Advanced Programming Assessed Exercise

Butler, Aidan 2281611B@student.gla.ac.uk

February 23, 2017

Contents

$^{\circ}$	\sim 1 1					l* . *			
0.1	(Alular	out∩moto	cimilization.	and	MICHA	lızatı∧n	programme of	lecion	- 4
Ο. Ι	Cciuiai	automata	Jiiiiuiatioii	anu	visua	nza tion	programmic	acoigii	 _

CelularAutomataSimulation ≪interface≫ Runnable Each Creature runs on its **TestWorld** own thread. + run() + main(String[]) 1 Node 1 - horiz : int Creature World - vert : int # fitness : double – neighbourNodes : # height: int ArrayList<Node> # lifeTime : int # nodes : Node[][] - resident : Creature # name : String # oneSpaceMoves : int[][] # node : Node # width : int # getHorizontalPos(): # getName() : String # assignNeighbours() # getThread() : Thread # getNeighbourNodes(): # generateNodes() 1 0..1 ArrayList<Node> # populateNode(Node) # generateOneSpaceMoves : # getResident() : # reproduce() int[][] # setFitness() : double Creature # getNodeAt(int,int) # setLifetime() : int # getVerticalPos(): # identifyNeighbours(Node) : # setName() : String ArrayList<Node> # spawn(Node) : Creature # isEmpty() : boolean # populate() # setNeighbourhood-- murder(Creature) + toString : String -Nodes() + toString() : String # setResident(Creature) + toString() : String Species2 RoundWorld # setFitness() : double # setLifetime() : int # setName() : String # identifyNeighbours(Node) : # spawn(Node) : Creature ArrayList<Node> FlatWorld Species1 # identifyNeighbours(Node) : ArrayList<Node> # setFitness() : double # setLifetime(): int # setName() : String # spawn(Node) : Creature

Figure 1: Celular automata simulation and visualization programme design

0.1 Celular automata simulation and visualization programme design

This programme is designed to implement and display a command line visualisation of a cellular automata simulation containing two self-reproducing types of agents, or creatures, with differing attributes which can exist, reproduce and destroy one another.

The programme begins when TestWorld's main method generates a new World and begins visualising the state thereof periodically by printing a representation of the evolving world-state to the command line. The abstract World class can be instantiated as either a RoundWorld (wherein the edges of the World "wrap around") or a FlatWorld (wherein they do not). The differences between the two subclasses are determined by World's abstract identifyNeighbours() method, which both classes must implement. The World contains an array of Node objects, each of which has an ArrayList of neighbouring Nodes and either zero or one resident Creatures. The abstract Creature class has fitness, lifeTime and name attributes, which are set by abstract methods implemented in the Species1 and Species2 classes which inherit from it. A Creature will sleep for the duration of its lifeTime and if interrupted (murdered) will simply terminate. If uninterrupted, the Creature will murder other Creatures in the surrounding Nodes and reproduce in them, depending on the result of randomised function's which take the Creature's fitness attribute as an input.

The World constructor generates an initial population of Creatures by calling the World.populate() method. Each Creature is allocated its own thread. The Creatures are able to interact with one another by accessing a common heap containing the Nodes. Thready-safety is provided by ensuring that Creatures lock Nodes prior to making changes to them.

The use of abstract classes, such as Creature, containing abstract methods such as Creature.spawn() is designed to minimise code repetition and maximise class cohesion. As much code as possible is placed in abstract classes, meaning that changes can be carried out in inheriting classes in a uniform manner and by changing as few lines of code as possible. The use of threads is designed to allow each creature to function independently from the rest.

World-state visualisation is made possible by TestWorld calling the World's toString() method, which in turn calls the Node's toString() method, which in its turn calls Creature's toString() method.