Appendix C
Documentation

Table of Contents

1 package circle

This package is both front and back-end of the circle simulation.

1.1 package agents

This package contains internal agents representation for the simulation with an abstract class to be extended.

1.1.1 class AgentInfo

This class is an internal representation of an agent in the simulation. It contains agent's unique id, list of another agents this agent can see and an actual agent.

1.1.2 class AgentDelegate

This interface defines how agents see each other. It contains only one method getFlag that returns a flag raised by the agent in the last round.

1.1.3 class AbstractAgent

AbstractAgent class is a base of each agent. It contains a flag raised by the agent in the previous round and a list of visible agents. It also provides:

- an abstract method getNewFlag(int) that is called by the simulation to ask an agent what flag it wants to raise in the given round
- an implemented getFlag method from AgentDelegate interface that allows the agent to both get the flag it raised in the previous round and the flag raised by other agents in the previous round
- a method setFlag(int) that is called by the simulation at the end of each round after all agents have made their decisions
- a method setVisibleAgents(AgentDelegate[]) that is used for simulation initialisation

1.2 package implementedAgents

This package contains implemented agents with defined behaviours.

1.2.1 class RandomAgent

Implemented agent that raises a random flag from among available flags.

1.2.2 class FirstAgent

Deterministic agent. It assumes the simulation is played with two flags. In the first round it raises random flag. Then it tries to adjust to the largest set of agents in a stable state, for example in the following situation with visibility 3 it will adjust to the green state and hence it will raise a blue flag:

flags: ...RRB**R**RBRB...
states: ...GYY**Y**GGG...

This is a stubborn agent and hence it may not be able to achieve consensus, for example in the following situation with visibility 3 it will always raise a red flag, similarly agent to its right will always raise a blue flag:

flags: ...BRB**R**RBRB...
states: ...YYY**Y**GGG...

1.2.3 class SecondAgent

Deterministic agent. Its behaviour is very similar to FirstAgent's, with the only difference being that it does not consider itself when calculating what state the most agents are in.

1.2.4 class ThirdAgent

Non-deterministic agent. Similarly to SecondAgent, this agent calculates the number of agents that were in both green and yellow state and adjust to the larger of them (if they are equal it makes a random choice). However, given that it has made a choice, it does not raise the chosen flag immediately. This agent has a chance that it will behave reasonable (i.e. that it will raise its chosen flag). The chance is initially set to 100%. If the agent raises different flag that it has raised in the previous round, the chances drops by 10%, similarly if agent raises the same flag the chance raises by 10%. If agent behaves unreasonably and raises the other flag than it choice was, the chance is reset to 100%.

1.2.5 class ConsistentAgent

Non-deterministic agent. It remembers flags raised in the previous round by all agents it sees. Based on this, it calculates agents' consistency (including its own) with the following rules: if agent raises the same flag as in the previous round, the consistency is increased by 1, if it raises different flag, it drops by 1. Maximum is set to 20. The agent calculates what state the most agents are in, not by counting them, but by summing up their consistencies. Finally it uses its own consistency as a chance of reasonable behaviour – so for example with consistency 19 the agent will have 95% chance that it will raise chosen flag.

1.2 package main

This package contains simulation back-end and front-end. It also contains a Tester class that allows to run multiple tests without GUI.

1.2.1 class AgentLabel

Extended JLabel that represents an agent in the GUI.

1.2.2 class AgentMemoryAccessor

Extended MemoryAccessor. It is able to display agent's id.

1.2.3 class CircleSimulation

Circle simulation back-end. The constructor CircleSimulation(int, int, Class) allows to create a simulation with given number of agents, given visibility and given class of agent (must extended circle.agents.AbstractAgent). Provides methods to progress to the next round, get round number, check whether the consensus is achieved, access individual agents or access the history. History can be disabled by changing HISTORY_ENABLED to false.

1.2.4 class Flag

This class contains colours definitions for flags and hence also the number of flags. Agents access this class to check how many flags they can choose from.

1.2.5 class Main

Contains a main method that creates a simulation with given parameters and displays its GUI.

1.2.6 class SimulationGUI

Extended JFrame that displays simulation GUI and allows to:

- see a round counter
- progress to the next round repetitively (GUI freezes) until consensus is achieved (history can be access afterwards if enabled)
- progress to the next round
- reset the simulation (a new back-end is created)
- enable or disable visibility (displaying network edges)
- enable or disable agents' ids
- enable or disable state
- enable or disable the history mode
- access the memory of selected agent
- modify the flag raised by the chosen agent in the previous round – this will update a history and the simulation will call setFlag on the agent once again in this round. This feature should not be used if an agent remembers the flags raised by its neighbour or itself as these will not be updated (unless manually via AgentMemoryAccessor)

1.2.7 class Tester

This class provides methods to test agents' behaviours by running multiple simulations. It saves the progress to the text file. The two methods provided are:

- testAgents(int tests, int minAgents, int maxAgents, int visibility, Class agentClass) that performs given number of tests (simulation runs) for each number of agents between minAgents and maxAgents. tinAgents has to be even and only tests with even number of agents are performed. For example, if tests is set to 10,000, minAgents 20, and maxAgents 24, this method will run the simulation until consensus is achieved 30,000 times
- testVisibility(int tests, int agents, int minVisibility, int maxVisibility, Class agentClass) that performs given number of tests for each visibility in range

2 package network

This package contains everything related to any-network simulation.

2.1 package GUI

GUI components of the simulation and network creator.

2.1.1 package creator

It is a front-end of network.creator package that allows to create and manage networks.

2.1.1.1 class CreatorDrawablePanel

A panel that displays a network with its coloured vertices, edges, labels.

2.1.1.2 class CreatorMenuBar

A menu bar that provides most of the functionality of the network creator that allows to:

- create a new empty network
- use network generators to automatically generate network of given properties
- save/load network to file
- show network statistics
- print network to console
- enable network colouring
- enable showing edges intersections
- initialise simulation with the current network
- enable performance-affecting options such as anti-aliasing
- view help

2.1.1.3 class CreatorMouseListener

Provides functionality required to create a network, such as:

- creating/deleting nodes
- creating/deleting edges
- changing nodes' positions

2.1.1.4 class MainWindow

Connect together all other components within this package to produce user-friendly window.

2.1.2 package simulation

This package groups together simulation GUI, including simulation initialisation, its history and the actual simulation interface.

2.1.2.1 class AbstractDrawablePane

Buffered drawable pane. Uses buffers to speed up edges and nodes' labels drawing. Allows to define node colours, selection highlighting and infections highlighting via four abstract methods:

- Color getFlag(int id)
- boolean containsInfections()
- boolean isInfected(int id)
- int getSelectionID()

2.1.2.2 class AgentMemoryAccessor

Extends util.MemoryAccessor to provide access to agent's internal memory. Displays agent's class and id.

2.1.2.3 class HistoryWindow

A window for displaying simulation history. Does not allow to display rounds that took place after the window was created. Requires to provide Network object on which the game takes place, boolean array indicating infections, List of integer arrays (each such array represents single round) and an optional network name.

2.1.2.4 class InitialisationSettings

Used to group together choices made in simulation initialisation, such as agent and infected agent classes, number of flags or consensus mode, to be used again when a new simulation is initialisated.

2.1.2.5 class InitialisationWindow

Initialisation window is a final step before starting the simulation, after the network has been created. It displays a network with uncoloured nodes and no options to modify it. It allows a user to choose a behaviour of agents and infected agents, the number of flags that should be used in the game, the consensus mode (differentiation or colouring) and whether infected agents should be included in checking if consensus was achieved. It also allows to select multiple nodes that will be marked as “infected” and use different behaviour.

2.1.2.6 class SimulationContentPane

This is a content pane for the SimulationWindow. It includes a drawable panel where the network is drawn – it highlights selection and infections. It provides labels that show current round, current consensus status (yes/no) and a round at which consensus was achieved for the first time. It also provides the buttons with functionality such as progressing to the next round, playing until consensus is achieved (it progresses in the background, contains timeout) or displaying a history window. Finally, it provides AgentMemoryAccessor that allows to modify a memory of last select agent.

2.1.2.7 class SimulationWindow

Displays all functionality of the SimulationContentPane in a window. Ensures proper positioning of the window (depending on where last such window was closed during single runtime of the application) and sets the proper sizes of the window.

2.1.3 class Constants

GUI constants. Allows to easily change:

- node's diameter
- minimum/maximum number of flags
- selection colour
- infections highlighting colour
- edges intersections colour
- node moving path colour (if advanced node moving in network creator is disabled)

2.2 package creator

This package includes the network representation, automated network generators and network statistics.

2.2.1 class Network

Network representation stored as an adjacency list. Provides back-end functionality such as:

- creating/deleting nodes
- deleting nodes with keeping connections
- connecting/disconnecting nodes
- converting the network to user-readable String
- iterating over the nodes of the graph
- iterating over the edges of the graph
- getting position of the requested node
- getting the number of nodes
- getting the neighbours' ids of the requested node
- checking if two requested nodes are connected
- repositioning a node
- getting the network statistics
- checking if network contains edges intersections
- finding the closest node to the given position
- saving/loading the network to/from the file
- generating networks of given types and parameters describing these types

2.2.2 class NetworkStats

Network statistics that are used to describe a network. They include:

- number of nodes
- number of edges
- minimum degree (lowest number of neighbours)
- maximum degree (highest number of neighbours)
- degree mean (average)
- degree median (middle value of sorted degrees)
- degree mode (most common)

2.2.3 class Node

Representation of a node within a network. Contains an id (unique for the network) and double coordinates. Provides public accessors only (with no setters).

2.3 package graphUtil

This package contains tools that are useful in accessing the Network object but are not required by the Network to provide proper functionality.

2.3.1 class Edge

This class represents an edge as two points of integer coordinates in 2D space. It provides methods to check whether two edges are polygonal chain (i.e. they have common point) and to check whether two edges intersect.

2.4 package painter

This package provides a graph painter and other functionality related to assigning colours to graph nodes.

2.4.1 class GraphPainter

Contains the colour definitions to be used by many other components within any-network simulation. This class also provides a functionality of colouring the graph with a simple BFS algorithm.

2.5 package simulation

This package contains simulation back-end including its internal representation of an agent and agent superclass.

2.5.1 package agents

This package contains implemented agents.

2.5.2 class AbstractAgent

This is a superclass of an agent that allows to define agent's behaviour by implementing abstract method int getNewFlag(int round). Any-network simulation's AbstractAgent is very similar to circle network simulation's AbstractAgent. It provides methods for the simulation to set visible agents and set flag. Agent can access its previously raised flag via getFlag() method and its neighbours via an array with AgentDelegates. Also, the most important change is that this AbstractAgent's only constructor requires to provide a number of flags that are used in the game – this final field can be accessed from subclasses.

2.5.3 class AgentDelegate

AgentDelegate is an interface that define how agents see each other. It contains only one method getFlag() that returns a flag raised by an agent in the last round.

2.5.4 class AgentInfo

AgentInfo is an internal representation of an agent in the simulation. It contains unique agent's id, agent (of class AbstractAgent) and neighbours as AgentInfo objects.

2.5.5 class Simulation

Simulation back-end that creates AgentInfo objects and connects them as defined in given network. It provides four constructors:

- Simulation(Network network, Class agentClass, int maxFlags, boolean consensus, int[] initialRound, boolean history)
- Simulation(Network network, Class agentClass, int maxFlags, boolean consensus, boolean history)
- Simulation(Network network, Class[] agentsClass, boolean infected[], int maxFlags, boolean includeInfected, boolean consensus, int[] initialRound, boolean history)
- Simulation(Network network, Class[] agentsClass, boolean infected[], int maxFlags, boolean includeInfected, boolean consensus, boolean history)

The first two constructors create a simulation without infections. They require a Network object defining the connections between agents, Class that is a descended of AbstractAgent, integer defining how many flags can be used in game, boolean defining a consensus mode (differentiation or colouring) and a boolean value for enabling/disabling history. The first constructor also requires an array with flags raised in the first round – array has to be of the same length as number of agents and its values have to be non-negative and smaller than maxFlags. In case of using this constructor, the simulation will set flags in the initial round without asking agents what choice they want to make, the first call of getNewFlag(int round) from AbstractAgent will have round equal to 1 (instead of 0).

The last two constructors allow to create a simulation with infections. Class array defines a class of each agent. Boolean array infected defines which agents should be marked as infected. Boolean includeInfected defines whether infected agents should be considered in consensus or not. All arrays have to have the same length equal to the number of nodes in provided Network.

Simulations provides methods that allow to:

- get the network used to initialise simulation (although it can be modified, it will not affect the simulation, as the Network object is used only for initialisation)
- check whether the consensus was achieved (it depends on consensus mode and includeInfected value if the simulation contains infections)
- get ids of infected agents
- progress to the next round
- get flag of requested agent
- get current round count
- get AgentInfo object of requested agent
- check if history is enabled
- get history (as unmodifiable list)

2.6 package tests

This package contains experiment scheduler and experiment definitions.

2.6.1 class AbstractExperiment

Experiment, when started, will play the simulation (until consensus) given number of times, and save all the information to the log file. Its only abstract method has to create a simulation.

2.6.2 class ExperimentScheduler

Experiment scheduler allows to add tests to it that will be later run. Scheduler allows to specify on how many threads experiments should be run simultaneously.

2.6.3 class NormalExperiment

The most simple experiment with differentiation goal and no infections.

2.7 class Main

Entry point of any-network simulation GUI mode. Creates a network creator window.

2.8 class Tests

Main class for running tests. Creates an ExperimentScheduler and runs all experiments added to it.

3 package util

This package contains various tools and utilities.

3.1 class MemoryAccessor

Memory accessor takes any Java object and creates a GUI interface that displays all fields of this object and allows to modify them (assuming that access modifiers do not prevent it). It works with any object and can be used to access agents' memory without necessity to implement this functionality for each agent separately.

4 package exceptions

This package contains various exceptions.

4.1 class FeatureDisabledException

This exception is thrown when the program tries to use a feature that is disabled, such as accessing non-existing history of the simulation.

4.2 class IncorrectUsageException

Exception thrown when the method (or methods) are used incorrectly, for example if overridden method is supposed to return a value in some range, but returns a value outside this range.

4.3 class ShouldNeverHappenException

This exception indicates programmer's serious error. It is thrown in the cases that should never happen and hence ensures that further modifications of the class do not violate these cases.