Procedurally Generated Tower Defense 3D

Tori Broadnax[1], Jeffrey Do[1], Richard Roberts[1], and Minh Vu[1]

[1]George Mason University

December 9, 2021

Abstract

The overall goal of this project was to create a 3D tower defense game that has procedurally generated levels. The report is broken up into three primary sections: Requirements, Technologies, and Timeline. In the requirements section, we will be quoting mostly from the Project Proposal–you do not need to reference it because enough context will be provided. We will identify what was planned to do, what was accomplished, how it was accomplished, and what was failed to accomplish. In the technology section, we will describe what technologies were used and how they aided us. Additionally, we will describe what parts beyond that given technology that were implemeneted by us. Lastly, the timeline section will chronologically detail what steps we took to reach our final product.

Contents

1 Requirements **3**

1.1 Procedurally Generated Level 3

1.2 User Interface 3

1.3 Weapons 4

1.4 Enemies 4

1.5 Reach Requirements 4

2 Technologies **4**

3 Timeline **5**

4 References **5**

1 Requirements

1.1 Procedurally Generated Level

> The tower defense level will be procedurally generated. There should be at least one path from enemy spawn to the tower. The tower will be placed randomly. There should be tiles that cannot have weapons placed on them (like trees), and tiles that can. Weapons should not be able to be placed on the path.

This was the primary requirement of the project, and it was the most difficult to implement. The intial stages of the algorithm were kind attrocious looking back on it. The algorithm then, and still, currently only works for a ten by ten grid/map size. The basis of this algorithm is that a start and end point would be generated. The start would be where enemies spawn, and the end would be where the tower is or the enemies' goal. A depth first search would then be conducted and start until it found the endpoint, it would then set all the tiles that it cross to get to that path to be a value of "walkable".

```
procedure GENERATE-MAP($map$)
    $start \leftarrow (0, \text{RANDOM}(map.size.y))$
    $end \leftarrow (map.size.y - 1, \text{RANDOM}(map.size.y))$
    DEPTH-FIRST-SEARCH($start, end$)
end procedure
```

The problem with the algorithm was that it would produce a path that would be touching itself, making it uninteresting and kind of illogical. Additionally, there would be no way for the enemy to traverse this path in an "attractive" manner most of the time. The solution was to implement something nodal pathfinding using A*. The concept is that we would choose a set of waypoints and pass them in to A* so that it would produce an optimal path between them.

```
procedure GENERATE-MAP($map$)
    $start \leftarrow (0, \text{RANDOM}(map.size.y))$
    $end \leftarrow (map.size.y - 1, \text{RANDOM}(map.size.y))$
    $waypoints \leftarrow \emptyset$
    SET-ADD($waypoints, start$)
    for $i \leftarrow 3, i < 7, i = i + 2$ do
        $pos \leftarrow (i, \text{RANDOM}(map.size.y))$
        SET-ADD($waypoints, pos$)
    end for
    SET-ADD($waypoints, end$)
    for $i \leftarrow 0$ do
        A*-PATHFINDING($waypoints[i], waypoints[i + 1]$)
    end for
end procedure
```

This solution proved to work well.

1.2 User Interface

There should be a tower health indicator. There should be a menu for selecting weapons to build. Once selected, the player should be able to select a tile, and the weapon should build if funds are sufficient. There should be a currency indicator. There should be buttons for starting and stopping the movement of enemies (like a pause).

1.3 Weapons

There should be mutiple types of weapons that vary in damage, rate of fire, and range. Weapons will fire projectiles at enemies in their range. The player should be able to build these weapons on tiles in exchange for currency.

1.4 Enemies

Enemies will have pathfinding towards to tower. When enemies reach the tower, a corresponding tower health deduction should take place. Enemies will have varying health, speed, and currency drops. Enemies should detect collisions with projectiles and take damage from them.

1.5 Reach Requirements

- Upgradable Weapons
- Weapon Selling

2 Technologies

- Unity

3 Timeline

11/07	Added some camera controller movement using free asset.
11/09	Very simple procedural algorithm with DFS to generate path. Ther was still a problem that the path touched.
11/25	Added A* pathfinding and used nodal pathfinding to generate enemy path. Added enemies, added enemy spawner, added enemy movement using the path from the map, documented the enemy script and destroyed the enemy object when it reached the tower.
11/26	Added weapons and weapons used LERP to rotate towards the nearest enemy. Added the balista and the canon prefabs with their respective projectiles.
11/28	Added blaster projectile. Added a building system so that when players clicked a tile, a corresponding weapon would be placed there. Before the weapons were just spawned in before hand. Additionally, a UI element was added so that only tiles that were valid building tiles would be highlighted yellow. Added a weapon menu so that users could select which weapon they wanted to build. Also added github actions so that the project would be automatically run through the default compile tests on commit. Futhermore, if the project is pushed with a tag, it will create a release for Windows.
11/29	Fixed some bugs with the tile highlighting UI.
11/30	Added the foundations for the player, which would be a singleton and contain the health, score, and currency.
12/03	Added player, health, score, and currency indicators on the UI.
12/07	Added basis for the player and enemies taking damage.
12/09	Refactored player damage from a fixed number to dynamic one determined by the projectile prefab. Score now increments and currency increases when enemies are destroyed. The currency dropped based on how much the enemy is worth. Created the techincal report using LaTeX(this file), presentation, and demo video.

4 References

- 3D Tilemap in Unity
- Brackeys Unity Tower Defense
- Manual Tilemap Unity
- Procedural Tilemap
- Kenney Tower Defense
- Unity GitHub Actions