PacMan Lab

In this lab, you will be implementing several components in a PacMan-like game. Specifically, you will be implementing the various items which move around the maze. Most of the rest of the program is implemented for you. It is necessary to understand a bit about how the project is organized – the classes and their methods so that you can utilize these functions in your code.

First, all important information is, as usual, conglomerated and coordinated in a model class: PacManModel. A view/control class, PacManControl takes care of displaying the state of the game, interacting with the user, and keeping track of the passage of time. The state of the game consists of the maze, PacMan, the ghosts, and the fruit. All can be obtained from an instance of the model class. Additionally, each actor (that is, PacMan, the ghosts, and the fruit) can be accessed via its name (a String). Since there could conceivably be more than one actor with the same name, there are functions to deal with this as well.

public class PacManModel {

public PacManMaze getMaze() { . . . }

public PacMan getPacMan() { . . . }

public int getNumGhosts() { . . . }

public PacManGhost getGhost( int i ) { . . . }

public Fruit getFruit() { . . . }

public PacManActor getActor( String n ) { . . . }

public int getActorCount( String n ) { . . . }

public PacManActor getActor( String n, int i ) { . . . }

}

The maze (class PacManMaze) consists of a grid of squares (class PacManSquare). Each square can be accessed using a Location object (which contains row and column):

public class PacManMaze() {

public int getWidth() { . . . }

public int getHeight() { . . . }

public PacManSquare getSquare( Location l ) { . . . }

}

Each square is either a wall or not and may contain a pellet (class PacManPellet) which may be a power pellet (class PacManPowerPellet extends PacManPellet):

public class PacManSquare {

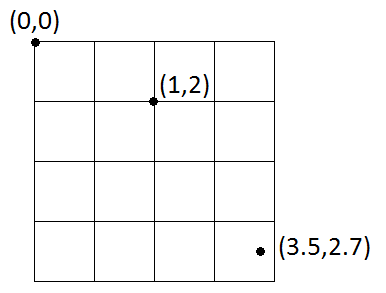
public boolean isWall() { . . . }

public PacManPellet getPellet() { . . . } // returns null if no pellet

public void addPellet( PacManPellet p ) { . . . }

}

The Location class is used to keep track of row and column pairs. Every square in the maze and every actor in the game uses Locations to keep track of their positions within the maze. Because actors need to move smoothly from one square to another (rather than “teleporting” instantaneously), the row and column fields need to be doubles. Integer values indicate that the actor is centered at the upper left corner of the corresponding square. Non-integer values indicate that the actor’s center is some percentage of the way into the square. For example, if the position of an actor is <row = 3.5, column = 2.7>, this indicates that the actor is within the square at <row = 3, column = 2>, along the center line of the square, 70% of the way to the next square. See diagram below for a few examples.



public class Location {

public Location( double r, double c ) { . . . } // constructs a Location object from a row and

// column

public double getRow() { . . . }

public double getColumn() { . . . }

}

Finally, all actors (PacMan, ghosts, fruits) will derive (extend) from the PacManActor base class. This class takes care of moving actors smoothly around the maze. Each actor maintains a pointer to the model object (so that each actor can access the state of the rest of the game). Additionally, each actor has a Location, a speed, and a direction (0 = up, 1 = right, 2 = down, 3 = left). Of these, only the direction can be directly manipulated, and even that won’t necessarily take effect immediately (since the actor may be moving along a wall). Instead, an actor can indicate what the next direction it wishes to travel is. When it reaches a point where it can turn in that direction (usually at the center of a square), the new direction is adopted. As an actor moves through the maze, several trigger methods will get called as it enters, leaves or reaches the center of a square. These methods can be overridden to provide different behaviors (such as eating a pellet or choosing a new direction). The actor class has an update method which handles the actual movement through the maze. This method can also be overridden. Finally, there are several additional convenience methods which retrieve the next square that an actor would occupy etc.

public class PacManActor {

// creates a new PacManActor in location l with speed s moving in direction d and model m

public PacManActor( Location l, double s, int d, PacManModel m ) { . . . }

public int getDirection() { . . . }

protected void setDirection( int d ) { . . . } // sets the next desired direction

public double getSpeed() { . . . }

public Location getLocation() { . . . }

protected PacManModel getModel() { . . . }

public void update( double dt ) { . . . }

protected void enterSquare( PacManSquare sq, int dir ) { . . . }

protected void leaveSquare( PacManSquare sq, int dir ) { . . . }

protected void reachCenterOfSquare( PacManSquare sq, int dir ) { . . . }

protected void commitDirectionChange() { . . . } // actually changes the direction

// gets the neighbor location of l in the direction dir

protected static Location getNeighborLocation( Location l, int dir ) { . . . }

// gets the neighbor square in the given direction

protected final PacManSquare getNeighborSquare( int dir )

// gets the next square that this actor will occupy (takes into account the desired next direction)

protected final PacManSquare getNextSquare()

}

Exercise 1 – PacMan

Class PacMan represents the hero of the video game. He is a yellow disc with a mouth which opens and closes. As he moves around the board, he eats pellets. If he eats a power pellet then, for a short period, he can eat any ghost whose path crosses his. Detailed instructions for implementing PacMan are provided in PacMan.java. However, once you have implemented PacMan you will also need to take the following steps:

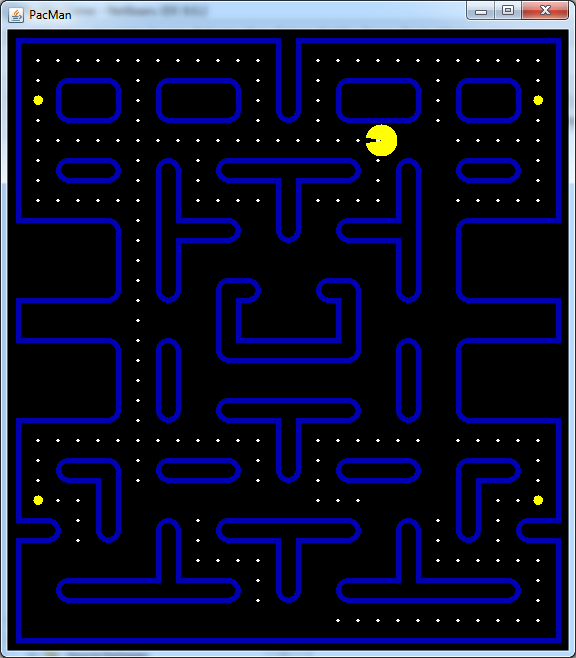
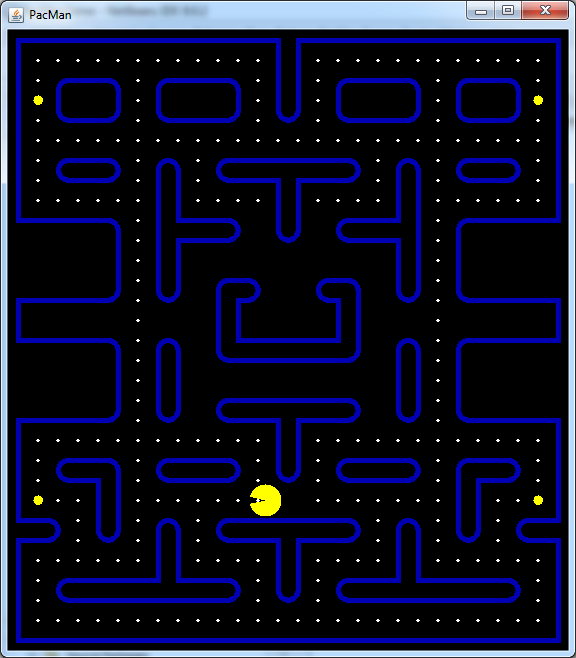
1. Uncomment lines 129 and 130 in PacManModel.java:

pacman = new PacMan(l, this);

addActor( "PacMan", pacman );

1. Uncomment the first half of the body of checkPacManIntersections() (lines 302-322) in PacManModel.java
2. Uncomment the body of function paintPacMan (lines 208 – 235) in PacManView.java

Test by running the project. You should see PacMan opening and closing his mouth. By using the arrow keys, you should be able to move PacMan around the maze. As PacMan moves around the maze, pellets should disappear. By the way, if you press the space bar, the program will pause/unpause.



The PacManGhost class represents all the ghosts in the game. Each ghost is slightly different from the others. The PacManGhost class takes care of implementing the similarities. First, since the ghosts move around the maze, the PacManGhost class extends PacManActor. Secondly, each ghost makes a decision when it reaches the center of a square which direction to continue in. At any given time, each ghost has a target location that it is trying to reach. Which square it is trying to reach depends on the state of the game and which ghost. There are four modes that a ghost can be in:

1. Chase mode – the ghost is chasing PacMan. Each ghost targets a location near PacMan (based on where PacMan is currently and what direction he is facing). Each ghost targets a different location. Some go straight for PacMan. Others target squares in front of PacMan or behind him. This has the effect of having the ghosts seem to set ambushes for PacMan.
2. Scatter mode – periodically, all the ghosts change to scatter mode. Each ghost targets a different corner of the board. This has the effect of drawing them away from PacMan. After a certain amount of time, they resume chase mode, re-converging on PacMan.
3. Frightened mode – when PacMan eats a power pellet, the ghosts turn blue and enter Frightened Mode for a few seconds. During this period, PacMan can eat them. Each ghost targets a random location on the maze and switches this location periodically.
4. Eye mode – once PacMan eats a ghost, the ghost’s body disappears, leaving only its eyes. To regenerate, the ghost must reach the regeneration square on the board. Each ghost targets this location.

Of these modes, all ghosts have the same behavior when in Frightened and Eye modes. These methods cannot be overrided (they are final). However, each ghost can provide different target locations for the other two modes. Therefore, override the methods which obtain the locations for Chase and Scatter modes. In fact, both of these methods are abstract meaning that any (non-abstract) sub-class of PacManGhost must provide implementations of them:

public class PacManGhost extends PacManActor {

protected abstract Location getChaseLocation();

protected abstract Location getFrightenedLocation();

}

Exercise 2: Implement the Ghosts

2a. Blinky – Blinky is red and moves at 110% of the standard ghost speed (GHOST\_SPEED) (specify this information when you implement the constructor). When in chase mode, he targets PacMan’s current position. When in scatter mode, he targets the upper left corner of the maze. Implement the two methods getChaseLocation and getFrightenedLocation to take care of this.

After implementing Blinky, you will need to modify the data file PacManBoard.txt. Currently it looks like this:

31 28

XXXXXXXXXXXXXXXXXXXXXXXXXXXX

X............XX............X

X.XXXX.XXXXX.XX.XXXXX.XXXX.X

X\*XXXX.XXXXX.XX.XXXXX.XXXX\*X

X.XXXX.XXXXX.XX.XXXXX.XXXX.X

X..........................X

X.XXXX.XX.XXXXXXXX.XX.XXXX.X

X.XXXX.XX.XXXXXXXX.XX.XXXX.X

X......XX....XX....XX......X

XXXXXX.XXXXX XX XXXXX.XXXXXX

XXXXXX.XXXXX XX XXXXX.XXXXXX

XXXXXX.XX XX.XXXXXX

XXXXXX.XX XXX XXX XX.XXXXXX

XXXXXX.XX XXXR XXX XX.XXXXXX

X . XX XX . X

XXXXXX.XX XXXXXXXX XX.XXXXXX

XXXXXX.XX XXXXXXXX XX.XXXXXX

XXXXXX.XX F XX.XXXXXX

XXXXXX.XX XXXXXXXX XX.XXXXXX

XXXXXX.XX XXXXXXXX XX.XXXXXX

X............XX............X

X.XXXX.XXXXX.XX.XXXXX.XXXX.X

X.XXXX.XXXXX.XX.XXXXX.XXXX.X

X\*..XX.......P .......XX..\*X

XXX.XX.XX.XXXXXXXX.XX.XX.XXX

XXX.XX.XX.XXXXXXXX.XX.XX.XXX

X......XX....XX....XX......X

X.XXXXXXXXXX.XX.XXXXXXXXXX.X

X.XXXXXXXXXX.XX.XXXXXXXXXX.X

X..........................X

XXXXXXXXXXXXXXXXXXXXXXXXXXXX

Fruit:

It’s pretty self-explanatory. The R (on line 14) indicates the ghosts’ re-spawn location. The F indicates where the Fruit will spawn (exercise 3). The P indicates where PacMan starts. To add some ghosts, you’ll need to add a ‘b’ (for Blinky) at line 12, column 13 (two squares above the R). In addition, you’ll need to add the line “b = Blinky” at the bottom of the file:

31 28

XXXXXXXXXXXXXXXXXXXXXXXXXXXX

X............XX............X

X.XXXX.XXXXX.XX.XXXXX.XXXX.X

X\*XXXX.XXXXX.XX.XXXXX.XXXX\*X

X.XXXX.XXXXX.XX.XXXXX.XXXX.X

X..........................X

X.XXXX.XX.XXXXXXXX.XX.XXXX.X

X.XXXX.XX.XXXXXXXX.XX.XXXX.X

X......XX....XX....XX......X

XXXXXX.XXXXX XX XXXXX.XXXXXX

XXXXXX.XXXXX XX XXXXX.XXXXXX

XXXXXX.XX b XX.XXXXXX

XXXXXX.XX XXX XXX XX.XXXXXX

XXXXXX.XX XXXR XXX XX.XXXXXX

X . XX XX . X

XXXXXX.XX XXXXXXXX XX.XXXXXX

XXXXXX.XX XXXXXXXX XX.XXXXXX

XXXXXX.XX F XX.XXXXXX

XXXXXX.XX XXXXXXXX XX.XXXXXX

XXXXXX.XX XXXXXXXX XX.XXXXXX

X............XX............X

X.XXXX.XXXXX.XX.XXXXX.XXXX.X

X.XXXX.XXXXX.XX.XXXXX.XXXX.X

X\*..XX.......P .......XX..\*X

XXX.XX.XX.XXXXXXXX.XX.XX.XXX

XXX.XX.XX.XXXXXXXX.XX.XX.XXX

X......XX....XX....XX......X

X.XXXXXXXXXX.XX.XXXXXXXXXX.X

X.XXXXXXXXXX.XX.XXXXXXXXXX.X

X..........................X

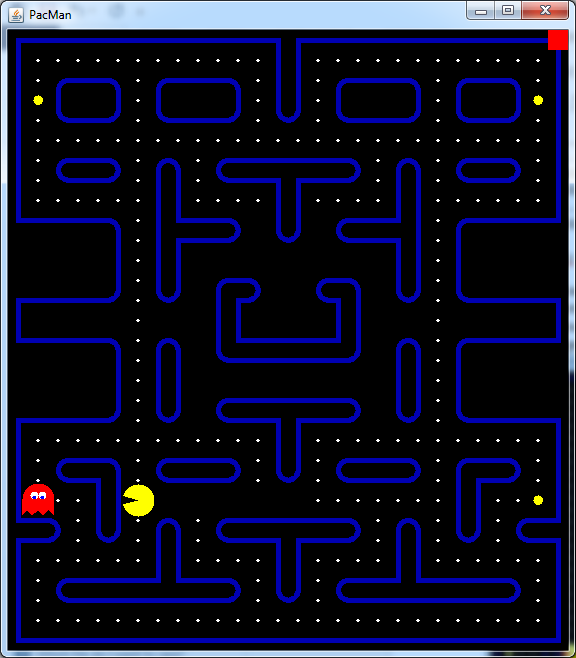
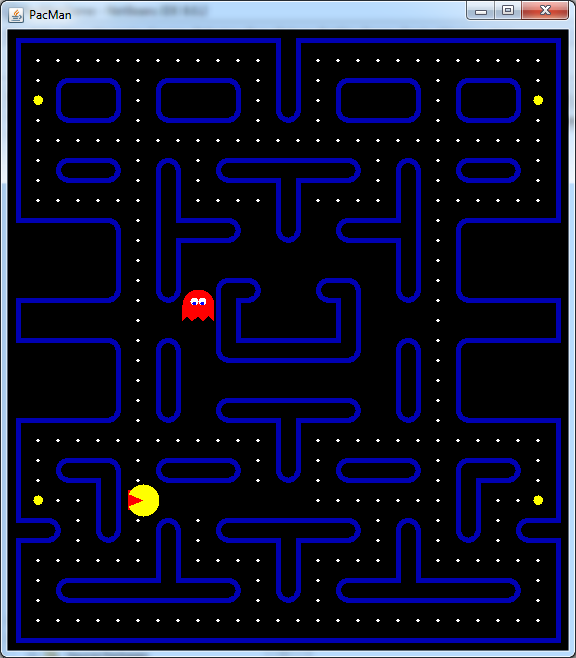
XXXXXXXXXXXXXXXXXXXXXXXXXXXX

Fruit:

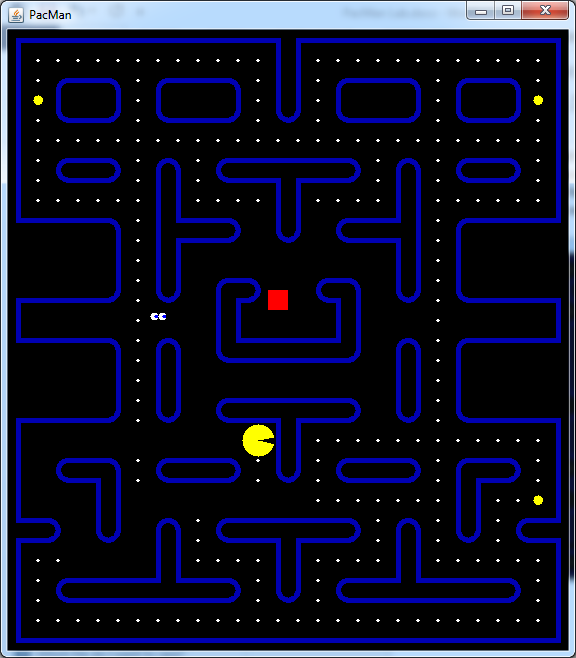
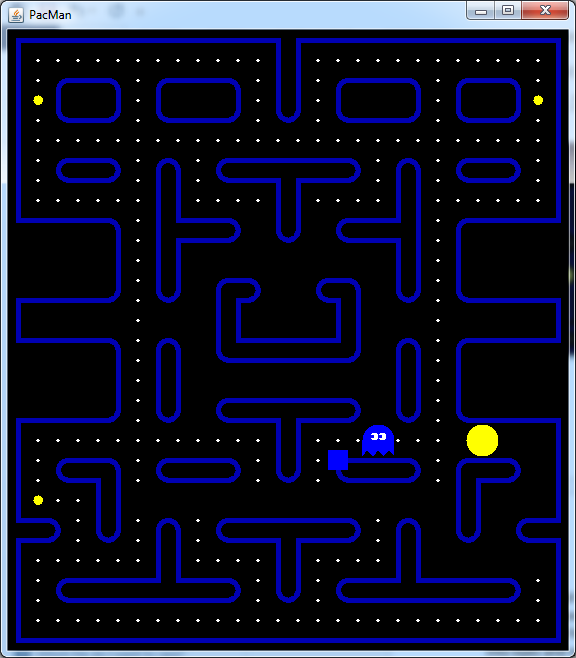
b = Blinky

As you add additional ghosts, you’ll need to add additional lines and starting locations to this file.

Run the program and you should see Blinky. If you press the ‘t’ key, you’ll see a red box indicating Blinky’s current target (you can see it underneath PacMan when his mouth is open). Wait a few seconds and Blinky should change to ScatterMode. Blinky will reverse direction when this happens. At this point, the target square should switch to the top right corner. Note that Blinky will not harm PacMan if he touches him. This feature has not been implemented in order to make testing easier.



Also, you need to test eating a power pellet. When this happens, PacMan should notify the model that he’s eaten a power pellet. Blinky should turn blue and target a random square (this targeting is already implemented for you in PacManGhost). If PacMan eats Blinky while in Frightened mode, Blinky will turn into eyes and move back towards the ghost regeneration square:



2b – 2g : Implement the rest of the ghost classes (Suggested order: Clyde, Pinky, Jinky, Inky, Dinky, Hinky). The details of each class are in the comments for that class. As you add ghost classes, you’ll need to add ghosts to the maze data file. At the bottom of the file use the format:

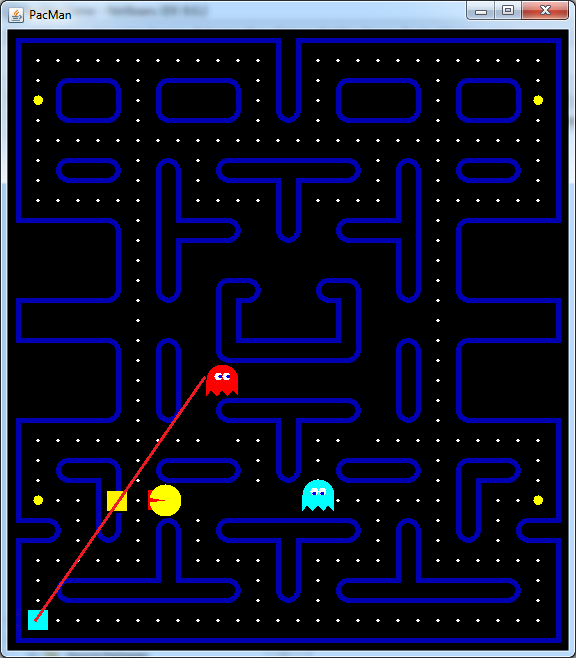
<letter> = <class-name>

Where letter is a single lowercase letter and class-name is the name of the class that you implemented. Place 1 (or more) instances of the lowercase letter in the maze itself. Some additional hints/explanation follow for particular ghosts:

Several of the ghosts refer to either PacMan’s location or the location of another ghost. You can obtain a pointer to another actor in the game by using the getActor method on the model. The method takes a String as input. The String should be the name of the class of the actor that you’re interested in. For example, to get a pointer to Blinky, you can use the line:

PacManActor blinky = getModel().getActor( “Blinky” );

For Inky, the chase target square is obtained by considering Blinky’s location and the square two squares in front of PacMan. Inky’s target is found by extending the line from Blinky through the square two in front of PacMan an equal distance on the other side as shown below:



The yellow square indicates the square two in front of PacMan. The light-blue square is Inky’s chase target. Notice that the yellow square is the midpoint of the Blinky’s location and the target square.

For Hinky, you will most likely need to store information in an instance variable.

Exercise 3 – Fruit

2a – b : Fruit and Cherry

Periodically during the game, a piece of Fruit (a cherry, strawberry, apple etc) spawns in the middle of the board and moves around. It’s worth extra points. In our game, the fruits will each have different characteristics.

Your first task is to implement class Fruit. Since Fruit moves around the board, there’s a class you should extend. You need to Override the reachCenterOfSquare method to pick a new location to go to. There are details in Fruit.java.

Next, implement class Cherry. All you need to do here is extend Fruit and provide the constructor. Next, in the maze data file, you need to add “Cherry” to the end of the line which starts with Fruit:

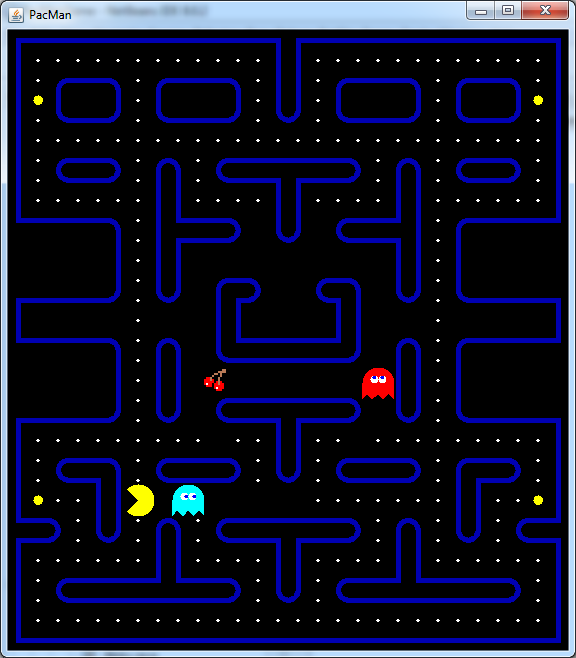
Fruit: Cherry

Finally, uncomment line 293 in PacManModel.java:

addActor( "Fruit", fruit );

the second half of the body of function checkPacManIntersections (lines 324-334) in PacManModel.java and the body of function paintFruit (lines 240 – 252) in PacManView.java.

Run the program and wait for about 10 seconds. You should see this:



2c-d: Implement Strawberry and Apple. The details are in the source files for those classes.

Bonus exercises. These are not due when the project is due. You may do these at any point in the semester. However, if more than one person does them, I will reduce the points awarded:

1. The fruits in PacMan bob up and down very slightly as they move around the board. Change class Fruit so that this happens. Do not modify any other classes. Also, do not override the update function of the Fruit class. Hint: You have to spoof the view into drawing the fruit in a slightly different location.
2. Change PacManModel so that the ghosts eat PacMan when they are not in Frightened or Eye modes. When PacMan is eaten, the board should reset all actors to their starting locations without resetting pellets. Also add a lives feature. The game should start with 3 lives and is over when all three lives are gone.
3. Add a score feature. The score should be displayed on the screen. Pellets are worth 10. Power pellets are worth 50. When ghosts are eaten, the first ghost is worth 200, the second 400, the third 800, the fourth 1600. This resets every time a power pellet is eaten. Fruit are worth 100 for cherries, 200 for strawberries and 300 for apples.
4. When all the pellets in the maze are eaten, a new maze should be loaded. The score should not reset.