# Programming Exercises

The following require you to open the skeleton program and make modifications. They are written in examination style and illustrate how you should prepare your answers.

Question 1

*This task refers to the Main procedure*

Alter how the menu displays so that:

– There is a new option '3. Rabbit Paradise'

– The 'Exit' option is now numbered 4

Evidence you need to provide:

* Copy of your amended code
* Screen capture of it executing

5 marks

Question 2

*This task refers to the Main procedure*

Code option 3 so that when it is selected the simulation is run with the following parameters:

– A landscape size of 20

– 20 warrens

– 0 foxes

– Locations are not fixed

– Variability is 1

Evidence you need to provide:

* Copy of your amended code
* Screen capture of it executing

5 marks

Question 3

*This task refers to the Simulation class*

Add an option to the game menu:

'0. Advance 10 time periods hiding detail'

Code this option.

Evidence you need to provide:

* Copy of your amended code
* Screen capture of it executing

7 marks

Question 4

*This task refers to the Rabbit class*

Change *Rabbit****’****s* constructor so that it receives in an extra variable that will allow the ratio of male to female rabbits to be altered. Use the identifier *genderRatio* for the new variable.

Set the default value to 50 so that the constructor can be called without specifying a value for *genderRatio*.

Evidence you need to provide:

* Copy of your amended code

2 marks

Question 5

*This task refers to the Fox class*

Add *Gender* to the *Fox* class.

Make the ratio of males to females 1 : 2.

Alter the *Inspect* method so that the gender of a fox is reported.

Change *ReproduceThisPeriod* so that only female foxes can reproduce.

Evidence you need to provide:

* Copy of your amended code
* Screen capture of an inspection of the Fox at 2,10

12 marks

Question 6

*A new subclass must be created for this task, as well as changes to the createLandscapeAndAnimals procedure*

*in Simulation*

Create a subclass of *Warren* called a *GiantWarren*.

– A giant warren has a maximum capacity of 200 and can always spawn a new warren even if it has done so already.

– A giant warren has a default rabbit.

– Add a giant warren to the default game at position (11,4) with a starting population of 115.

Evidence you need to provide:

* Copy of your amended code
* Screen capture of a default simulation executing

11 marks

Question 7

*A new subclass must be created for this task, as well as changes to the Location class and createLandscapeAndAnimals, drawLandscape and AdvanceTimePeriod procedures in Simulation*

Create a *Den* class that can exist in a location.

– The den will spawn 1 new fox per 3 time periods.

– The den will store how many foxes it has created as a private instance variable.

– The fox will appear at a random position.

– If there is already a fox in this location, it is replaced by the new fox.

– Position the den at (2,3) in a default game.

– The den will be displayed on the map as a D plus the number of foxes it has spawned, e.g. D2.

Evidence you need to provide:

* Copy of your amended code
* Screen capture at time period 3 of a default game running

18 marks

Question 8

*This tasks refers to the Fox class*

The average age of death of foxes needs to be known.

– Create a class variable called \_*TotalDeadFoxes* to store the total foxes who have died.

– Create a class variable called \_*TotalFoxAge* to store the sum of the ages of all foxes who have died.

– When a fox dies, the \_*TotalDeadFoxes* needs to be incremented and its age added to \_*TotalFoxAge*.

– An accessor method in Fox called *getLifeExpect* will return the average age of a fox at death.

– A message stating 'The average life expectancy of a fox stands at X' should be printed under the landscape each time it is displayed.

– If no foxes have yet died, the default lifespan should be returned.

Evidence you need to provide:

* Copy of your amended code
* Screen capture of default simulation at time period 0
* Screen capture of default simulation at time period 4

14 marks

Question 9

*This task refers to the Simulation class*

Create a menu option in the simulation: **'**6. Find biggest warren’.

The coordinates of the biggest warren will then be displayed: **'**Biggest warren at (X,Y) **'**.

Create a new procedure called findBiggest to search the warren array in a linear fashion and display the message.

Evidence you need to provide:

* Copy of your amended code
* Screen capture of option 6 running

12 marks

Question 10

*This task refers to the Rabbit class*

Make rabbit death probability go up by 10% with age.

Evidence you need to provide:

* Copy of your amended code
* Screen capture of a warren inspected (showing individual rabbits) at time period 2

2 marks

Question 11

*This task requires changes to Warren, Rabbit and Simulation classes*

Create a menu option: '7. Inspect all rabbits'.

It should display a list of all rabbits in all warrens, showing their details.

The rabbits must be shown in age order, oldest to youngest.

Bubble-sort the rabbits after adding them all to one list.

An accessor method to get the rabbits list out of a warren must be created.

An accessor method to get a rabbit's age out of a rabbit must be created.

Evidence you need to provide:

* Copy of your amended code
* Screen capture of option 7 running

18 marks

Question 12

*This task requires changes to Simulation as well as creation of new classes*

Beneath the warrens are secret tunnels connecting them. Not every warren is connected to every other. No warren is connected to more than two other warrens. This data must be stored in a *WarrenGraph*.

|  |  |
| --- | --- |
| WarrenGraph | |
| -nodes[] | |
| +addNode(theNode)  +adjList() | |
| Node |
| -selfX  -selfY  -leftBranchX  -leftBranchY  -rightBranchX  -rightBranchY |
| +getCoord(l/r/s) |

Each warren connected to another has the coordinates of itself and its connecting warrens stored in a node. *WarrenGraph* contains a list of all nodes. The procedure *getCoord* returns the x- and y-coordinates of these based on arguments (l)eft, (r)ight and (s)elf.

The *adjList* method displays an adjacency list and should be executed by a new option: '8. Display adjacency list'.

The following data should be used to initially populate the graph.

|  |  |  |
| --- | --- | --- |
| self | left | right |
| (1,1) | (2,8) | (9,7) |
| (2,8) | (13,4) | (1,1) |
| (9,7) | (1,1) | (13,4) |
| (13,4) | (9,7) | (2,8) |

Evidence you need to provide:

* Copy of your amended code
* Screen capture of option 8 running

22 marks

Question 13

*This task requires changes to Simulation and WarrenGraph*

Create a new procedure in *WarrenGraph* called *adjMatrix*. It will display the graph as a matrix instead of a list. It will be executed by '9. Display adjacency matrix'. A 1 should be used to indicate a connecting burrow.

Evidence you need to provide:

* Copy of your amended code
* Screen capture of option 9 running

17 marks

Question 14

*This task requires changes to WarrenGraph*

Amend your solution for task 13 to replace the '1' with the actual distance between the nodes/warrens.

Use Pythagoras' theorem to calculate the distance between the two points.

Distances should be rounded to 1 decimal place.

Evidence you need to provide:

* Copy of your amended code for adjMatrix
* Screen shot of option 9 running

9 marks

Question 15

*This task requires changes to Simulation and WarrenGraph*

Create a procedure to find whether there is a route between two warrens.

It will be executed by Option 10.

Evidence you need to provide:

* Copy of your code
* Screen capture of option 10 running showing no route between warrens
* Screen capture of option 10 running showing a route between warrens

13 marks

# Answers and Solutions

## Complete UML Diagram

# 