

**3D Games Programming Assignment**

**Report**

**Soheil Salehinia (K2144160)**

**Maze Runners**

**Tasks, Aims, and Objectives**

Maze Runners is an endless third-person shooter game in which the player should compete against an AI to exit through randomly generated mazes. These mazes come in different sizes based on the level of the game, and contain some enemies to slow down or distract the player and AI. Two keys in the environment should be collected by the player and AI to exit from the exit point.

The main task was to implement an algorithm to create these random mazes based on the defined size. These mazes should always have a solution and should have only one solution to complete. The other task was implementing the AI that can find the correct way to collect a key and exit through the exit point.

There are some other tasks which I mentioned below:

* Implementing enemies and their actions
* Placing enemies in the environment in a fair random way
* Placing keys and exit point
* Implementing player and AI’s actions
* Implementing UI system
* Implementing Scene Manager
* Implementing Save System

**The plot of the Game, and User Instructions**

The plot of the game is to finish the maze faster than AI, and the player should first collect a key and then exit through the exit point. In this path, there are enemies that players should engage with them. In addition, the player should try to slow down the AI by doing specific actions. The AI also does the same to the player to win the game.

The game will start with a 3 by 3 random maze which does not include enemies and by exiting this maze, a new bigger maze will generate with enemies inside. As the player progress the game there will be always bigger mazes to explore and the game will never end. If the AI exit faster than the player, the maze will restart and be generated randomly with the same size.

The player can continue with the last size he was playing from the Main Menu at any time. He can also retry each level whenever he thought it is necessary. He can also start a new game from the Main Menu to begin with a 3 by 3 maze.

The attacking actions of the player, AI, and enemies have cooldowns. Therefore, they can not do an action continuously. There is also a cooldown at the beginning of the level for attacking actions so that the player and AI do not engage each other at the beginning.

Player actions are Move, look, Sprint, Aim, Shoot, Punch, Drop Bomb. The player can aim anywhere and shoot a bullet. If the bullet hits the AI, it will slow him down, and if the bullet hits enemies, it will destroy them. When the player punches, if the AI is in the same cell as the player, it will slow down the AI. The player can also drop a bomb which will explode after moving out of the cell and create fire for a limited amount of time. If the AI or player traverses through the fire, it will slow them down.

The AI actions are Choose Path, Move Fast, Move Slow, Shoot, Punch, Drop Bomb. Attacking actions are as same as the player’s attacking actions. He can shoot or punch the player to slow him down or drop a bomb to create fire.

There are three types of enemies in the game which are drones, bombers, and runners, I will briefly explain their functionality below.

Drones move randomly with low speed and when the player or AI be in their range and sight they will shoot, and if the bullet hits the player or AI, it will slow them down. Therefore, the player cannot sprint anymore and the AI will move slow for a limited amount of time.

Bombers also move randomly with low speed and the player or AI be in their range they will drop a bomb. As soon as the bomb hits the ground, if the player or AI is not in the same cell, it will explode. This explosion creates a fire in the same cell, and if the player or AI tries to traverse through that cell, they will become slow for a limited amount of time. The fire also will vanish after some time.

Runners move randomly at high speed and try to distract the player. There are harmless, but they may choose the correct or the wrong way. Therefore, it is better to kill the source of distraction.

In the below table I mentioned user instructions:

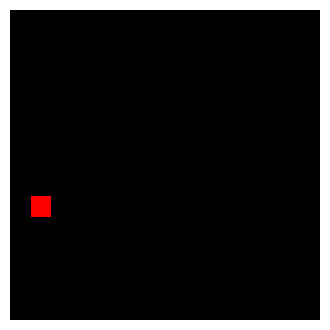
|  |  |  |
| --- | --- | --- |
| **Action** | **Mouse and Keyboard** | **Controller** |
| Move | W/A/S/D | Left Stick |
| Look | Mouse Pointer | Right Stick |
| Sprint | Left Shift | Button North (Triangle) |
| Aim | Right Mouse Button | Left Trigger (L2) |
| Shoot | Left Mouse Button | Right Trigger (R2) |
| Punch | Control | Button East (Circle) |
| Drop Bomb | F | Button West (Square) |
| Pause | Escape | Select (Options) |
| UI Navigate | Up/Down/Left/Right | D-Pad |
| UI Submit | Enter | Button South (Cross) |

**Major Techniques, Algorithms, and Tools**

**Random Maze Generator: [[1]](#footnote-1)**

I used a famous algorithm called Backtracking Generator to create the mazes. In this algorithm, we create a number of cells based on the size of the maze with walls around each cell. We start from a random cell and randomly choose a wall. Then we delete that wall and go through it to the next cell. We repeat the same job for the new cell, but we should only consider the walls that open a path to an unvisited cell.

We should continue the same procedure until there will be no adjacent unvisited cells remaining. At this point, we go back to the previous cell and repeat the same functionality. This algorithm will be finished when we go back to the cell from which we started.



**AI Path Finding:**

The AI uses a heuristic approach to find the path first to the key and then to the exit point. AI has a memory of cells which at the start it is empty and the path he took. He randomly chooses an available path and goes to the next cell. He will continue this until he reaches a dead-end. He will add cells’ information and the path he used to his memory as he traverses the maze. Then he will backtrack to the last cell in his path which has an available new path to explore. He continues this procedure until he finds a key. Again, the AI will continue the same until he finds the exit point.

If the AI finds the exit point before finding a key, he will remember the exit location by adding it to a stack of cells. From there he tries to find a key and, in his path, he will add every cell which is not duplicated to the stack. When he finds the key, he knows exactly the shortest way to get back to the exit point by popping cells from the stack and going through them one by one.

**Player Third Person Shooter:**

I used the Starter Assets - Third Person Character Controller[[2]](#footnote-2) from Unity Asset Store for the basic implementation of my player controller. I modified this package to add aim, shoot[[3]](#footnote-3), drop the bomb and punch functionality, and animations[[4]](#footnote-4).

**AI Choose Action:**

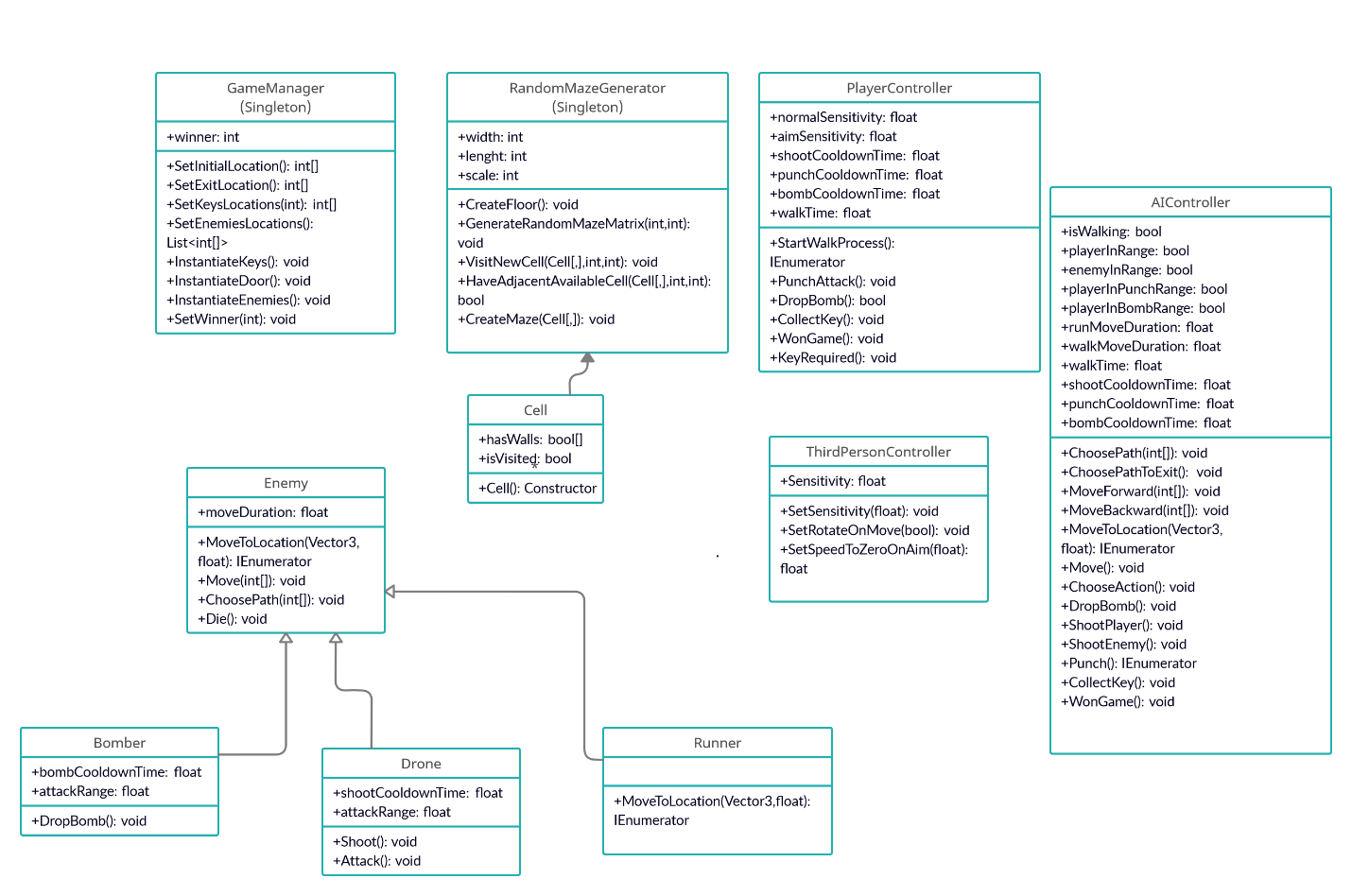
The AI will choose action when it reaches a new cell. He can choose between move and attacking options to enemies or the player. He chooses these actions randomly based on the availability of the action at that point. I also modified Starter Assets - Third Person Character Controller to use the same system without input system and camera for the AI and the Runner enemy.

**Placing Objects:**

If we consider the maze a matrix of cells, the player and AI always start from the first cell of this matrix. The keys and exit points will place in the corners of the maze. Then we divide this maze into 3 by 3 small mazes and in each one except the one who the player and AI are in, we place an enemy randomly in a random place in these small mazes.

**Specification of the source code**

Below is the diagram of major classes, attributes, and methods in the game.[[5]](#footnote-5)



I will explain briefly about each class and its attributes and methods in the following.

**Random Maze Generator:**

This singleton class is responsible for creating the random maze based on the width and the length attributes. The scale variable changes the scale of the walls and floor. The “Create Floor” method instantiates the floor based on the size and scale of the maze. The “Generate Random Maze Matrix” method creates the matrix of cells using the backtracking generator algorithm. “Visit New Cell” and “Have Adjacent Available Cell” methods are responsible for the steps of the algorithm. “Create Maze” will instantiate the walls based on the completed matrix.

**Cell:**

Each cell has an array Booleans called hasWalls which is show that the cell has the left, down, right, or up wall or not. The isVisited attribute uses for the backtracking generator algorithm to see if the cell is visited already or not. In its constructor, the variables are set to their default value.

**Game Manager:**

This singleton class is responsibleforsetting the locations of the player, AI, keys, exit point, and enemies. It also set the winner of the game after the game finishes. The “Set Initial Location” method sets the player and AI location. “Set Exit Location”, “Set Keys Locations”, “Set Enemies Locations” methods are responsible for setting exit point, keys, and enemies’ locations respectively. “Instantiate Keys”, “Instantiate Door”, and “Instantiate Enemies” are responsible for instantiating the objects inside the maze. The “Set Winner” method sets the winner based on the outcome of the game to either player or AI.

**Player Controller:**

This class which is attached to the player is responsible for player actions. The normalSensitivity variable is for the speed of the camera when the player is not aiming and the aimSensitivity is for when the player is aiming. shootCooldownTime, punchCooldownTime, and bombCooldownTime are setting the action’s cooldowns. walkTime is also sets the time which the player will move slowly because of the events that happened. The methods are self-explanatory by their names. “KeyRequired” will call when the player collects the key. The shooting action of the player happens in the Update method.

**Third Person Controller:**

This class was from the Starter Assets - Third Person Character Controller package. I added Sensitivity to it to change the sensitivity of camera move based on being in the aim state or not. “Set Rotate on Move” is responsible for rotating the player when aiming towards the aim direction. “Set Speed to Zero on Aim” will set the speed of the player to zero when he is aiming to force the player to not move when aiming.

**AI Controller:**

This class which is attached to the AI is responsible for AI actions. The attributes’ names show that what are they responsible for. runMoveDuration and walkMoveDuration will set the speed of AI when he is fast or slow based on the events that happened in the game. “Choose Path”, “Choose Path to Exit”, “Move Forward”, “Move Backward”, and “Move to Location” are the implementation of the heuristic pathfinding algorithm. “Choose Action” is the method responsible for choosing the appropriate action in each cell. The actions are “Move”, “Drop Bomb”, “Shoot Player”, “Shoot Enemy”, and “Punch”. The “Collect Key” method will call when the AI collects the key and the “Won Game” method will call when the AI exit the maze and win the game.

**Enemy:**

This abstract class is the parent of all types of enemies which is responsible for pathfinding of the enemies. The moveDuration variable sets the speed of the enemy to move from a cell to another one. “Move To Location”, “Move”, and “Choose Path” methods are responsible for the random pathfinding. The “Die” method will call when the enemy is been hit by a bullet.

**Bomber, Drone, and Runner:**

Each class is responsible for the specific action that each enemy does. Bomber has bombCooldownTime and attackRange attributes and drop a bomb in the “Drop Bomb” method. The drone has shootCooldownTime and attackRange attributes and shoots a bullet towards the player or AI. Runner overrides the “Move To Location” method to include the animation and higher speed.

**References**

* <https://github.com/john-science/mazelib/blob/master/docs/MAZE_GEN_ALGOS.md#backtracking-generator>
* <https://assetstore.unity.com/packages/essentials/starter-assets-third-person-character-controller-196526>
* <https://unitycodemonkey.com/video.php?v=FbM4CkqtOuA>
* <https://www.mixamo.com/>
* <https://creately.com/>
* <https://assetstore.unity.com/packages/vfx/particles/fire-explosions/fire-explosion-vfx-48795>
* <https://assetstore.unity.com/packages/3d/props/rust-key-167590>
* <https://assetstore.unity.com/packages/3d/vehicles/air/simple-drone-190684>
* <https://assetstore.unity.com/packages/2d/textures-materials/sky/starfield-skybox-92717>
* <https://assetstore.unity.com/packages/3d/props/weapons/yughues-free-bombs-13147>
* <https://www.myfreetextures.com/seven-free-grass-textures-or-lawn-background-images/>
* <https://www.textures.com/download/WoodFine0079/124251>

1. <https://github.com/john-science/mazelib/blob/master/docs/MAZE_GEN_ALGOS.md#backtracking-generator> [↑](#footnote-ref-1)
2. <https://assetstore.unity.com/packages/essentials/starter-assets-third-person-character-controller-196526> [↑](#footnote-ref-2)
3. <https://unitycodemonkey.com/video.php?v=FbM4CkqtOuA> [↑](#footnote-ref-3)
4. <https://www.mixamo.com/> [↑](#footnote-ref-4)
5. <https://creately.com/> [↑](#footnote-ref-5)