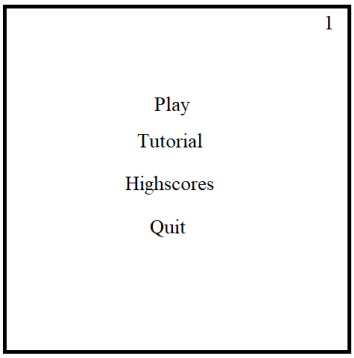
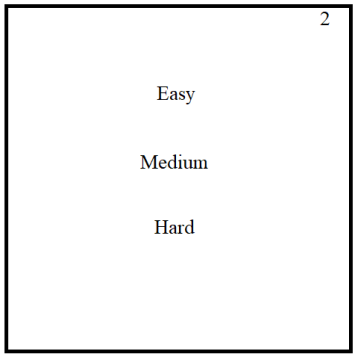
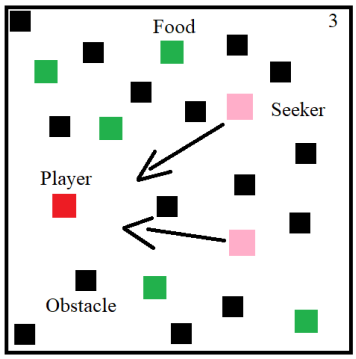
Soft Computing Assignment

Task 2 – Storyboard







Task 6 – Vector3.Distance and A* Pathfinding

Vector3.Distance and A* pathfinding essentially do the same thing of calculating the distance between 2 objects. There are different distance based formulas that can be used for specific implementation but in its simplest form especially games (a-b).magnitude is used.

Vector3.Distance is the exact same as the aforementioned formula but in a function format. This is more space efficient and easier to process for developers as it gives more context to what is happening.

Using these distance methods can only get us so far however, especially when obstacles enter the scene. These methods get the position of a and b and practically draw a straight line to connect the 2 points meaning if an obstacle had to be in the way the seeker would be stuck.

This is where A* pathfinding differs. A* calculates not only the distance between the 2 points but many other routes to calculate the fastest route the seeker could take. These calculations include any obstacles that might be in the way and will try to look around.

Task 7 – Storyboard Evaluation

As the title of the game suggests, this is Pacman… only better, faster and harder! This game can attract all players from all ranges. From those who want a casual gaming experience to those who want a challenge and prove they can do better, this game can do it all.

Pacman reborn offers an endless array of levels to test how far you can reach before losing all your 3 lives. A maze generates at the beginning of each level which was done using a recursive maze algorithm making the map completely random each time the game is played.

The aim of the game is to negate the maze to find and collect food whilst trying to avoid the 2 seekers that are trying to reach and defeat you. You progress to the next level by eating all the food inside the level which can be anywhere between 15 and 20 food objects. Each time the player eats one of these food objects the seekers start fleeing away to a location which is 10 tiles away from the player along with alternating colours to indicate the fleeing. When the seekers catch the player, they get moved to a location which is at least 10 tiles away from the player and the player loses a life.

The seekers' logic and scripting were done using Aaron Granberg's A* Pathfinding package which is a neat rework on unity's built-in pathfinder. The simple yet effective grid-based pathfinding system works brilliantly for this game and makes the experience of working and playing with it that much better.

When all the food has been eaten, the game then loads a new map and places a different number of food at random points in the map along with 2 seekers and the controllable character. The score will be carried over to this scene until the player has been defeated and is then taken to the highscores page where the top 5 highscores of all time are displayed for the player.

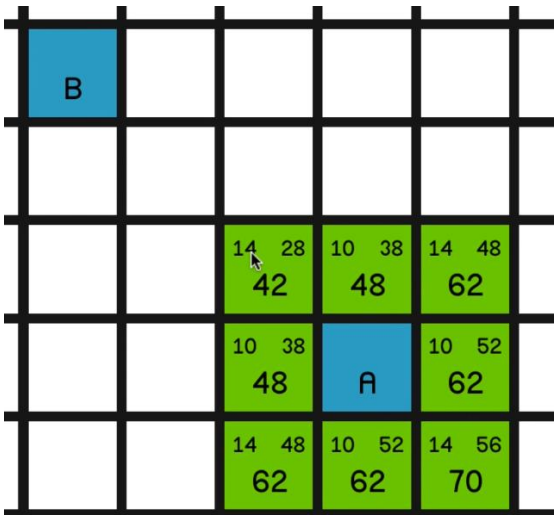
The game is very barebones in terms of design and there is no music or sound effects, these are definitely two aspects of the game where improvements can be done to bring the game to a much higher standard than it currently is. The graphics and sound will attract more players to the game due to a more audio visual experience.

Task 8 – A* Pathfinding

When a path is being calculated from point a to point b a set of calculations are being made for each node. These are the G cost, H cost and the F cost.

The G cost is how far away the current node is from the starting node. Inversely the H cost is the distance from the current node to the end node. The F cost is the sum of the G and H cost that would give the final cost.

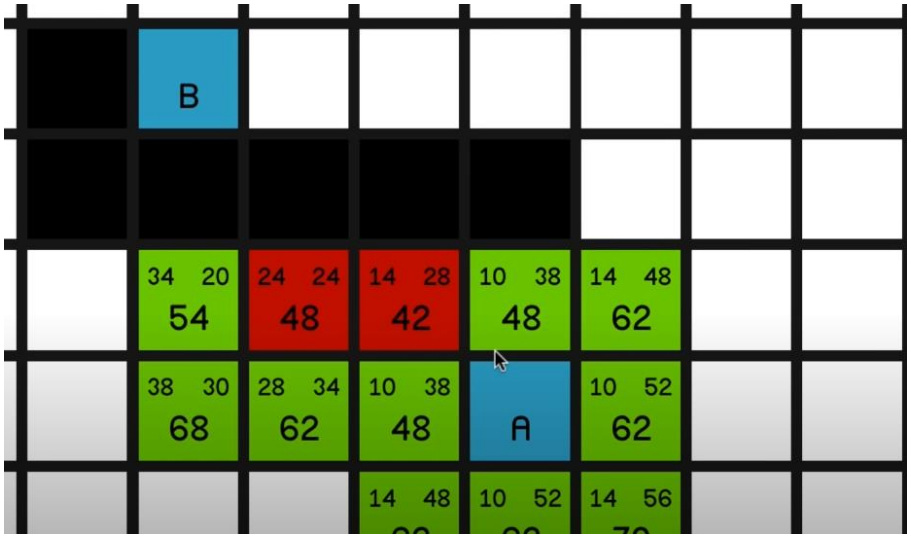
From the diagram on the right, we see 2 nodes, A and B, Node A is the starting point. Then the 9 surrounding nodes are given values relative to the notes. The distance between 2 adjacent nodes is usually 1 and the distance diagonally is 1.4 according to Pythagoras' theorem, for simplicities sake the values are multiplied by 10 so that the values are integers.



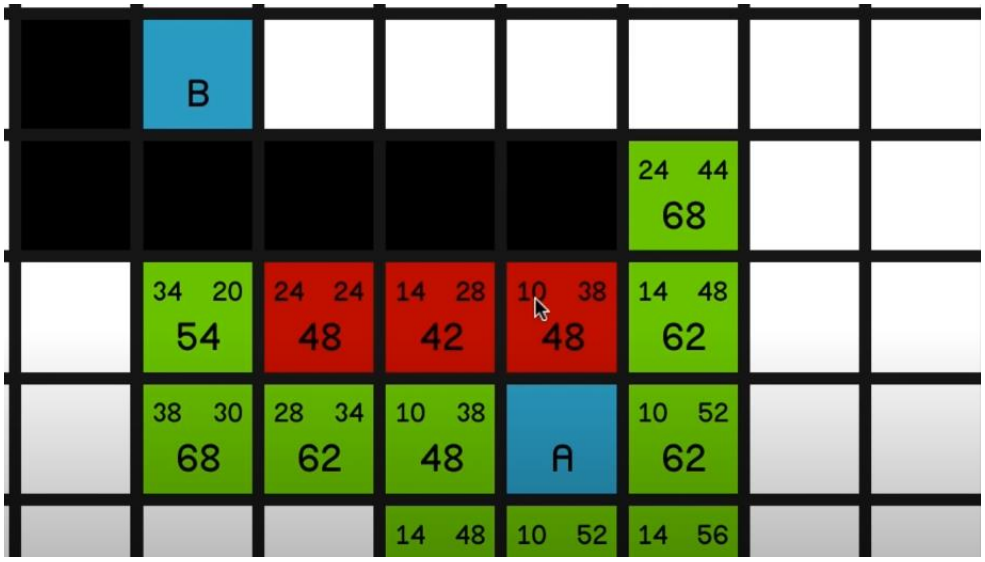
In each node there are 3 values, the top left number is the G cost, the top right number is the H cost, and the bigger number underneath them is the F cost.

The algorithm will then look at all the surrounding nodes and choose the node with the smallest F cost and marks this particular node as closed. The algorithm will then repeat this process for all subsequent nodes until the path has been found and completed.

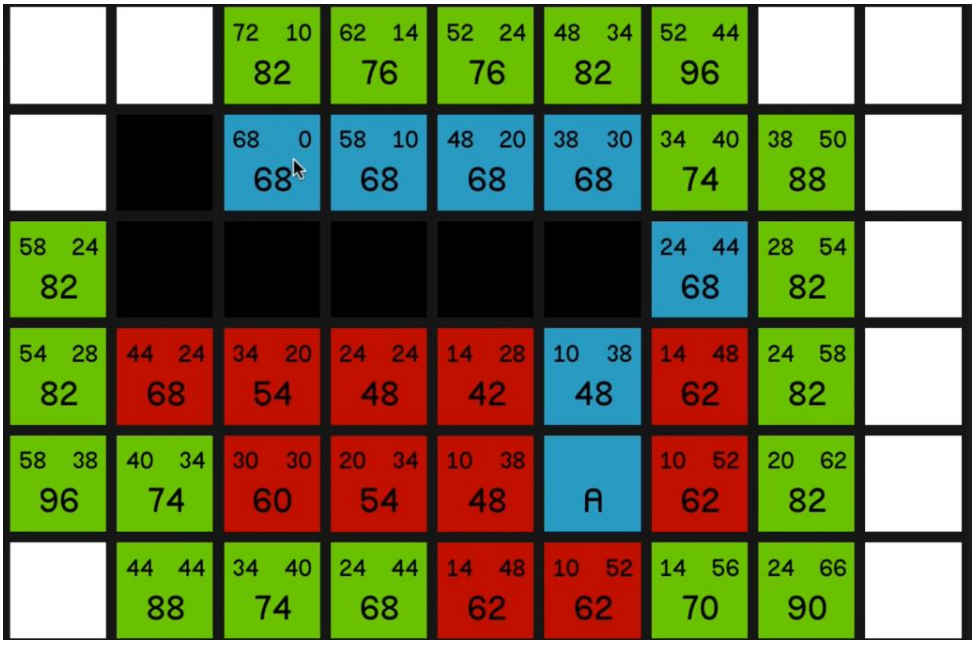
When an obstacle is in the way of the 2 nodes the process is more or less the same with the exception that the algorithm is checking for different routes as it is calculating due to lower F costs.



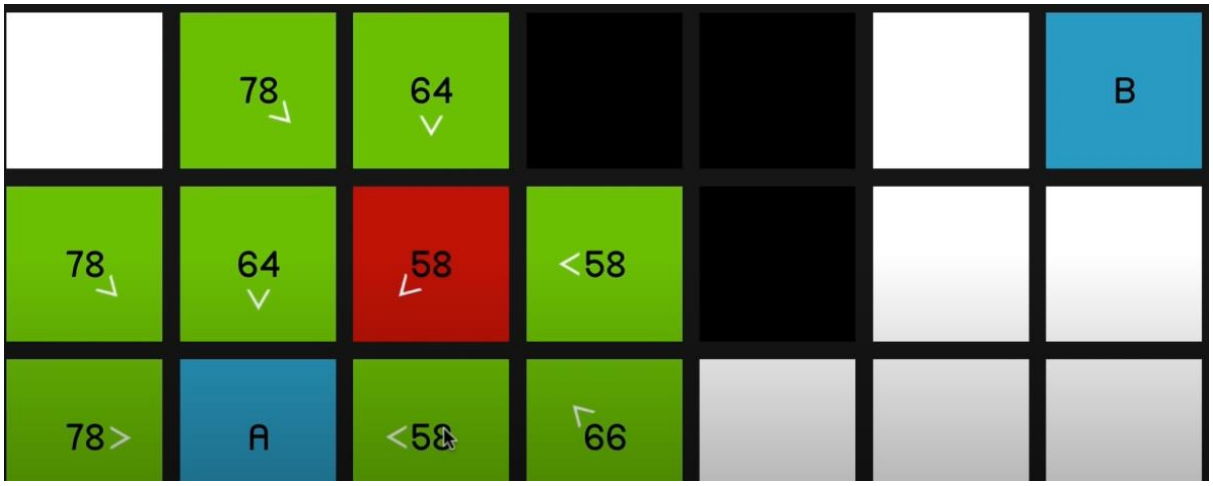
From the diagram above we see that 2 nodes have been chosen that have the lowest F cost and that are closest to the node. Now the algorithm will choose the node with the lowest cost, in this case 48.



The process of choosing the node with the lowest F cost continues until all nodes have been explored and the path has been computed as seen in the diagram below.

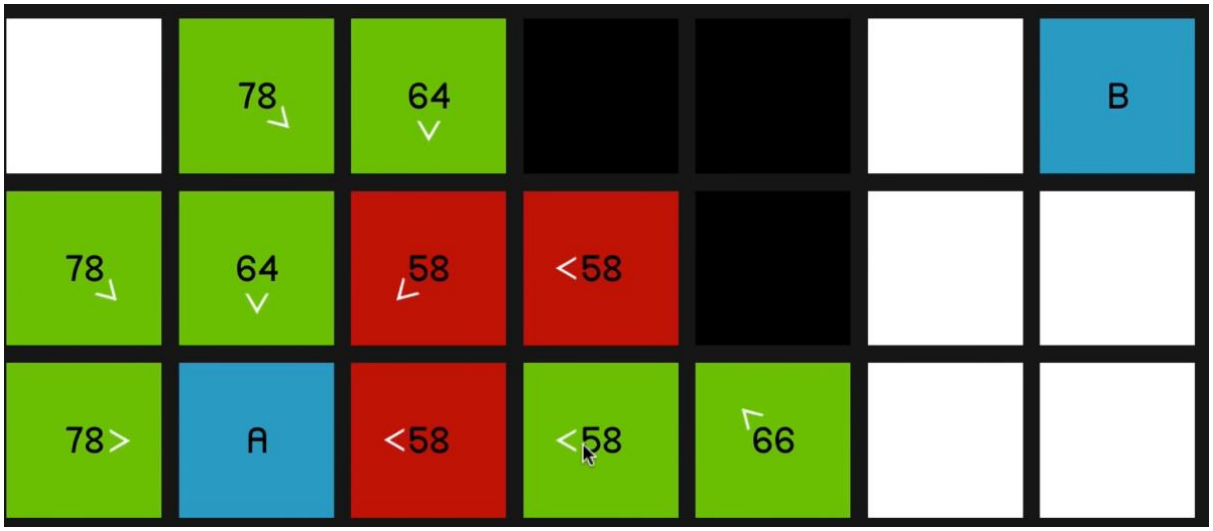


Since a single node might have different values based on which node is being calculated at a given time, the chosen nodes come with a direction indicator to show from which node the values are originating.



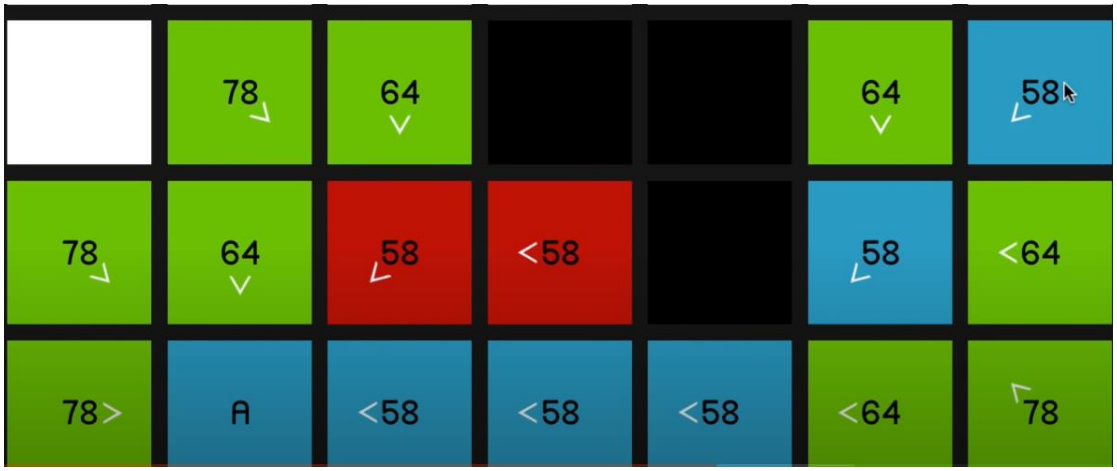
We see that node with cost 58 has originated from node A and the node 66 is originating from node 58. We can also notice that not all of node 58's neighbours are originating from it, simply because going to node 64 from 58 would increase its F cost and make it inefficient and therefore the node is originating from Node A. So, this algorithm will decide the cheapest cost for a node depending on its origin.

When the furthest 58 is chosen the nodes below it still shows the slower diagonal paths. However, when the 58 closest to the node is chosen see how the values will update in the diagram below.



We now see how the value of the node has been updated and now that there is a clear path to the end point the F cost will remain unchanged.

The final path should look something like this.



Task 9 – Game Evaluation

Instruction Manual

The copy of the game will be received in a zip folder called “PacmanReborn.zip”.

Move to a directory of choice, usually a folder specific for games.

Inside the chosen directory create a new folder and call it “Pacman Reborn”.

Move the zip folder to the “Pacman Reborn” folder and extract the contents of the zip folder inside the empty “Pacman Reborn” folder.

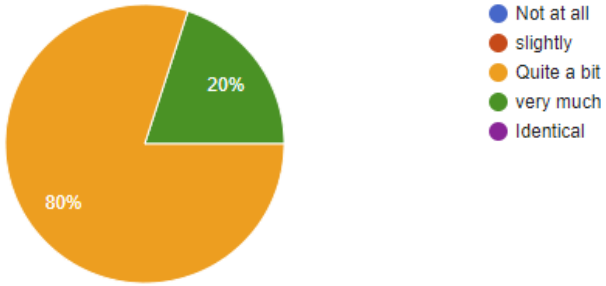
Once extraction is complete there will be an executable file (a .exe file) called “Pacman Reborn.exe”. This is the file you need to access to play the game.

Enjoy!

Final Evaluation

How similar is the game to the original Pacman?

5 responses



What are your favourite/ the strongest aspects of the game?

5 responses

- I enjoy the endless nature and the degree of difficulty
- The simplicity of the game and how well the maze is made with the algorithm.
- the randomly generated maze was a nice touch, it makes it replayable
- The highscore system to compete with friends, the increasing difficulty making it way more challenging than before!
- THE MAZE!

What suggestions or improvements can be done to the game?

5 responses

- I would improve collision with the walls
- The levels getting progressively more difficult as you go on.
- add more enemy variants and more maps would've been nice
- Better graphics
- More Levels

From the results that have been gathered from the questionnaire it is clear that the game was a success, more notably the neat maze algorithm that I chose to implement and the highscore system.

The players' experience was one of a positive outcome, no bugs were reported, and the game was very playable. The process of making the game was a very straightforward one, the A* pathfinding system facilitates implementing AI which makes the game development process that one of ease. Since the game lacked quite a lot in terms of graphics and audio, I could really put my focus in the functionality and playability of the game. Because of this all targets that were set were reached with conviction and the feedback from the play testers is evidence of this.

As with every game, there is always room for improvement and this game is no exception. The feedback that was received really shed light on what can be done to improve the game.

Improved collisions with walls: the tester is referring to a gameplay hindrance that occasionally happens when the tester mistimes a turn. The player could turn early or too late and get stuck on the wall, obviously they can reverse and readjust but this takes away from the experience. A way to go about this is to make the players movement limited to the graph that has been made and the player can turn when the centre of the player is on the node.

The levels getting more difficult as you go on: This is another very interesting tip that can push the game's standard even higher. Currently, when the player chooses the difficulty, the enemy is set to that difficulty for the duration of the player's life. What can be done is when the player starts progressing further, I could speed up the enemies or start adding more enemies of the same speed, another way could be to add more food in the level as the levels go by.

Better graphics: This is undoubtedly the weakest aspect of the game as the only real object in the game right now is a square with different colours to represent the different objects of the game. Sky's the limit when it comes to design however I would start redesigning my game by creating player and enemy assets that will complement the game and give them specific animations so that the game is more pleasing to the eye and would keep players interested in the game for longer. An addition to this would be to implement sound effects and music, music is always underestimated and forgotten most of the time but it can drive a games standard tenfold to the gameplay.