University of Missouri - Columbia

Final Report

By Zachary Summers, Isaac Sexe

5/12/22

CMP_SC 4610 Computer Graphics I

Professor Ye Duan

Project Summary

Introduction

For our project, we decided to do a 3D World Generation game that used a procedurally generated dungeon as the environment. The player spawns in the dungeon after it's been generated and they are tasked with reaching the boss room and defeating the boss to move onto the next level of the dungeon. Currently, our game is still an alpha build and doesn't feature too much variety or gameplay mechanics and instead focuses on the generation of the dungeon and moving through it.

Roles

Our team split focus between the dungeon itself and the character models. Isaac Sexe focused on the dungeon generation, while Zach Summers worked on developing the characters and their interactions. Afterward, Isaac Sexe merged the two parts. Both of us focused on our respective roles while completing the presentation and this report as well.

Dungeon Generation

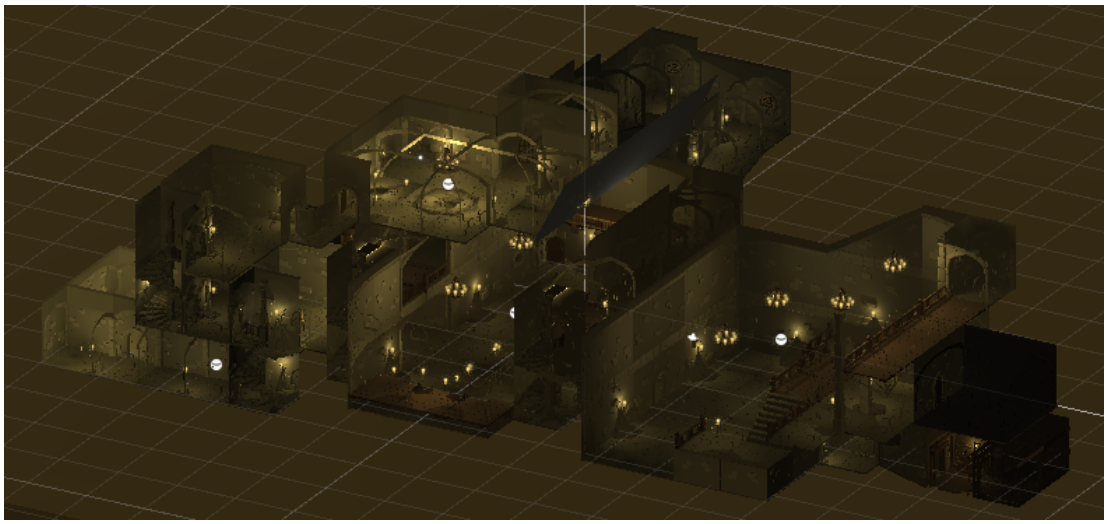
Introduction

The dungeon generation included the algorithm for generation, building, and populating the dungeon. This is all done at the beginning of the scene and almost everything is instantiated through scripts. After the dungeon is fully built a NavMesh is generated to allow for the AI to navigate the environment.

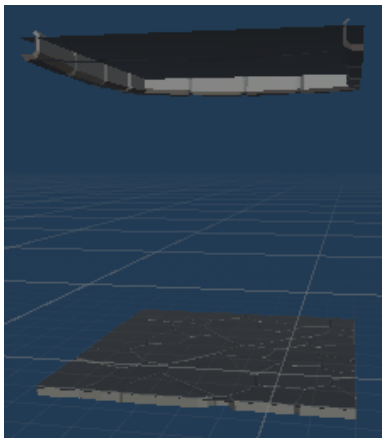
Assets

The assets used for the dungeon generation include models used for the dungeon generation. These assets are from Ultimate Low Poly Dungeon by Broken Vector on the Unity Asset Store. The main models used were the modified versions of the Dungeon_Demo Scene, Dungeon_Custom_Center Prefab, Dungeon_Wall_Var1 Prefab, Chandelier Prefab, and the Torch_Wall Prefab, which were all included in the Asset Store package. The Demo Scene was used for the main menu, the room prefab was used as a room unit, the wall prefab was used as a wall for the dungeon, and the Chandelier and Torch_Wall were placed in the dungeon for lighting.

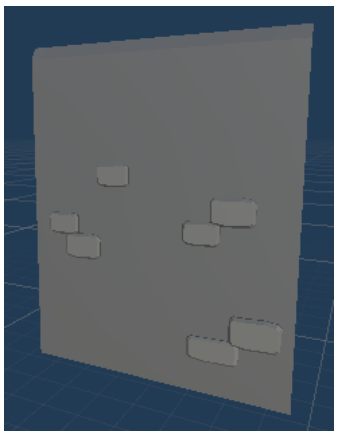
Dungeon_Demo Scene



Dungeon_Custom_Center Prefab



Dungeon_Wall_Var1 Prefab



Chandelier Prefab

Torch_Wall Prefab

Implementation

The dungeon generation was implemented using a modified version of Vazgriz methods found on his website, https://vazgriz.com/119/procedurally-generated-dungeons/, and his GitHub repository, https://github.com/vazgriz/DungeonGenerator. Originally, we intended to implement our method based on the description of his algorithm, but due to wasting time on debugging and stumbling on the generation implementation, we decided to take his original implementation and modify it for our purposes.

Data Structures

For the dungeon generation, the algorithm first builds a 2D grid of rooms and then works with that. For this, there are several different data structures featured in the implementation with varying degrees of complexity and importance. We will be discussing the main ones that are featured in the dungeon generation, but there is some quality of life ones that Vazgriz used and those will be linked in the references. The data structures we will be covering in the report are the data structures used for building the 2D grid and for the grid itself.

Grid2D Object

```
public class Grid2D<T> {
    3 references
    T[] data;

    10 references
    public Vector2Int Size { get; private set; }
    4 references
    public Vector2Int Offset { get; set; }

    2 references
    public Grid2D(Vector2Int size, Vector2Int offset) {
        Size = size;
        Offset = offset;

        data = new T[size.x * size.y];
    }
```

CellType Enumerator

```
enum CellType {
    10 references
    None,
    3 references
    Room,
    4 references
    Hallway
}
```

Room Object

```
class Room {
    38 references
    public RectInt bounds;

    2 references
    public Room(Vector2Int location, Vector2Int size) {
        bounds = new RectInt(location, size);
    }
```

The Grid and Room objects both have methods associated with them that help will the algorithm but their uses are straightforward forward so we will be omitting them.

Dungeon Generation

Once the scene starts, the script initializes the minimap location and creates empty GameObjects that are used for organization in the Hierarchy Panel. Afterward, the script starts generating the dungeon.

```
void Start() {
    1 reference
    GameObject minimap = GameObject.FindGameObjectWithTag("SecondaryCamera");
    minimap.transform.position = new Vector3((size.x/2) * roomScale, 100, (size.y/2) * roomScale);
ferences

    23 references
    demoAnchor = new GameObject("demoAnchor");
    8 references
    dungeonAnchor = new GameObject("dungeonAnchor");
erences

    6 references
    Debug.Log("Generating Dungeon...");
    Generate();
    5 references
    Debug.Log("...Finished!");
23 references
}
```

The generation begins by initializing variables that are used throughout the script. The generation begins and is done in 6 parts; generating the rooms, calculating a Delaunay triangulation, trimming the graph, pathfinding hallways, building the dungeon, and populating the rooms.

```
0 references
void Generate() {
    // Allows for customs seeded runs
    if(seed < 0) seed = new Random().Next();
    random = new Random(seed);

    grid = new Grid2D<CellType>(size, Vector2Int.zero);
    rooms = new List<Room>();

    Debug.Log("...Generating rooms...");
    PlaceRooms();

    Debug.Log("...Generating Triangulation...");
    1 reference
    Triangulate();
    // BuildDemoDelaunay();

    Debug.Log("...Generating Hallways...");
    CreateHallways();
    // BuildDemoHallways();

    Debug.Log("...Pathfinding Hallways...");
    PathfindHallways();

    Debug.Log("...Building Dungeon...");
    BuildDungeon();

    Debug.Log("...Populating Dungeon...");
    PopulateDungeon();
}
```

Generating the Rooms

Room generation starts by finding a random location and size for the room. A room object is created using these parameters and a buffer room is created using a room 1 unit bigger than the previous room object. The buffer is then compared with every room created so far and if it intersects with any of them the loop ends and that room will not be added. The room is then checked to ensure it is within bounds. If both of these checks are okay, the room is added to the rooms list and the cube is created to represent to room on the minimap. After that, the room object is added to the grid and the loop continues. This loop continues for however attempts are given in the script or until the max number of rooms are created.

```
void PlaceRooms() {
    for (int i = 0; i < roomCount; i++) {
        Vector2Int location = new Vector2Int(
            random.Next(0, size.x),
            random.Next(0, size.y)
        );

        Vector2Int roomSize = new Vector2Int(
            random.Next(roomMinSize.x, roomMaxSize.x + 1),
            random.Next(roomMinSize.y, roomMaxSize.y + 1)
        );

        1 reference
        bool add = true;
        Room newRoom = new Room(location, roomSize);
        Room buffer = new Room(location + new Vector2Int(-1, -1), roomSize + new Vector2Int(2, 2));

        foreach (var room in rooms) {
            if (Room.Intersect(room, buffer)) {
                add = false;
                break;
            }
        }

        if (newRoom.bounds.xMin < 0 || newRoom.bounds.xMax >= size.x
            || newRoom.bounds.yMin < 0 || newRoom.bounds.yMax >= size.y) {
            add = false;
        }

        if (add) {
            rooms.Add(newRoom);
            PlaceRoom(newRoom.bounds.position, newRoom.bounds.size);

            foreach (var pos in newRoom.bounds.allPositionsWithin) {
                grid[pos] = CellType.Room;
            }
        }
        // breaks if room exceeds a certain number
        if(rooms.Count >= maxNumRooms) break;
    }
}
```

Delaunay Triangulation

Vazgriz implements a version of Simon Zeni's script that can be found on his GitHub repository, https://github.com/Bl4ckb0ne/delaunay-triangulation. This performs Bowyer-Watson's algorithm for calculating a Delaunay triangulation given as a series of vertices. These vertices are the center of each room object.

```
void Triangulate() {
    List<Vertex> vertices = new List<Vertex>();

    foreach (var room in rooms) {
        vertices.Add(new Vertex<Room>((Vector2)room.bounds.position + ((Vector2)room.bounds.size) / 2, room));
    }

    delaunay = Delaunay2D.Triangulate(vertices);
}
```

Trimming the Graph

To trim the graph, Vazgriz uses Prim's algorithm for calculating a minimum spanning try, for which the weight of an edge is the distance from the vertices and the starting point is the first edge's U vertex. This minimum spanning tree then has its edges randomly added back to it so create some loops in the dungeon.

```
void CreateHallways() {
    List<Prim.Edge> edges = new List<Prim.Edge>();

    0 references
    foreach (var edge in delaunay.Edges) {
        edges.Add(new Prim.Edge(edge.U, edge.V));
    }

    List<Prim.Edge> mst = Prim.MinimumSpanningTree(edges, edges[0].U);


    selectedEdges = new HashSet<Prim.Edge>(mst);
    var remainingEdges = new HashSet<Prim.Edge>(edges);
    1 reference
    remainingEdges.ExceptWith(selectedEdges);

    foreach (var edge in remainingEdges) {
        if (random.NextDouble() < 0.125) {
            selectedEdges.Add(edge);
        }
    }
}
```

Pathfinding Hallways (A*)

Vazgriz implements a version of an A* algorithm to search for the optimal path from a starting point to the endpoint. The starting point and endpoint are the vertices from the edges calculated during the trimmed graph. The cost for the heuristic function is calculated using the distance from the current position to the end position, +10 points if moving into a room cell, +5 points if moving into an empty cell, and +1 points if moving into a hallway.

```
void PathfindHallways() {
    DungeonPathfinder2D aStar = new DungeonPathfinder2D(size);

    0 references
    foreach (var edge in selectedEdges) {
        var startRoom = (edge.U as Vertex<Room>).Item;
        var endRoom = (edge.V as Vertex<Room>).Item;

        var startPosf = startRoom.bounds.center;
        var endPosf = endRoom.bounds.center;
        var startPos = new Vector2Int((int)startPosf.x, (int)startPosf.y);
        var endPos = new Vector2Int((int)endPosf.x, (int)endPosf.y);

        1 reference
        var path = aStar.FindPath(startPos, endPos, (DungeonPathfinder2D.Node a, DungeonPathfinder2D.Node b) => {
            var pathCost = new DungeonPathfinder2D.PathCost();

            pathCost.cost = Vector2Int.Distance(b.Position, endPos);    //heuristic

            if (grid[b.Position] == CellType.Room) {
                pathCost.cost += 10;
            } else if (grid[b.Position] == CellType.None) {
                pathCost.cost += 5;
            } else if (grid[b.Position] == CellType.Hallway) {
                pathCost.cost += 1;
            }

            pathCost.traversable = true;

            return pathCost;
        });
```

After a path is calculated the grid is updated and a cube is placed to represent the hallway on the map.

```
        if (path != null) {
            for (int i = 0; i < path.Count; i++) {
                var current = path[i];

                if (grid[current] == CellType.None) {
                    grid[current] = CellType.Hallway;
                }

                if (i > 0) {
                    var prev = path[i - 1];

                    var delta = current - prev;
                }
            }

            foreach (var pos in path) {
                if (grid[pos] == CellType.Hallway) {
                    PlaceHallway(pos);
                }
            }
        }
    }
}
```

Building the Dungeon

After the grid is completed, the dungeon is ready to be built. The grid is iterated through and any cell that is a room has a room built there, and any cell that is a hallway has a hallway built there. Afterwards, the Chandeliers are placed in the center of each room.

```
void BuildDungeon(){
    for(int x=0; x<grid.Size.x; x++){
        for(int y=0; y<grid.Size.y; y++){
            CellType cell = grid[x,y];
            1 reference
            switch(cell){
                case CellType.Room:
                    BuildRoom(x,y);
                    break;
erence

                case CellType.Hallway:
                    BuildHallway(x,y);
                    break;
            1 reference
            }
        }
    }

    foreach(var room in rooms){
        // hard code height of chandelier
        // could just set y in prefab and use that but for now this good
        Vector3 pos = new Vector3(room.bounds.center.x * roomScale, 4.6f, room.bounds.center.y * roomScale);
        GameObject temp = Instantiate<GameObject>(chandelierPrefab, pos, chandelierPrefab.transform.rotation);
        temp.transform.SetParent(dungeonAnchor.transform);
    }
}
```

This is the function used to build each hallway. The room function is similar with minimal changes to some values.

```
void BuildHallway(int x, int y)
{
    GameObject temp = Instantiate<GameObject>(roomPrefab,
        new Vector3(x * roomScale + roomScale / 2, 0, y * roomScale + roomScale / 2),
        Quaternion.identity);
    temp.transform.SetParent(dungeonAnchor.transform, true);

    GameObject prefab = wallPrefab;
    if (random.NextDouble() < 0.25) prefab = litWallPrefab;

    //check walls
    // for each cell next to current check...
    //      if edge end of grid, if so build wall
    //      else if edge is empty, if so build wall
    if (y - 1 < 0)
    {
        GameObject wallTmp = Instantiate<GameObject>(prefab,
            new Vector3(x * roomScale + roomScale / 2, 0, y * roomScale + roomScale / 2),
            prefab.transform.rotation);
        wallTmp.transform.SetParent(dungeonAnchor.transform, true);
    }
    else if (grid[x, y - 1] == CellType.None)
    {
        GameObject wallTmp = Instantiate<GameObject>(prefab,
            new Vector3(x * roomScale + roomScale / 2, 0, y * roomScale + roomScale / 2),
            prefab.transform.rotation);
        wallTmp.transform.SetParent(dungeonAnchor.transform, true);
    }
    // This is repeaded for [x, y+1], [x-1, y], and [x+1, y]      You, now • Uncommitted
```

Populating Rooms

Now that the dungeon is built each room needs to be populated with either a boss, player, or enemies. The Boss room is calculated by finding rooms that have one entry point and no exit. The player room is randomly selected from the remaining rooms, and the rest are enemy rooms.

```
void PopulateDungeon()
{
    List<Room> roomsLeft = new List<Room>(rooms);
    Room bossRoom;
    Room spawnRoom;

    List<Room> tempRooms = new List<Room>();
    foreach (var edge in selectedEdges)
    {
        tempRooms.Add((edge.U as Vertex<Room>).Item);
        tempRooms.Add((edge.V as Vertex<Room>).Item);
    }
    List<Room> possibleRooms = new List<Room>();
    foreach (var room in roomsLeft)
    {
        int count = 0;
        foreach (var tempRoom in tempRooms) if (room == tempRoom) count += 1;
        if (count == 1) possibleRooms.Add(room);       You, 1 second ago • Uncommitted change
    }
    if (possibleRooms.Count > 0) bossRoom = possibleRooms[random.Next(possibleRooms.Count)];
    else bossRoom = rooms[0]; // backup incase there is not a single room
    roomsLeft.Remove(bossRoom);

    spawnRoom = roomsLeft[random.Next(roomsLeft.Count)];
    roomsLeft.Remove(spawnRoom);
```

After the rooms are found, each one has a spawner instantiated at its center.

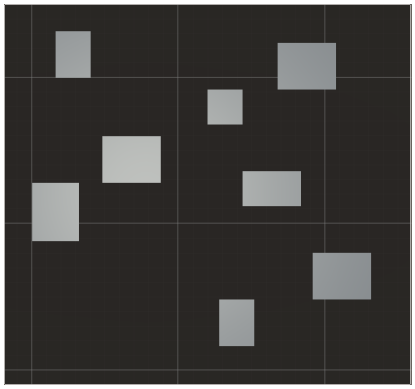
```
GameObject temp;
// Spawn boss
temp = Instantiate<GameObject>(bossSpawner);
temp.transform.position = new Vector3(bossRoom.bounds.center.x * roomScale, 0, bossRoom.bounds.center.y * roomScale);
temp.transform.SetParent(dungeonAnchor.transform);

// Spawn enemies
foreach (var room in roomsLeft)
{
    temp = Instantiate<GameObject>(enemySpawner);
    temp.transform.position = new Vector3(room.bounds.center.x * roomScale, 0, room.bounds.center.y * roomScale);
    temp.transform.SetParent(dungeonAnchor.transform);
}
// Spawn player
temp = Instantiate<GameObject>(playerSpawner);
temp.transform.position = new Vector3(spawnRoom.bounds.center.x * roomScale, 0, spawnRoom.bounds.center.y * roomScale);
temp.transform.SetParent(dungeonAnchor.transform);
```

NavMesh Generation

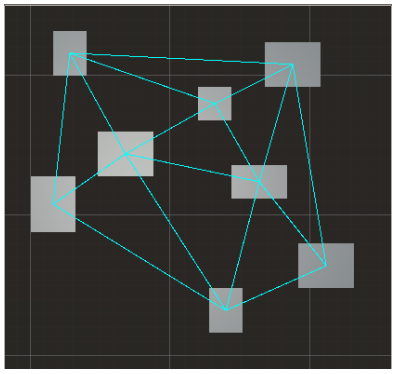
The NavMesh is generated at runtime using Unity Technology's NavMeshComponents package from their GitHub repository, https://github.com/Unity-Technologies/NavMeshComponents. The room prefabs had a plane object added to them that included a NavMeshSurface script added to. A script called NavMeshBaker that bakes a NavMesh 0.5 seconds after the scene loads.

Results

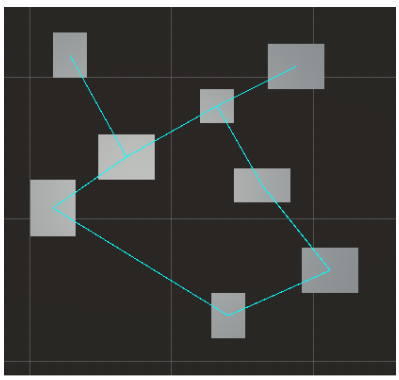
Placing Rooms



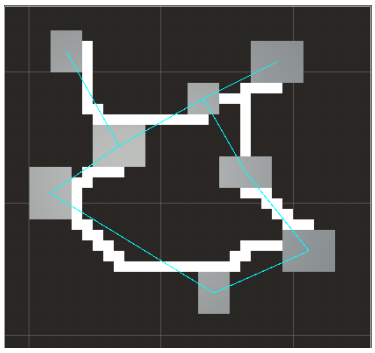
Delaunay Triangulation



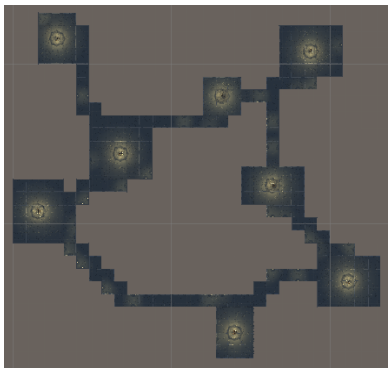
Trimming the Graph



Pathfinding Hallways



Building the Dungeon



Populating the Rooms

Each colored circle represents a specific type of room. The black is a boss, the blue is a player, and the red is an enemy room.

Game Sandbox

Introduction

The game sandbox consists of the components that make up the player experience. This includes possible player actions and enemies.

Player

The player is implemented using an open-source first-person controller which can be imported using the unity store.

(https://assetstore.unity.com/packages/3d/characters/modular-first-person-controller-189884)

Additional modifications have been made on top of this system. It allows the player to move, run, jump and crouch. The addition of multiple weapons that can be switched between was made, allowing the player to attack with either the sword (1 on the keyboard) or the rifle (2 on the keyboard) by clicking.

When clicking with the rifle equipped, a bullet prefab is instantiated at the location and with the direction needed.

```
if (Input.GetMouseButtonDown(0))
{
    Instantiate(bullet, transform.position + new Vector3(0, .11f, 0), transform.rotation);
    bullet.GetComponent<Rigidbody>().AddForce(bullet.transform.forward * 10);

}
// transform.Rotate((Input.GetAxis("Mouse Y") * 2.3f), 0, 0, Space.Self);

// Should add something to check for max rotation so weapon doesn't do flips
transform.rotation = Quaternion.Euler(-Camera.main.transform.rotation.eulerAngles.x, transform.rotation.eulerAngles.y, transform.rotation.eulerAngles.z);
```

In the above code section, when the mouse is clicked, a bullet is instantiated at the rifle, with slight adjustments. Then force is added to make it move forward. Outside of this, the rifle is always moving to align itself with the player's center of vision by rotating to aim toward the mouse.

The UI was implemented using a canvas set to display itself on top of the camera at all times. This UI displays the player's health, which is decreased when touching enemies.

Enemy

The enemy in the game has a zombie model imported from the unity store. Using the Unity NavMesh system, the enemy can follow the player and deal damage to it on collision.

```
if(disFromPlayer < 10){
    agent.destination = Playerpos.position;
}
```

When a collision is detected with the sword or a bullet, the enemy takes damage and is destroyed upon losing all of its health.

```
void OnTriggerEnter(Collider collision)
{
    if(collision.tag == "Bullet")
    {
        print("collision");
        health -= 10;
        Destroy(collision.gameObject);
    }
}
```

Project Development

Issues

There are quite a few issues with the game currently. The gun isn't aligned properly so when you fire the bullets tend to be miss-aligned. The sword doesn't function properly either and when swung counts as the player hit the box so it will damage the player. The animations don't work so the enemies just glide to the player. The fadeout screen doesn't work on teleportation. There are also some inefficiencies with the implementation of the dungeon scripts and they all could be optimized.

Future Goals

The future goals for this project would be not only fixing the previously stated issues but also expanding upon the game. Currently, the game isn't much of a game and there's not much to do other than move from level to level. This could be improved by adding a leaderboard, reward rooms, rewards for beating the bosses, more enemies, more characters, etc. We could also add sound and improve the visuals of the game to make it more enjoyable.

References

Dungeon Generation

1. https://vazgriz.com/119/procedurally-generated-dungeons/
    - Vazgriz's Website
2. https://github.com/vazgriz/DungeonGenerator
    - Vazgriz's GitHub Repository
3. https://github.com/Unity-Technologies/NavMeshComponents
    - Unity Technology's NavMeshComponents Package
4. https://assetstore.unity.com/packages/3d/environments/dungeons/ultimate-low-poly-dungeon-143535#publisher
    - Broken Vector's Ultimate Low Poly Dungeon Asset Package