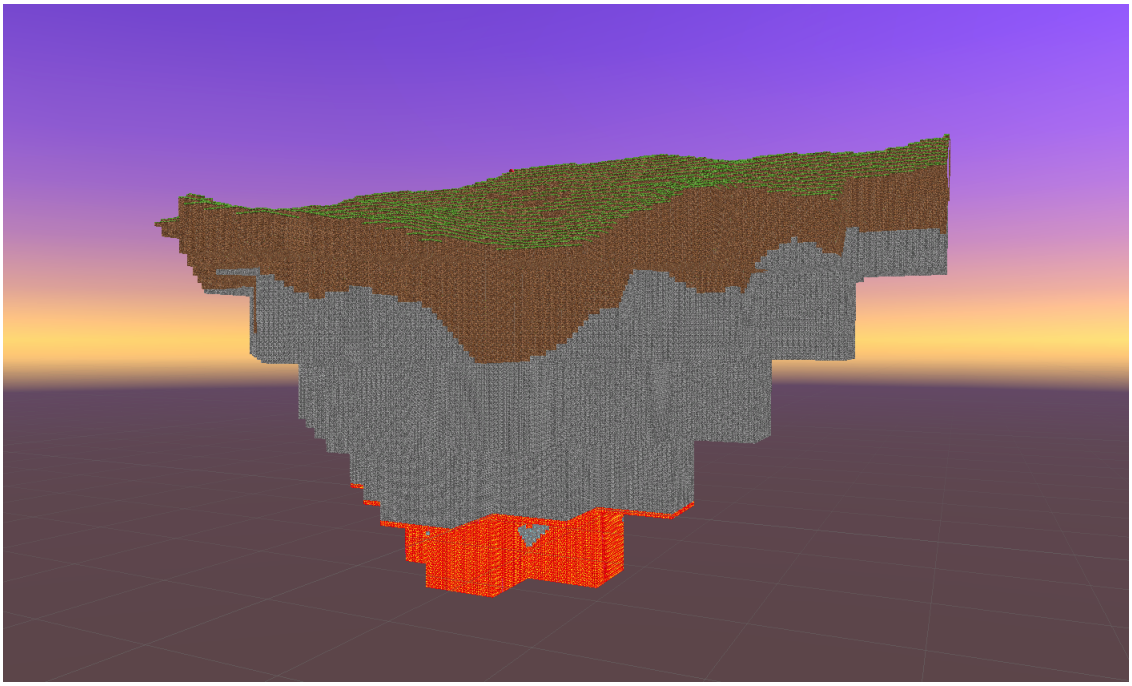
Instituto Politécnico de Lisboa (IPL)

Instituto Superior de Engenharia de Lisboa

Departamento de Engenharia da Eletrónica e Telecomunicações e de Computadores (ADEETC)

Interacção em Ambientes Virtuais - LEIM - 2425SV

Trabalho nº 1 - Mundo Minecraft



Docente: Arnaldo Abrantes

Realizado por:
Diogo Santos - 48626
Tatiana Damaya - 50299

23 de março de 2025

Conteúdo

Lista de Figuras

1 Introdução

Com este primeiro trabalho da unidade curricular de Interação em Ambientes Virtuais, temos como objetivo desenvolver um jogo similar ao Minecraft, aplicando os conceitos aprendidos em aula para criar um ambiente voxelizado gerado proceduralmente.

Para a construção do mundo virtual, utilizamos uma abordagem baseada em voxels, onde cada elemento do cenário é representado por um Block (bloco). Estes blocos são organizados em grupos chamados de Chunks, que por sua vez compõem o World (mundo) do jogo. Essa estrutura hierárquica permite otimizar o carregamento e a renderização, garantindo um melhor desempenho.

A geração do terreno é feita através do algoritmo de Ruído de Perlin, que permite criar paisagens naturais e suaves ao invés de terrenos completamente aleatórios. Esse método é amplamente utilizado em jogos para gerar terrenos de forma procedural, tornando cada mundo único.

Para otimizar a manipulação do ambiente, implementámos o Flood Fill Algorithm, que é utilizado para detecção e propagação de mudanças no cenário, por exemplo, quando um bloco é removido ou alterado.

Além disso, para melhorar a eficiência do código e evitar travamentos na execução, utilizámos Corotinas, que permitem dividir a execução de tarefas pesadas em múltiplos frames, garantindo que a geração dos chunks ocorra de forma mais fluida sem comprometer a taxa de frames por segundo (FPS).

2 Alterações - Afinação dos parâmetros do(s) gerador(es) de ruído de Perlin e desempenho para diferentes soluções adotadas (em FPS, número de triângulos, vértices, etc.)

Após terminadas as aulas, a performance do jogo precisava ser melhorada devido à geração de *chunks* estar a causar travões no jogo assim como *bugs* na geração de terreno.

Estes problemas não foram corrigidos a 100%, mas conseguiram ser mitigados. Foi sacrificada *performance* inicialmente no momento de geração do mundo. O raio de renderização foi deixado em 3 e a altura em que grutas começam a ser geradas abaixo da linha do mar foi reduzido de -10 para -25. Esta última mudança foi feita pois a geração de grutas foi o elemento mais prejudicial para a causa de *bugs* na geração do mundo. Assim, a experiência de terreno se tornou mais agradável, mesmo com pequenos travões.

Isto implica que o número de triângulos foi reduzido em cerca de 20% em cada instância. Os travões de *gameplay* foram reduzidos consideravelmente, sacrificando um pouco de *render distance*, tentando ao máximo não quebrar a imersão.

3 Implementação

Nesta secção, apresentamos a estrutura do código e os principais algoritmos utilizados na construção do mundo voxelizado. Foram implementadas as seguintes classes principais:

- **Block.cs** - Representa os blocos individuais no mundo voxel.
- **Chunk.cs** - Responsável por organizar e gerar os blocos dentro de uma determinada área.
- **World.cs** - Gere o mundo como um todo e controla a geração dos chunks.
- **Utils.cs** - Contém funções auxiliares, incluindo a geração de terreno usando Perlin Noise.
- **BlockInteraction.cs** - Lida com a interação do jogador com os blocos (colocação, remoção, etc.).

3.1 Classe Block

A classe `Block` representa um bloco individual dentro do mundo voxel. Cada bloco pode ser de diferentes tipos, incluindo GRASS, DIRT, STONE, AIR e adicionámos LAVA, DIAMOND, BUSH, CACTUS para o mundo que idealizamos.

Listing 1: Definição da Classe Block

```
public class Block {
    public enum BlockType {
        GRASS, DIRT, STONE, LAVA, DIAMOND, BUSH, CACTUS, AIR
    };
    BlockType bType;
    Chunk owner;
    Vector3 pos;
    bool isSolid;
}
```

A implementação da lava e do diamante foi feita adicionando novos tipos de bloco na enumeração `BlockType`. A lava foi pensada para emitir luz e causar dano ao jogador quando tocada e programada para junto com os blocos de diamante aparecerem aleatoriamente em camadas profundas do terreno.

3.2 Classe World

A classe `World` gere a geração dos chunks e organiza o mundo do jogo. Ela controla a inicialização dos blocos e a renderização do ambiente.

Listing 2: Inicialização do Mundo

```
void Start() {
    for (int x = 0; x < worldSize; x++) {
        for (int z = 0; z < worldSize; z++) {
            CreateChunk(x, z);
        }
    }
}
```

Cada chunk é criado e armazenado em uma estrutura de dados, garantindo melhor desempenho na renderização.

3.3 Classe Utils

A classe `Utils` fornece funções auxiliares para gerar o terreno usando o algoritmo de Ruído de Perlin.

Listing 3: Geração Procedural com Perlin Noise

```
public static int GenerateHeight(float x, float z) {
    return (int)(maxHeight * Mathf.PerlinNoise(x * scale, z * scale
   ));
}
```

Esse método permite criar paisagens suaves e naturais, evitando terrenos completamente aleatórios.

3.4 Classe BlockInteraction

A classe `BlockInteraction` lida com a remoção e adição de blocos no mundo. Ela utiliza um sistema de *raycasting* para detectar colisões e interações do jogador com o ambiente.

Listing 4: Interação com Blocos

```
if (Input.GetMouseButtonDown(0)) {
    RaycastHit hit;
    if (Physics.Raycast(ray, out hit)) {
        Destroy(hit.collider.gameObject);
    }
}
```

Esse sistema permite que o jogador minere blocos, incluindo os de diamante e lava.

3.5 Tentativa de Adição de Cactos e Arbustos

Tentamos adicionar elementos decorativos ao mundo, como cactos e arbustos. A ideia era gerar essas estruturas aleatoriamente em biomas apropriados, como desertos.

Listing 5: Tentativa de Geração de Cactos

```
void GenerateCactus(int x, int y, int z) {
    int height = Random.Range(2, 5);
    for (int i = 0; i < height; i++) {
        placeBlock(x, y + i, z, BlockType.CACTUS);
    }
}
```

Infelizmente, essa funcionalidade apresentou problemas na renderização e precisaria de mais tempo para ser corrigida. O problema parece estar na forma como os blocos são gerados dentro dos chunks.

4 Conclusão

Neste projeto, conseguimos implementar um ambiente voxelizado similar ao Minecraft, utilizando técnicas de geração procedural e otimização de chunks. Através do uso do Ruído de Perlin, conseguimos criar um terreno dinâmico e visualmente interessante, e a introdução de corotinas ajudou a mitigar problemas de desempenho durante a geração do mundo.

Para a movimentação do player, reutilizámos um código desenvolvido no ano passado na unidade curricular de Animação e Ambientes Virtuais (AAV).

Durante o desenvolvimento, tentámos adicionar elementos decorativos, como arbustos e cactos, para enriquecer a paisagem. No entanto, não conseguimos obter um resultado funcional dentro do tempo disponível, pelo que essa funcionalidade acabou por não ser incluída na versão final.

Se tivéssemos mais tempo, gostaríamos de aprofundar a otimização da geração do mundo, melhorar o desempenho da renderização e adicionar mais elementos interativos ao ambiente, garantindo um ambiente mais rico e interativo. Apesar dos desafios encontrados, o projeto permitiu consolidar conhecimentos sobre voxels, geração procedural e otimização gráfica em Unity.