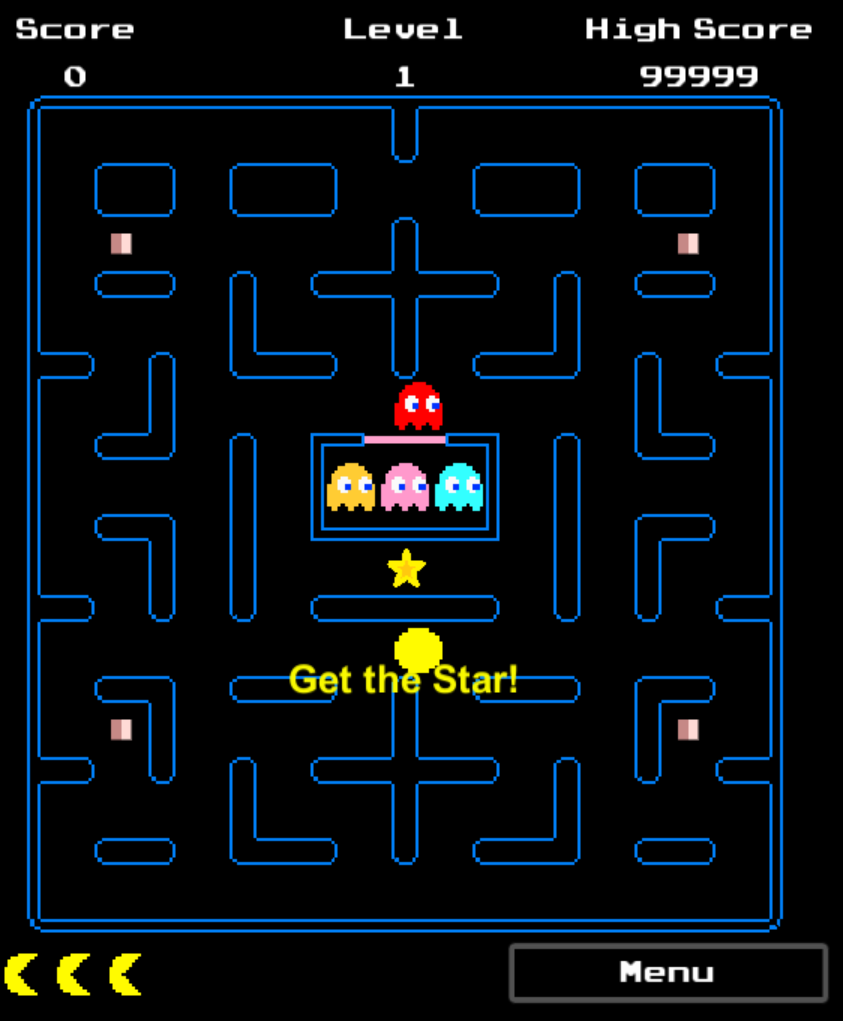
Pac-Man: Search Algorithm and Machine Learning



Final Report

Introduction to Artificial Intelligence

Dr. Chuan

Team: Ana Garcia, Elizabeth Moreno, Michael O’Donnell

Abstract

The objective of this project to create an implementation of a searching algorithm for an enemy behavior and have a machine learning application as well. A searching algorithm is widely applicable to a game environment when there is a player vs. environment situations. These applications generate more fun tension in game play, but also have other applications such as Google-Maps, GPS, shortest path length, and with weights can be used for a knapsack problem. In order for machine learning to be effective there must be a training phase in where the enemy goes from knowing nothing to connecting ideas and adjusting behavior based on the learned information. In this situation we will be placing an enemy in an unknown environment, let it discover its environment then find a path to the player.

Introduction/Motivation

For this project, we planned on making use of the different search algorithms presented in class and represent and demonstrate them through the ghosts in our own take on the game Pac-Man. Every member of this team is part of the Gaming and Application Development Club here at UNF and we are currently working with the Unity 2D engine in order to develop our own version of the game Pac-Man using a different map, sprites, and scripting for the game. With this in mind, we decided to go ahead and base our project along this line of work in order to get more experience using the Unity Engine and how to go about working with the scripting and coding necessary to achieve a rather difficult and machine learning AI for the ghosts, or your equivalent sprite, in the game. With this experience working on this project and focusing on the artificial intelligence, we can hopefully have a better working knowledge of how to go about working with Unity 2D and C# scripting/coding for the game.

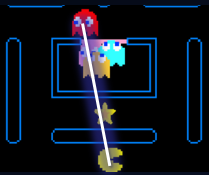
Related Work

In order to aid us on this project, in that we have never really fully used Unity 2D before or really get into C# that much in the past, we relied on a Unity Pac-Man game tutorial online to steer us in the right direction for the game portion of the project (Noobtuts - Unity 2D Pac-Man tutorial). It gave us a step-by-step instruction on how to go about setting up the map, the colliders for the map onto all the walls which has to be done manually, set up the sprites with their positions and animations and movements (including movement animation), and pac pellets, or energizers, in the corners of the map. Upon further research, we discovered that it is not known exactly what AI was programmed into the original Pac-Man for the ghosts, so a best guess is made. In our case, we searched online for an idea on the implementation for machine learning and A\* search to the ghosts in Pac-Man. What we found was to implement a target for the ghosts to get to, in this case using A\* search through every possible direction at an intersection in order to find the quickest and shortest route to reach the target. It was also mentioned that at every intersection it would check a direction, and if that direction was into a wall, it would mark that direction as so, thus learning not to take that direction and re calculate which direction to go to and pick one that wasn’t a wall and that led to Pac-Man in the shorted route possible. The important thing to note is that the original game’s AI is not known, so it is up to us to make our best effort in order to understand how it worked and be able to implement our own take on it using these different algorithms and processes.

Methodologies, Algorithms, Approaches:

Enemy AI

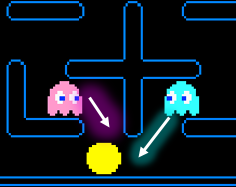
***A\* Search - Blinky***

The A\* search is composed of two parameters, a heuristic value and a recursive function. The A\* search equation is given as: **f(n)** = **g(n)** + **h(n)**, where **g(n)** = cost to reach state and **h(n)** = heuristic. It is a complete algorithm, meaning that if there is a solution it will find one and it is optimal when h(n) is admissible.

A suitable heuristic value for this experiment was a function called **Vector3**(x,y,z) in the Unity class. The function returns a numeric value that is a linear distance between the object specified and the player. It’s adaptive to both the 2D and 3D environment. In order for the A\* to fully work the enemy must have full knowledge over the map before it begins moving, to simplify the complication the map was turned into a 2 Dimensional array in the enemy’s data, that way it would be able to calculate the path dynamically without having to check with the unity colliders. Rigid bodies are not supposed to go past colliders, and if the script isn’t properly handles with proper functions and way points. The AI will begin moving every single iteration of its calculation.

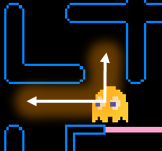
***Ambush - Inky and Pinky***

Having the complete A\* algorithm completed ahead of time allowed the designers to mess around the other enemies AI’s. Under testing phase, it was noticed that if all enemies take the same path to chase the user the game becomes simple and not challenging. In order to prevent a bring trail of enemies behind you, the heuristic values of Inky and Pinky were modified to over and underestimate the location of the user. This will create and ambush behavior since they will both take different paths.



***Random -******Clyde***

Was originally meant to be the machine learning algorithm, due to faulty implementation he was changed to pick a random direction on a 2D plan at an intersection. Meaning he will keep moving in his chosen direction until he encounters an intersection that allows him to move in another direction.



***Machine Learning***

Machine learning is an application where through training exercises, and series of test runs the machine will be able to learn more about it’s environment and be able to adapt accordingly.

For our implementation a ghost will be given a location where the ghost will attempt to move right, left, up and down. The information will be stored in a 2D array and for every move, will record if it successfully moved. There were two approaches in this implementation. The ghost will attempt to move at a certain direction and if it interacts with a collider, then it will remember the coordinate position and record it in it’s 2D Array that it could not pass through.

|  |  |
| --- | --- |
| Value | Meaning |
| 0 | Wall |
| 1 | No Wall |

This proved to be a faulty approach because there was no clear interaction between the Script of the colliders on the enemy and the colliders in the environment. Not only that, the grid on the engine is made up of doubles, so moving one unit wasn’t really moving one unit the engine. The dimensions on the map had to match properly, in on our case didn’t since we had a premade made designed already.

It was then supposed to use this data to find a path towards the player. The implementation proved to be a failure, and caused a lot of bugs in the game while testing, such as the ghost not remembering what it learned upon level reset and the ghost not properly reading a collider causing it to move past the collider when executing the script. The programmers had a backup plan, by hard coding the maze into a 2D array there will be a known map and a learned map, by comparing the values of these two arrays the machine will eventually fill up the map to match the known map. Due to lack of time, the implementation resulted to be incomplete. There is still a lot to learn about Unity and we believe that with more experience this would have been a completely feasible task.

Experiments and Results

When the game is launched, a prompt will appear asking for what resolution to be displayed in and whether to be in windowed mode or not. Depending on your preference on what you pick, the main menu’s animations below the title may be a little off the box it is to be placed in. When playing, the menu will have 3 options, play, high scores, and exit. The high score doesn’t work in our latest build for the game so that is printed in the menu. When play is clicked, the game will start right away with the first level being loaded. The played inputs any direction using either the arrow keys or WASD to move Pac-Man and the ghost speed is nearly 0, due to the fast growth that the speed will increase at. Collecting the stars and moving to a higher level, the ghosts will begin to move faster and the AI will become more apparent. Through testing, Blinky will begin by always going to the left and Inky to the right. Whereas Blinky will begin to make a beeline straight to us, Inky will immediately, at the first intersection, try to decide which path to take in order to get behind and ambush Pac-Man. This results in Inky usually taking a different path when a level is usually loaded. Pinky emerges from the ghost start area a little later and chooses a direction, either left or right, based on the position of Pac-Man in order to get in front of him and ambush him. Clyde, as being the random control ghost, will go in a random direction every time. At the start of the level, it was found that if Pac-Man attempts to stay in the bottom half of the level, then there is an increased chance that he surrounded in either one of the two bottom corner, relying on the fact that Blinky is closing in, and Inky and Pinky become aware of this and make appropriate movements to get to position in the bottom half of the map. If the player moved up, they have a better chance of survival at the start due to the first waypoints of the ghosts, specifically Clyde and Blinky, placing them in the bottom half of the map, whereas Inky and Pinky have less waypoints and are able to immediately begin searching/leaning which position to be in to get to Pac-Man. If Pac-Man is able to survive until levels above 10, then the ghost speed becomes a little too much to handle, showcasing how fast the AI able to think and search for Pac-Man with the increased variable speed of themselves and Pac-Man. At this point, the ghosts will be able to capture Pac-Man due to increased speed and algorithm processing that keeps up with the speed of the ghost movement.

Conclusions

In conclusion, we were able to implement both A\* search and machine learning, to an extent, into our own take of a game of Pac-Man revolving around randomly generated starts that must be picked up in order to proceed to the next level. The A\* search and machine learning was able to operate and process in a fast manner, even in the wake of increased Pac-Man and ghost movement speed. Through this project, we were able to learn a lot about the Unity 2D engine and how it works, especially coinciding with the task of coding in scripts for the animations, movement, map, menus, and ghost AI, to name some. Using this knowledge, we should now have a better hold on the topic and an idea of what to do when focusing on any future AI endeavors and our club projects.

References

List of Unity Functions – Vector Retrieved April 19, 2016, from <http://docs.unity3d.com/ScriptReference/Vector3.html>

N.d. (2016) Unity 2D Pac-Man Tutorial. Retrieved April 12, 2016, from <http://noobtuts.com/unity/2d-pacman-game>

Basic Unity for Beginners. Retrieved April 15, 2016, from https://www.youtube.com/user/Unity3D

Scripting in C# in Unity. Retrieved April 20, 2016, from http://forum.unity3d.com/