AI Coursework Report - CW 1 Johannes Janousek

Introduction

To overcome the challenges of the first course work I built an agent using versions of the breadth-first-search algorithm. The relatively small size of the environment and the finite possible actions allowed me to use a relatively computationally expensive algorithm. Building an BFS-agent seemed like a solid and effective way of tackling the functionality tests.

Strategy

The agent's strategy consists of three BFS algorithms, of which two are concerned with finding nodes with food and one is concerned with finding nodes with ghosts. The overall strategy is to avoid using the same paths multiple times, so visiting all the food nodes in an efficient manner, and to just use minimal evading manoeuvres for the ghosts.

Firstly, Pacman checks if ghosts are visible or audible. Should there be ghosts nearby, the ghost-BFS constructs a path and passes the first node that would lead Pacman closer to the getAction function. Here however, Pacman will move into the opposite direction of the given node. If the opposite direction is not possible because it lies within a wall, he chooses one of the other two remaining directions at random. In this situation a future performance improvement would be if Pacman chooses a node that is either food or on the way to food as an evading path from the ghost which would reduce the frequency with which Pacman travels the same paths.
Should there be no ghosts in detectable reach, Pacman looks for the closest food or capsule node. The food-BFS constructs a path and passes the closest node to the getAction function. It works exactly as the previously described ghost-BFS only that Pacman moves into the actual direction of the given food node.
Once all the food in Pacman's sight has been consumed, Pacman switches to the third BFS which looks for the closest way point - a grid point that is not a wall-coordinate and that has not been visited so far. Pacman has an internal memory that removes way points that have been visited during the current game. A future improvement for this strategy could be a second internal storage that remembers all the food nodes Pacman has seen but not eaten and prioritises those over the (potentially empty) way points. This would reduce the amount of times Pacman travels to the middle point of the map where the ghosts spawn in the mediumClassic map as these coordinates have no food and are a great trap for Pacman getting caught by ghosts.

Methodology

My agent consists of six components. An internal storage for unvisited nodes, three versions of the breadth first search algorithm, a function that constructs a path and the getAction function which interprets the result from the BFS.

In an earlier version of my agent, I paired a single BFS with a corner seeking agent. This was an agent that was successful almost 90% of all the games played on mediumClassic without ghosts. However, when this agent was put into an environment with ghosts that success rate dropped to 10%. Essentially, how it worked was that once the BFS could not detect any food nodes or ghosts, the agent switched to a corner seeking strategy. This worked fine without ghosts as Pacman could just rely on a heuristic approach and eventually stumble across all the food nodes. However, this method caused games to last significantly longer and as a result exposed Pacman for a longer time to the risk of colliding with a ghost. When I took the corner seeking agent out and instead implemented the internal storage for unvisited way points which I then fed into a BFS, the amount of times Pacman got caught by a ghost decreased significantly. He even won a couple of games on the tricky map with four ghosts.

Creativity

- 3 breadth-first-search algorithm that successfully switch on depending on the situation within Pacman's environment
- The way points: collect all the way points of any given map in an internal storage for Pacman to visit and remove visited nodes from the list.

Results

In my first test (Test 1), Pacman went through 100 games on the mediumClassicNoGhosts map. The purpose of this test was to check the functionality of the food-BFS and way-points-BFS.
In Test 2, I tested Pacmans performance in the same map with ghosts. Here, I wanted to see how effective the interplay between waypoints-BFS and ghost-BFS is. Pacman played this map for 1000 games.
In Test 3, I wanted to put my agent into a completely new environment which I have never seen when building the agent. My agent played 100 games on the trickyMap. The aim of this test was to test the agents performance in a completely new environment with two additional hostile agents.
In Test 4 I wanted to test the impact the 'def final' function has on my win rate. Pacman played 100 games on mediumClassic without the final function in his code.

Analysis of the results

Test 1 - Win Rate: 100/100
Test 1 proved to be an uninteresting test as there can nothing be commented besides that the breadth first search algorithm in combination with the way points storage is working.

Test 2 - Win Rate: 834/1000 - Avg Score: 1213.477
This test proved that the waypoints-BFS in combination with the ghost-BFS works significantly better than a BFS in combination with a corner seeking agent.

Test 3 - Win Rate: 39/100 - Avg Score: 709.03
Here, I was actually surprised that the agent won almost 40% of his games. It showed that the agent performs relatively well in a completely new environment.

Test 4 - Win Rate 32/100 - 605.04
The fourth test shows that not resetting the init variables after each game has a disastrous effect on the overall performance. With the function in the code, Pacman had a win-rate of 83.4%. Without, it dropped to 32% which is lower than the agents performance on the trickyClassic map.

References

Ics.uci.edu. (1996). *BFS and DFS*. [online] Available at: https://www.ics.uci.edu/~eppstein/161/960215.html [Accessed 28 Oct. 2018].

Redblobgames.com. (2014). *Introduction to A**. [online] Available at: https://www.redblobgames.com/pathfinding/a-star/introduction.html [Accessed 28 Oct. 2018].

Velardo, V. (2017). *How to Implement Breadth-First Search in Python*. [online] Python in Wonderland. Available at: https://pythoninwonderland.wordpress.com/2017/03/18/how-to-implement-breadth-first-search-in-python/ [Accessed 28 Oct. 2018].