Marcus Pearce, Justin Hu, Vincent Hwang

APCS Peck

Period 5

5/23/17

**APCS Final Project Specification: Peckman**

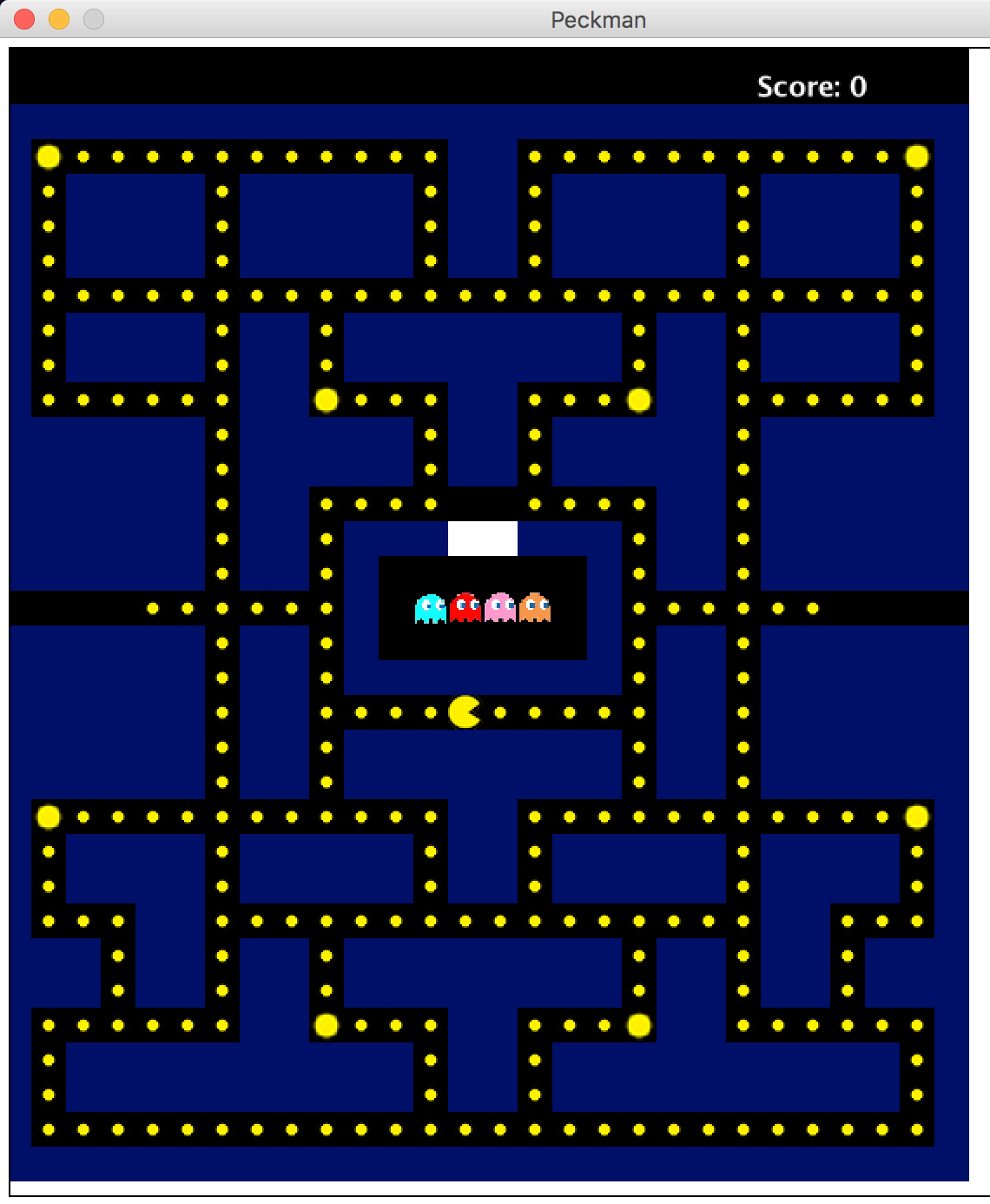
**---------------------------------------------------------------------------------------------------------------------**

**Summary**

In this project, we are re-creating the game Pacman. Pacman is an iconic and classic video game played and enjoyed by everybody. The objective of the game is to have Pacman eat all the dots before he is eaten by any of the four ghosts. Whenever Pacman occupies the same location as an enemy, he is considered to have collided with that ghost. However, if Pacman eats a big dot, a temporary “hypeMode” is activated, and Pacman will be able to eat the ghosts. During this mode, the ghosts run away from Pacman, and if they are eaten they return to their spawn point for a little while before re-entering the world. To create Pac-Man, we are using the java-oriented environment “Greenfoot” to help create the GUI. We developed the individual classes and methods in Eclipse in Java.

Objective:

The goal is for Pacman to eat all the dots and achieve the maximum score without being eaten by a ghost. As time progresses, more ghosts spawn (up to four), making the game progressively more difficult. Special “BigDot” change the game to a temporary “hypeMode,” where Pacman can eat the fleeing ghosts for extra points.



**---------------------------------------------------------------------------------------------------------------------**

**Structural Design**

Queue:

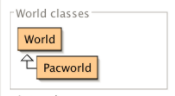
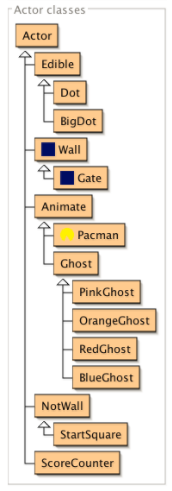
The queue is used to track the order of the ghosts that have been eaten. The queue is implemented in the Pacman class to keep track of the ghosts that Pacman has eaten. The ghost class has a method that returns the color of the ghost to differentiate which ghost has been eaten and correctly add the ghost to the queue in order. A queue is used because as ghosts die they need to respawn in the order they died in. A queue, which is first-in, first-out, is perfect for this.

**---------------------------------------------------------------------------------------------------------------------**

**Object-Oriented Design**

**Class Diagram:**

World Classes Actor Classes



**Class Summaries:**

*World*

*World* is the world that *Actors* live in. It is a two-dimensional grid of cells. All *Actor* are associated with a *World* and can get access to the world object. The size of cells can be specified at world creation time, and is constant after creation. Simple scenarios may use large cells that entirely contain the representations of objects in a single cell. More elaborate scenarios may use smaller cells (down to single pixel size) to achieve fine-grained placement and smoother animation.

*Pacworld*

*Pacworld* extends *World*, a class that is imported from Greenfoot. This class creates and manages the world. *Pacworld* creates the world for the game by reading the info in from a given text file. *Pacworld* is referenced by *Actor* for the timer (overall and hypeCounter) and *ScoreCounter*. It creates the map and GUI for the user to see and play the game.

*Actor*

An *Actor* is an object that exists in the Greenfoot world. Every *Actor* has a location in the world, and an appearance (that is: an icon). An *Actor* is not normally instantiated, but instead used as a superclass to more specific objects in the world. Every object that is intended to appear in the world must extend *Actor*. Subclasses can then define their own appearance and behaviour. One of the most important aspects of this class is the act method. This method is called when the 'Act' or 'Run' buttons are activated in the Greenfoot interface. The method here is empty, and subclasses normally provide their own implementations.

*Animate*

*Animate*extends *Actor* and includes overarching methods for the movement of all moving actors (Pacman and Ghosts).

*Pacman*

*Pacman,* which extends *Animate***,** is directly controlled by the user. Pacman can eat Dot, BigDot, and ghosts (when hypeMode is in effect). Movement is controlled by user input using the up, down, left and right arrows. Pacman can be eaten by ghosts if hypeMode is not on. Pacman references Pacworld to increment score according to what he eats.

*Ghost*

*Ghost* extends *Animate*and is the parent class for all ghosts. It implements overarching methods for movement and for tracking the desired location. All ghosts move towards a given target location, with that target varying according to specific ghosts. All ghosts can only change direction at junctions and cannot double back (turn 180 degrees). Ghosts can eat Pacman unless hypeMode is on, in which case they can be eaten. In hypeMode all ghosts temporarily change their movement to run away from Pacman or “scramble,” and their images change to the dark blue DeadGhost. They spawn from the centre of the map at given time intervals, and return there if they are eaten. Once eating, they respawn according to the queue.

*PinkGhost*

*PinkGhost* extends *Ghost*. The pink ghost aims in front of Pacman’s current direction as oppose to its current location. Its desired location is four units in front of Pacman’s current path.

*OrangeGhost*

*OrangeGhost* extends *Ghost*. If Pacman is within a certain distance of OrangeGhost, OrangeGhost will use random movement. Otherwise, it will target Pacman’s current location (just like RedGhost).

*RedGhost*

*RedGhost* extends *Ghost*. It chases Pacman by targeting its current location and taking the most direct route to it.

*BlueGhost*

*BlueGhost* extends *Ghost*. Initially, the blue ghost tends to retreat to the bottom right corner. The blue ghost takes the position of both Pacman and red ghost to create a vector, which is then doubled in order to determine the blue ghost’s point of interest.

*Edible*

*Edible* extends *Actor* and is an abstract class for items to be eaten by Pacman (Dot and BigDot).

*Dot*

*Dot* extends *Edible* and is meant to be eaten by Pacman. In the class the image is set, and objects of the *Dot* class are referenced for score.

*BigDot*

*BigDot* extends *Edible* and its purpose is to temporarily change the status of the world to hypeMode when eaten. In the class the image is set, and objects of the BigDot class are referenced for score.

*Wall*

*Wall* extends *Actor* and it establishes the boundaries for other actors. The class itself just uses an image and is used as reference by Animate objects. Animate objects cannot move through walls.

*NotWall*

*NotWall* extends *Actor* and is an empty space which animate objects can move through. The class itself sets the image.

*ScoreCounter*

*ScoreCounter* extends *Actor* and increments and updates the score. The score is displayed in the upper right hand corner of the map and constantly updates (created in PacWorld).

*Gate*

*Gate* extends *Wall* is a special wall with different image that ghosts move through when they spawn. After the ghosts spawn, they cannot re-enter the gate. Pacman can never go through the *Gate*.

*StartSquare*

*StartSquare* extends *NotWall* and is the location Ghosts go to when spawning. It is directly next to the gate and next to the spawner for the Ghosts. Ghosts use a different act() method to get to *StartSquare* then use their respective movement algorithms. Ghosts can move through the Gate until it reaches *StartSquare*.

**GUI (Graphical User Interface):**

The GUI in this project is the equivalent to a screen in a gaming console. The map and characters (shown in the screenshot above) are created, and the actors move and interact visually as per user input.

**---------------------------------------------------------------------------------------------------------------------**

**Detailed Design**

The detailed specs for each of the classes in the Pacman game are provided through Javadoc documentation. Refer to Javadoc comments for in description of methods, instance variables, etc.

**---------------------------------------------------------------------------------------------------------------------**

**Testing**

We made a boolean tester class to test and print whether a method passed or failed. JUnit testing is not compatible with Greenfoot, so the tester class is an alternative. We tested the methods that incorporated GUI by running the program and making sure all the images were correct and everything was in the right place. We used a spreadsheet to document all the results of the GUI testing.