Automated Pacman Game – Report

Introduction

Within this project I have created a Pacman game that is fully automated, I have achieved this by making a reactive agent that will consider various sensors within the world and then make decisions based on what sensors are activated. I created this type of agent due to the low processing power needed and a reactive agent was the most intuitive way for the agent to navigate the world by reacting to how it was changing and make a decision for movement based on what it could see in front of it.

Description – Strategy

To complete the first coding task, making the pacman can win the game without any ghosts, I decided that the pacman should traverse the map by chasing the closest piece of food until all the pieces of food have been eaten and the win criteria is me. To do this I accessed the X and Y positions for the food and compared them to the pacman's current location on the map, if the values aren't the same it will move towards the first food item on the list. This list is separated into a list of X axis values and Y axis values, so the agent will first check if there is any food pieces in the X axis it will move towards it, if it cannot find any food in the X axis it will instead look for any food on the Y axis, if it cannot find any on either axis it will make a random move until it finds any food on either plane. If pacman finds himself looking for a place to go it will use this strategy because it will be an effective way of making sure that all the pieces of food have been eaten and in turn winning the game.

At each point in the game the pacman is checking to see what moves are legal, for example, if the pacman reaches a wall on the far west side of the map it will change the value of Directions.WEST to illegal meaning if the agent has to make a random decision on movement then it will not consider a move in the west direction.

For the second coding task the overall strategy was for the pacman to move any way that it could meaning it would be further away from the closest ghost on the map. To do this the agent needs to have to access the distance from the ghost at each point, its next move, its last move and all the legal moves available to it. I will get the distance away from ghost from the sensing agent, the next move will be calculated by getting the current position and adding or subtracting an X or Y value depending on the direction it is moving. The key to this strategy is knowing what the legal moves are, it is important to remove the last move from legal moves so reverse the action of getting closer to the ghosts. To utilise these variables, I will check if in the pacmans next position it sees a ghost, if it sees a ghost it will make a change in direction to increase to the distance from the ghost.

At the starting point the pacman has not got a variable for "last distance" so I set the value to X, this means as soon as it sees a ghost it will replace the x with what the distance is currently – meaning is there is a move that decrease distance to ghost it will change direction on the next move. If pacman doesn't see a ghost is will have the same strategy that was laid out from the first coding task, except that it is now constantly checking for ghosts given its next position on the map. Given the next position on the map sees a ghost in its path the move which decreased the distance to the ghost will be removed from the legal choices, thus increasing the distance from ghost and allowing the pacman to continue the search for food.

Description – Methodology

For the task of winning the game without ghosts I originally set the agent to only check the X axis for food items, if there was none for the agent to find then it would produce a random move. I changed this to check the Y axis before making a random movement as this will reduce the number of random moves the agent makes thus giving the agent more purpose and completing the task quicker than before.

When creating the agent that can win the game when there are ghosts I started by using the same basic movement as the agent outlined earlier but now when the agent sees a ghost in its vision it starts moving towards a corner, when it reaches the corner it will 'tick' the corner off the list so the next time it sees a ghost it will move towards a different corner this is because the agent will clear the food quicker than if it returned to the same corner each time it saw a ghost. However, when I tested this methodology I found that it was still running into ghosts on its way to a corner and as the order of the corners had been hard coded the agent would not react to the environment and often die on its way to the corner when there are two ghosts in the game.

The next method I tried for completing the game with ghosts was to reverse the pacman movement when it saw a ghost e.g. if the pacman saw a ghost it would make a move in the opposite direction it had just moved in. This methodology had very limited success. To develop this idea further I changed the hard coded change of direction into a more automated agent by accessing the pacman's last position (*self.last*) and removed this from the legal choices. After removing the last choice, I asked the pacman to make a random choice that was legal (now not containing the last choice it made), this was more successful than previous methodology most of the time but crashed when the pacman was chase around the corner (giving the error ***legal.remove(self.last[x]) -> x is not in list***).

After these failed attempts with just accessing if the pacman sees ghosts in its view I decided to base pacmans decision on if it should move away from the food searching method on if the pacman was getting closer to a ghost, compared to the distance in its last position. To do this I created a variable called "distanceFromGhost", using the method "util.manhattanDistance" to compare pacman and the ghost location. This method was producing wins but still not as often as I required as the pacman could not see any ghosts that were coming from around a corner, so would often die traversing the map. To fix this problem I predicted what the next pacman position would be based on the direction it was travelling, for example, if the pacman was travelling east then the next pacman position would be its current X value plus 1 (as the pacman's X coordinate will increase by one if it is travelling east). This allowed me to pretend the pacman was in its next position and if it could see a ghost it would then change away from the food searching method. Again, this was producing more wins and nearing consistent results but sometimes would struggle to make a decision when it came face to face with a ghost, the final additions to the code I made was to make the agent choose a new direction if the distance to a ghost was less than 2 although this initially failed as it would leave the pacman with no choices, I added a check before the ghost removed a choice that there was a legal option if you removed this one ("if len(legal) > 1").

Tests

The tests I conducted throughout the process of creating the game range from testing the ability to move without errors, to increasing likelihood of success with ghosts. Test 1 is Pacman playing the game on layout "mediumClassicNoGhosts", this test is looking to see if the pacman can navigate around the map successfully without any errors. Test 2 is also played on layout "mediumClassicNoGhosts", this is testing the efficiency of my food searching method and that it

completes the game within a reasonable time. Test 3 is played on layout “mediumClassic”, this test is looking to see if the pacman will move towards a corner if there is a ghost in its view and how often the Pacman achieves victory. Test 4, played on the same layout at test 3, is testing to see if the pacman will reverse its movement when it sees a ghost in its view and in turn charting how often it achieves a victory. At this point in my testing I realised my final strategy for the pacman to be successful in a game with ghosts so test 5, also played on “mediumClassic”, is checking that the variable “Last distance from Ghost” and “Distance from ghost” is updating based on my predicted next position for the pacman. I am testing these variables as they are the key figures that decide if the pacman should run away or not. Test 6, played on “mediumClassic”, is testing to see if the pacman acknowledges the decrease in distance from the ghost and tries to increase distance from ghost and the amount of times a win is achieved. Test 7 is testing an improved model of Test 6.

Results

For Test 1 the results are as follows, when running the agent in any layout pacman can navigate through it successfully without crashing, thus we can conclude that the agent is smart enough to recognise walls and to remove any illegal options from its next choice.

For Test 2, I ran the agent with my food searching algorithm, the pacman completed the game but not in a suitable amount of time. With this considered I asked the pacman to check its Y axis after it had checked all of its X axis, meaning I am reducing the number of random movements, increasing the efficiency of my method. When we can see there is food to be eaten in the game, then pacman will move towards it, from this we can conclude that the game will be won by pacman eventually as it will not stop until all the food has been eaten and win state has been achieved.

For Test 3, having successfully implemented a ‘corner seeking’ agent into the code pacman, on seeing a ghost, will now move towards a certain corner of the map if it has not been to that corner before. As you can see in Table 1 the amount of victories using this method were very low. So, when the agent seeks a corner on viewing a ghost the chances for success are very minimal as it will often run into a ghost on the way to a corner.

Having seen the results of test 3 I changed my strategy for the agent, now believing that if the pacman reversed his movements until not in sight of the ghost anymore the agent would be able to win the game. Test 4 was unsuccessful on my first try as the code I implemented would only change the pacmans direction for one movement, after the first movement it would often try to revert to the direction it had just come from as it could see a piece of food in that axis. After the initial failure I hard coded the change of direction depending on the direction the pacman was heading in before, but this still had the same effect as the initial test. After these two options failed I changed the strategy, now removing the last choice the pacman had made from the legal list of moves and made it take a random choice from available moves. This method was more successful, but as you can see from Table 1 this was still very infrequent and should be improved. We can see that when the pacman makes any choice but its last when in view of a ghost it will have limited success, so we can conclude that this is not an efficient way to navigate around ghosts because it will often die when it cannot see a ghost directly coming from in front of it.

The next strategy that I used to help the pacman successfully navigate the map without running into ghosts was checking the view the pacman has in the next position it will be in and that this distance is further away from a ghost than in the last position. This test wasn’t successful when first implemented as the last distance wasn’t updating and was stuck on whatever the initial value for the last distance, to fix this I changed the initial value for ‘Last distance’ to the string ‘x’ this meant that

that first value it saw would be the initial value rather than the value of 100, which I had set initially. When running successfully the output on the console looked like what you can see in Figure 1. We can see that when the pacman sees a ghost getting closer to it, it will output the next distance it predicts and the last distance, thus we can conclude that pacman will be able to see the distance to the ghosts, update and store these values so they can access it when deciding whether to divert from the food searching method.

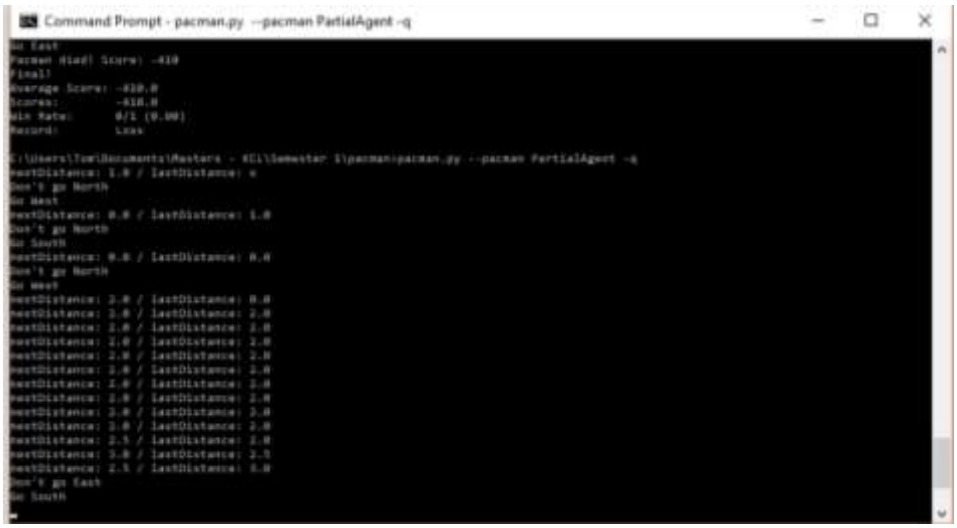

Figure 1 - Output when successfully updating 'next distance' and 'last distance'

The last two tests were to test that pacman will move away from the ghosts if it sees a ghost in the next position, this test failed at first because the pacman was still running into ghosts if it came face to face with a ghost, when I added a check for a ghost within 2 spaces the code ran successfully. We can see that when 'Distance from ghost' is greater than 'last distance from ghost' OR when the 'distance from ghost' is smaller than 2, pacman will make a random movement that is anything but its last choice. So, we can conclude that pacman will win at least 1 in 5 games because when the pacman sees a ghost it will run away from it and continue the search for food.

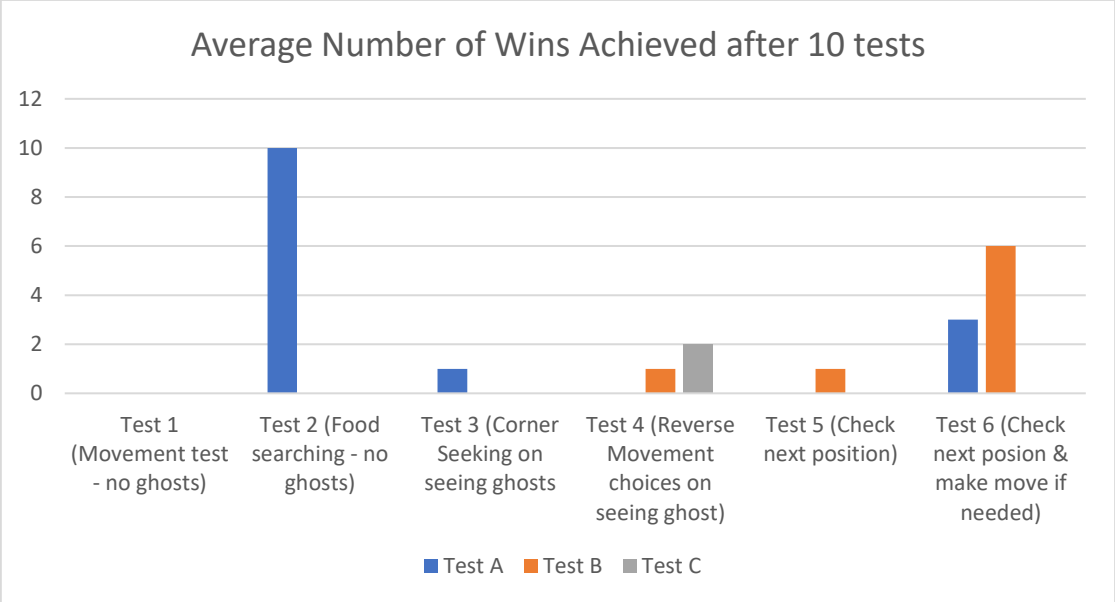

Table 1 - Average number of wins per test

Test Name	Intention	Outcome	Pass / Fail	Comments	Changes
Movement Test – No Ghosts	Pacman to move through map, changing direction when running into a wall	Pacman moves successfully without any system errors	Pass	N/A	N/A
Food Searching – No Ghosts	Pacman to chase the food spread across the map	Pacman chases food.	Pass	Successful but takes excessive time to reach win state	Add searching along Y axis
	Pacman to chase the food spread across the map	Pacman chases food along X axis, then Y Axis	Pass	N/A	N/A
Corner Seeking on Seeing Ghost	Pacman to run into the corners on seeing a ghost	Pacman runs into a corner when seeing a ghost	Pass	Pacman can run into ghosts on the way to a corner, very rarely achieving win state	Change strategy
Reverse Choice on seeing Ghost	Pacman will make reverse his movements on seeing a ghost	Pacman reverses only one movement when seeing ghost	Fail	As Pacman only reverses most recent move, will often turn back into ghost if there is a piece of food in that axis	Hard code change of direction
	Pacman will make reverse his movements on seeing a ghost	Pacman still reverses only one movement when seeing ghost	Fail	As Pacman only reverses most recent move, will often turn back into ghost if there is a piece of food in that axis	Remove last move from legal choices.
	Pacman will make random choice for movement, but cannot go the same way it just moved	Pacman moves away from ghost	Pass	As Pacman only makes random choice away from ghost, although does not see ghosts in around corners	Change strategy
Checking Next Pacman Position	Console will update the distance to ghost based on predicted position and store the last distance it saw a ghost at.	Console updates distance away from ghost, but last distance remains at initial value	Fail	Pacman will not move away from ghosts if this value doesn't update as it will not think it is getting closer to a ghost	Change initial value
	Console will update the distance to ghost based on predicted position and store the last distance it saw a ghost at.	Console updates both values	Pass	N/A	Add random choice apart from last choice to move away from ghosts

Checking Next Position & Making Move if Needed	Pacman will move away from ghosts when it can see a ghost from its predicted next position	Pacman sees a ghost from around the corner but does not react when face to face with ghost	Fail	Pacman will not win games if it encounters a ghost directly in front of it	Add a check for any ghosts within a certain distance
	Pacman will move away from ghosts when it can see a ghost from its predicted next position	Pacman makes random movement away from ghost if it can see a ghost	Pass	Pacman will now win on average 2 out of 5 games. The number is not greater than 2 because it only access one of the ghosts location, if there is two ghosts in close proximity it will only react to one, meaning it may turn into the path of the other ghost	N/A