Reinforcement Learning for Pac-Man

Introduction

For this assignment you will implement Q-learning using feature functions and use them to drive Pac-Man through mazes to collect food dots while trying to avoid ghosts. You will be provided with simulation infrastructure in the form of a JAR file so you will not need to develop any graphics or simulation code. You will also be provided with API documentation in the form of PDF files for the classes in the JAR file that you are allowed to use in your agent program. You will also be provided with the three mazes.

Your task will be to develop a reinforcement learning agent that implements the Q-learning using a set of linearly weighted feature functions, as discussed in lecture. Your agent will learn the feature weights through a set of training episodes. The number of episodes (or epochs) will be a command line parameter that will be input to the program. You will need to determine appropriate values for: (a) the learning rate (alpha); (b) the future reward discount factor (gamma); and (c) the exploration factor (epsilon). The exploration rate will be fixed throughout all episodes. Exploration will be limited by the number of training episodes that will be specified by the command line input.

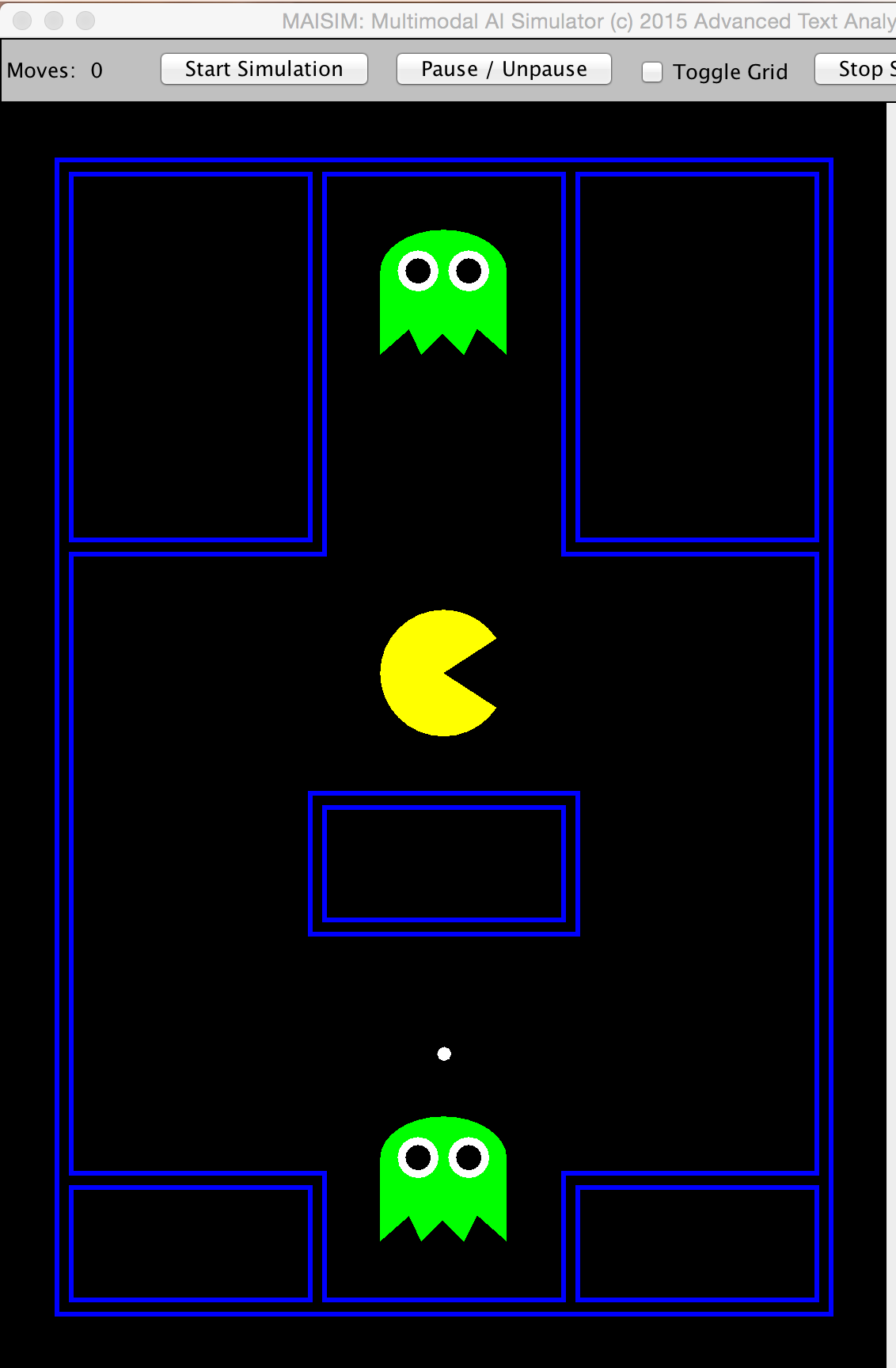
The agent program must run its learning algorithms during all moves of all training episodes, but when training is finished it must stop exploring and simply follow the policy that was learned.

Test Mazes

The following mazes are provided. The first one, "trap", is for development, and the other two, "game-small-1" and "game-small-2" are for testing. Your grade on this assignments will be based on how your agent program performs against the two "game-small" mazes. They are described briefly below.

**trap:**

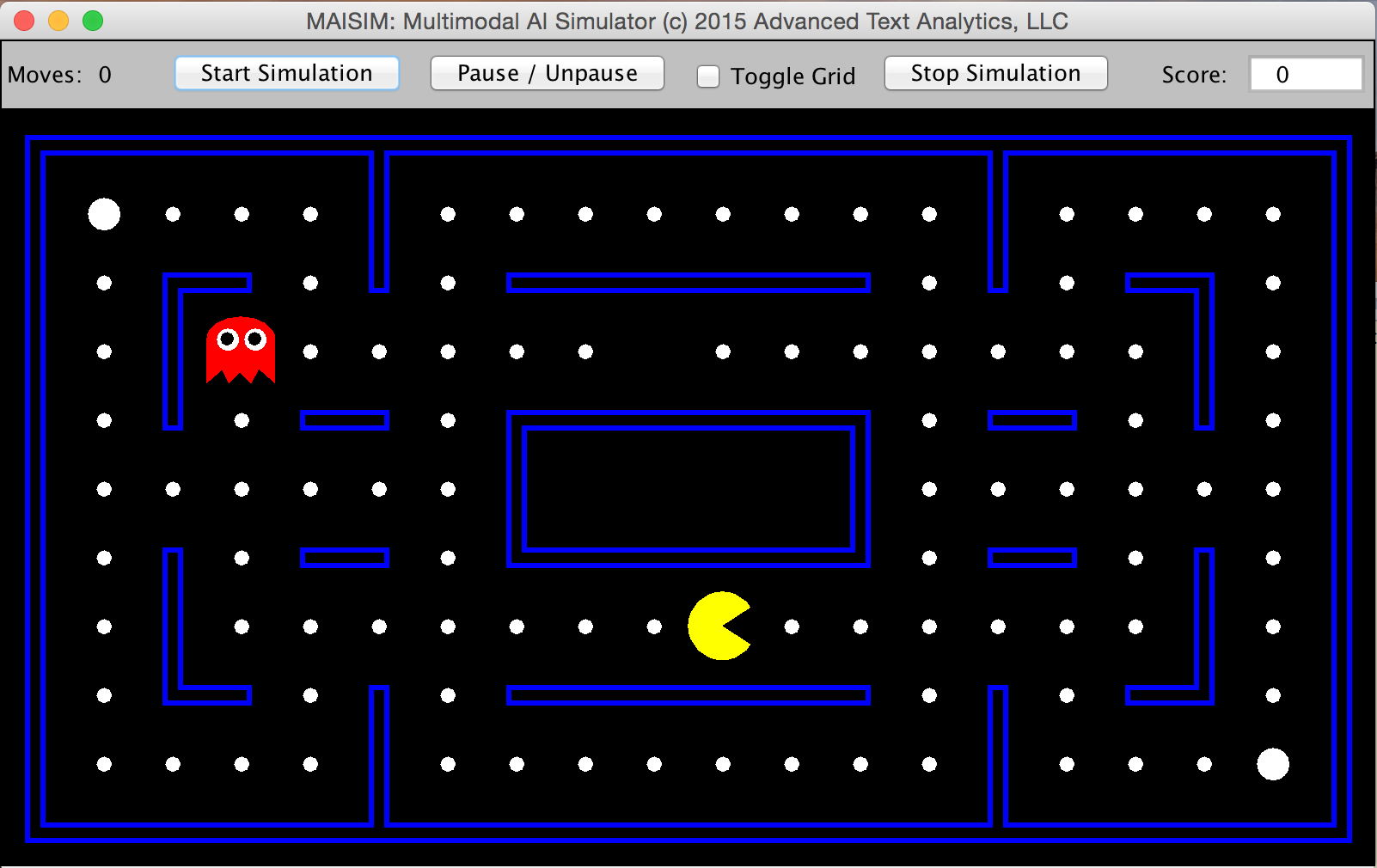
This maze has two ghosts that move randomly, but they surround Pac-Man. There is only one food dot and the goal is to get the food before Pac-Man is eaten by a ghost. This maze is for development testing and will not be used for grading. If your learned policy is successful about 40% of the time, that is great performance.



**game-small-1:**

This is a good size maze with only one ghost. The ghost is "Blinky.  Blinky alternates between SCATTER mode (for 7 turns), in which he targets one of the four corners of the maze, and CHASE mode (for 20 turns) in which he always moves to the adjacent cell that places him closest to Pac-Man's current position. Blinky starts in SCATTER mode. If Pac-Man eats a power pellet, then the current state is suspended for 20 turns and Blinky reverses direction, turns gray, and moves around randomly. After this FEAR mode suspension, the previous mode is resumed.

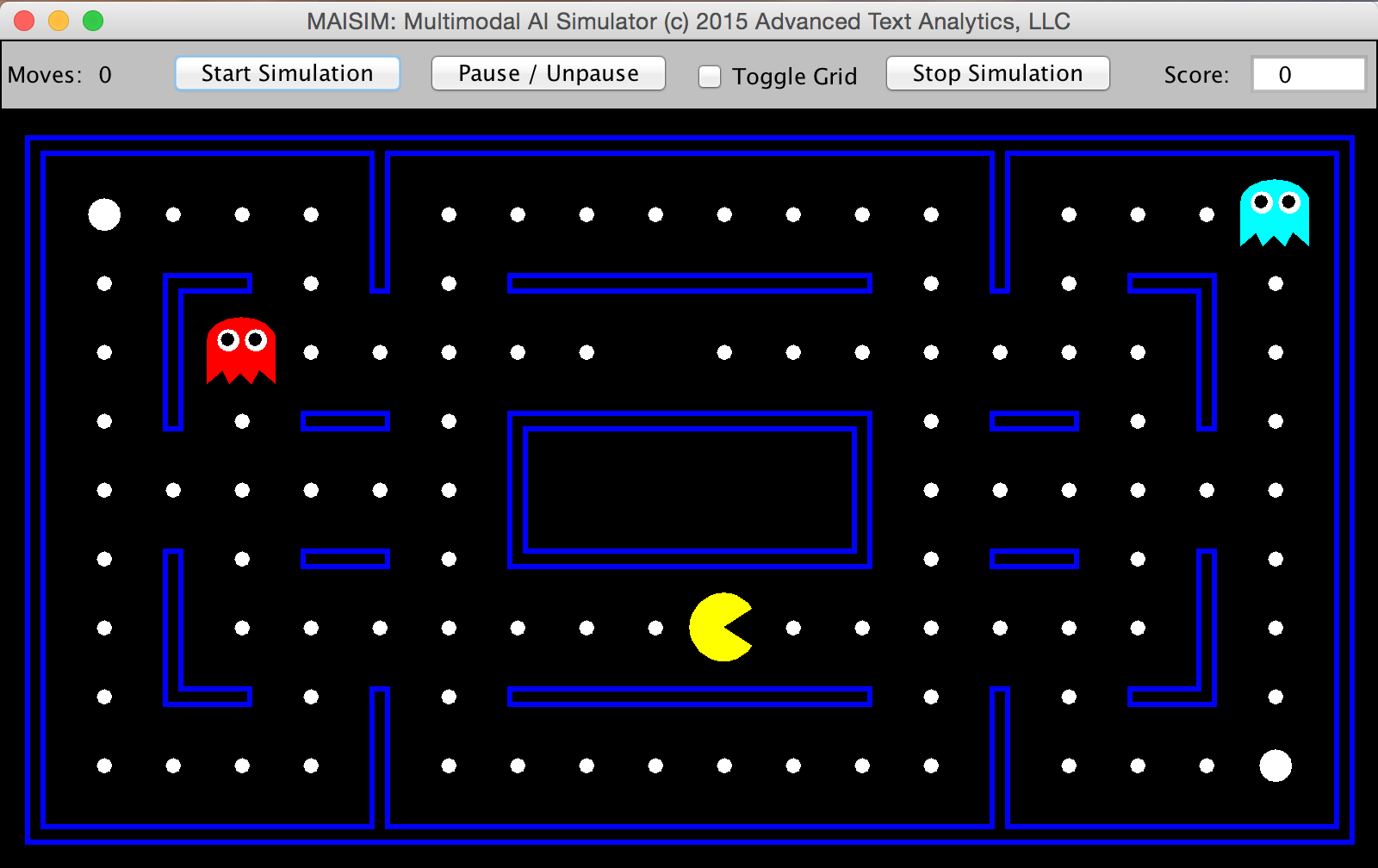
This maze will be used for grading. Your agent program should learn to avoid Blinky and eat all the dots. Success should be 100% for this maze.



**game-small-2:**

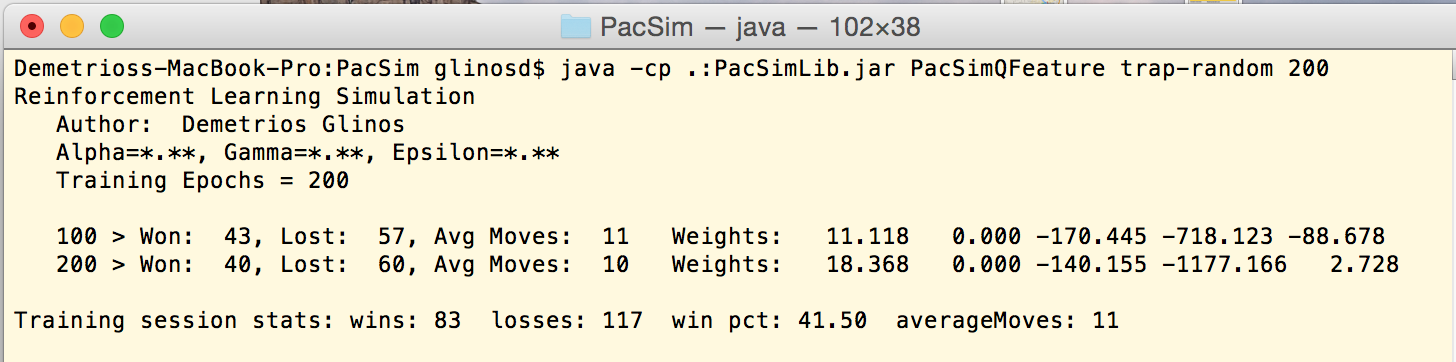
This is a good size maze with two ghosts. The second ghost is "Inky", who targets the cell that is at the position that is reflected about the point that is two cells in front of Pac-Man in the direction he is facing, and which is located at the same distance from that point, and in the same direction, as the line segment from Blinky to that point. The net result is that when Blinky is close to Pac-Man, Inky will generally not be far away. Mode switching for Inky is the same as for Blinky.

This maze will also be used for grading. It is difficult and a success rate of about 1.5% is OK. What matters here is whether it appears that Pac-Man can avoid multiple ghosts at all while still chasing after food dots. If you run your agent program multiple times and one of those times it can last for close to 100 moves in this gladiatorial arena, this should be considered success.



Required Output:

The program should report the key simulation parameter in the following format. An important part of this format is the reporting of the won-loss record and average move count for each hundred training episodes and the ending feature weights at that point. The feature functions themselves should be described in a brief, but complete, comment that appears as a header for your program file.



 At the end of training, the program should report the overall won-loss record and average move count across all training episodes.

Files Provided

[PacSimLib.jar](https://webcourses.ucf.edu/courses/1134955/files/48479106/download?wrap=1) - this is an enhanced simulation engine that should completely replace any previous version used.

[trap-random](https://webcourses.ucf.edu/courses/1134955/files/48480108/download?wrap=1) - this is the development test maze

[game-small-1](https://webcourses.ucf.edu/courses/1134955/files/48480091/download?wrap=1) - test maze with 1 ghost

[game-small-2](https://webcourses.ucf.edu/courses/1134955/files/48480098/download?wrap=1) - test maze with 2 ghosts

[PacSim.pdf](https://webcourses.ucf.edu/courses/1134955/files/48479411/download?wrap=1) - Javadoc for the PacSim class with boolean methods pacmanEaten() and ghostEaten()

[PacUtils.pdf](https://webcourses.ucf.edu/courses/1134955/files/48479169/download?wrap=1) - Javadoc for the PacUtils with some new methods that may be useful

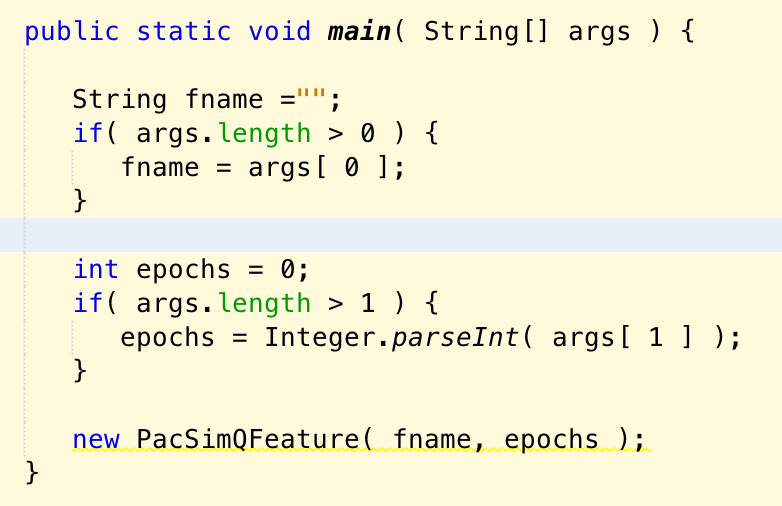
Programming Considerations:

As in the previous program assignment, your program must be written in Java and must implement the PacAction interface. Your action() method will drive Pac-Man by returning the PacFace direction in which Pac-Man should move for the next turn. The action() method will be invoked during both training and testing, but the GUI will be suppressed so the training will proceed much more quickly.

There are several special considerations for training that must be taken into account:

**1. Command Line Input**

The number of training episodes will be variable during test and will be determined by a command-line argument to the main() method, which should look something like the following:

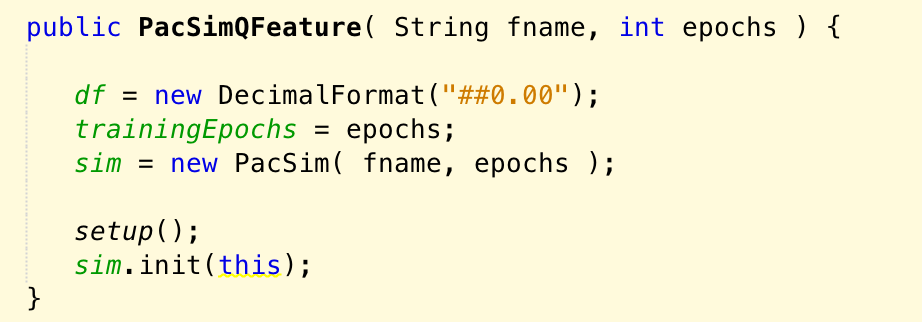


The training episodes parameter should be passed to the constructor for your class.

Please note that if the training episodes value is missing, your program should use a value of zero so that the simulation engine will immediately run in simulation mode. This mode should be backward-compatible with previous programs for Pac-Man that do not involve training.

**2. Initialization:**

The init() method that is required by the PacAction interface will be called before every training episode and also before the simulation run at the end of training. Therefore, if there is any data structure initialization to be done, it should be done in a separate method (e.g., "setup()") that is called from the class constructor.  An example of how to place the call to setup() is shown here:



Please note that the training episodes parameter that is received from the main method should be passed to the simulation engine as its second input parameter.

What to Submit

You should submit a single Java source file (.java, not .class). This file should contain your Q-learning agent class and any supporting classes as well. Do not put any support classes in separate files. Simply do not declare them to be public and just add them to the bottom of your agent class file.

Your source file should include a header identifying UCF, this course and semester, and identifying the program author. The header comments should also describe your feature functions in sufficient detail that can be replicated by the reader without reading your program code.

The program must be submitted on Webcourses. Email submissions will not be graded.