Rabbit Season

Andrew Rosen
Original David Matuszek

Abstract

This assignment serves as a review of content from CIS 1068. It is an engaging program that will help you get back into the swing of things. It's a graphical program, so you can explore the basics of how GUI's work in Java. You will also start learning how to work with others, as this is a two person project.

1 Premise

This assignment is different than the others, as I've given you a full working program, but you will need to improve it to get a grade.

A `Fox` is hunting the `Rabbit` in a small field, represented by a 2D grid. The field has many spaces, some of which have bushes, which block the view of the fox and the rabbit. The goal of the fox is to end up on the same space of the rabbit, where he catches and eats it. The goal of the rabbit is to evade the fox for 100 turns.

You play as the rabbit.

1.1 Setup

To get started, download the zip file, unzip it, and put all the java files into the same project.[1] Run `RabbitHunt.java` and you will get something like the image below.

That big red dot is the fox, who is very hungry and is going to chase down and eat the rabbit (the brown dot). The green things are bushes.

Play with the program for a while. Notice that the Fox always chases down the rabbit. This is because the Fox has a strategy, which you can look at in the `Fox.java` code. The Rabbit gets eaten all the time because its strategy is to just chooses a random direction to move.

[1] You might surprised that after you create a new project, its as easy as copy-pasting or drop and dragging the files into the project.

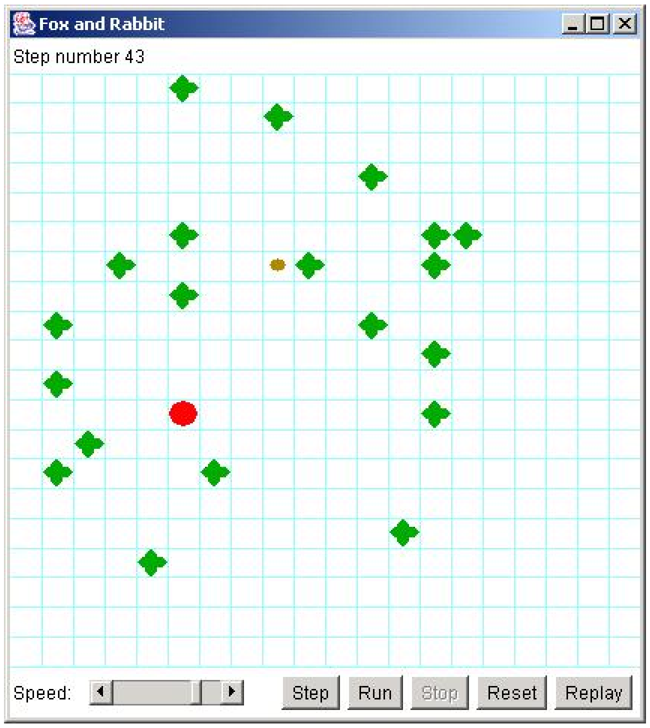

Figure 1: The game board. If you get a window, but no grid, just hit one of the buttons in the window.

1.2 Your Task

You need to fix that so that your Rabbit survives to turn 100 as much as possible. The `Rabbit` has a constructor `Rabbit(model, row, column)`, and it has a method `decideMove()`, which decides how the rabbit is going to move at each turn. You can make as many changes to the Rabbit class that you want.

Your grade is the percentage of times that the Rabbit makes it to turn 100 out of 300 runs (with some adjustment). It's probably impossible to get a 100.

1.3 The One Rule

You can only modify `Rabbit.java`. You can look at everything else, but only change `Rabbit.java`.

2 Code Tour

This program is built using the Model-View-Controller paradigm.

- The `Model` holds the rules of the game and handles how the pieces interact with each other.[2] The `Model` makes the board, places the `Fox`, the `Rabbit`, and a number of `Bush` objects.
- The `View` is the code for what you see. `View`'s job is to show you, the user, what's going on in the `Model`. In this program, that means `View` shows the game board.
- `Controller` lets you run the simulator. It tells the `Model` when to start and stop.

Finally, `RabbitHunt.java` has the main method and just instansiates `Model`, `View`, and `Controller` objects, and turns control over to the `Controller` object.

You can safely ignore and never open the `Controller` or `View` to complete this assignment. This means your code will use resources provided by `Model` and `Animal`. Those are the parts you need to get a good grasp of.

2.1 The Model

The Model class:

- places the fox, rabbit, and bushes in the field
- gives the rabbit and the fox each a chance to move (one moves, then the other–they don't both move at the same time)
- tells the `View` to display the result of these two moves, and
- determines which animal won

Model also provides you with many useful constants. Many aspects of the program, such as direction and what an animal can see are represented as an `int`. Rather than remembering what number represents what, `Model` provides a number of constants.

[2]Technically, the Model is made of five classes: `Model`, all the animals, and `Bush`

Model.N	indicates ”north” (straight up)
Model.NE	indicates ”northeast” (up and to the right)
Model.E	indicates ”east” (to the right)
Model.SE	indicates ”southeast” (to the right and down)
Model.S	indicates ”south” (straight down)
Model.SW	indicates ”southwest” (to the left and down)
Model.W	indicates ”west” (to the left)
Model.NW	indicates ”northwest” (up and to the left)
Model.STAY	indicates ”no move”
Model.MIN_DIRECTION	the numerically smallest direction (not including STAY)
Model.MAX_DIRECTION	the numerically largest direction (not including STAY)
Model.BUSH	indicates a bush
Model.FOX	indicates a fox
Model.RABBIT	indicates a rabbit
Model.EDGE	indicates the edge of the board

Finally, Model provides the following method:

`static int turn(int direction, int amount)` Given a starting direction and the number of 1/8 turns to make clockwise, this method returns the resultant direction.

For example, `turn(x,4)` would return the direction opposite of `x`.

2.1.1 The Other Model Classes

Other classes that form part of the Model are:

Bush This class doesn't do anything. The Model creates bushes and places them, but the bushes themselves just sit there and get in the way.

Animal This is the superclass for both the Rabbit class and the Fox class. It provides methods that are the same for both the rabbit and the fox: looking in a particular direction, measuring the distance to an object, and moving.

Fox The fox is an animal that tries to catch and eat the rabbit.

Rabbit The rabbit is an animal that tries not to get caught and eaten.

The Animal class provides the following methods. Since Fox and Rabbit are subclasses of Animal, they inherit these methods, and can use them just as if they had defined the methods themselves.

2.1.2 Animal Methods

`int look(int direction)` Return one of the constants Model.BUSH, Model.FOX, Model.RABBIT, or Model.EDGE, depending on what the animal sees in that direction.

`int distance(int direction)` Returns the number of steps the animal would have to make in order to land on top of the nearest object (or go off the edge of the playing area).

`boolean canMove(int direction)` Tells whether the animal can move in the given direction without being stopped by a bush or the edge of the board. Does not tell whether it's a good idea to move in that direction.

3 Hints

You should check out `Model.java` and see what static resources it provides, as well `Animal.java`, as it provides all the common actions an `Animal` can do.

If you're at a loss on where to begin, check out `Fox.java`, since it provides the algorithm for what the `Fox` does to chase down `Rabbit`.

4 Rubric

Most assignments are weighted at 100 points. This one will be weighted at 120.

100 points The rabbit survival rate. Your grade is the square root of the percentage of times the rabbit survives times ten.

20 points You must explain your rabbit's survival strategy to me or the TA to get any credit at all for the assignment. If you're nervous about describing your strategy in front of the Professor or your TA, I recommend commenting your code so you can read your comments like a script.