Computer Science 565 Project 2 Final Documentation



Introduction:

The main focus on our design was to test the effectiveness of the A\* algorithm. The layout of our map defeated the purpose of the explorer path, therefore we set way points along the specified explorer path, for the first few waypoints, and then modified the explorer path to go up and down instead of from right to left. The reason for such medications is because our map layout consisted of a pseudo maze which would help us test our A\* more efficiently.

We understood the purpose of this assignment as implementing AI for the Non-Playable Character (NPC) and flocking for the “Ghost Dogs”. The goal of the AI introduced in this game is to aid the NPC in capturing all the treasures before you do. The purpose of flocking is for Professor Barnes amusement. This document will discuss each of these tasks in detail, and how they were implemented in game at a functional level.

|  |  |
| --- | --- |
| Tasks to Complete | Completed |
| (1) Modify inspector pane | Yes (Removed Old Version) |
| (2) Collision testing for NPC and Player | Yes (No Sphere Based Collision See Below) |
| (3) Exploration for NPC | Yes (More Focus On Complex Path) |
| (4) Treasure-directed path for NPC | Yes |
| (5) Flocking for dogs | Yes |
| (6) One complex path | Yes (actually multiple) |
| (7) Game ends when 5 treasures have been found | Yes |

Solutions (Detailed Descriptions):

1. Updates to Controls and Interfaces.

None of the controls requested have been modified except I have removed the Inspector Pane to facilitate the things I was implementing. I did add in one additional control.

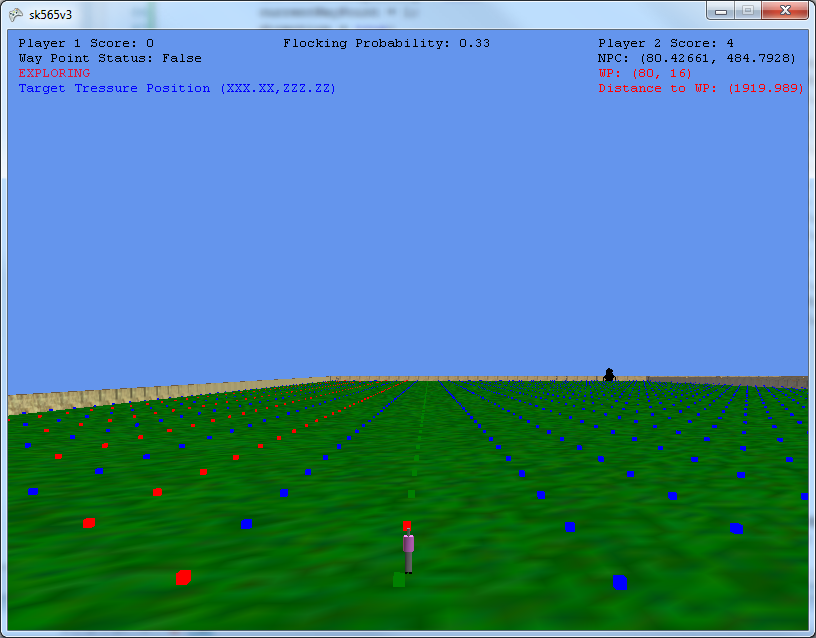
|  |  |
| --- | --- |
| Control | Function |
| D | Teleports dogs to player if they are stuck.  *We attempted to do collision detection on dogs, but parts of the flocking algorithm move them through walls.* |

Code Implementations:

|  |  |
| --- | --- |
| Files | Description (Function) |
| Scene.cs | protected override void Update(GameTime gameTime) –  Added a control ‘D’ to this function |
| Scene.cs & NPAvatar.cs | protected override void Draw(GameTime gameTime) –  Added in a Sprite that take care of the “new” inspector pane, this sprite gets most of its input from the properties inside the NP-Avatar |

9

Explanation of OUR inspector pane



7

6

5

8

1

4

3

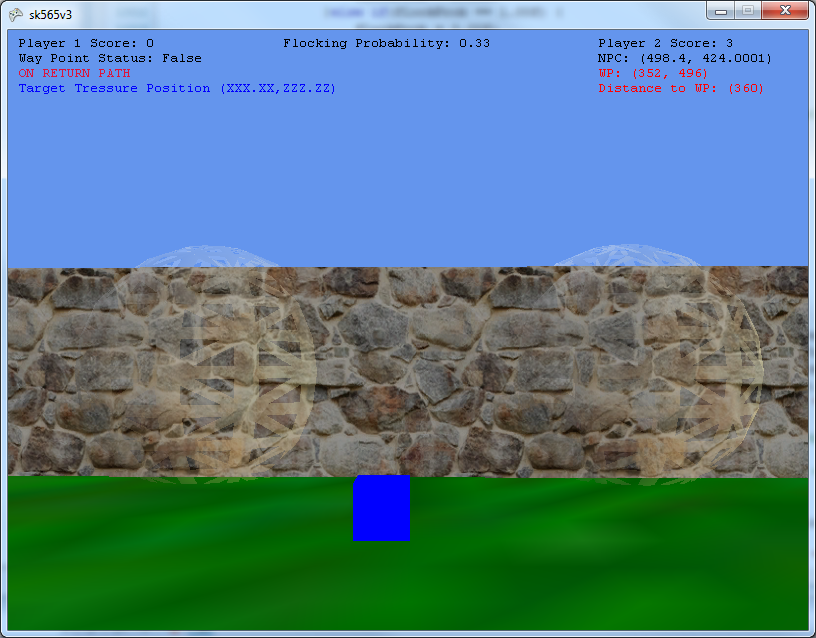
2

|  |  |
| --- | --- |
| Number | Function |
| 1 | Number of treasures captured by player 1 |
| 2 | Has NPC arrived at next waypoint |
| 3 | This indicates what the NPC is doing at the present moment |
| 4 | If the NPC has found a treasure it will indicate to the user the (X,Z) location of the treasure the NPC is pursuing. |
| 5 | Number of treasure captured by NPC |
| 6 | NPC’s current location |
| 7 | The current point NPC is headed to. Its “bearing” |
| 8 | The distance to NPC’s next way point |
| 9 | The probability to flock set by the user by pressing the “P” button. |

1. Collision testing for NPC and Player (Not Sphere Based)

Collision testing was done differently than required to handle the layout of our map. Had we used bounding spheres for collision detection, it would have been very difficult for my NPC and the player to move through the world. The reason for this is the structure of the maze and the fact that we used different walls than provided by the professor. The way we handled collision detection is as follows. Inside Scene.cs is a variable defined as follows “private char[,] gridMap;”. “gridMap” is a 512 by 512 array that contains information about where objects are placed. As the character moves through the world, both forwards and backwards he looks one grid cell at a time in front and in back of him. If the cell he is going to move into contains a ‘b’ then the cell is not free to move into. If the cell contains a ‘f’ or a ‘w’ the cell can be moved into without a problem. Because the character does not only move vertically XOR horizontally, he uses a multiple of his @ vector to determine what is infront and what is in back of him no matter what angle he is headed. Grid map is populated during XNA’s call to “protected override void LoadContent()”inside Scene.cs. Inside “LoadContent()” the user defined function “public void buildWalls()” is called, which places the obstacles into the world and into “gridMap”.

Once again the reason for taking this approach is because we were able to obtain higher levels of percision in our collision detection calculations with a simpler algorithm. Below is an image of the bounding spheres produced for our walls in game.



As can be seen the sphere would have produced gaps in our walls allowing for the NPC and Player to pass through.

Code Implementations

|  |  |
| --- | --- |
| Files | Description (Function) |
| Scene.cs | public void buildWalls()–  This is called by protected override void LoadContent() at the start of the game, it opens up the file located at "//Content//Walls.csv" and reads in the locations of the walls, the type of wall is denoted by a number, and the program uses that number to bring in that type of wall. The walls were created in AC3D by Matthew Hoggan. Both functions above are in Scene.cs. This information is stored in the member variable, private char[,] gridMap; instantiated as gridMap = new char[512, 512];. Because it is 512 X 512 it requeires the program to divide by private const int spacing = 150; to map into gridMap with some level of round-off error. |
| NPAvatar.cs | private bool checkForCollisionFront()–  This function is used to look at .5 \* the “Forward” vector of the NPAvatar. It is called from private void setStep() which determines if the NPAvatar can move forward. Or if she needs to adjust her angle to avoid a wall. |
| Player.cs | public override void Update(GameTime gameTime) – This function which is part of XNAs framework is called once per frame, and thus if the player is holding down the forward or the back button it checks to see if the cell that he/she is moving into is free or not. The data structure it checks against, is in side Scene.cs |
| SavageBeast.cs | private bool isCollision() –  Is the equivalent function for the “Dogs”. This function attempts to to side collision too. However, it is not 100% effective. |

1. Exploration for NPC

First it is important to mention that there are two types of waypoints. A waypoint can be considered as a vertex that occurs when V(x,y)%8 == 0. Another type of way point is a waypoint specified to the NPC when the game first starts. Waypoints had to be specified in order to aid the NPC when navigating through the walls of our maze, and conform to some degree to the requirement of an exploration path. When the Explorer path for the NPC is first calculated all boxes turn green. The explorer path is calculated in chunks based on the specified set of waypoints. Once the explorer arrives at its next waypoint, the next path to its next waypoint in the “Explorer Path” is calculated. The path is calculated using the A\* algorithm which will be discussed shortly. Once the NPC calculates its next segment of the “Explorer Path” it turns the boxes red as it moves through them. Currently, if you allow the NPC to follow through on one full cycle of its explorer path, it will have obtained all 5 treasures. If you wish you can move the treasures around inside Scene.cs to test out the Explorer’s Path please feel free. The NPC should return back to the start of the explorer path in an inverse fashion once completing a full cycle. It is also important to note that a “State Machine” has been developed with 5 variables which help the NPC know which movement state it is in.

|  |  |
| --- | --- |
| Files | Description (Function) |
| Scene.cs | private char[,] gridMap; --  Used to tell the NPC what objects are immediately in front of it. |
| NPAvatar.cs | public void STATE\_MACHINE\_FUNCTION\_TO\_DETERMINE\_MOVEMENT\_METHOD\_THIS\_IS\_KEY() and public override void Update(GameTime gameTime)are the two functions that we would recommend reading first. The name of the first, we would hope, should tell you the exact purpose of the function. There are a lot of helper functions envolved in these two functions, and we would hate to try and explain them all. |

1. Treasure-directed path for NPC

When the Treasure Directed Path is first calculated it turns all Nodes in that path to chocolate. Note that when the NPC is in this mode the boxes or waypoints she crosses will be changed to purple to indicate the nodes she has crossed in that path. The implementation of the treasure directed path is handled as follows. The NPC moves along the Explorer Path until it is within 4000 units of a treasure. It then takes its current point and the point of the treasure and passes it to Graph.cs where the A\* path is handled. The coordinates are passed into the graph as “OrderedPairs”. They are then loaded into a Queue and that is returned back to the NPC as a set of “OrderedPairs”. In the case that a treasure is found while the NPC is in treasure moving mode. It will push that treasure onto a queue. Once it gets to the designated treasure, it pushes the return path onto a stack of queues, and it proceeds onto the next treasure. Once all treasure are removed from the queue it takes one return path at a time off the stack and returns back to the explorer path, until there are not more return paths, and then it returns back to where it left off in the explorer path. The one thing we would like to do in future enhancements is to take the union of the return paths so that it does not go back over redundant waypoints. If no other treasures are selected and placed on the queue then on the return to the explorer path the boxes turn back to blue. It should be noted at this point that each return path will have the boxes colored perl.

|  |  |
| --- | --- |
| Files | Description (Function) |
| NPAvatar.cs | public void STATE\_MACHINE\_FUNCTION\_TO\_DETERMINE\_MOVEMENT\_METHOD\_THIS\_IS\_KEY() and public override void Update(GameTime gameTime)are the two functions that we would recommend reading first. The name of the first, we would hope, should tell you the exact purpose of the function. There are a lot of helper functions envolved in these two functions, and we would hate to try and explain them all. |
| Graph.cs | public Queue<OrderedPair> findShortestPath(OrderedPair myloc, OrderedPair tresloc) and public void loadQueue(Queue<OrderedPair> retrn, List<Node> path, List<Node> closed\_set, Node reverse\_engineer\_node, Node myloc\_node ) are the two functions we recommend reading. findShortestPath() is the A\* algorithm, and loadQueue() is a recursive function that reassembles the path once the final point is found. Based on how we have things configured there should always be a path. |

1. Flocking for dogs

Flocking for dogs needs to be enhanced, using different types of forces. However, the general concept is there. To facilitate grading, and because of the obsessive use of walls, we provided a spot at the beginning of the map that will allow you to test our flocking algorithm. This is labeled as the “Dog Park”. Please note this area is not considered as part of the NPC explorer path. It is only used for you. I also attempted to use collision detection for the dogs, however it is not perfected. Therefore, if you hit ‘d’ the dogs will teleport to you.

|  |  |
| --- | --- |
| Files | Description (Function) |
| Scene.cs | private void placeSavageBeasts() – is used to create a container of savagePack[dogId] = new SavageBeast(this, "dog " + dogId, new Vector3(initialX, initialY, initialZ), new Vector3(0, 1, 0), 0.0f, "dogV3", avatar, dogId); which contains all the dogs |
| Savage Beast.cs | private void flockingAlgorithm() – Should help you trace our algorithm. We have made the function calls inside that function very decriptive and located within the class itself. There are quite a few helper functions, and explaining them all would take more than a few lines. |