CPSC 481 Final Project: Improving PAC-MAN

Project Lead: Michael Romero
Miles McCloskey
Diego Franchi
Date: 5/8/2018

Baseline Score: 100
Personal Score: 1/3, 1/3, 1/3
Algorithm Multiplier: 1 (Reinforcement Learning / qLearning, A*, Heuristics)

The main idea behind our implementation of PAC-MAN A.I. was to strive to be close to the original behavior of the game with respect to the logic of the original game's ghosts, while taking advantage of modern path-finding and decision making algorithms.

At the start of this project, we had to spend a significant amount of time researching Machine Learning, as it would be weeks before it was covered in class. We eventually discovered reinforcement learning and decided that was an appropriate algorithm to strive towards. We then had to spend a significant amount of time reading through the PAC-MAN codebase to learn how everything interacted in order to determine how best to inject our code into the existing codebase.

We started by taking care of some aesthetic issues, i.e. color of the ghosts were not matching the original, and none of the provided game maps matched the original either. We then created a custom map, and added getters/setters for ghost colors.

The ghosts which came with the demo were incredibly bad at hunting PAC-MAN. We needed to improve the ghosts, and decided it would be good to follow the original ghost behaviors from the classic PAC-MAN arcade version. We have added the terminal options of '-o' and '--originalGhosts' in order to trigger each of the four ghost agents we created which each follow the behavior of the original ghosts Blinky, Pinky, Inky, and Clyder.

Each ghost utilizes the A* algorithm and a Manhattan distance heuristic for path-finding. They also use unique logic for each ghost that controls the targeted tile.

We added the ability to display a visible path matching the color of the ghost it belongs to as a way of troubleshooting our targeting algorithms. This feature is also simultaneously an easy way to demonstrate to the class the target of each individual ghost and the intended path towards its target.

Towards the end of our project, once we were content with how deadly our ghosts had become, we decided to switch our development to creating a more intelligent PAC-MAN that could automatically play against the ghosts we created. We created a set of 'features' that drive the behavior of PAC-MAN, i.e. the distance to the closest ghost, whether the next state results in a higher score, whether a pellet was eaten, etc. There are many, many additional features that could be added to drive PAC-MAN's behavior. We settled on five.

Once PAC-MAN had a decently capable set of decision points, we decided to implement a reinforcement learning method called "QLearning" that dynamically modifies a set of weights based on whether or not a given movement is the most responsible for the largest difference in state reward values. If the result of a given weight, multiplied by the function's return value is found to cause the greatest difference (both negatively or positively), that particular weight is "blamed" for the difference, its weight adjusted by ".01". Added a "--reinforcementLearning" command line option, which requires a set of initial weights (5 floats, comma separated) that it then uses to run the game. Each state transition when "—reinforcementLearning" is specified causes our CPSC481Agent to execute an "adjustWeights()" function which updates the weights as described above, storing the value on the GameState object, that is then carried on to subsequent GameState objects, and eventually returned when the game ends. It is then passed in as an argument to the following game as its set of starting weights. Thus, each subsequent state, and each subsequent game is capable of receiving the adjusted weight values and at some point will hopefully reach equilibrium of perfect weights.

We may not have reached that equilibrium, however, we got pretty close :D

Files Modified:

commands.txt
Created commands that initialize our Ghosts in the game. This document contains example executions of the program and how we are running the game with options that we have created

Trigger our Ghost Agents... Blinky / Pinky / Inky / Clyde, using the pacmanClassic map we created.
```
python pacman.py -l pacmanClassic -o
```

Allow PacMan to play against Blinky and draw Blinky's intended path....
```
python pacman.py -l pacmanClassic --blinky -z .5 -d -p CPSC481Agent
```

PacMan play's against all four ghosts, not drawing intended paths.
```
python pacman.py -l pacmanClassic -o -z .5 -p CPSC481Agent
```

Provide some weights to start reinforcementLearning from... play 100 games quitely with CPSC481Agent against Blinky
```
python pacman.py --blinky -p CPSC481Agent --reinforcementLearning 1,1,1,1,1,1,1 -l smallClassic -n 100 -q
```

Watch a fairly intelligent PAC-MAN completely fail against all four of our ghosts
```
python pacman.py -o -p CPSC481Agent --reinforcementLearning 1,1,1,1,1,1,1 -l pacmanClassic
```

ghostAngents.py
Created separate classes for each AI ghost to implement the classic ghost AIs from the famous PAC-MAN game. These ghosts currently use a state space search to minimize the distance between PAC-MAN and the ghost itself using all legal paths. We currently use an A* algorithm with a Manhattan distance heuristic for path finding. Each ghost class is capable of drawing a visible path to their target location for presentation and debugging.

graphicsDisplay.py
Modified color of ghosts to better match the original PAC-MAN game, created functions that allowed us to set ghost colors, and a function that allowed us to visibly draw a provided path, which takes in a list of directions and a color as input.

pacman.py
This file controls the creation of the game. We have modified it to instantiate our ghost classes which use AI modeled after the original PAC-MAN's ghost behaviors. Our ghosts are triggered using the "-o" command line argument. We also added a flag -d to display the paths ghosts intended targets. Also, we made it so you can play against one or as many of the ghosts that you would like to play against, specified in ARGV via --blinky or --pinky, etc. Lastly, we added --reinforcementLearning in with a set of 5 weights. Added getBetterPacmanPosition().

searchAgents.py
Added a function that returns the set of directions from start location (x1,y1) to end location (x2, y2). This is converted by the ghost class into a series of cell coordinates representing a path that is then drawn onto the screen displaying the ghosts intended path towards its target.

multiAgents.py
We added CPSC481Agent class to drive the behavior of PAC-MAN. CPSC481Agent can be used in conjunction with "–reinforcementLearning <set of weights>" to initialize reinforcement learning. These set of weights dynamically change the behavior of PAC-MAN's decisions based on the location of the ghosts, food, power pellets, and more.

PAC-MAN now uses getBetterPacmanActions() which removes the "Stop" direction as an option, making him seem more aggressive.

game.py
Modified to allow for the setting and the getting of weights which are being used in reinforcement learning. Ensuring the weights are always kept on the game state. These changes are allow being made when the "--reinforcementLearning" command is being set or called in the terminal. Added a getBetterPacmanActions() function which removes his ability to stop, and makes him seem more aggressive.

search.py
We created the A* search ghost function which is used for path finding

Files Created:

pacmanClassic.lay
This is a layout file built to match the classic PAC-MAN map. We created this as a way to make this version of PAC-MAN to feel more like the original.

pacmanClassicNoPowerPellets.lay
This is a layout file built to match the classic PAC-MAN map, with the power pellets removed. This was used while testing values for our CPSC481Agent's reinforcement learning algorithm. At one point, we needed the ghosts to stop being frightened so we could tune PAC-MAN's behavior to better evade the ghosts.