Ernie Ledezma

Dani Odicho

Arnold Santos

Cesar Zalzalah

**COMP 565 Project 1**

Our game has a space theme, the black, gray, white colors imply that the person is on the dark side of the moon (Pink Floyd). There are also spaceships floating around from planet Monotron that came to invade the moon. Little did they know, the moon was already flooded with aliens with red eyes that don’t seem very friendly. That’s why the spaceships are just observing the moon and rotating in mysterious way to wait for the right chance. Meanwhile, two humans have made it to the moon in the search for valuable coins. However, when they try to grab the coins something weird happens …>:)

**Additional Models**

Location: GameName1\GameName1Content

**alien.x (Aliens)** - Built with AC3d by Ernie Ledezma

**gold.x (Gold coin)** - Built with AC3d by Ernie Ledezma

**ufo.x (UFO/Spaceship)** –Built with AC3d by Ernie Ledezma

|  |  |  |  |
| --- | --- | --- | --- |
| ***Class*** | ***Type*** | ***Name*** | ***Description*** |
| *Agent.cs* | *variable* | *int treasures* | *# of treasures found* |
| *Agent.cs* | *method* | *Treasures()* | *gets and sets number of treasure* |
| *Treasures.cs* | *variable* | *nav Node* | *Stores an embedded NavNode used to determine distance to treasures* |
| *Treasures.cs* | *variable* | *bool tag* | *Boolean value if treasure has been found* |
| *Treasures.cs* | *method* | *Node()* | *gets or sets node variable* |
| *Treasures.cs* | *method* | *Tag()* | *gets or sets tag variable* |
| *Treasures.cs* | *method* | *Update()* | *moves the treasure to the other side of the terrain when tagged* |
| *Stage.cs* | *variable* | *treasure3D (Model)* | *treasure marker* |
| *Stage.cs* | *variable* | *treasure2 (Model)* | *tagged treasure marker* |
| *Stage.cs* | *variable* | *pathMode (bool)* | *pathfinding/treasure finding mode* |
| *Stage.cs* | *variable* | *TreasureList(List<Treasures>)* | *List of treasure objects* |
| *Stage.cs* | *method* | *LoadScene()* | *Pass TreasureList to player and npagent classes* |
| *Stage.cs* | *method* | *setSurfaceHeight()* | *Modified for interpolation* |
| *Stage.cs* | *method* | *LoadContent()* | *Modified to add treasures and change models* |
| *Stage.cs* | *method* | *Update()* | *add n key listener and added tagged treasures and np mode to inspector mode.* |
| *Player.cs* | *variable* | *int tagDistance* | *tag treasure when user is less than 200 units away from treasure* |
| *Player.cs* | *variable* | *List<Treasures> TreasureList* | *List of treasure objects passed from stage.cs* |
| *Player.cs* | *method* | *Update()* | *move player with arrow keys, tag treasure, move it , and play sound if player gets close to it* |
| *Player.cs* | *method* | *playSound(String path)* | *plays a file located in the bin folder* |
| *NPAgent.cs* | *variable* | *List<Treasures> TreasureList* | *List of treasure objects passed from stage.cs* |
| *NPAgent.cs* | *method* | *ChangePath()* | *switch from following path to following nearest treasure* |
| *NPAgent.cs* | *method* | *nextUntagged()* | *turn towards closest untagged treasure, return false if none exist* |
| *NPAgent.cs* | *method* | *Update()* | *turn toward next untagged treasure, move towards it, tag it, Move treasure, play sound.* |
| *NPAgent.cs* | *method* | *playSound(String path)* | *plays a file located in the bin folder* |

**Added methods and variables in project 2:**

|  |  |  |  |
| --- | --- | --- | --- |
| ***Class*** | ***Type*** | ***Name*** | ***Description*** |
| *NavGraph* | *variable* | *Stage stage* | *Stores instance of the stage* |
| *NavGraph* | *variable* | *Int nodeSpacing* | *Stores the maximum spacing value between the nodes* |
| *NavGraph* | *Variable* | *List<NavNode> open, closed, path, aStarNodes.* | *Lists needed for the A\** |
| *NavGraph* | *Variable* | Dictionary<String, NavNode> graph | *A keyvalue pair to keep track of the nodes* |
| *NavGraph* | *variable* | *Bool pathComplete* | *Indicates when the path was calculated* |
| *NavGraph* | *variable* | *Int totalAdjacents* | *Number of all adjacency connections on the graph* |
| *NavGraph* | *method* | Void createQuadTreeNodes(int x1, int y1, int x2, int y2) | *Recursively create the quad tree nodes based on existence of collidable objects* |
| *NavGraph* | *method* | bool objectExists(int x1, int y1, int x2, int y2) | *Checks if there is a collidable object inside the rectangle bounded by x1,y1 and x2,y2* |
| *NavGraph* | *method* | Path aStar(Vector3 startPosition, NavNode, destination) | *Calculates and returns the shorted path from the start position to destination* |
| *NavGraph* | *method* | Void sortByCost(NavNode node1, NavNode node2) | *Used to sort lists by cost* |
| *NavGraph* | *method* | Void Connect(NavNode node1, NavNode node2) | *Add each node to adjacency list of the other* |
| *NavGraph* | *Method* | String skey(int x, int y) | *Returns a formatted string used for the key of the nodes* |
| *NPAgent* | *Variable* | Path treasurePath | *Path to a treasure calculated by A\** |
| *NPAgent* | *Variable* | NavNode nextTreasurePathNode | *Next node on the path to a treasure* |
| *NPAgent* | *variable* | NavNode lastSeen | *The location of the npagent to get back to after tagging a treasure* |
| *NPAgent* | *variable* | bool tagged | *Determines the treasure was tagged* |
| *NPAgent* | *Variable* | Int treasureDetection | *Treasure radar radius (4000)* |
| *NPAgent* | *Method* | Update() | *patrol between nav nodes until a treasure fall within 4000 untis, find the best path with A\* then go back to where the NPAgent was* |
| *NavNode* | *Variable* | Double cost | *cost from source to node + node to destination* |
| *NavNode* | *variable* | Double distanceToSource | *Distance from node to source* |
| *NavNode* | *variable* | Double distanceToGoal | *Distance from node to gaol* |
| *NavNode* | *variable* | List<NavNode> adjacent | *List of nodes that are adjacent to the current nodes* |
| *NavNode* | *Variable* | NavNode pathPredecessor | *The node that precedes the current node on the path to the destination* |
|  | *method* | NavNode(Vector3 pos, NavNodeEnum enum, int dist) | *New constructor with a specified distance that represents the offset distance from nodes created on the same level of the quad tree* |
| *Stage* | *Method* | onLoadContent() | *Added fifth treasure and modified positions of treasures and NPAgent* |
| *Path* | *Method* | Path reversePath(Path p) | *Returns the reversed path of the given path* |

**How to run the project:**

1- Extract the AGMGSK.zip file

2- Open the AGMGSK.sln file

3- Click Start to run the project.

**AGMGSK Modified Controls**

The added control keys are:

‘n’ which is used to alternate the NPAgent behavior between following the path between the blocks, and walking towards the nearest untagged treasure.

‘p’ which is used to alternate between flocking probabilities of 0.33,0.66 and 0.99

**Treasures Locations**

Locations of the treasures can be found in the stage.cs file:

(500, 500)

(490, 470)

(447, 453)

(500, 425)

(480, 420)

The treasures were placed in the relatively flat area near the player to make it easier to test the game. There’s also one treasure inside the walls as instructed

**Fixed issues with project 1:**

We made sure to fix all of the issues that you pointed out in project one, and fixed them according to the specs.

* We had no treasure inside the walls so we added one.
* We had the path type set to reverse instead of Loop, and we changed it to loop.
* When pressing n the NPAgent use to turn towards a treasure, tag it, then turn to another treasure, until getting all the treasures. We fixed the issue by making the NPAgent go back to path mode after tagging a treasure.
* Our NPAgent use to get stuck in the side of the level. We no longer have this issue.

**QuadTree Graph:**

We implemented the quad tree algorithm to efficiently place nodes on the terrain. Which we handled in the NavGraph class. The graph has a keyvaluepair to distinguish the nodes by their locations. In the constructor of the graph, we call a createQuadTree() method which takes the location of two points, which represent the corners of a box where 9 nodes will be placed. The method checks whether the distance between the x positions of the points is less than the minimum allowed sacing between nodes. We set the spacing to 125 instead of 150 because it produced better results. With 150 spacing, the 9 nodes do not fit inside the walls, and if we tried to use 4 nodes or even 1 node instead, we were faced with the issue of two nodes being adjacent even though there is a wall in between. That’s why we decided to stick to 125 befause it gave enough room for the NPAgent to walk through, while maintaining the correctness of the adjacencies of the nodes.

The createQuadTree() checks if an object exists in the area using the objectExists() method, or if the distance between the two corners is greater than 50 (Spec asks for 25 between each two nodes , but the 50 value is used because we create 9 nodes at a time, so the works because it’s the distance between three nodes rather than two.

The method calls itself recursively if one of the two conditions was met, which breaks the area to 4 smaller areas. When finally no object is found on the area, 9 nodes are places, and the area around each node is scanned to see create adjacencies.

**Navigation for NPAgent**

We changed the behavior of the NPAgent according to specs. The NPAgent will be in path mode initially, following the nodes that are initially created by the same path in project 1.

We modified the update method in the NPAgent class to keep checking between the distance between the NPAgent and the closest treasure. When the distance becomes less than 4000, the changePath() method gets called which is used to switch back and forth between the two path modes. Then the nextUntagged() method gets called which calls the AStart Method to find the shortest path between the NPAgent position and the treasure location. The A\*algorithm was implemented according to the pseudo code in the notes, and the method returns a Path object which is of type revers. We are assuring that the player does not get stuck by turning towards a nav

**npAgent Algorithm**

The NPAgent class handles the pathfinding and treasure handling. At first a linked list of treasures is passed in as a parameter in the method, where it gives itself as a value for a private variable inside the class. This allows the list of treasures to be shared between other classes such as stage, player, and NPAgent as well as be modified by any one of those classes.

Next up is the nextUntagged and Update methods. In nextUntagged the list of treasures is traversed, and the distance of each untagged treasure to the agent is measured. When all distances have been measured, the algorithm either turn the agent to face the closest untagged treasure or return a value of “false” if there exists no untagged treasures. To make it more simple, the regular “path” that the NPAgent follows has been simplified to a square and only 4 treasures are placed outside of the square. Different treasures are closer to each regular path node to demonstrate the ability of the algorithm to find the closest treasure.

Lastly we made changed to the update method inside the NPAgent class. The code we added allows for traversing through all untagged treasures, checks the distance between the agent and each treasure, and if any distance is within range, it is marked as tagged in the code and the model just runs away rapidly. When a treasure is tagged, its necessary to turn the agent to face the following untagged treasure in the list. To do this a boolean variable “turn” is set when a treasure is tagged and then the agent turns to face the next untagged treasure.

**Surface Height Algorithm**

The surface height algorithm was implemented in the setSurfaceHeight() method inside Stage.cs class, it was kept there so that it can be used by both Agent.cs and Pack.cs to position the player and the pack properly on the terrain. The method was applied according the the lecture notes of the class. By creating a square of four vector3 nodes A,B,C,D and calculating the heights of those points. then looking at two triangles. ABC, BCD. If the player was located in the upper triangle (ABC) do the lerp according to AB and AC. if the player was in the other triangle. apply the lerp according to DC and DB. The algorithm significantly smoothes the position of the player and the pack of aliens on the terrain.