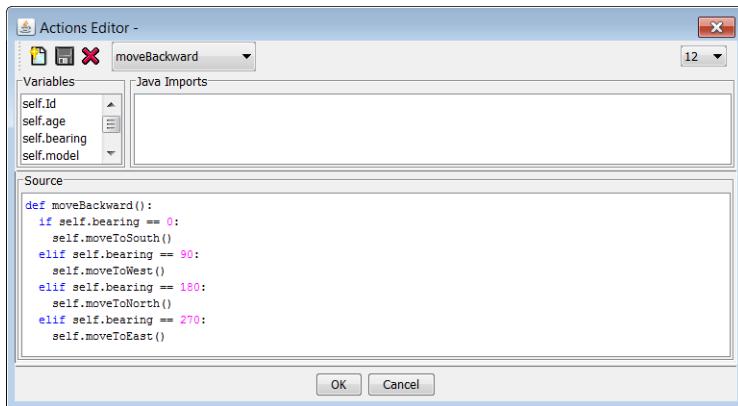
- 19** In the Environment panel, click Cougar. In the Property panel, click the Edit button to the right of the Actions property and click Edit to open the Actions Editor.
- 20** In the Actions Editor, select moveForward from the Actions drop-down list and uncomment the code as shown in the following figure.



In the moveForward action, if the agent's bearing is equal to 0 (`if self.bearing == 0:`), it means that the agent is facing north, so to move forward it needs to move north (`self.moveToNorth()`). Similarly, if the bearing is equal to 90 (`if self.bearing == 90`), the agent is facing east, and it needs to move east, and so on.

Moving backward is similar to moving forward.

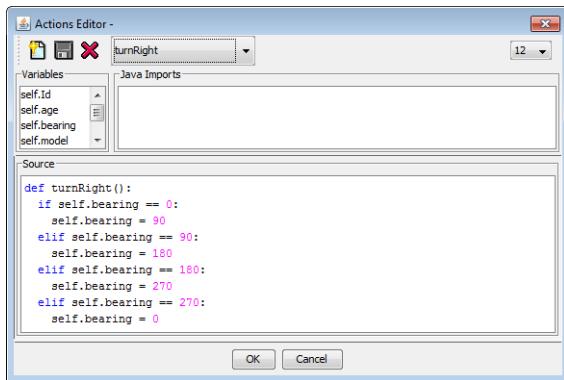
- 21** In the Actions Editor, select moveBackward from the Actions drop-down list and uncomment the code as shown in the following figure.



The agent changes its position based on its own bearing. One point worth mentioning is that by moving backward, we mean the agent moves one step farther in the opposite direction to its current bearing, without changing the bearing—in other words, the agent doesn't turn around.

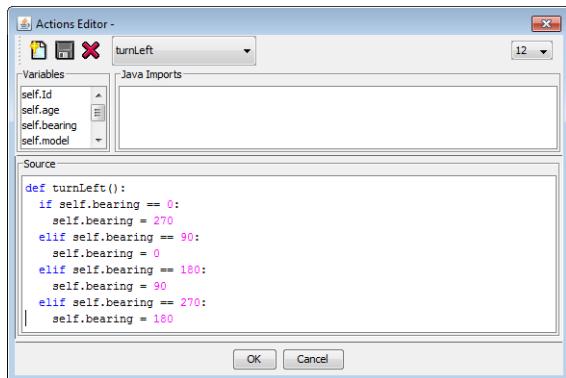
The actions turnRight and turnLeft (already available in the model) allow your agent to rotate and turn around by changing its bearing.

- 22** In the Actions Editor window, select the turnRight and then the turnLeft actions from the Actions drop-down list and uncomment the code so that it matches the code in the following figures.



The preceding code shows the implementation of the turnRight action. Based on the current value of the bearing field of the agent, a new value is set to make the turn happen. For instance, if the bearing is equal to 0 (`if self.bearing == 0:`), then to turn right it must be set to 90 (`self.bearing = 90`) and so on.

The turnLeft action is similar to the turnRight action.

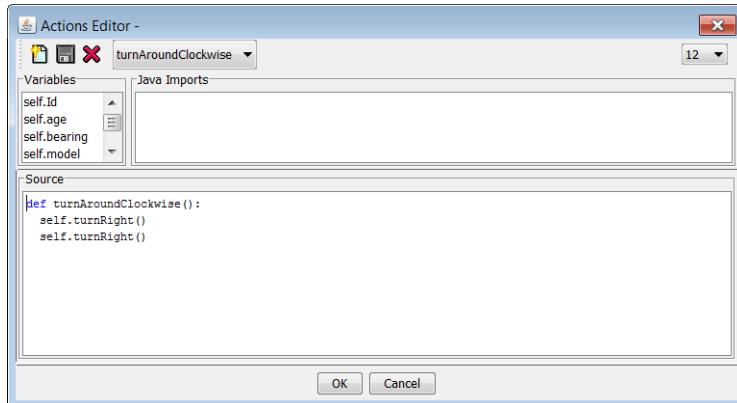


Here, the turnLeft action follows the same logic. For example, if the bearing is equal to 0 (if `self.bearing == 0`), then to turn left it needs to be changed to 270 (`self.bearing = 270`).

Next, you can combine these actions to make more complicated moves. For instance, the agent can turn around by making two consecutive right turns (clockwise) or two consecutive left turns (counterclockwise). The turnAroundClockwise and turnAroundCounterClockwise actions are already implemented for you.

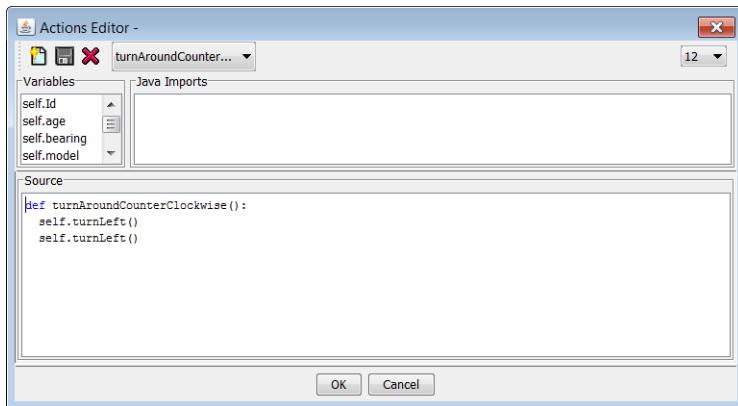
11

- 23** In the Actions Editor, select the turnAroundClockwise action and then the turnAroundCounterClockwise action from the Actions drop-down list and uncomment the code so that it matches the code in the following figures.



To turn around clockwise (by 180 degrees), the agent needs to perform two consecutive turnRight actions (`self.turnRight`).

The turnAroundCounterClockwise follows the same logic.



Here, to turn around counterclockwise (by 180 degrees), an agent needs to perform two consecutive turnLeft actions (`self.turnLeft()`).

- 24** Click OK to close the Actions Editor.
- 25** Close Agent Analyst and ArcMap without saving your changes.

As you'll see in the following exercises, in addition to these simple movements you want your agents to be able to directly move toward a target in a specific location. The target can be a point, in which case the agent moves from point to point, or it can be a vector (a boundary line, for instance).

Moving a point agent toward a stationary point agent

In this exercise, you will modify the model you developed for the previous exercise to illustrate the agent's movement toward a stationary or fixed point target.

Exercise 11b

In this exercise your agents have a field named Id. You will assign a unique identifier to each cougar agent and make the agents move toward the first agent (the agent with Id equal to 0), which is not moving.

Load the ArcMap document

- 1 Start ArcMap. Navigate to C:\ESRIPress\AgentAnalyst\Chapter11. Open ex11b.mxd.

In the ArcMap display you will see a series of points representing the locations of cougars. In ArcToolbox, the Chapter11Ex11b toolbox has been created for you. If you do not see any cougar agents in the display, right-click on the cougars layer in the Table Of Contents and click Zoom To Layer.

11

Reset the cougars shapefile

If you let the model run for some time in the preceding exercise (remember this exercise uses the cougars shapefile that you wrote to in the previous exercise as its input) you can replace the cougars shapefile with the copy you made in step 14 of exercise 11a. If you did not make a copy of the cougars shapefile in exercise 11a, you might want to remove and reinstall the chapter 11 data files.

- 2 To use the copy of the cougars shapefile you created in step 14 of exercise 11a before running the model again, use Catalog to navigate to and expand C:\ESRIPress\AgentAnalyst\Chapter11\Data\shapefiles.

You should see two shapefiles, cougars and cougarsCopy.

- 3 Right-click cougarsCopy and select Copy. Right-click the shapefiles directory and select Paste.

A new copy of cougarsCopy.shp is created as cougarsCopy2.shp.

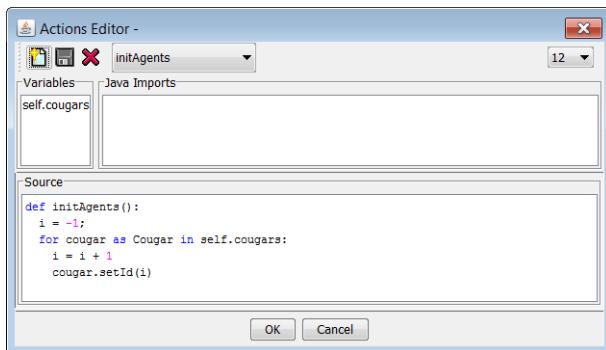
- 4 Right-click cougars and click Delete. Right-click cougarsCopy2 and select Rename. Rename cougarsCopy2 to cougars and press Enter. You might also need to add cougars.shp to ArcMap while an ArcMap exercise document is open by using the Add Data button.

Load the Agent Analyst model

- 5 Right-click the Chapter11Ex11b toolbox, point to New, and select Agent Analyst Tool.
- 6 Once the Agent Analyst window appears, click File and select Import. Do not save the changes when prompted.
- 7 Navigate to C:\ESRI\Press\AgentAnalyst\Chapter11\Models and open ex11b.sbp.

Add the initAgents action to assign a unique ID to each agent

- 8 In the Environment panel, click Simple Model. In the Property panel, click the Edit button to the right of the Actions property to open the Actions Editor.
- 9 In the Actions Editor, select initAgents from the Actions drop-down list and uncomment the code shown in the following figure.

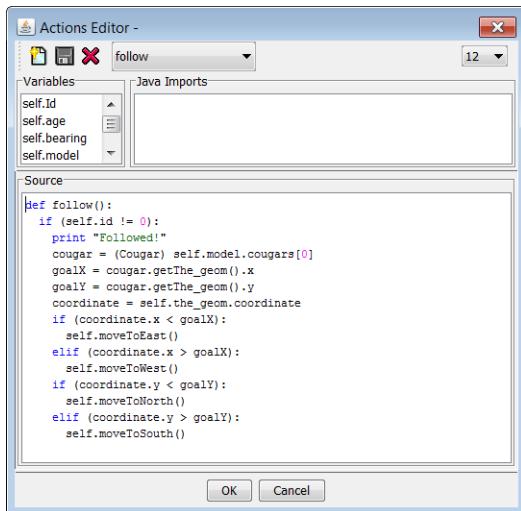


In this code, integers (stored in the variable *i*) get assigned to agents as their IDs. The first agent will have 0 as its ID, and the integers 1, 2, 3, and so on will be assigned to the rest of the agents (*cougar.setId(i)*).

To give the agents the opportunity to move toward a stationary cougar agent, you will define a new follow action for the cougar agent. In this action, you will first have the cougar agents move toward a stationary cougar agent, and in the next exercise the stationary cougar agent will also be moving, hence the name of the action, *follow*.

- 10 In the Environment panel, click Cougar. In the Property panel, click the Edit button to the right of the Actions property to open the Actions Editor.

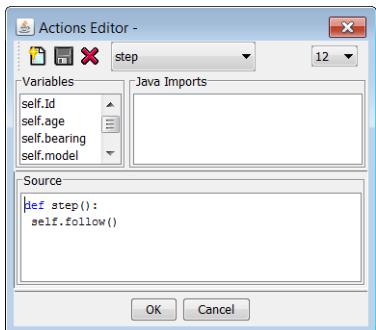
- 11** In the Actions Editor, select follow from the Actions drop-down list and uncomment the code as shown in the following figure.



In this action, an agent first checks its own ID. If its ID is equal to 0, it does nothing (`if (self.id != 0) :`). Otherwise, it starts moving toward the first target agent. To do so, an agent consults the model to know where the first target agent is (`cougar = (Cougar) self.model.cougars[0]`) and stores the first agent's coordinates in the variables `goalX` and `goalY` (`goalX = cougar.getThe_geom().x`) and (`goalY = cougar.getThe_geom().y`). The coordinates of the agent itself are also stored in the variable `coordinate` (`coordinate = self.the_geom.coordinate`). Then, based on a simple comparison of its own position with the target position, the agent performs a simple move: if the agent's x coordinate is less than the first target agent's x coordinate, then it needs to move east (`if (coordinate.x < goalX) : self.moveToEast()`); otherwise it moves west (`self.moveToWest()`). Similarly, if the agent's y coordinate is less than the first target agent's y coordinate, the agent needs to move north (`if (coordinate.y < goalY) : self.moveToNorth()`); otherwise it moves south (`self.moveToSouth()`). Therefore, all the agents, except for the first one, move toward the first agent until they reach it.

The step action needs to be changed to use the newly defined follow action.

- 12** In the Actions Editor, select step from the Actions drop-down list and uncomment the code as shown in the following figure.

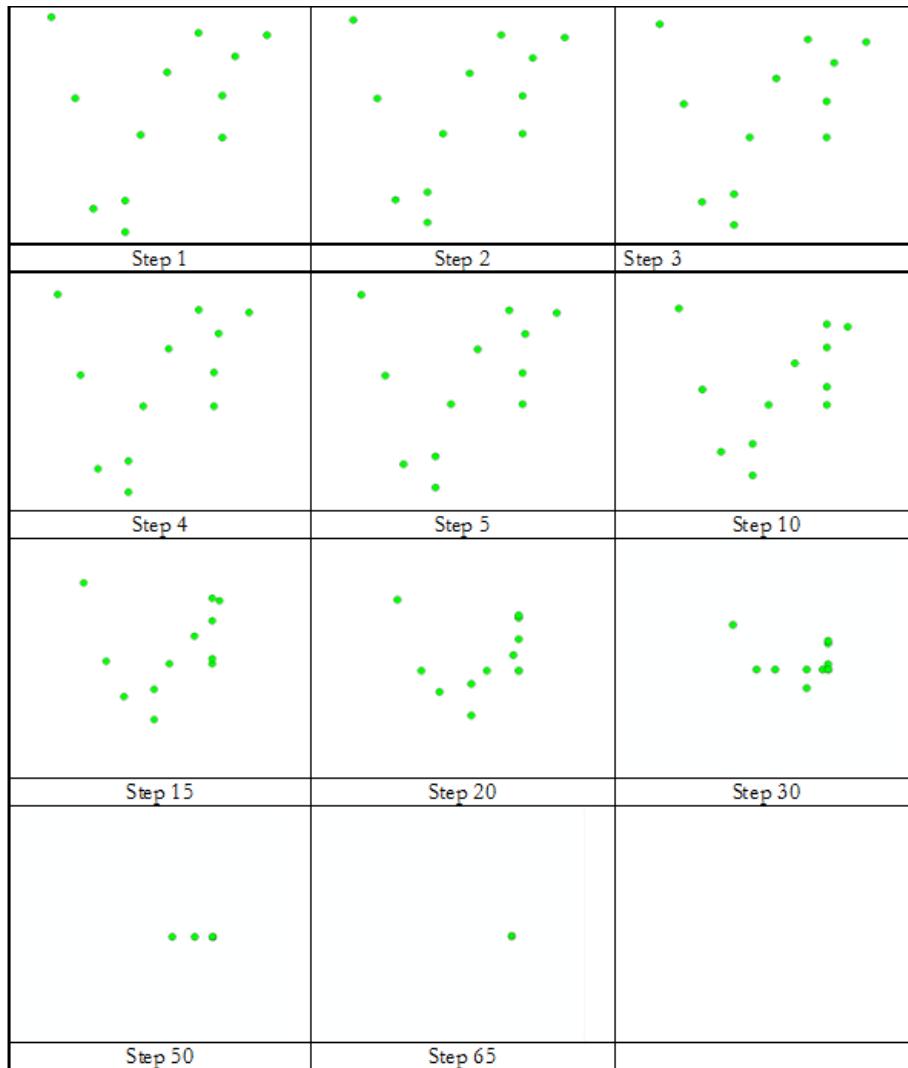


The code here calls the follow action of the agent (`self.follow()`).

- 13** Click OK to save your edits and close the Actions Editor.
14 Run the model.

Note: You may want to highlight the target agent so that you can see where the other agents are moving toward. To do so, right-click the cougars layer and click Open Attribute Table. Select the first row (Id = 0) by clicking in the gray box to the far left of the row (probably with an arrow sign). The row will turn light blue. Close the attribute table, and you'll see that the target agent is now a different color than the other agents.

The following image illustrates the positions of the agents moving toward a fixed point agent in different time steps. The starting locations of the cougar agents are different from the starting locations in your model, demonstrating that similar patterns will result regardless of the beginning locations. It will probably take your agents much longer to reach the target.



11

- 15 Close Agent Analyst and ArcMap without saving your changes.

Moving a point agent toward a moving point target

In this exercise, you will model point agents moving toward a target point that is not fixed. In other words, the target point is a moving point agent. You will modify the previous model so that the first agent is also moving. In each step, the first agent, which is the target agent, performs a random movement. All other agents follow the first agent.

Exercise 11c

Load the ArcMap document

- 1 Start ArcMap. Navigate to C:\ESRIPress\AgentAnalyst\Chapter11. Open ex11c.mxd.

In the ArcMap display you will see a series of points representing the locations of cougars. If you do not see any cougar agents in the display, right-click on the cougars layer in the Table Of Contents and click Zoom To Layer.

Reset the cougars shapefile

If you let the model run for some time in the preceding exercise (remember this exercise uses the cougars shapefile that you wrote to in the previous exercise as its input), you can replace the cougars shapefile with the copy you made in step 14 of exercise 11a. If you did not make a copy of the cougars shapefile in exercise 11a, you might want to remove and reinstall the chapter 11 data files.

- 2 To use the copy of the cougars shapefile you created in step 14 of exercise 11a before running the model again, use Catalog to navigate to and expand C:\ESRIPress\AgentAnalyst\Chapter11\Data\shapefiles.

You should see two shapefiles, cougars and cougarsCopy.

- 3 Right-click cougarsCopy and select Copy. Right-click the shapefiles directory and select Paste.

A new copy of cougarsCopy.shp is created as cougarsCopy2.shp.

- 4 Right-click cougars and click Delete. Right-click cougarsCopy2 and select Rename. Rename cougarsCopy2 to cougars and press Enter. You might also need to add cougars.shp to ArcMap while an ArcMap exercise document is open by using the Add Data button.

In ArcToolbox, the Chapter11Ex11c toolbox has been created for you.

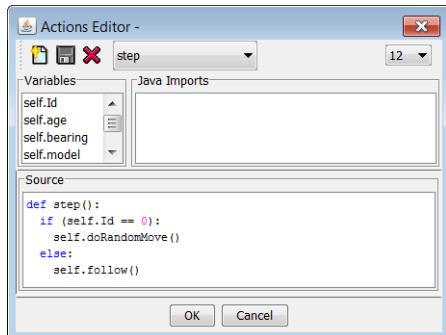
Load the Agent Analyst model

- 5 Right-click the Chapter11Ex11c toolbox, point to New, and select Agent Analyst Tool.
- 6 Once the AgentAnalyst window appears, click File and select Import. Do not save the changes when prompted.
- 7 Navigate to C:\ESRIPress\AgentAnalyst\Chapter11\Models and open ex11c.sbp.

As mentioned in Exercise 11b, here the first agent randomly moves and the rest of the agents try to follow it.

Modify the step action

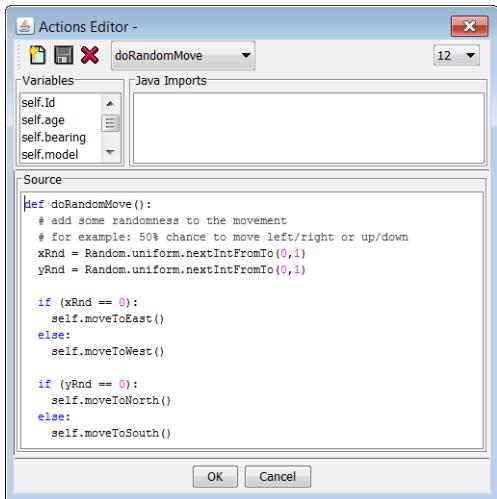
- 8 In the Environment panel, click Cougar. In the Property panel click the Edit button to the right of the Actions property to open the Actions Editor.
- 9 In the Actions Editor, select step from the Actions drop-down list and uncomment the code as shown in the following figure.



The first agent (Id=0) is the moving target, so if the agent's ID is equal to 0, then it must make a random move (`if (self.Id == 0): self.doRandomMove()`). If the agent is not the first agent, it must follow the first agent, and this can be done by calling its follow action (`self.follow()`).

Based on the preceding code, the first agent makes a random move in every tick (or time step) by calling the `doRandomMove` action. You need to define this action for the cougar agent with Id=0.

- 10 In the Actions Editor, select `doRandomMove` from the Actions drop-down list and uncomment the code as shown in the following figure.



The doRandomMove action is aimed at introducing randomness to the direction in which the agent moves. In the action's code, two random integers, which can be either 0 or 1, are generated and stored in the variables xRnd and yRnd (`xRnd = Random.uniform.nextIntFromTo(0,1); yRnd = Random.uniform.nextIntFromTo(0,1)`). The first if statement (`if (xRnd == 0)`) determines the direction of the x movement of the agent: if the value of xRnd is equal to 0, the agent moves to the east (`self.moveToEast()`); otherwise it moves to the west (`self.moveToWest()`). The second if statement (`if (yRnd == 0)`) determines the direction of the agent's y movement: if the value of yRnd is equal to 0, the agent moves to the north (`self.moveToNorth()`); otherwise it moves to the south (`self.moveToSouth()`).

11 Run the model.

Note: You may want to highlight the target agent so you can see where the other agents are moving toward. To do so, right-click the cougars layer and click Open Attribute Table. Select the first row (Id = 0) by clicking in the gray box to the far left of the row (probably with an arrow sign). The row will turn light blue. Close the attribute table, and you'll see that the target agent is now a different color than the other agents.

Because the cougar agent that is being followed moves randomly in all directions, over time it tends to stay in the same place (since, each time step, it has a 50 percent chance of moving east or west and north or south). Therefore, the other cougar agents will be able to converge with it almost as quickly as when the agent was in a fixed location in the previous exercise.

12 Close Agent Analyst and ArcMap without saving any changes.

As another example of how simple movements can be used to implement more complex movements, the current model will be modified in exercise 11d to give the agent the ability to evade a moving agent.

Having a point agent evade a moving point agent target

In this exercise, all the agents evade the first agent. In each step, the first agent performs a random movement.

Exercise 11d

Load the ArcMap document

- 1 Start ArcMap. Navigate to C:\ESRIPress\AgentAnalyst\Chapter11. Open ex11d.mxd.

In the ArcMap display you will see 12 points representing the locations of cougars. In ArcToolbox, the Chapter11Ex11d toolbox has been created for you. If you do not see any cougar agents in the display, right-click on the cougars layer in the Table Of Contents and click Zoom To Layer.

Reset the cougars shapefile

If you let the model run for some time in the preceding exercise (remember this exercise uses the cougars shapefile that you wrote to in the previous exercise as its input), you can replace the cougars shapefile with the copy you made in step 14 of exercise 11a. If you did not make a copy of the cougars shapefile in exercise 11a, you might want to remove and reinstall the chapter 11 data files.

- 2 To use the copy of the cougars shapefile you created in step 14 of exercise 11a before running the model again, use Catalog to navigate to and expand C:\ESRIPress\AgentAnalyst\Chapter11\Data\shapefiles.

You should see two shapefiles, cougars and cougarsCopy.

- 3 Right-click cougarsCopy and select Copy. Right-click the shapefiles directory and select Paste.

A new copy of cougarsCopy.shp is created as cougarsCopy2.shp.

- 4 Right-click cougars and click Delete. Right-click cougarsCopy2 and select Rename. Rename cougarsCopy2 to cougars and press Enter. You might also need to add cougars.shp to ArcMap while an ArcMap exercise document is open by using the Add Data button.

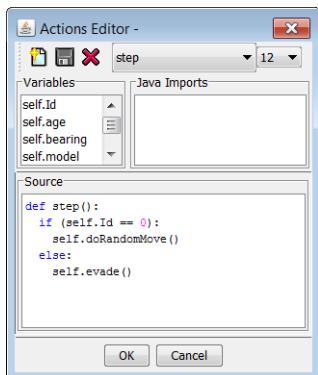
Load the Agent Analyst model

- 5 Right-click the Chapter11Ex11d toolbox, point to New, and select Agent Analyst Tool.
- 6 Once the Agent Analyst window appears, click File and select Import. Do not save the changes when prompted.
- 7 Navigate to C:\ESRI\Press\AgentAnalyst\Chapter11\Models and open ex11d.sbp.

In this exercise, the first agent makes a random move at each step, and the rest of the agents try to evade it. So, for this exercise, the step action of the cougar agent needs to be modified to give agents the ability to evade the first agent.

Update the step action

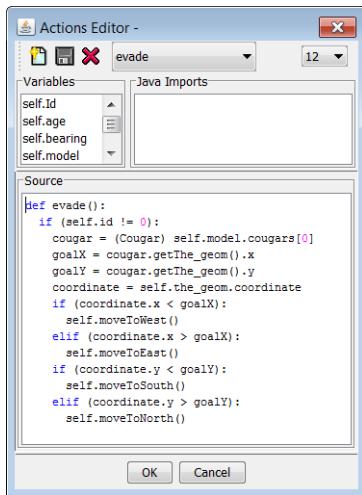
- 8 In the Environment panel, click Cougar. In the Property panel click the Edit button to the right of the Actions property to open the Actions Editor.
- 9 In the Actions Editor, select step from the Actions drop-down list and uncomment the code as shown in the following figure.



Here, if the agent's ID is equal to 0, it makes a random move (`if (self.Id == 0): self.doRandomMove ()`). If the agent is not the first agent, it must evade the first agent, and this can be done by calling the evade action (`self.evade ()`).

Define the evade action

- 10 In the Actions Editor window, select evade from the Actions drop-down list and uncomment the code shown in the following figure.



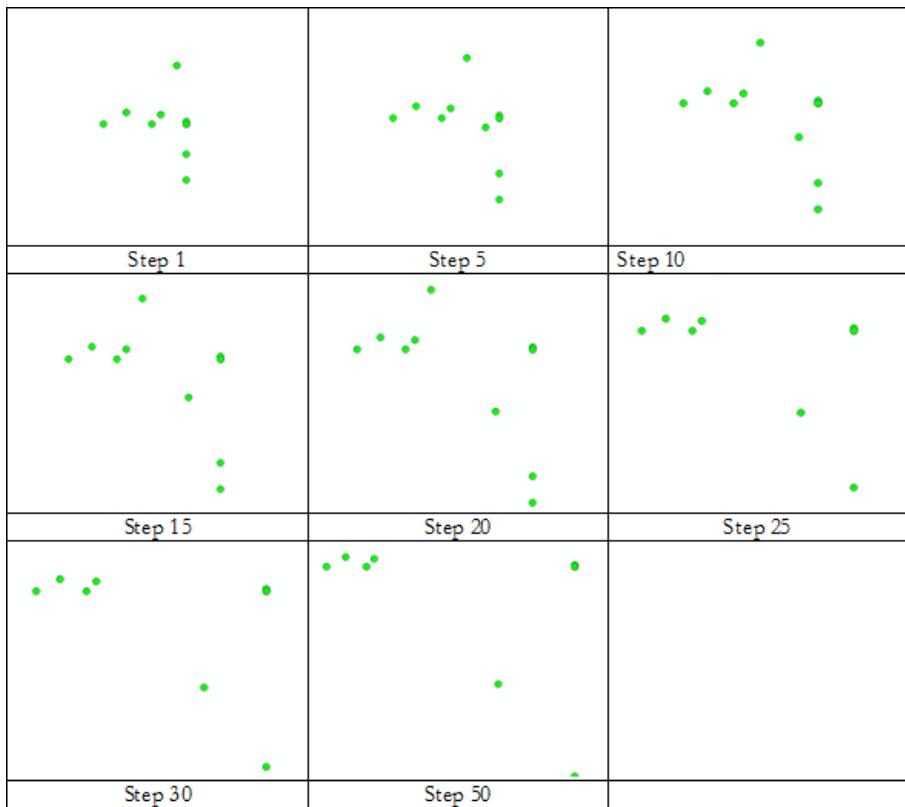
In this action, an agent first checks its own ID. If its ID is equal to 0, then it does nothing in this action (`if (self.id != 0):`), even though the agent with Id=0 did move randomly earlier in the `doRandomMove` action called by the step action. In this action, if the agent's ID is not equal to 0, it tries to evade the first agent. It first consults the model to learn the location of the first agent (`cougar = (Cougar) self.model.cougars[0]`) and stores these coordinates in the variables `goalX` and `goalY` (`goalX = cougar.getThe_geom().x; goalY = cougar.getThe_geom().y`). The coordinates of the agent itself are also stored in the `coordinate` variable (`coordinate = self.the_geom.coordinate`). Then, based on a comparison of its own position with the target position, the agent performs an evasive move: if the agent's x coordinate is less than the first agent's x coordinate, it needs to move west (`if (coordinate.x < goalX): self.moveToWest()`); otherwise, it moves to the east (`self.moveToEast()`). Similarly, if the agent's y coordinate is less than the first agent's y coordinate, the agent needs to move south (`if (coordinate.y < goalY): self.moveToSouth()`); otherwise it moves north (`self.moveToNorth()`). These moves serve to increase the distance between the agents and the agent they are trying to evade (the cougar agent with Id=0).

11

11 Run the model

Note: You may want to highlight the target agent so that you can see where the other agents are moving toward. To do so, right-click the cougars layer and click Open Attribute Table. Select the first row (Id = 0) by clicking in the gray box to the far left of the row (probably with an arrow sign). The row will turn light blue. Close the attribute table, and you'll see that the target agent is now a different color than the other agents.

The following image illustrates the positions of the agents evading a moving point target in different time steps. The starting locations of the cougar agents are different from the starting locations in your model, demonstrating that similar patterns will result regardless of the beginning locations.



12 Close Agent Analyst and ArcMap without saving your changes.

Polygon agents

In the previous four exercises you generically moved point agents, then moved them toward stationary and moving targets, and then had them evade another agent. In the next three exercises you will go through a similar progression with polygon agents. In these exercises, the polygon agents will represent fixed entities on a landscape (boundaries defining ZIP Codes). In this case, polygon agents really do not move, as the point agents did earlier. Movement, as defined in this fixed geometry, does not occur by the polygon agents but by what the agents can host. The movement can be an aggregate agent (e.g., the residents of a region or neighborhood) that is spatially linked to a polygon (e.g., a ZIP Code region). The residents can move between ZIP Code regions.

In the next three exercises you will have the features that the polygon hosts move toward stationary and moving targets and evade features moving between polygon agents. This general movement will be referred to as “moving polygon agents.”

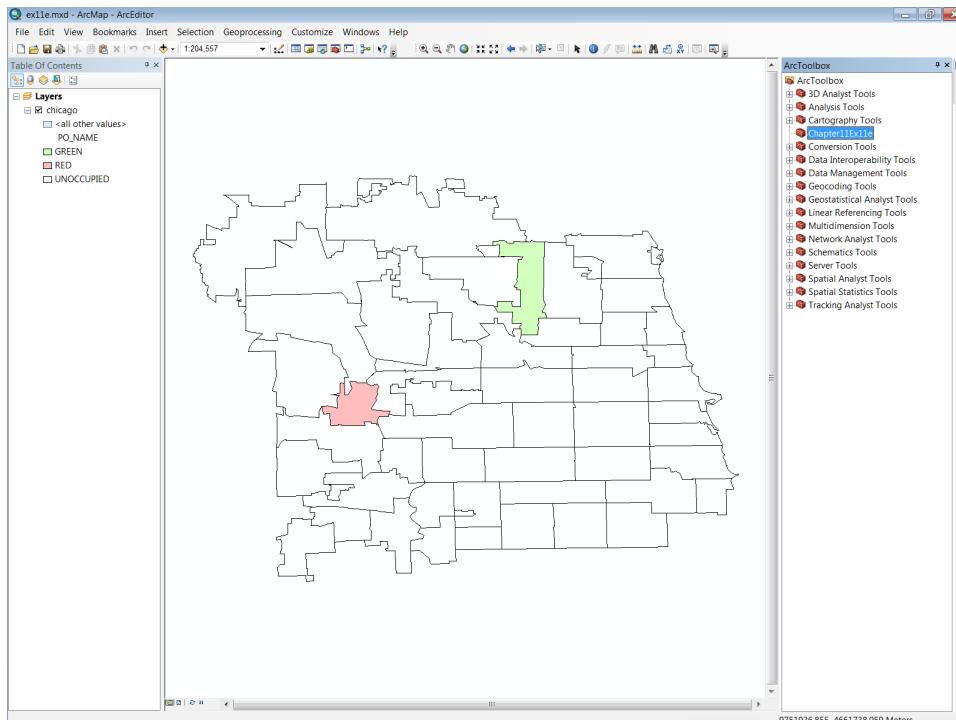
Moving a polygon agent toward a fixed polygon agent

In this exercise you move a polygon agent (or have the features associated with a polygon agent move between the various fixed polygon agents) toward a stationary target—a nonmoving polygon agent.

Exercise 11e

Load the ArcMap document

- 1 Start ArcMap. Navigate to C:\ESRIPress\AgentAnalyst\Chapter11. Open ex11e.mxd.



In the ArcMap display you will see the ZIP Code boundaries for Chicago. In ArcToolbox, the Chapter11Ex11e toolbox has been created for you.

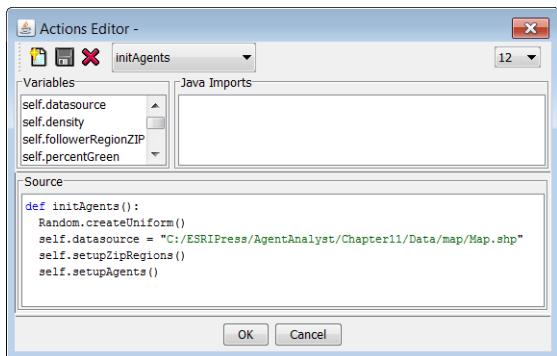
Load the Agent Analyst model

- 2 Right-click the Chapter11Ex11e toolbox, point to New, and select Agent Analyst Tool.
- 3 Once the Agent Analyst window appears, click File and select Import. Do not save the changes when prompted.
- 4 Navigate to C:\ESRIPress\AgentAnalyst\Chapter11\Models and open ex11e.sbp.

In this model there are two types of agents: ZipRegion and Resident. These agents are created from an input shapefile. ZipRegion agents are derived from the polygons identifying the ZIP Code boundaries of Chicago.

Define the initAgents action to initialize your agents

- 5 In the Environment panel, click Polygon Model. In the Property panel, click the Edit button to the right of the Actions property to open the Actions Editor.
- 6 In the Actions Editor, select initAgents from the Actions drop-down list and uncomment the code as shown in the following figure.

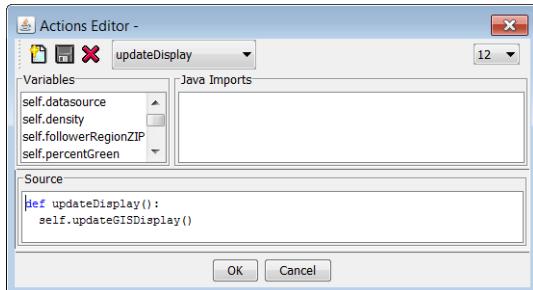


In this action, first a random distribution of numbers is generated (`Random.createUniform()`). Next, the agent's data source is set (`self.datasource = "C:/ESRIPress/AgentAnalyst/Chapter11/Data/map/Map.shp"`). Each type of agent has a specific action for initialization. Those actions are used to initialize the agents (`self.setupZipRegions(); self.setupAgents()`).

To update the display, the `updateDisplay` action is needed so that the screen will reflect the latest changes in the agents' locations.

Define the updateDisplay action

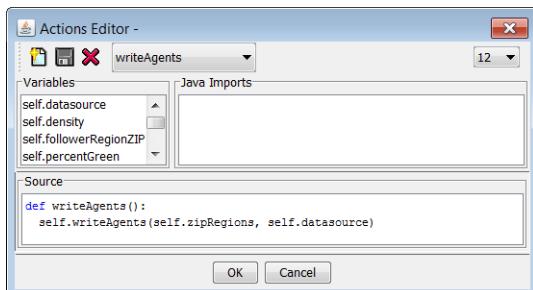
- 7 In the Actions Editor, select updateDisplay from the Actions drop-down list and uncomment the code as shown in the following figure.



This code calls the updateGISDisplay action to update the display (`self.updateGISDisplay()`).

However, prior to displaying the results, you need to write the new locations of the agents back to the Map.shp shapefile.

- 8 In the Actions Editor, select writeAgents from the Actions drop-down list and uncomment the code as shown in the following figure.

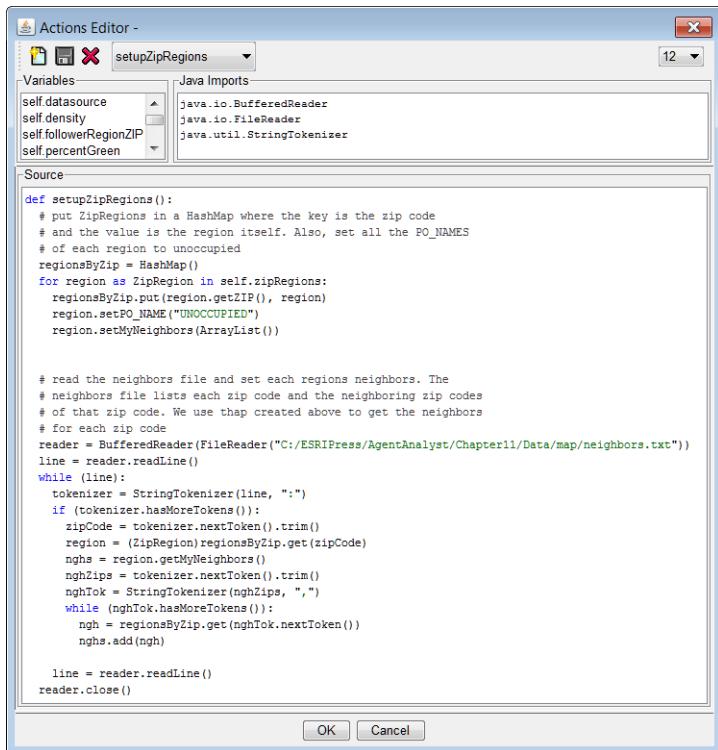


This code writes the agents' locations to the shapefile (`self.writeAgents(self.zipRegions, self.datasource)`). ArcMap will later use the updated information to display the agents.

Define the setupZipRegions action

The setupZipRegions action creates the required ZipRegion agents and identifies the neighbors of each ZipRegion.

- 9 In the Actions Editor, select setupZipRegions from the Actions drop-down list and uncomment the code shown in the following figure.



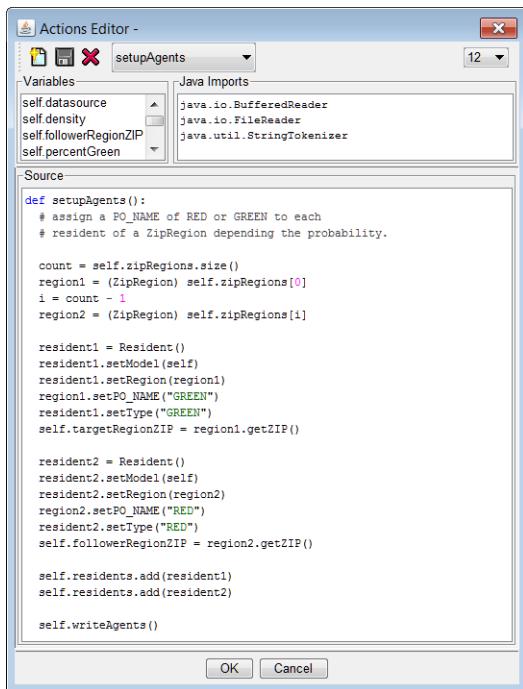
In this action, the code first defines the variable `regionsByZip` as a hash map (`regionsByZip = HashMap()`) in which the key is the ZIP Code and the value is the region itself (`regionsByZip.put(region.getZIP(), region)`). All the Zip-Regions initially have the “unoccupied” state (`region.setPO_NAME("UNOCCUPIED")`). Then, for each region, an empty array of neighbors is also allocated (`region.setMyNeighbors(ArrayList())`).

There is a file named `neighbors.txt` that lists all the ZIP Codes along with their neighboring ZIP Codes. The file is read (`reader = BufferedReader(FileReader("C:/ESRIPress/AgentAnalyst/Chapter11/Data/map/neighbors.txt"))`; `line = reader.readLine()`). In the while loop, each line read from the `neighbors.txt` file is first segmented upon seeing a colon (`tokenizer = StringTokenizer(line, ":")`). The first segment is the ZIP Code, so it is stored in the variable `zipCode` (`zipCode = tokenizer.nextToken().trim()`). The region associated with that ZIP Code is then retrieved (`region = (ZipRegion)regionsByZip.get(zipCode)`). The neighbors of that region are determined (`nghs = region.getMyNeighbors()`). The comma-separated list of the ZIP Codes of the neighboring regions is then stored in the variable `nghZips` (`nghZips = tokenizer.nextToken().trim()`). This sequence of numbers is then segmented

upon seeing the comma (`nghTok = StringTokenizer(nghZips, ",")`). The regions corresponding to those ZIP Codes are then retrieved (`ngh = regionsByZip.get(nghTok.nextToken())`) and added to the list of neighbors (`nghs.add(ngh)`).

The next type of agent that needs to be initialized is Resident. These are the agents that will move between the fixed polygon ZipRegion agents. Start by defining the `setupAgents` action. The aim of the `setupAgents` action is to create two resident agents: the “target resident” and the “follower resident.” To create the agents, this action first finds the two regions where the two resident agents need to be positioned. The region in which the target resident resides is called the “target region,” and the region in which the follower resident resides is called the “follower region.”

- 10** In the Actions Editor, select `setupAgents` from the Actions drop-down list and uncomment the code shown in the following figure.

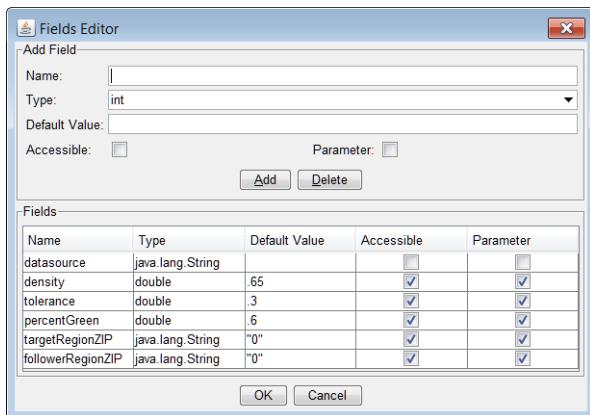


11

In this code, the first region in the `zipRegions` array is assigned to the target region (`region1 = (ZipRegion) self.zipRegions[0]`). The follower resident is positioned on the last region in the `ZipRegion`. To determine the follower region, the total number of regions is stored in the variable `count` (`count = self.zipRegions.size()`), and the element of the `zipRegions` array at the index `count - 1` (`i = count - 1`) is the last region (`region2 = (ZipRegion) self.zipRegions[i]`). The target region has a green state (`resident1.setType("GREEN")`), and the follower region has a red state (`resident2.setType("RED")`).

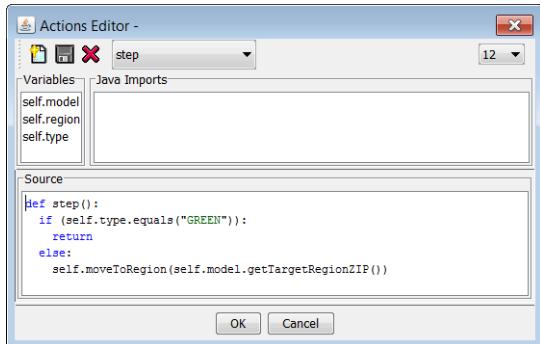
- 11** Click OK to save your changes and close the Actions Editor.

For the follower resident to move toward the target region, it must first know where the target region is. To implement this, two fields of type `java.lang.String` have been defined at the model level, as shown in the following figure. These fields are named `targetRegionZIP` and `followerRegionZIP`. The first field stores the ZIP Code of the target region (`self.targetRegionZIP = region1.getZIP()`), and the second stores the ZIP Code of the follower region (`self.followerRegionZIP = region2.getZIP()`).



Define the step action

- 12** In the Environment panel, click the Resident agent. In the Property panel, click the Edit button to the right of the Actions property to open the Actions Editor.
- 13** In the Actions Editor, select step from the Actions drop-down list and uncomment the code as shown in the following figure.

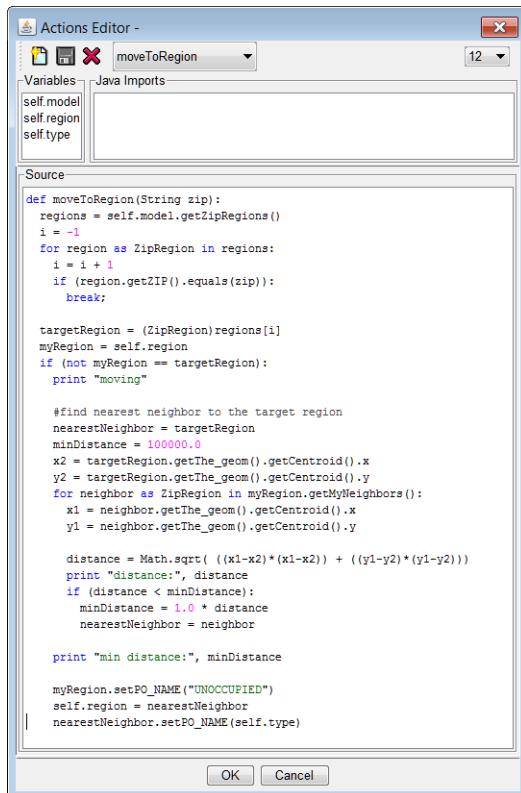


In this code, in each step the resident agent checks its own type (`if (self.type.equals ("GREEN"))`). If it is a green resident, it does nothing (`return`), because you

want the green region to be fixed. Otherwise, the agent first consults the model to retrieve the ZIP Code of the target region, and then it starts to move toward that region (`self.moveToRegion(self.model.getTargetRegionZIP())`).

Define the moveToRegion action

- 14** In the Actions Editor, select `moveToRegion` from the Actions drop-down list and uncomment the code shown in the following figure.



11

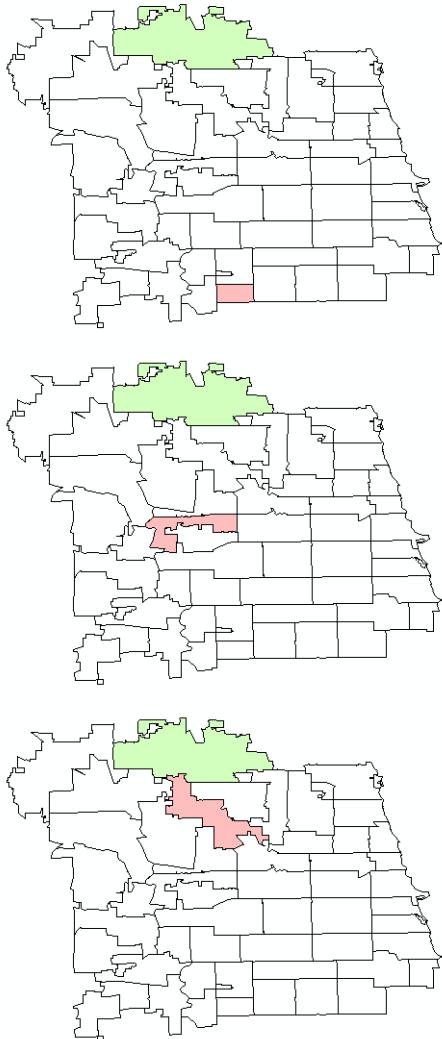
In this action, the follower resident agent first locates the target region by using its ZIP Code. This is done by retrieving all the regions (`regions = self.model.getZipRegions()`) and then using a for loop to compare the ZIP Code of the target region with the ZIP Codes of the other regions one by one until a match is found (`if (region.getZIP().equals(zip)): break;`). Then the follower agent evaluates its own neighbors and selects the neighbor that is the nearest to the target region. The goal is to find that neighbor and move to it to get closer to the target.

To do this, the action first stores the coordinates of (the centroid of) the target region in the variables x2 and y2 (`x2 = targetRegion.getThe_geom().getCentroid().x;` `y2 = targetRegion.getThe_geom().getCentroid().y`). Then, in a for loop, for each neighboring region, the coordinates of (the centroid of) that region are stored in variables x1 and y1 (`x1 = neighbor.getThe_geom().getCentroid().x;` `y1 = neighbor.getThe_geom().getCentroid().y`). Then the distance between these two points is calculated based on the mathematical formula for calculating the distance between two points on a plane (`distance = Math.sqrt(((x1-x2) * (x1-x2)) + ((y1-y2) * (y1-y2)))`). If the resulting value is less than the current value for the minimum distance (which is initially set to a large number), this value is assigned to the minimum distance (`minDistance = 1.0 * distance`), and the corresponding region is assigned to a variable storing the nearest neighbor (`nearestNeighbor = neighbor`). Once all neighboring regions are evaluated, the follower resident moves to the neighbor that gets the agent closest to the target agent by changing its own region (`self.region = nearestNeighbor`).

- 15 Click OK to save your changes and close the Actions Editor.
- 16 Run the model.

The following images illustrate the red follower resident agent moving toward the green target resident agent.

Note: If a RePast Output window opens and a series of statements, including warning messages, are displayed, you can minimize the window and ignore the messages.



11

- 17 Close Agent Analyst and ArcMap without saving any changes.

Having a polygon agent follow a moving polygon agent

In this exercise, you will modify the previous model to implement a model in which a polygon follower agent moves toward a moving polygon target agent. In other words, it follows the target agent.

In this model, the green target agent performs a random move in each time step—that is, it randomly selects one of its neighboring regions and moves toward that region. The red follower agent, as before, moves toward the target region in each time step.

Exercise 11f

Load the ArcMap document

- 1 Start ArcMap. Navigate to C:\ESRIPress\AgentAnalyst\Chapter11. Open ex11f.mxd.

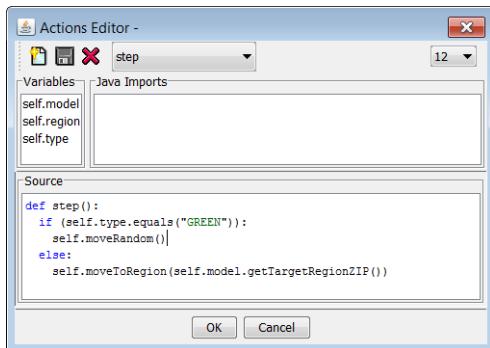
In the ArcMap display you will see the ZIP Code boundaries for Chicago. In ArcToolbox, the Chapter11Ex11f toolbox has been created for you.

Load the Agent Analyst model

- 2 Right-click the Chapter11Ex11f toolbox, point to New, and select Agent Analyst Tool.
- 3 Once the Agent Analyst window appears, click File and select Import. Do not save the changes when prompted.
- 4 Navigate to C:\ESRIPress\AgentAnalyst\Chapter11\Models and open ex11f.sbp.

Modify the step action of the resident agent

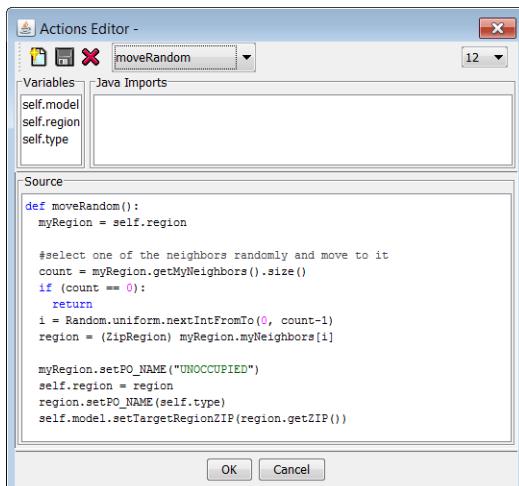
- 5 In the Environment panel, click Resident. In the Property panel, click the Edit button to the right of the Actions property to open the Actions Editor.
- 6 In the Actions Editor, select step from the Actions drop-down list and uncomment the code as shown in the following figure.



Here, in each step the resident agent checks its own type. If it is a green resident (`if (self.type.equals("GREEN"))`), it performs a random move (`self.moveRandom()`); otherwise, it moves toward the target agent (`self.moveToRegion (self.model.getTargetRegionZIP())`).

Define the moveRandom action

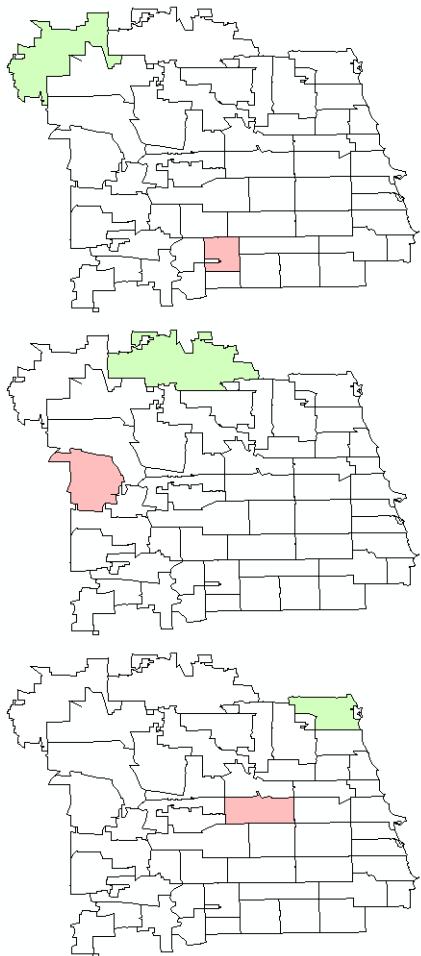
- In the Actions Editor, select moveRandom from the Actions drop-down list and uncomment the code as shown in the following figure.



In this code, the agent selects one of its neighboring regions and moves into it. To do that, the total number of the agent's neighbors is stored in the variable count (`count = myRegion.getMyNeighbors().size()`). If there are no neighbors, the agent does nothing (`if (count == 0): return`); otherwise, a random integer is generated that can be between 0 and count - 1 (`i = Random.uniform.nextIntFromTo(0, count-1)`). This integer is the index to refer to the selected neighbor (`region = (ZipRegion) myRegion.myNeighbors[i]`). The agent's current PO_NAME is set to "UNOCCUPIED" (`myRegion.setPO_NAME("UNOCCUPIED")`), its region is updated (`self.region = region`), its type is assigned to the new region's PO_NAME (`region.setPO_NAME(self.type)`), and the action updates the target region's ZIP Code (`self.model.setTargetRegionZIP(region.getZIP())`).

- 8 Click OK to save your changes and close the Actions Editor.
- 9 Run the model.

The following images illustrate the red resident (follower agent) following the green resident (target agent). Once the follower agent catches the target agent, it will move with it each time step. If you review the code of the `moveToRegion` action, you will see that the follower identifies where the target agent is and then determines the closest neighboring region to the target. Since the closest neighboring region is the region the target agent is in, the follower will also move into it.



11

- 10** Close Agent Analyst and ArcMap without saving any changes.

Having a polygon agent evade a moving polygon agent

In this exercise, you will modify the previous model to implement a more complex model in which a polygon agent follows a moving polygon target agent, and the target polygon agent evades the follower agent.

In each step, the green target agent first consults the model to find out where the follower agent is. This is done by retrieving the ZIP Code of the region where the follower agent resides. The target agent evaluates its own neighbors to find the region that is farthest away from the region containing the follower agent. Then it moves to that neighbor to evade the follower agent. The follower agent, in each time step, moves one region toward the green target agent.

Exercise 11g

Load the ArcMap document

- 1 Start ArcMap. Navigate to C:\ESRIPress\AgentAnalyst\Chapter11. Open ex11g.mxd.

In the ArcMap display you will see the ZIP Code boundaries for Chicago. In ArcToolbox, the Chapter11Ex11g toolbox has been created for you.

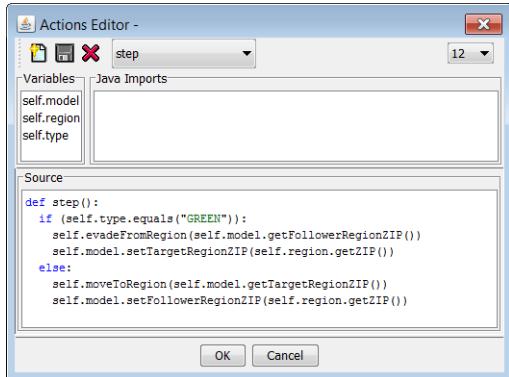
Load the Agent Analyst model

- 2 Right-click the Chapter11Ex11g toolbox, point to New, and select Agent Analyst Tool.
- 3 Once the Agent Analyst window appears, click File and select Import. Do not save the changes when prompted.
- 4 Navigate to C:\ESRIPress\AgentAnalyst\Chapter11\Models and open ex11g.sbp.

Modify the step action of the resident agent

- 5 In the Environment panel, click Resident. In the Property panel, click the Edit button to the right of the Actions property to open the Actions Editor.

- 6 In the Actions Editor, select step from the Actions drop-down list and uncomment the code as shown in the following figure.

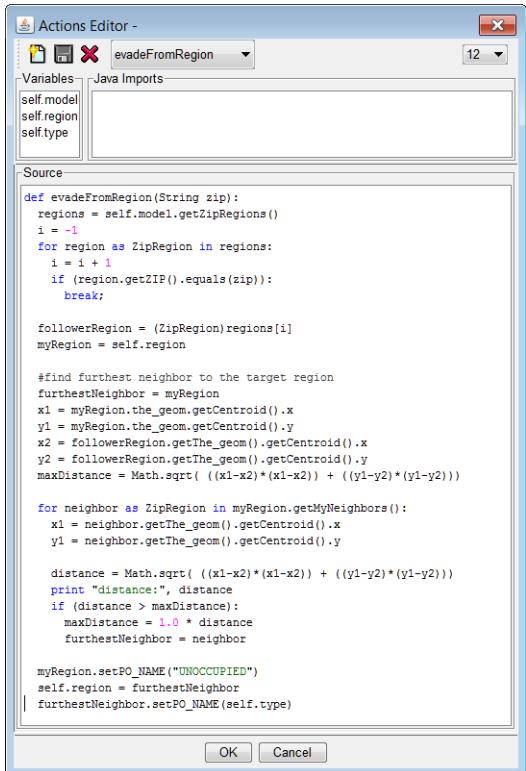


In this code, in each time step the resident agent checks its own type. If it is a green resident (if (self.type.equals("GREEN"))), it tries to evade the region the follower agent is currently in (self.evadeFromRegion(self.model.getFollowerRegionZIP())) and the target region agent's ZIP Code gets updated (self.model.setTargetRegionZIP(self.region.getZIP())). If the agent's type is not green, it is a red (follower) agent, so it moves toward the target region agent (self.moveToRegion(self.model.getTargetRegionZIP())) and updates the follower region agent's ZIP Code to the ZIP Code of the region it has just moved to (self.model.setFollowerRegionZIP(self.region.getZIP())).

11

Define the `evadeFromRegion` action, which helps the target resident agent evade the region where the follower agent currently is, given its ZIP Code

- 7 In the Actions Editor, select `evadeFromRegion` from the Actions drop-down list and uncomment the code as shown in the following figure.



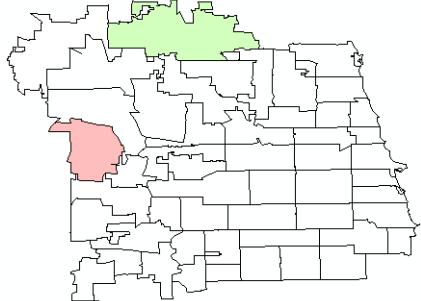
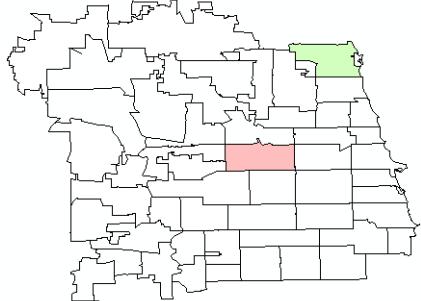
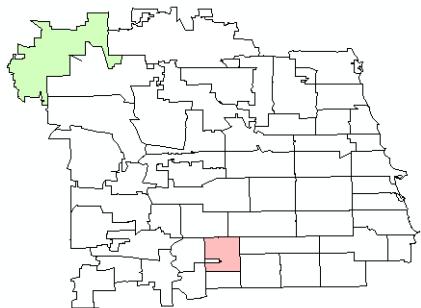
In this action, the target resident agent first locates the follower region agent using its ZIP Code. This is done by retrieving all the regions (`regions = self.model.getZipRegions()`) and then using a for loop to compare the ZIP Code of the follower region agent with the ZIP Codes of the other regions one by one until a match is found (`if (region.getZIP().equals(zip)): break;`). Then the target agent evaluates its own neighbors and selects the neighbor that is the farthest away from the follower region agent. The goal is to find that neighbor and move into it to get as far as possible away from the follower agent. To do this, the action first stores the coordinates of the target region agent into the variables `x1` and `y1` (`x1 = myRegion.the_geom.getCentroid().x; y1 = myRegion.the_geom.getCentroid().y`). The current coordinates of the follower region agent are also stored in the variables `x2` and `y2` (`x2 = followerRegion.getThe_geom().getCentroid().x; y2 = followerRegion.getThe_geom().getCentroid().y`). Then the distance between these two points is calculated using the mathematical formula for calculating the distance between two points on a plane (`maxDistance = Math.sqrt(((x1-x2)*(x1-x2)) + ((y1-y2)*(y1-y2)))`). In a for loop, for each neighboring region surrounding the region containing the target agent, the coordinates of that region are stored in the variables `x1` and `y1` (`x1 = neighbor.getThe_geom().getCentroid().x; y1 = neighbor.getThe_geom().getCentroid().y`). The distance between the neighboring region and the follower region agent is calculated according to the same formula (`distance = Math.sqrt(((x1-x2)*(x1-x2)) + ((y1-y2)*(y1-y2)))`). If this distance is greater than the current maximum distance, the maximum distance is updated to this value and the neighbor is set as the furthest neighbor. Finally, the target region's PO name is set to "UNOCCUPIED", the follower region's PO name is set to the target region's type, and the follower region's PO name is set to the target region's type.

((y1-y2) * (y1-y2))). If the resulting value is greater than the current value for the maximum distance (`if (distance > maxDistance)`), this value is assigned to the maximum distance (`maxDistance = 1.0 * distance`) and the corresponding region is assigned to the farthest neighbor (`furthestNeighbor = neighbor`). Then the target resident agent moves to that neighbor by changing its own region (`self.region = furthestNeighbor`).

8 Click OK to save your changes and close the Actions Editor.

9 Run the model.

The following images illustrate the green resident (target agent) evading the red resident (follower agent).



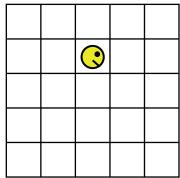
11

10 Close Agent Analyst and ArcMap without saving any changes.

Moving a point agent relative to a raster surface

In this exercise you will implement a modified version of the complete set of simple actions discussed in exercise 11a. However, in this exercise the agents will move relative to a raster surface. The major difference between the movement actions here and the ones in exercise 11a is that the unit of movement is “cell size”—that is, the size of the cells that make up the surface. Another major difference between vector surfaces and raster surfaces is that raster surfaces do not have an unlimited size, so you need to make sure that the agents consistently move within the borders.

You need to make some changes to the previous actions—for instance, you will make required changes to stop an agent from moving outside the raster boundaries. From exercise 11a you already know that every agent has a location, which includes a pair of integers (x, y) and also a bearing. If you limit your agents to movement only in four directions—north, south, west, and east—then it is possible to use a simple convention for representing the bearing of the agent: 0 for north, 90 for east, 180 for south, and 270 for west. For example, in the following image, an agent is shown with the position (2, 1) and the bearing equal to 90 (facing east). Note that the left-uppermost point in the image has a row/column position of (0, 0). However, in Agent Analyst when moving an agent you usually do not move it using row/column space. You generally track agent locations in map units; when moving the agents over a raster, you move them the cell-size distance (but still in map units).



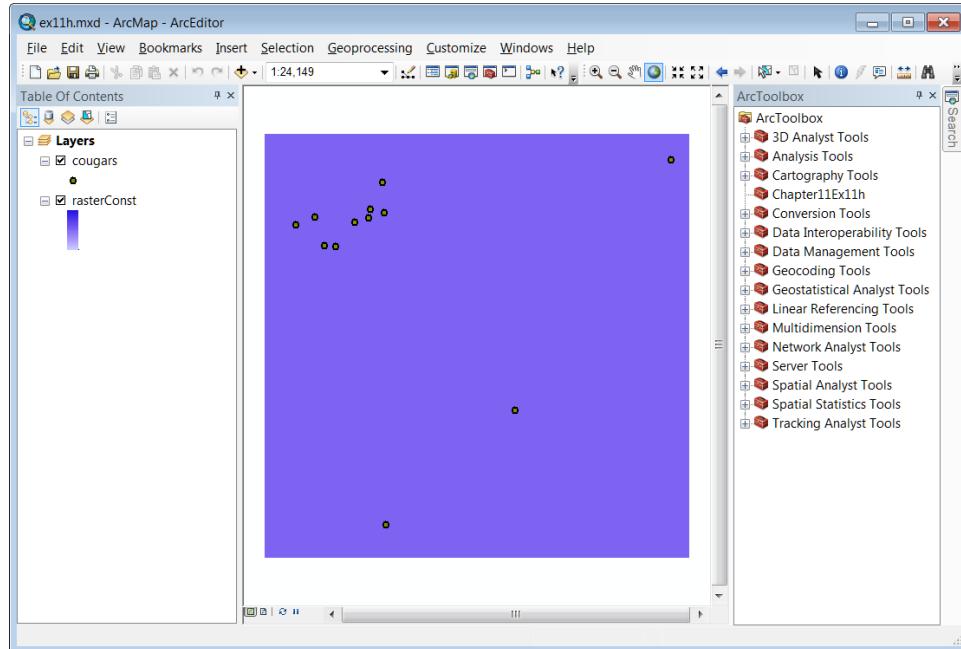
In this exercise, each agent is initialized to have a bearing of 90 (facing east). There is a trap in the raster, and if an agent goes into the trap it cannot exit. The trap is a rectangular area. In each time step, the agent first checks to see if it is in the trap. If it is, it does nothing. Otherwise, a random integer from 1 to 5 is calculated. The random number defines the number of cells to move in that time step, and then the agent moves forward that many cells. In addition, a random integer from 1 to 2 is defined. If the random value is 2, the agent turns right. Your objective is to see how long it takes for these randomly moving agents to enter the trap.

Exercise 11h

Load the ArcMap document

- 1 Start ArcMap. Navigate to C:\ESRIPress\AgentAnalyst\Chapter11. Open ex11h.mxd.

In the ArcMap display you will see a series of points distributed throughout the landscape. If you do not see any cougar agents in the display, right-click on the cougars layer in the Table Of Contents and click Zoom To Layer. The configuration of the agents in your display may differ from that in the following figure.



In ArcToolbox, the Chapter11Ex11h toolbox has been created for you.

Reset the cougars shapefile

If you let the model run for some time in the earlier exercises (remember this exercise uses the cougars shapefile that you wrote to in an earlier exercise as its input) you can replace the cougars shapefile with the copy you made in step 14 of exercise 11a. If you did not make a copy of the cougars shapefile in exercise 11a you might want to remove and reinstall the chapter 11 data files.

- 2** To use the copy of the cougars shapfile you created in step 14 of exercise 11a before running the model again, use Catalog to navigate to and expand C:\ESRIPress\AgentAnalyst\Chapter11\Data\shapefiles.

You should see two shapefiles, cougars and cougarsCopy.

- 3** Right-click cougarsCopy and select Copy. Right-click the shapefiles directory and select Paste.

A new copy of cougarsCopy.shp is created as cougarsCopy2.shp.

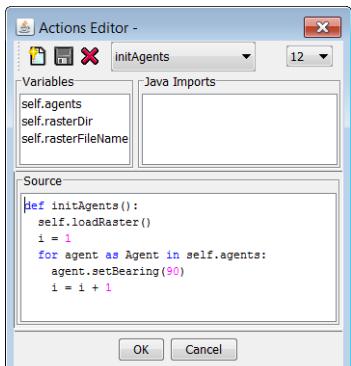
- 4 Right-click **cougars** and click Delete. Right-click cougarsCopy2 and select Rename. Rename cougarsCopy2 to cougars and press Enter. You might also need to add cougars.shp to ArcMap while an ArcMap exercise document is open, using the Add Data button.

Load the Agent Analyst model

- 5 Right-click the Chapter11Ex11h toolbox, point to New, and select Agent Analyst Tool.
- 6 Once the Agent Analyst window appears, click File and select Import. Do not save the changes when prompted.
- 7 Navigate to C:\ESRIPress\AgentAnalyst\Chapter11\Models and open ex11h.sbp.

Add the initAgents action to initialize agents and set their bearing field so that they all face east

- 8 In the Environment panel, click GIS Model. In the Property panel, click the Edit button to the right of the Actions property to open the Actions Editor.
- 9 In the Actions Editor, select initAgents from the Actions drop-down list and uncomment the following code.

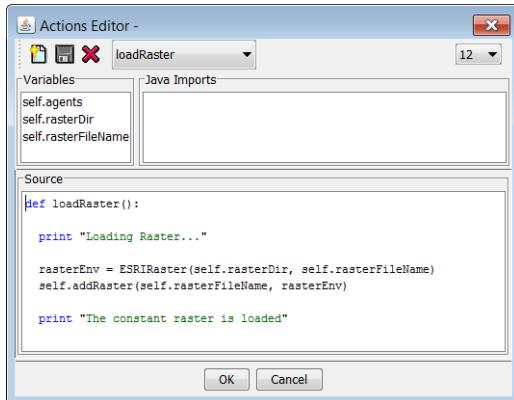


In this code, the loadRaster action is called to load the raster surface the agents are to move on (`self.loadRaster()`). Then, in the for loop, the bearing field of all agents is set to 90, which means that initially they are all facing east (`agent.setBearing(90)`).

- 10 Click OK to save your changes and close the Actions Editor.
- 11 In the Environment panel, click the Name agent. In the Property panel, click Group Name, and change the value to **agents**. Then click Name, and change the value to **Agent**.

Define the loadRaster action

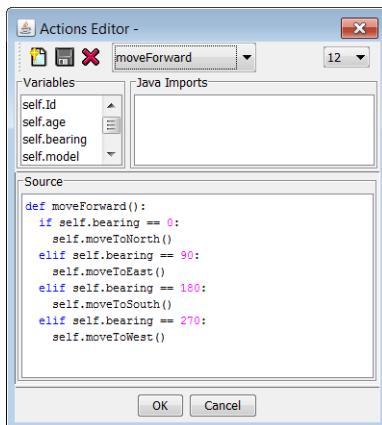
- 12 In the Environment panel, click GIS Model. In the Property panel, click the Edit button to the right of the Actions property to open the Actions Editor.
- 13 In the Actions Editor, select loadRaster from the Actions drop-down list and uncomment the code as shown in the following figure.



In this code, a raster object is created and stored in the variable rasterEnv (`rasterEnv = ESRIRaster(self.rasterDir, self.rasterFileName)`). The parameter `self.rasterDir` refers to the path where the raster dataset is stored and `self.rasterFileName` refers to the name of the raster dataset. Both parameters are stored as fields in the model. Then you register the raster object with the current model (`self.addRaster(self.rasterFileName, rasterEnv)`).

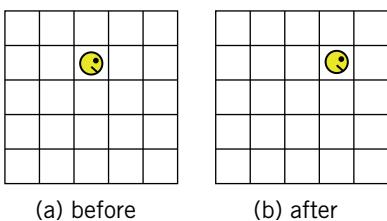
The next couple of actions are the basic movement actions of agents.

The moveForward action allows the agent to move forward.

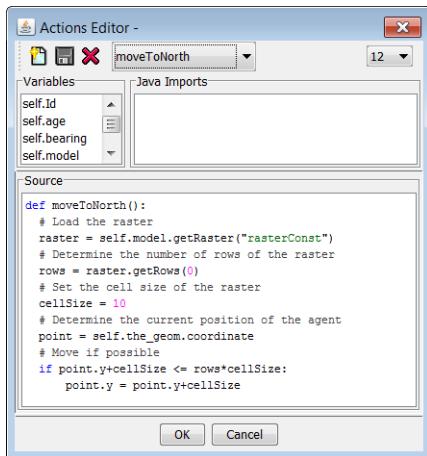


In this action, if the agent's bearing is equal to 0 (`if self.bearing == 0:`), it means that the agent is facing north, so to move forward it needs to move north (`self.moveToNorth()`). Similarly, if the bearing is equal to 90 (`elif self.bearing == 90`), the agent is facing east and needs to move east, and so on.

In the following images, an agent is shown before and after performing a moveForward action. Before the action (a), the agent is at (2, 1) with bearing 90, and after the action (b), the agent is at (3, 1) with bearing 90.



The `moveToNorth` action allows the agent to move north.



In this code, first the number of rows of the surface is determined (`rows = raster.getRows(0)`). The cell size is then set to 10, which is the size used when creating the raster surface (`cellSize = 10`). To make sure that the agent remains within the surface boundaries, the following steps are necessary. First, the current location of the agent is identified and stored in the `point` variable (`point = self.the_geom.coordinate`). To move the agent to the north, the agent's `y` field must be incremented by the cell size, but before doing that you must make sure that if you increment the `y` field by the cell size, it will still have a valid value (less than or equal to the number of rows multiplied by cell size). This condition is checked in an `if` statement (`if point.y+cellSize <= rows*cellSize`). If the condition is true, the location of the move is still within the boundaries of the raster, therefore the agent's `y` field is incremented by the cell size (`point.y = point.y+cellSize`).

It is important to point out that the boundary condition (`if point.y+cellSize <= rows*cellSize`) is making the test in map units, not in row/column units. The lower left is the origin of the study extent of the map units. Multiplying the number of rows by the cell size identifies the map coordinates of the northern boundary of the study extent. In this case, the map coordinates of the origin (the lower left) of the raster are (0, 0).

Calculating the cell size of a raster surface and moving the agents to the neighboring map cells

A raster surface is a generalization of the landscape into cells. Each cell is (generally) a square, and at any given time an agent can only be at the center of the cell it is in. In the cougar model, during each time step a cougar agent will either move into one of the eight surrounding cells or not. Since the agents on a raster surface can move only from cell center to cell center, the cell size of the raster surface must be known to make the movement possible.

Calculating the cell size

There are two ways to deal with cell size:

The cell size is known beforehand: This could be the case when, for example, you create a raster surface where the cell size can be set manually (for instance, by using the Raster Creation tool under Spatial Analyst Tools in ArcMap). Or, the cell size can be derived by querying the raster.

The cell size is not known beforehand: If you are using an existing raster, you may not know the cell size. To determine the cell size of such a raster, you first need to know where the agent is. Then you need to calculate the location of one of its eight surrounding cells. Once these two values are calculated, you need to convert them from row/column coordinates to map coordinates. Based on which cell you have chosen, the cell size can be calculated in one of these two ways:

Choosing a cell on a cardinal direction: If any cell directly to the north, south, east, or west is chosen, cell size can be calculated by subtracting x values or y values. For example, if the current location of the agent is (200, 300) and the cell to its north is located at (200, 310), the cell size is equal to 10 (the difference between 300 and 310).

continued

Choosing a cell on a diagonal direction: If any of the cells on the diagonal directions (north-east, south-west, and so on) is chosen, the procedure is a bit more complicated. First of all, the geometric distance between the current cell center and the cell center of the selected neighboring cell must be calculated. This distance can be calculated using the mathematic formula for calculating the distance between two points in a two-dimensional (2D) plane. Let's name the result of this calculation *Distance*. Given two points, namely (a, b) and (c, d), on a 2D plane, their geometrical distance is:

$$\text{Distance} = \sqrt{(a - c)^2 + (b - d)^2}$$

These two points are also located on the two ends of the hypotenuse of a hypothetical right (and at the same time isosceles) triangle. Hence, the cell size is calculated as:

$$\text{Cell Size} = \text{Distance} \times \frac{\sqrt{2}}{2}$$

Moving the agents

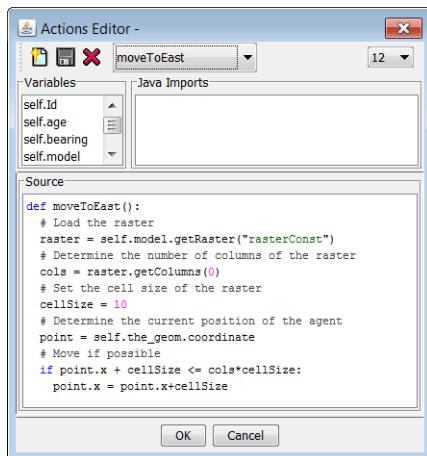
Since each cell on a raster surface is surrounded by eight neighboring cells, there are two possible types of moves for agents on a raster surface:

Cardinal moves (north, south, east, or west): These moves correspond to moving to the cell that is above, below, to the right, or to the left. In such cases, *one* of the coordinates of the current location of the agent (which—as we know—is located at the central point of a cell) will change by the cell size. For instance, if the current location of the agent is (400, 500) and it wants to move to the east and the cell size is 20, this value will be added to the x coordinate, yielding (420, 500) as the new coordinates of the agent.

Diagonal moves (north-west, south-east, and so on): These moves allow the agent to move diagonally. Here, both coordinates of the current location of the agent will change by the cell size. For instance, if the current location of the agent is (300, 400), the destination cell is the one to the north-west, and the cell size is 20, to calculate the coordinates of the destination point, 20 must be subtracted from the x value and added to the y value, resulting in (280, 420) being the center of the cell on the north-west.

It is worth mentioning that all the movements in exercise 11h that involve changes in the agent's position are implemented in terms of the four basic movements (moveToNorth, moveToSouth, moveToEast, and moveToWest), and it is guaranteed that the agent's coordinates after making any of the four basic moves will be valid.

The moveToEast action allows the agent to move east.

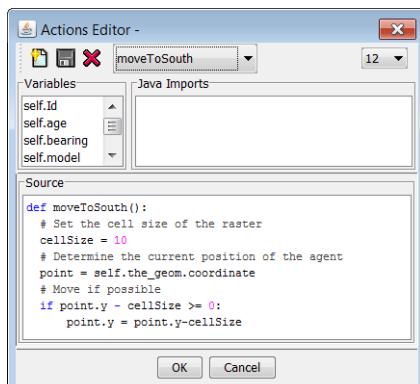


11

In this code, first the number of columns of the surface is determined (`cols = raster.getColumns(0)`). The cell size is then set to 10, which is the size used when creating the raster surface (`cellSize = 10`).

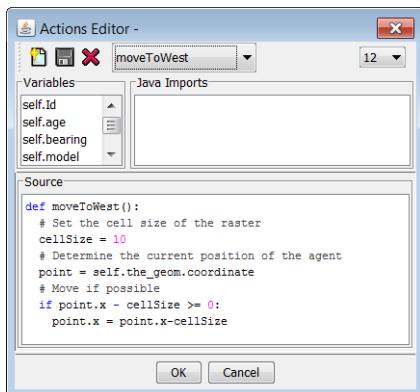
Next, the current location of the agent is identified and is stored in the `point` variable (`point = self.the_geom.coordinate`). If the agent's current value of the `x` field will be valid after adding the cell size to it (less than or equal to the number of columns multiplied by cell size) (`if point.x + cellSize <= cols*cellSize`), then increment the agent's `x` field by the cell size (`point.x = point.x+cellSize`).

The moveToSouth action allows the agent to move south.



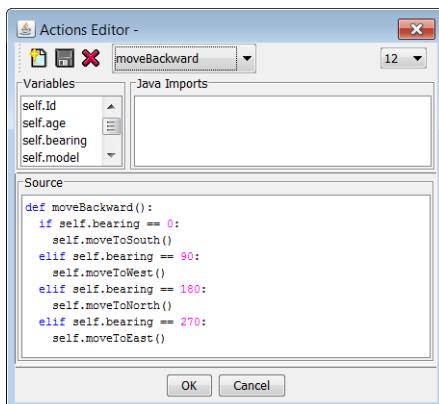
Here, the current location of the agent is identified and is stored in the point variable (`point = self.the_geom.coordinate`). To move the agent to the south, the agent's y field must be decremented by the cell size, but before doing so you must make sure that if you decrement y by the cell size, it will still be less than or equal to 0 (`if point.y - cellSize >= 0`). If true, the cell size is subtracted from the agent's y field (`point.y = point.y - cellSize`).

The `moveToWest` action allows the agent to move west.



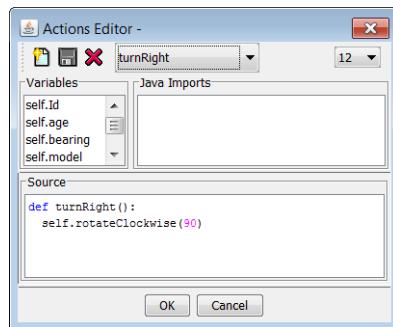
In this code, the current location of the agent is identified and is stored in the point variable (`point = self.the_geom.coordinate`). To move the agent to the west, the agent's x field must be decremented by the cell size, but before doing so you must make sure that if you decrement the x field by the cell size, it will still be greater than or equal to 0 (`if point.x - cellSize >= 0`). If true, the x value for the agent will be decremented by the cell size (`self.x = self.x - cellSize`).

The `moveBackward` action allows the agent to move backward.



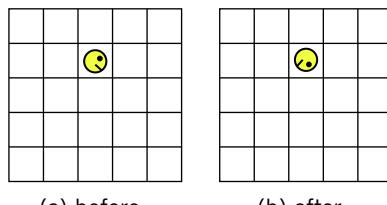
In this action, if the agent's bearing is equal to 0 (`if self.bearing == 0:`), it means that the agent is facing north, so to move backward it needs to move to the south (`self.moveToSouth()`). Similarly, if the bearing is equal to 90 (`elif self.bearing == 90`), the agent is facing east and needs to move to the west, and so on.

The `turnRight` action allows the agent to turn right.

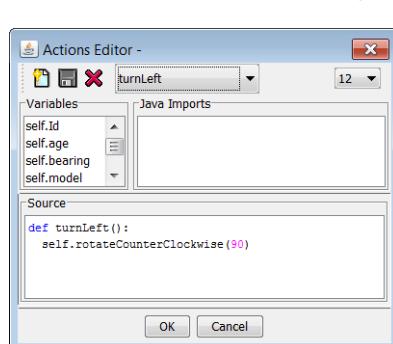


For an agent to turn right, it needs to rotate clockwise for 90 degrees (`self.rotateClockwise(90)`).

The following images illustrate an agent performing a `turnRight` action. Before the action (a) the agent is at (2, 1) with a bearing of 90. After the action (b), the agent is at (2, 1) with a bearing of 180.

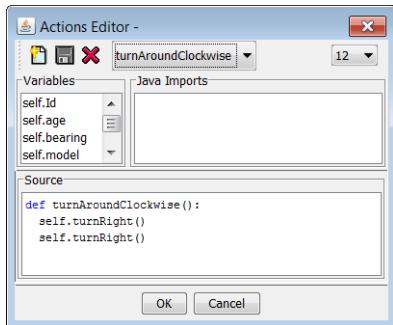


The `turnLeft` action allows the agent to turn left.



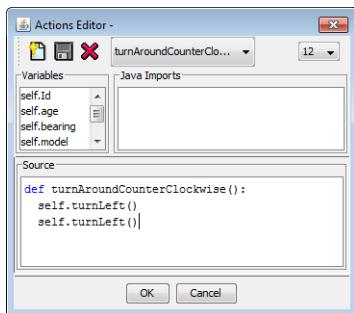
For an agent to turn left it needs to rotate counterclockwise for 90 degrees (`self.rotateCounterClockwise(90)`).

Now we need to define the action `turnAroundClockwise`.



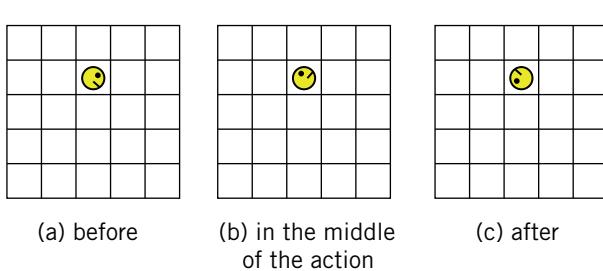
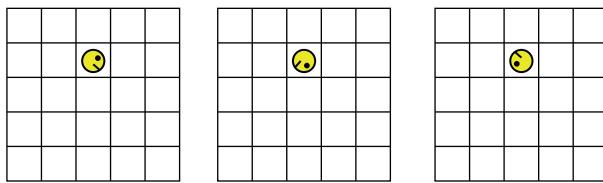
In this code, to turn around clockwise, the agent turns right twice (`self.turnRight()`).

We also need to define `turnAroundCounterClockwise`.



To turn around counterclockwise, the agent turns left twice (`self.turnLeft()`).

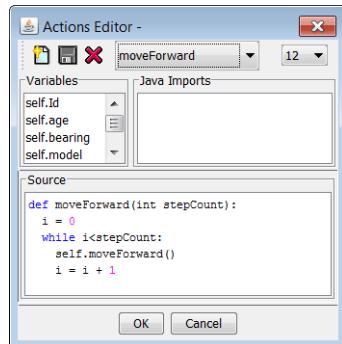
The result of performing the `turnAroundClockwise` and `turnAroundCounterClockwise` actions is illustrated in the following figures.



The top illustration shows an agent performing a turnAroundClockwise action: (a) before the action, (b) in the middle of the action, and (c) after the action. The bottom illustration shows an agent performing a turnAroundCounterClockwise action: (a) before the action, (b) in the middle of the action, and (c) after the action.

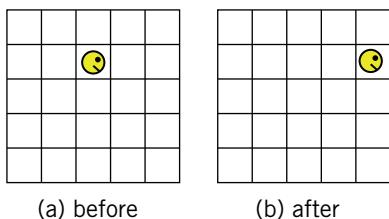
A second version of the moveForward action is shown in the following figure. The action allows the agent to move more than one cell distance forward in one time step.

Note: For demonstrating the sequence of the expanded logic, we will add complexity to the basic moveForward action by creating three different versions of the action. In actual practice, if you want to have three variations of the same action—that is, if you want to have three different actions—you should name them uniquely.

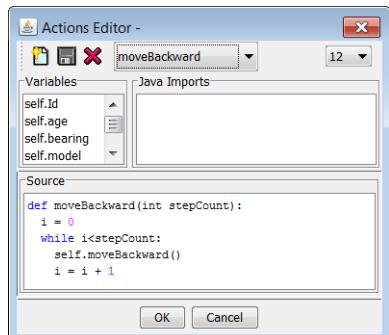


In this code, the number of desired forward moves is passed to the action by the stepCount variable (`moveForward(int stepCount)`). In the while loop, the agent calls the `moveForward` action (`self.moveForward()`) and increments the variable `i`, which was initially set to 0 (`i = i + 1`). The while loop guarantees that the `moveForward` action is executed as many times as the value of `stepCount`.

In the following figure an agent has performed a `moveForward` action with `stepCount` set to 2 (moving forward two steps): before the action (a), the agent is at (2, 1) with a bearing of 90, and after the action (b), the agent is at (4, 1) still with a bearing of 90.

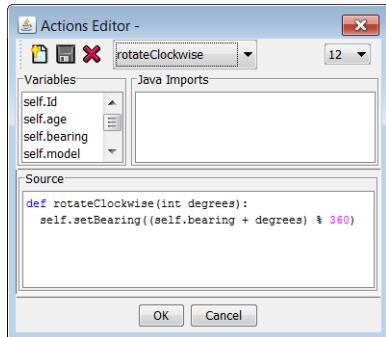


A second version of the `moveBackward` action is shown in the following figure. The action allows the agent to move backward more than one cell in a time step.



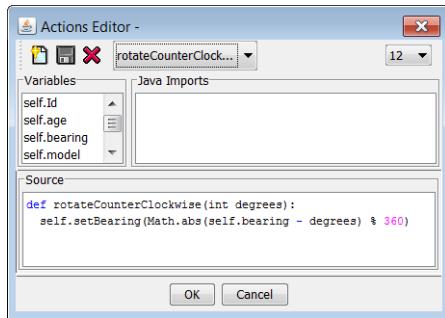
Here, the number of desired backward moves is passed to the action by the stepCount variable (`moveBackward(int stepCount)`). In the while loop, the agent calls the `moveBackward` action (`self.moveBackward()`) and increments the variable `i`, which was initially set to 0 (`i = i + 1`). The while loop guarantees that the `moveBackward` action is executed as many times as the value of `stepCount`.

The rotateClockwise action allows the agent to rotate clockwise by certain degrees.



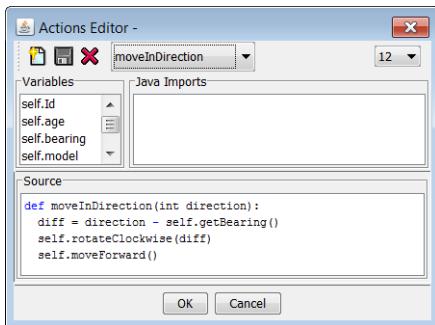
In this action, the agent adds the requested degrees of rotation to its bearing. The modulus operator is applied—the sum being divided by 360—and then the result (the remainder) is set as its new bearing (`self.setBearing((self.bearing + degrees) % 360)`).

The rotateCounterClockwise action allows the agent to rotate counterclockwise by certain degrees.



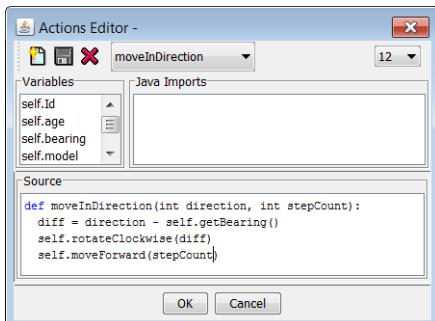
In this action, the agent subtracts the requested degrees of rotation from its bearing. The modulus operator is applied—the difference being divided by 360—and then the result (the remainder) is set as its new bearing (`self.setBearing(Math.abs(self.bearing - degrees) % 360)`).

The action `moveInDirection` is a combination of clockwise rotation and moving forward actions. This action allows the agent to make a single move forward in a particular direction.



In the `moveInDirection` action, the direction is passed to the action as an integer (`moveInDirection(int direction)`). The action first calculates the difference between the requested direction and the agent's current bearing (`diff = direction - self.getBearing()`). The agent then performs a clockwise rotation equal to whatever the result is (`self.rotateClockwise(diff)`), and finally it moves one step forward (`self.moveForward()`).

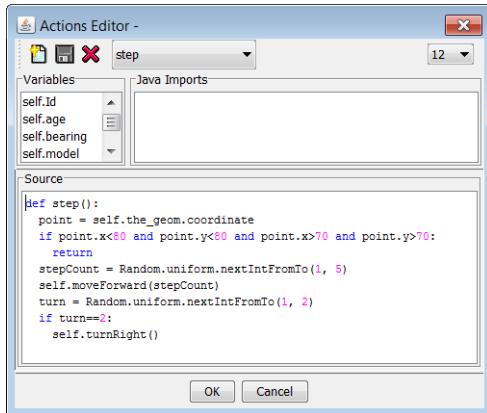
A variation (second version) of the `moveInDirection` action allows the agent to move more than one cell in a time step in a particular direction.



In this code, the direction as well as the desired number of cells to move are passed to the action as two integers (`moveInDirection(int direction, int stepCount)`). The action first calculates the difference between the requested direction and the agent's current bearing (`diff = direction - self.getBearing()`). The agent performs a clockwise rotation equal to whatever the result is (`self.rotateClockwise(diff)`) and then moves forward as many cells as specified by `stepCount` (`self.moveForward(stepCount)`).

Edit the step action to implement the trap scenario

- 14 In the Environment panel, click Agent. In the Property panel, click the Edit button to the right of the Actions property to open the Actions Editor.
- 15 In the Actions Editor, select step from the Actions drop-down list and uncomment the code as shown in the following figure.



In this code, first the agent's current position is identified and stored in the point variable (`point = self.the_geom.coordinate`). If the agent's current position is within the trap area, which is defined as a rectangle with the lower-left coordinate of (70, 70) and the upper-right coordinate of (80, 80), it does nothing (`if point.x<80 and point.y<80 and point.x>70 and point.y>70: return`). (Note that the lower-left and upper-right coordinates of the trap are in map units not row/column space.) Otherwise, it generates a random integer from 1 to 5 to define the number of cells to move (`stepCount = Random.uniform.nextIntFromTo(1, 5)`) and then moves forward that many cells (`self.moveForward(stepCount)`). Another random number is defined, either 1 or 2 (`turn = Random.uniform.nextIntFromTo(1, 2)`). If the random number is equal to 2, the agent turns right (`if turn==2: self.turnRight()`).

So far, you have developed various movements for agents. To observe the agents' movements in ArcMap, you first need to store the changes in their locations and their bearings in the shapefile. To do so, you can use the `writeAgents` action, which has already been defined for you. Then you need to update the display with the latest locations of the agents by using the `updateDisplay` action, which has also been defined for you.

Uncomment the writeAgents and updateDisplay actions (which are the same as in exercise 11a)

- 16** In the Environment panel, click GIS Model. In the Property panel, click the Edit button to the right of the Actions property to open the Actions Editor.
- 17** In the Actions Editor, select the writeAgents action and then the updateDisplay action from the Actions drop-down list and uncomment the code.
- 18** Click OK to save your changes and close the Actions Editor.
- 19** Run the model.

You will notice the agents moving around the raster surface randomly, but they stay within the study area extent. Since the agents' movement is random, and there is equal chance for the agent to move in any given direction, you will observe that the agents tend not to move very far or seem to be moving in circles around their starting points. Because the area of the trap is small compared with the size of the raster, and considering where the cougars begin, it may take some time before any of them fall into the trap. But when one does, you will see that that agent stops moving.

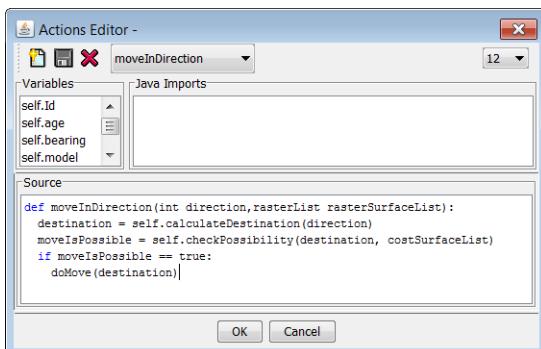
- 20** Close Agent Analyst and ArcMap without saving any changes.

Complex raster surfaces

Our surfaces thus far have been abstract raster surfaces. Now it is time to consider more realistic environments. In raster representation, a surface is made up of continuous square-shaped cells, with each cell neighboring eight others. Each agent can be in only one cell at any given moment. We use the notions of “cost surface” and “local opportunity surface” as a possible approach when modeling with raster data, as follows.

- **Cost surface.** In the general case, the raster surface on which the agent moves might present different *constraints* to the agent’s movement. For instance, it can have slope, a barrier like a wall or a river, or (in an urban environment) legal constraints such as restricted areas. We capture all such constraints in the notion of cost surface. In the following discussion, the phrase “cost surface” is used in a generic sense to mean a raster with cell values; each cell value represents the value of some environmental constraint at that cell location. In case there is more than one cost surface involved, the agent would have to take all of them into account in order to make the move.
- **Local opportunity surface.** Each agent has particular capabilities to deal with environmental constraints—for example, their speed, their energy, their determination, and so on. The overall effect of these capabilities gives rise to what we call the “local opportunity surface” (LOS) for the agent. In the simplest case, the extent of LOS is determined by the MaxSpeed property of the agent.

With these concepts, you can now have the agent move in more realistic environments using more complex methods. You need to add an extra parameter to the previous methods to incorporate the cost surface. The new moveInDirection action would be similar to the following pseudocode. (Not all the actions are defined here.)



This method takes two parameters (each with the given data type):

- A direction (east, west, northeast, and so on)
- A list of cost surfaces (if there are no constraints, this list will be empty)

Note that here the action has been implemented at a higher level. This method first calls calculateDestination to find out the destination of the movement. After calculating the destination, the checkPossibility action is called to check whether it is possible for the agent to move to the destination. To perform this check, the destination and costSurfaceList are sent to this action. If the result of this method is true, then the movement is possible and the agent performs a move to the destination. Obviously, checking the possibility of movement is application-specific; therefore, we didn't specify special implementation for this action.

It is worth mentioning that you can improve the moveInDirection method in different ways. For instance, the checkPossibility method mentioned earlier returns just a Boolean value, either true or false. Therefore, the requested move is considered either possible or impossible. It might be more realistic if you replace this method with one that not only evaluates the possibility of the requested move, but also helps to find other candidate moves if the desired move cannot be performed.

You can alter the moveInDirection action to realize these concepts in the following manner. First, before making the move, the agent needs to evaluate the neighboring cells in terms of their cost. For instance, the agent may want to move east, but the cell immediately to the east has a very steep slope or a wall, in which case the agent has to find a less costly way of moving east—for example, by “moving around” the wall or by going north and then east. The agent will need to negotiate this constraint. You can create a helper action (called evaluateDirection, for example) to assist the agent to do so. The moveInDirection action might first call this action. To weight the surrounding cells, the evaluateDirection action may call the weights action, which weights the surrounding cells in terms of their cost. To make the behavior of the agent more realistic, some degree of indeterminacy (uncertainty) should be built into its behavior. Commonly, modelers do this by introducing some randomness into the behavior. In this case, rather than the agent always moving to the next least costly cell, it might look at the three least costly cells and makes a random choice among them. One such approach was demonstrated in chapters 5 and 8.

This chapter introduced you to the basic concepts of movement. Even though movement may sometimes appear to be complex and application-specific, at its essence movement is created from simple building blocks.

GIS data, fields, and actions dictionaries

Table 11.1 Data dictionary of GIS datasets

| Dataset | Data type | Description |
|---------|-----------|-------------------------|
| Badger | raster | Distribution of badgers |

Table 11.2 Fields dictionary for the models

| Dataset | Data type | Description |
|-------------|-----------|---|
| cougars | shapefile | Locations of the cougars (exercises 11a, b, c, and d) |
| map | shapefile | Census blocks (exercises 11e, f, and g) |
| rasterconst | raster | A constant raster surface (exercise 11h) |

Table 11.3 Actions dictionary for the models

| Declared actions | Description |
|------------------|--|
| GIS model | |
| loadRaster | Loads the raster file |
| initAgents | Initializes the simulation by loading the raster (via the loadRaster action) and setting the bearing property of the agents to 0 |
| writeAgents | Writes the latest changes of the agent to the cougars shapefile |
| updateDisplay | Reflects the latest changes in position of the agents to the screen |
| Agent | |
| step | In each tick, each agent runs its step action to move. The step action in turn calls other move and turn actions. |
| moveToNorth | Moves the agent to the north |
| moveToSouth | Moves the agent to the south |
| moveToEast | Moves the agent to the east |
| moveToWest | Moves the agent to the west |
| moveForward | Causes the agent to move forward, based on the value of the bearing field. For instance, if the agent is facing east (bearing = 90), it will move the agent to the east. |

Table 11.3 Actions dictionary for the models (*continued*)

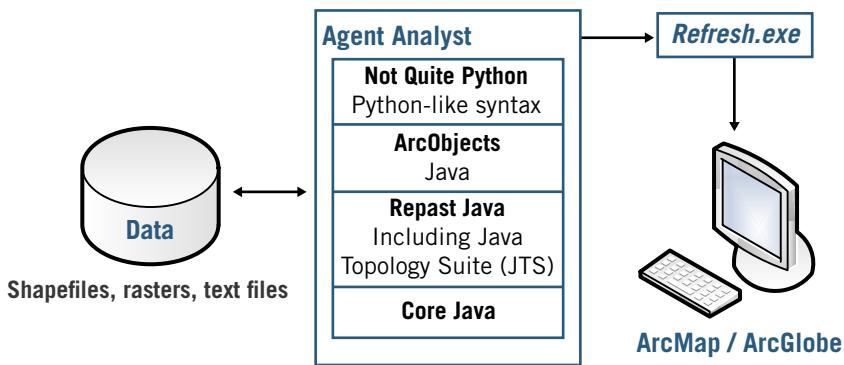
| Declared actions | Description |
|----------------------------|---|
| moveForward(int) | Similar to moveForward, except that it takes the number of steps as an argument. For instance, if 3 is passed as the number of steps, the agent will move forward three times. |
| moveBackward | Causes the agent to move backward, based on the value of the bearing field. For instance, if the agent is facing east (bearing = 90), it will move the agent to the west. |
| moveBackward(int) | Similar to moveBackward, except that it takes the number of steps as an argument. For instance, if 5 is passed as the number of steps, the agent will move backward five times. |
| turnRight | Causes the agent to turn right |
| turnLeft | Causes the agent to turn left |
| turnAroundClockwise | Turns around the agent clockwise (180 degrees) |
| turnAroundCounterClockwise | Turns around the agent counterclockwise (180 degrees) |
| rotateClockwise | Rotates the agent clockwise to whatever degrees is passed to the action as an argument |
| rotateCounterClockwise | Rotates the agent counterclockwise to whatever degrees is passed to the action as an argument |
| moveInDirection(int) | Moves the agent forward in a certain direction (angle) specified by the integer argument |
| moveInDirection(int, int) | Similar to moveInDirection(int), except for an additional argument that is the number of steps the agent is supposed to move in that direction |

Appendix

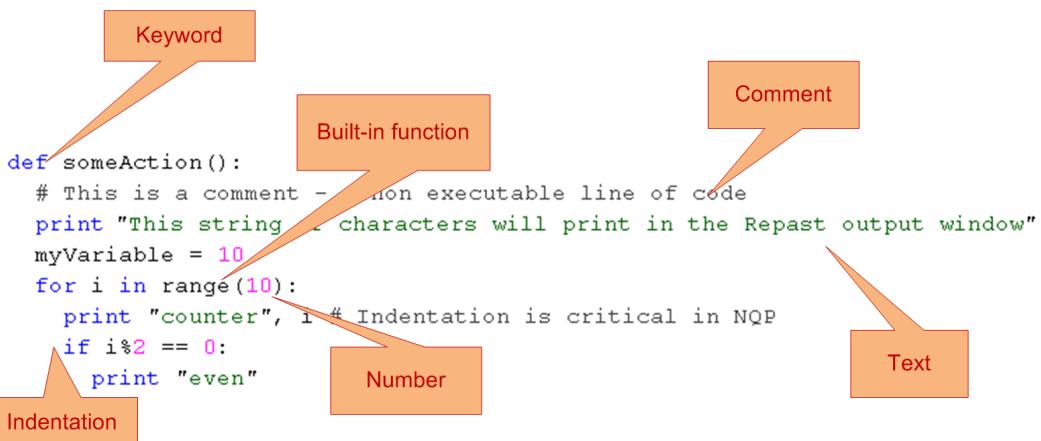
Not Quite Python

by Arika Ligmann-Zielinska

This appendix provides a quick overview of Not Quite Python (NQPy) and its relation to Java. NQPy is an object-oriented programming language that utilizes Python-flavored syntax to access the Repast Java simulation toolkit (<http://repast.sourceforge.net/>). It has the look and feel of Python and allows you to access a full suite of tools useful for spatial agent-based modeling: reading and writing shapefiles and Esri raster grids, full access to Repast Java packages (<http://repast.sourceforge.net/api/index.html>), access to ArcObjects, and access to standard Java libraries (see later in this appendix). But NQPy is not the Python programming language. Consequently, you cannot access Python modules; use Python lists, dictionaries, tuples, or built-in functions; or define custom classes using Python syntax. However, as described later, NQPy implements many of these functionalities directly or allows for a seamless use of Java equivalents instead.



NQPy borrows Python syntax to provide a relatively easy way to specify agent behavior. This appendix is a brief reference of the basic NQPy commands. The colors used for NQPy syntax in the Agent Analyst Actions Editor are shown in the following figure.



Keywords

The following is a list of reserved words (keywords) used in NQPy. These words cannot be used as names for functions or variables.

```
and      break     continue
def      elif      else
for      if        import
in       not       or
print    return    while
```

Printing

To print text to the RePast Output window (a command line console) you use a print statement as follows:

```
print "This is a simple way to print to the RePast Output console"
print "There are",10,"agents in a list"
print "you can use:\nspecial\t\tcharacters"
```

Note that no conversion is necessary to print a combination of text and numbers. Use a comma to separate the items. The comma adds a space before and after the item.

A

Comments

A comment is a line prefixed by #. Any line that is prefixed using # is ignored by the compiler.

Indentation

To delimit any self-contained block of code, you need to use whitespace indentation. Consider the following code:

```
rows = 3
cols = 2
for i in range(rows):
    print "inside the i =",i,"block"
    for j in range(cols):
        print "cell coordinates:",i,j
print "end of all i"
```

It results in the following output:

```
inside the i = 0 block
cell coordinates: 0 0
cell coordinates: 0 1
inside the i = 1 block
cell coordinates: 1 0
cell coordinates: 1 1
inside the i = 2 block
cell coordinates: 2 0
cell coordinates: 2 1
end of all i
```

Variables

A variable is a container that refers to a value stored somewhere in memory. In NQPy, variables are not explicitly declared. In other words, you do not have to assign a type to a newly created variable, which is different from the underlying Java. Note, however, that once an assignment is made, you cannot dynamically change the type, which is different from Python, which allows you to dynamically change the type of data the variable points to. For example:

```
i_var = 2      # integer
d_var = 4.0    # double (floating point number)
d_var = i_var * 2
```

will generate a compilation error because `i_var` points to an integer, and `d_var` points to a double.

Notice that variables are case sensitive. For example:

```
features = "Freeways.shp"
print Features
```

will result in an error because `features` and `Features` are two different variable names.

Primitive types

NQPy supports three primitive types: integer (int), double (floating point number), and boolean (true/false value):

```
i_var = 1      # integer
d_var = 1.0    # double (floating point number)
b_var = true   # boolean
```

Strings

A string is a sequence of characters representing text. String values are surrounded by single (‘ ’) or double (“ ”) quotation marks. For example:

```
agent1 = "resident"
agent2 = 'developer'
agent3 = "developer"
print "agents:", agent1, agent2, agent3
```

Note that strings in NQPy are Java strings, and hence you can call Java string methods on them.

```
print "agent1 name length:", agent1.length()
print agent3.toUpperCase()
filename = "neighbors.txt"
if filename.endsWith(".txt"):
    print "It is a text file"
```

Strings can be combined (concatenated) and sliced (divided into substrings). For example:

```
name = "datafile"
extension = ".dat"
print name[:4], name[4:6], name[4:], name+extension
```

results in:

```
data fi file datafile.dat
```

Every character in a string has an index (position) starting from zero. For example, “data” has four characters, where the index of “d” is 0, the index of “t” is 2, and so on. Note that a substring [m:n] will end with a character at the position m-1 of the input string.

You can convert to and from strings:

```
int_var = 100
str_var = str(int_var)
str_int = "17"
int_var = int(str_int)
str_dbl = "17.17"
dbl_var = float(str_dbl)
```

A

Java conversion is also valid:

```
str1_var=Integer.toString(int_var)
str2_var=Double.toString(dbl_var)
```

Note that, in this case, you need to use the Java class equivalents of the primitive types—Integer for int and Double for double. Both of these classes come with a variety of attributes and methods. For example, Integer.MAX_VALUE and Double.POSITIVE_INFINITY allow you to initialize a number (integer and double, respectively) using a very large value.

You can learn more about Java strings at <http://download.oracle.com/javase/6/docs/api/java/lang/String.html>.

Flow control

Conditional statements in NQPy use the following convention:

```
if condition:  
    statement_block  
elif condition:  
    statement_block  
else:  
    statement_block
```

The condition is some expression that evaluates to true or false. The statement block is a collection of one or more NQPy instructions that are executed if the condition is true. The colon indicates the beginning of the conditional block of statements. The level of indentation indicates which statement is part of the block. Notice that 0 is equivalent to false, and any other number is true. For example, in the following code:

```
a=4  
if a:  
    print a  
    a = a + 1  
b = 3
```

“print a” and “ $a = a + 1$ ” will be executed since “a” is true, and “ $b = 3$ ” will be executed regardless of the value of a. However:

```
a=0  
if a:  
    print a  
    a = a + 1  
b = 3
```

will result in no action because “ $a=0$ ”, which defaults to false.

Logical operators and Boolean expressions

NQPy uses the standard comparison operators, as shown in the following example:

```
x == y # x is equal to y
x != y # x is not equal to y
x > y # x is greater than y
x < y # x is less than y
x >= y # x is greater than or equal to y
x <= y # x is less than or equal to y
```

Moreover, there are three logical operators: and, or, and not. The semantics of these operators is similar to their meaning in English.

```
if agent2 == "governor" and agent2 == "governor":
    print "two agents that are alike"
if not agent1 == agent2:
    print "two different agents"
if i_var == 4 or d_var == 4:
    print "i_var or d_var equals 4"
```

A

Loops

Loops are used for repetitive tasks. Two types of loops are available: for-loop and while-loop. A for-loop iterates over each value in a list. The loop will execute once for each value in the list. A while-loop is repeated as long as its condition is true. Iterating through loops is done as follows:

```
for x in list:
    statements

while true_condition:
    statements
```

For example:

```
i = 0
while i < 5:
    print "ABM"
    i = i + 1
# or
for i in range(5):
    print "ABM"
```

The keywords “continue” and “break” are also supported. For example:

```
for i in range(20):
    if i%2:
        continue
    print "i =",i
    if i == 10:
        break
```

will print even values from 0 to 10 and then break out of the loop.

Object orientation

Like Java and Python, NQPy is based on the object-oriented programming technology, meaning that most of the data structures used in NQPy are put into self-contained smaller programs called objects. These objects symbolize real-world entities (like generic agents, parcel lots, census tracts, and residents) or abstract concepts (like attitudes, beliefs, and rules of behavior). Objects have a body that is built from data called attributes and may perform actions through functions referred to as methods. Programs are coded by using objects as building blocks and defining relationships between them. To define an object, you create its template, called a class. You can think of a class as a user-defined compound type, which is more complex than the primitives (e.g., integers, doubles, or booleans). After specifying the class, you can create a number of instances from it, which are the objects themselves.

Objects have attributes (also called fields or instance variables) and methods (also called actions or functions). Attributes may be some predefined parameters (e.g., a data source for a vector agent or a model for a generic agent), Java parent fields, or user-defined fields. In NQPy, methods can be actions defined by the user, functions inherited from Java (e.g., String object methods), or functions generic to NQPy (e.g., the step action). Attributes and methods are referenced using the dot (.) notation, where the object reference is on the left side of the dot and the method or the field is written on the right side of the dot:

```
objectName.fieldName
objectName.actionName()
```

The object reference is mandatory, even when the action or the field is defined directly in a given component. In this case, you need to use `self` as the object reference. For example, if you define a “home” field as part of a resident vector agent and then you want to call this field within any of the resident agent actions, you need to use `self.home` to refer to this field. If you want to refer to the same field outside the agent component, you need to define the home field as being accessible during field definition.

You can retrieve or modify the value of a field directly, or you can use getters and setters, respectively. Hence, both of the following are valid pieces of NQPy code:

```
aHome = self.getHome() is equivalent to aHome = self.home
```

```
self.setHome(newHome) is equivalent to self.home = newHome
```

Note that getters and setters are functions and, as a consequence, require parentheses.

You can convert between different object types. This procedure is called casting. For instance, if you want to retrieve a value from a list of agent names and convert it to a string variable called agentLabel, you need to use the following code:

```
agentLabel = (String)agentNames.get(5)
```

Importing

Core Java classes (i.e., `java.lang`) and most Repast classes (e.g., `uchicago/src/sim/util/Random`) do not have to be imported. Other Java libraries (e.g., `java.io`) must be explicitly imported. If uncertain, you should include the full path to your class in the Import pane in the Actions Editor. Paths to any external user-defined Java classes or .jar files that you want to include in your project can be specified using a semicolon-separated list in the Class Path property of the Environment component. They will be included as imports during model compilation. For example, if you have complex behavior classes located in `behavior.jar` on your C drive and you want to use them in your model, you can include `c:\behavior.jar` in the Class Path property.

A

Lists

A list is a changeable ordered collection of values. Every item in the list has a well-defined position referred to as its index. The indexing in a list starts from zero. For example:

```
iterator = range(10)
print iterator
```

will print values from 0 to 9. Note the built-in `range()` function. It has a form of

```
range(optional from_value, required to_value, optional step_value)
```

For example, `range(2,8,2)` will contain the values 2, 4, and 6. To create a list of more complex types, you use the Java `ArrayList` class (<http://download.oracle.com/javase/6/docs/api/java/util/ArrayList.html>). For example:

```
alist = ArrayList()
agent = "resident"
for i in range(6):
    alist.add(agent+str(i+1))
```

results in (“resident1”, “resident2”, ..., “resident6”).

`ArrayList` comes with a few useful methods, as demonstrated in this code:

```
if alist.contains("resident1"): # checks for membership
    print "Agent 1 is present"
print alist.size() # prints the number of
# elements in the list
alist.set(1,"farmer") # assigns "farmer" to position 1
alist.remove("resident3") # deletes "resident3" from the list
alist.add("developer") # adds "developer" to the
# end of the list

print "current list:", alist
alist.clear() # deletes all items in the list
if alist.isEmpty(): # checks if there are
# any values in the list
    print "no more agents"
```

You can easily retrieve an element from a list. For example, the statements

```
print alist.get(1)
# or using Python-like indexing:
print alist[1],"is located at position 1"
```

will retrieve the second value in the list. You can also get part of a list (i.e., slice it; see the section “Strings” earlier for details). For example, the following code finds the middle position in a list and prints the second half of it:

```
middle = alist.size() / 2
print "Half the list:",alist[:middle]
```

You can iterate over lists as shown here:

```
alist = ArrayList()
stringAgent = "resident"
for i in range(6):
    alist.add(stringAgent+str(i+1))
for strAgent in alist: # simple iteration over list items
    print strAgent
```

This code prints:

```
resident1
resident2
resident3
resident4
resident5
resident6
```

However, now consider the following code:

```
for agent_name as String in alist:
    print "name length:",agent_name.length()
```

Such in-loop embedded conversion (casting) is necessary in NQPy. Without it, the compiler treats each element in the list as the most general Java Object and reports an error because it cannot find the specified method, like length() in this example. Consequently, if you want to apply properties and methods (fields and actions in Agent Analyst) defined for a particular class, you need to cast to that class within the for-loop.

ArrayList cannot contain primitive types like integers or doubles. Therefore, if you want to create a list of doubles, for example, you need to cast the double value to the Double object:

```
d1 = 12.0
d2 = 102.4
d3 = 17.1
dlist = ArrayList()
dlist.add(Double(d1))
dlist.add(Double(d2))
dlist.add(Double(d3))
print dlist
```

A

Hash maps

A hash map, called a dictionary in Python, is a Java mapping data type in which a unique key (identifier) is associated with some value:

```
amap = HashMap()
```

To add values to the initiated hash map, you need to use the put method:

```
for i in range(10):
    amap.put(Integer(i),"agent"+str(i)) # parameters: (key, value)
print amap
```

Like lists, hash maps have a variety of useful methods:

```

amap.isEmpty()                                # returns true if the
                                              # hash map is empty
print amap.size()                            # retrieves the number
                                              # of items in the map
key=Integer(5)
if amap.containsKey(key):                   # checks for a key
                                              # in the hash map
    print "for key = 5", amap.get(key)
val = "agent5"
if amap.containsValue(val):                 # checks for a value
                                              # in the hash map
    print "all values", amap.values()      # retrieves all values
                                              # in the hash map
    print "all keys", amap.keySet()        # retrieves all keys
                                              # in the hash map
amap.remove(key)                            # deletes an item from the
                                              # map using its identifier
print "a map now:", amap
amap.clear()                                # deletes all items
                                              # in the hash map
print "a map now:", amap

```

Functions

A function (an action) is a self-contained and labeled sequence of NQPy statements that performs specific operations. These operations are put into a function definition:

```

def function_name(parameters):
    statements
    (return some_value)

```

Each time a function is invoked with its name, the statements specified in its body are executed, and, if needed, a value is returned. The parameters and the return statement are optional. Thus, in its simplest form, a function can be defined as follows:

```

def printName():
    print "Agent-Based Model"

```

You invoke this function by calling its name, `printName()`. Since a function is always defined as part of a class, it needs to be called using an instance of that class. In particular, if `printName()` is an action of `someModel`, and you invoke it in another action of `someModel`, you need to use the following syntax:

```
self.printName()
```

If you want to pass a value to a function, you need to specify a parameter type and a parameter name for that value:

```
def printName(String aName): # type: String, name: aName
    print aName, "Agent-Based Model"
```

which is called as follows:

```
self.printName("Residential")
```

To return a value, you need to declare the type of the object to be returned, as follows:

```
def Double demoFunc(double x):
    # demonstrates the use of value return
    print "input value", x
    y = x * 1.14
    print "new value", y
    return Double(y)
```

When invoked, this function returns a modified floating point value, which needs to be further assigned to some variable, as shown in the following:

```
aValue = 2.3
newval = self.demoFunc(aValue)
print "returned value", newval
```

A

Default functions

NQPy comes with a few generic functions for handling simulation-specific tasks. These functions are described in the following list.

- pause is an action of the model component that pauses the simulation from within the code rather than interactively through the Repast toolbar.
- stop is an action of the model component that stops the simulation from within the code rather than interactively through the Repast toolbar.
- getTickCount is an action of the model component that returns the current tick using a double number.
- initAgents is a model-level action executed at the beginning of the simulation as part of the model setup.

- writeAgents is a model-level action that writes the vector agents to a specified shapefile, updating the features and attributes in that shapefile with current values from Agent Analyst. The default code reads:

```
self.writeAgents(vectorAgentGroup_name, path_to_shapefile)
```

- updateDisplay is a model-level function that refreshes the ArcGIS map view by invoking the Refresh.exe application. Note that you must call the writeAgents action prior to updateDisplay to render the changes made by the model.
- step is an action template (an action with an empty body) predefined for generic as well as vector agents. Any code that you put inside this method will be, by default, executed for every tick.

Generic agent

A generic agent is a type of a non-GIS agent—for example, a pedestrian walking in a city or a resident moving from one building to another. The following code initializes such agents from the model component level:

```
agent = SomeGenericAgent() # a class name of your generic
                           # agents, e.g. Pedestrian()
agent.setModel(self)
```

Math

Math functionality is often utilized in modeling. The `java.lang.Math` class handles the basic math operations and constants. There is no need to import this class:

```
print "PI & E", Math.PI, Math.E
x = -2.66
y = 2
print Math.abs(x), Math.ceil(x), Math.floor(x), Math.max(x,y)
print Math.pow(y,3),"\t",Math.round(x)
print Math.sqrt(4)
print "simple random",Math.random()
```

Other math functions are described at <http://download.oracle.com/javase/6/docs/api/java/lang/Math.html>.

Random

Repast is equipped with a custom Random class designed to facilitate stochastic simulations. It provides a wide variety of distributions, functions for handling random seeds, and other special functions like shuffle(). For example:

```
Random.createUniform() # The uniform distribution is initialized
# and can be used:
# print a few doubles
print "doubles:"
for i in range(4):
    print Random.uniform.nextDoubleFromTo(0,15)
# print a few integers
print "integers:"
for i in range(4):
    print Random.uniform.nextIntFromTo(0,15)
# now create a normal distribution (mean, std)
Random.createNormal(3.2, 1.2)
# generate a few numbers
for i in range(7):
    print "drawn from normal:", Random.normal.nextDouble()
```

For other distributions go to <http://repast.sourceforge.net/api/index.html>, select the uchicago.src.sim.util package, and then select the Random class.

A

You can fix the random seed, which results in “freezing” the stochastic component of your model. With a fixed seed, the model will generate the same result each time it is run. Setting the seed destroys previous distributions, so you must initialize them again:

```
Random.setSeed(1000)
Random.createUniform()
# print a few numbers
# note: each time you execute the following code,
# the same numbers are obtained
for i in range(7):
    print "seed fixed:", Random.uniform.nextIntFromTo(0,100)
```

The shuffle function randomizes the order of your list elements:

```
agents = ArrayList()
# populate the list, note that you can use
# semicolon to place multiple lines of code into one line:
agents.add("resident"); agents.add("farmer");
agents.add("developer"); agents.add("planner")
print "agents:",agents
for i in range(4):
    SimUtilities.shuffle(agents)
    print i, agents
```

For more information on using random numbers in your models consult http://repast.sourceforge.net/repast_3/how-to/random.html.

Java Topology Suite

Agent Analyst uses the Java Topology Suite (JTS) for performing spatial operations such as feature creation, comparison, and manipulation. JTS is a Java-based application programming interface (API) that implements the OpenGIS Simple Features Specification. Every vector agent in Agent Analyst is equipped with a field named *the_geom*, which is equivalent to the Shape field found in Esri products. This field is an entry point to various GIS-related operations like buffering and calculating centroids. The website <http://www.vividsolutions.com/jts> provides full documentation on spatial algorithms available through JTS. A few are presented in the following discussion.

Assume you have a polygon vector agent called aPolygon. First, you need to retrieve its geometry:

```
poly = aPolygon.the_geom
```

You can now use poly to perform a variety of calculations:

```
bounding_box = poly.getEnvelope()
area = poly.getArea()
perimeter = poly.getLength()
centroid = poly.getCentroid()
print "polygon centroid", centroid.X, centroid.Y
```

To calculate a buffer around this center point, you would use code such as this:

```
centerBuffer = centroid.buffer(30.0)
print "Buffer created."
```

A variety of spatial analysis operations are supported. For example, the point-in-polygon test can be performed as follows:

```
if pointAgent.the_geom.within(poly):
    print "point agent is within the polygon"
```

Distance between points can also be calculated:

```
print "distance from centroid", pointAgent.the_geom.distance(centroid)
```

Another example involves checking whether otherPolygon is within a given distance of poly:

```
distance = 55
if otherPolygon.the_geom.isWithinDistance(poly, distance):
    print "polygons are within a distance of 55 feet"
```

Sometimes it is useful to build a temporary geometry (point, polygon, ring, line, and so on) that is later used in spatial operations. One way to do that is to use the GeometryBuilder java class. Then you create an empty factory for geometry building:

```
factory = GeometryFactory()
builder = GeometryBuilder(factory)
```

You build a new geometry by adding the coordinates to your builder:

```
builder.addCoordinate(x1_coord, y1_coord)
builder.addCoordinate(x2_coord, y2_coord)
builder.addCoordinate(x3_coord, y3_coord)
```

With the list of coordinates in the builder, different geometries can be built:

```
aRing = builder.createLinearRing()
aPolygon = builder.createPolygon()
```

A

You can clear the current list of coordinates. New coordinates will have to be added before any geometries can be created:

```
builder.clear()
builder.addCoordinate(167.604, 28.368)
aPoint = builder.createPoint() # a point is created from the last
                             # added coordinate
```

Handling rasters

Grids must be explicitly loaded into the Agent Analyst model:

```
# step [1] create a raster object
rasterObject = ➔
    ➔ ESRIRaster(path_to_dir_containing_the_raster, raster_name)
# step [2] load the raster and give it a local label
# i.e. the name used within the model
self.addRaster("araster", rasterObject)
# note that the raster is loaded on the model level
```

After loading, you can access some of the raster properties as follows:

```
someRaster = self.getRaster("araster")
bandIndex = 0
print "raster properties"
print "rows:", someRaster.getRows(bandIndex)
print "cols:", someRaster.getColumns(bandIndex)
print "cell size:", someRaster.getMeanCellWidth(bandIndex)
print "cell size:", someRaster.getMeanCellHeight(bandIndex)
```

Agent Analyst comes with two methods for handling positions on a raster. The first one is based on the projection of the map—you simply retrieve the position using map coordinates. The second method uses the grid coordinate system of rows and columns. To convert the specified map projection coordinate (e.g., from a vector point called `thisPoint`) to a pixel grid coordinate for a specified raster band, you use the following notation:

```
pointObject = thisPoint.the_geom.coordinate
bandIndex = 0
pointPixel = someRaster.mapToPixel(pointObject.x, pointObject.y, ➔
    ➔ bandIndex)
```

The `mapToPixel` method returns the grid location information from a raster by passing the geographic coordinates. The coordinates can then be retrieved as `pointPixel.x` and `pointPixel.y`. The converse (pixel coordinates to map projection coordinates) is also possible:

```
pixelPoint = someRaster.pixelToMap(gridX, gridY, bandIndex)
```

You can read the value of a selected raster cell:

```
val = someRaster.getPixelValue(col_no, row_no, bandIndex)
```

You can also read a value at a location in map units (rather than as columns and rows):

```
val = someRaster.getPixelValueAtMap(mapX, mapY, bandIndex)
```

Thus, if you want to read the value of the underlying raster for a given `pointVectorAgent`, use this code:

```
pointGeom = pointVectorAgent.the_geom.coordinate
x = pointGeom.x
y = pointGeom.y
val = someRaster.getPixelValueAtMap(x, y, 0)
print "point raster value", val
```

To set a pixel in a specified band to a new value, you use the following syntax:

```
someRaster.writePixelValue(aValue, xGrid, yGrid, bandIndex)
```

Note that the raster is not automatically refreshed in ArcMap—it needs to be reloaded.

About the contributors

Kevin Johnston has been in software development at Esri (the developers of ArcGIS software) for the last 20 years focusing on the ArcGIS Spatial Analyst extension, the ArcGIS Geostatistical Analyst extension, and dynamic and statistical modeling. Kevin has an MLA in landscape architecture from Harvard University and a PhD from Yale University focusing on ecological modeling. Kevin has written a series of book chapters and journal articles on spatial modeling, in particular on wildlife distribution and movement. He has developed several models exploring the effects of climate change on species distribution. In addition to working at Esri, as a volunteer Kevin has been working on elephant-movement models for Amboseli in Kenya, snow leopard corridor models in Nepal, and agent-based models for cougar movement in Flagstaff, Arizona.

Daniel G. Brown (PhD in geography, 1992, University of North Carolina at Chapel Hill) is a professor in the School of Natural Resources and Environment at the University of Michigan. His work, published in more than 100 refereed articles, chapters, and proceedings papers, has aimed at understanding human-environment interactions through a focus on land-use and land-cover changes, through modeling these changes, and through spatial analysis and remote sensing methods for characterizing landscape patterns. Recent work has used spatial simulation, including agent-based models, to understand and forecast landscape changes. He has chaired the Land Use Steering Group, under the auspices of the U.S. Climate Change Science Program, and served as a member of the NASA Land Cover and Land Use Change Science Team, on a variety of panels for the National Research Council, NASA, the National Science Foundation, and the European Research Council, and on the editorial boards for *Landscape Ecology*, *Computers, Environment and Urban Systems*, and the *Journal of Land Use Science*. In 2009 he was elected fellow of the American Association for the Advancement of Science.

Nicholson Collier is a software engineer at the Center for Complex Adaptive Systems within the Decision and Information Sciences Division of Argonne National Laboratory. He has more than 12 years of experience designing and implementing simulation and visualization frameworks and applications for government agencies, private industry, and academia. Prior to working for Argonne, he worked for the Social Science Research Computing group at the University of Chicago and at PantaRei Corporation. He holds a PhD in the philosophy of religions from the University of Chicago.

Hamid Ekbia, PhD, is associate professor of information science and cognitive science, and the director of the Center for Research on Mediated Interaction at the School of Library and Information Science, Indiana University, Bloomington, where he teaches human-computer interaction (HCI), agent-based modeling, and geographic information systems. His research is focused on how technologies mediate interactions among people, organizations, and communities. He currently works in the area of patient-centered healthcare, organizational workflow and modeling, and games for health. Ekbia has written about technology and mediation in network organizations, design and research communities, and local populations. His recent book,

Artificial Dreams: The Quest for Non-Biological Intelligence (Cambridge University Press, 2008), is a critical-technical analysis of artificial intelligence.

Jo Fraley (BS in mathematics, 1991, East Tennessee State University) has spent the last 18 years applying geographic information systems to problems in a variety of industries and agencies. After beginning her career at the Tennessee Valley Authority, Jo joined Esri in 1994 as a technical marketing representative. Jo specializes in ArcGIS Server but supports the complete ArcGIS software system for the Esri technology center in Washington, DC. She currently spends most of her time building prototypes and demonstrations to show how ArcGIS software can be used to solve problems for the U.S. government.

Michelle A. Gudorf (BA in biology and geography, 1996, University of Colorado) has worked with the National Park Service GIS program modeling habitat suitability for bighorn sheep restoration west of the Rocky Mountains. In addition, she has completed other National Park Service GIS natural and cultural resource projects and provided metadata instruction. She has served as president of the Society of Conservation GIS and for the Vermont Association of Conservation Districts. Currently she is a consultant working with the U.S. Bureau of Land Management on modeling the effects of climate change on species habitat.

Elizabeth Groff (PhD in geography, 2006, University of Maryland) is an associate professor in the criminal justice department at Temple University. She is an applied researcher who has held positions in practice, policy, and academia. Her research interests include place-based criminology; modeling geographical influences on human activity; the role of technology in police organizations; and the development of innovative methodologies using geographic information systems, agent-based simulation models, and randomized experiments. She has published 13 articles and five book chapters and coauthored two books. She was elected a fellow of the Academy of Experimental Criminology in 2010.

Naicong Li is a senior research scientist and GIS analyst at Redlands Institute, University of Redlands, where she leads the research effort in spatial decision support (SDS) and helped establish the SDS Consortium. She is an expert in spatial modeling in various application domains, including endangered species recovery, nonnative invasive species management, and intrinsic landscape aesthetic resources evaluation. She was part of the team at Esri responsible for developing the ArcSketch add-on to ArcGIS software, which is now integrated into the functionalities of ArcGIS. Prior to entering the field of GIS, she worked in the field of artificial intelligence and machine translation. She holds a PhD in linguistics and an MS in computer science (University at Buffalo, SUNY). Her current research interests include SDS, urban planning and design, spatial cognition, dynamic modeling, and knowledge management.

Michael J. North, MBA, PhD, is the deputy director of the Center for Complex Adaptive Agent Systems Simulation within the Decision and Information Sciences Division of Argonne National Laboratory and a senior fellow in the joint Computation Institute of the University of Chicago and Argonne. He has more than 15 years of experience developing and applying advanced modeling and simulation applications for the U.S. government, state governments

in the United States, international agencies, private industry, and academia. He is the lead developer and a principal investigator for the widely used free and open source Repast agent-based modeling suite (<http://repast.sourceforge.net>) upon which Agent Analyst is based. He is the lead author of the book *Managing Business Complexity: Discovering Strategic Solutions with Agent-Based Modeling and Simulation* (Oxford University Press, 2007) and has authored or coauthored more than 50 journal articles and conference papers. He holds 10 college degrees, including a PhD in computer science from the Illinois Institute of Technology.

Arika Ligmann-Zielinska (PhD in geography, 2008, San Diego State University and University of California, Santa Barbara) is currently an assistant professor in the Department of Geography and Environmental Science and Policy Program at Michigan State University. Her research activities encompass a broad range of modeling approaches that capture the dynamic relationships within coupled human and natural systems. To date her core research has focused on spatiotemporal sensitivity analysis of model output, which allows for decomposing outcome variability and distributing it among the uncertain model inputs such as land characteristics, human behavior, and feedbacks. As part of her graduate teaching agenda, she has developed a graduate-level seminar entitled “Geosimulation,” which educates students in modeling human decision making and its impact on natural environment.

Derek T. Robinson (PhD in School of Natural Resources, University of Michigan, 2009) is an assistant professor of geographical information science at the University of Waterloo, where he performs research on coupled human-environment systems within the context of land-use science and geography. His research typically involves using agent-based models to integrate geographical information systems and ecological and human decision-making models to evaluate how socioeconomic contexts and policy scenarios affect land use, ecological function, and human well-being.

Nathan Strout is the technology manager at the Redlands Institute and an adjunct faculty member in the Master of Science in GIS program at the University of Redlands in California. Mr. Strout has many years of experience applying spatial methods and technologies to a wide array of projects. He manages a team of GIS, programming, and IT staff and acts as the principal investigator for multiple public and privately funded research projects for the University of Redlands. Mr. Strout has also been teaching programming in the Master of Science in GIS (MS GIS) program at the University of Redlands for the past two years.

Data source credits

Chapter 1

Data courtesy of United States Geological Survey

Chapter 2

Data created by the author

Chapter 3

Data courtesy of United States Census Bureau, Geography Division

Chapter 4

Data courtesy of the Michigan Center for Geographic Information

Chapter 5

Data courtesy of United States Geological Survey

Chapter 6

Data courtesy of City of Seattle

Chapter 7

Data courtesy of City of Redlands, California

Chapter 8

Data courtesy of United States Geological Survey

Chapter 9

Data courtesy of City of Seattle

Chapter 10

Data courtesy of Vermont Center for Geographic Information

Rivers—From Esri Data & Maps 2007, courtesy of ArcWorld

Elevation—From Esri Data & Maps 2008, courtesy of USGS EROS Data Center

States—From Esri Data & Maps 2010, courtesy of ArcUSA, US Census, Esri (Pop2010 field)

Movie courtesy of Tony Turner

Movies © Paul M. Torrens, 2010

Chapter 11

Data courtesy of United States Census Bureau, Geography Division

Esri Data License Agreement

Important

Read carefully before installing the package.

ENVIRONMENTAL SYSTEMS RESEARCH INSTITUTE, INC. (ESRI), IS WILLING TO LICENSE THE DATA AND RELATED MATERIALS TO YOU ONLY UPON THE CONDITION THAT YOU ACCEPT ALL THE TERMS AND CONDITIONS CONTAINED IN THIS LICENSE AGREEMENT. PLEASE READ THE TERMS AND CONDITIONS CAREFULLY BEFORE INSTALLING THE PACKAGE. BY INSTALLING THE PACKAGE, YOU ARE INDICATING YOUR ACCEPTANCE OF THE ESRI LICENSE AGREEMENT. IF YOU DO NOT AGREE TO THE TERMS AND CONDITIONS AS STATED, ESRI IS UNWILLING TO LICENSE THE DATA AND RELATED MATERIALS TO YOU. IN SUCH EVENT, YOU SHOULD RETURN THE MEDIA PACKAGE (WITH THE SEAL UNBROKEN) AND ALL OTHER COMPONENTS TO ESRI IF A MEDIA PACKAGE WAS PROVIDED, OR CLICK I DO NOT ACCEPT THE LICENSE AGREEMENT IF ACCESSING THE DATA AND RELATED MATERIALS ONLINE.

ESRI LICENSE AGREEMENT

This is a license agreement, not an agreement for sale, between you (“Licensee”) and **Environmental Systems Research Institute, Inc. (“Esri”)**. This Esri License Agreement (“Agreement”) gives Licensee certain limited rights to use the data and related materials (“Data and Related Materials”). All rights not specifically granted in this Agreement are reserved to Esri and its Licensors.

Reservation of Ownership and Grant of License: Esri and its Licensors retain exclusive rights and title to and ownership of the copy of the Data and Related Materials licensed under this Agreement and, hereby, grant to Licensee a personal, nonexclusive, nontransferable, royalty-free, worldwide license to use Data and Related Materials based on the terms and conditions of this Agreement. Licensee agrees to use reasonable effort to protect Data and Related Materials from unauthorized use, reproduction, distribution, or publication.

Proprietary Rights and Copyright: Licensee acknowledges that Data and Related Materials are proprietary and confidential property of Esri and its Licensors and are protected by United States copyright laws and applicable international copyright treaties and/or conventions.

Permitted Uses: Licensee may install Data and Related Materials onto permanent storage device(s) for Licensee’s own internal use.

Licensee may make only one (1) copy of the original Data and Related Materials for archival purposes during the term of this Agreement unless the right to make additional copies is granted to Licensee in writing by Esri.

Licensee may internally use Data and Related Materials provided by Esri for the stated purpose of GIS training and education.

Uses Not Permitted: Licensee shall not sell, rent, lease, sublicense, lend, assign, time-share, or transfer, in whole or in part, or provide unlicensed third parties with access to Data and Related Materials or portions of Data and Related Materials, any updates, or Licensee's rights under this Agreement.

Licensee shall not remove or obscure any copyright or trademark notices of Esri or its Licensors.

Term and Termination: The license granted to Licensee by this Agreement shall commence upon the acceptance of this Agreement and shall continue until such time that Licensee elects in writing to discontinue use of Data and Related Materials and terminates this Agreement. The Agreement shall automatically terminate without notice if Licensee fails to comply with any provision of this Agreement. Licensee shall then cease accessing, uninstall, remove, and destroy Data and Related Materials and any whole or partial copies, modifications, or merged portions thereof in any form and, if the media package was provided, return such media package to Esri. Upon request, Licensee shall execute and deliver evidence of such actions to Esri. The parties hereby agree that all provisions that operate to protect the rights of Esri and its Licensors shall remain in force should breach occur.

Disclaimer of Warranty: Data and Related Materials contained herein are provided "as is," without warranty of any kind, either express or implied, including, but not limited to, the implied warranties of merchantability, fitness for a particular purpose, or noninfringement. Esri does not warrant that Data and Related Materials will meet Licensee's needs or expectations; that the use of Data and Related Materials will be uninterrupted; or that all nonconformities, defects, or errors can or will be corrected. Esri is not inviting reliance on Data and Related Materials for commercial planning or analysis purposes, and Licensee should always check actual data.

Data Disclaimer: Data used herein has been derived from actual spatial or tabular information. In some cases, Esri has manipulated and applied certain assumptions, analyses, and opinions to Data solely for educational training purposes. Assumptions, analyses, opinions applied, and actual outcomes may vary. Again, Esri is not inviting reliance on this Data, and Licensee should always verify actual Data and exercise its own professional judgment when interpreting any outcomes.

Limitation of Liability: Esri shall not be liable for direct, indirect, special, incidental, or consequential damages related to Licensee's use of Data and Related Materials, even if Esri is advised of the possibility of such damage.

No Implied Waivers: No failure or delay by Esri or its Licensors in enforcing any right or remedy under this Agreement shall be construed as a waiver of any future or other exercise of such right or remedy by Esri or its Licensors.

Order of Precedence: Any conflict between the terms of this Agreement and any FAR, DFAR, purchase order, or other terms shall be resolved in favor of the terms expressed in this Agreement, subject to the government's minimum rights, unless agreed otherwise.

Export Regulation: Licensee acknowledges that this Agreement and the performance thereof are subject to compliance with any and all applicable United States laws, regulations, or orders relating to the export of data. Licensee agrees to comply with all laws, regulations, and orders of the United States with regard to any export of such technical data.

Severability: If any provision(s) of this Agreement shall be held to be invalid, illegal, or unenforceable by a court or other tribunal of competent jurisdiction, the validity, legality, and enforceability of the remaining provisions shall not in any way be affected or impaired thereby.

Governing Law: This Agreement, entered into in the County of San Bernardino, shall be construed and enforced in accordance with and be governed by the laws of the United States of America and the State of California without reference to conflict of laws principles. The parties hereby consent to the personal jurisdiction of the courts of this county and waive their rights to change venue.

Entire Agreement: The parties agree that this Agreement constitutes the sole and entire agreement of the parties as to the matter set forth herein and supersedes any previous agreements, understandings, and arrangements between the parties relating hereto.