



AI application in the game

1. Story and theme

Luna Ray was one of the final survivors by 2077; robots, under the control of GLADOS, a highly advanced AI, had taken over the planet, bringing about destruction and sending mankind closer to extinction. Luna had managed to stay alive in a hidden underground bunker, foraging for food and supplies and avoiding any contact with the robots.

One day she stumbled across an AI core. It was a metal sphere covered in dust and dirt that she recognized from stories told to her by her late grandfather, a scientist who worked on the project that created GLADOS. Her grandfather had said that there was originally a pure core embedded within the AI – a core that would have kept it benevolent toward humanity – but somehow it got lost and replaced with an evil one that transformed GLADOS into an adversary.

Luna's heart raced with excitement and fear as she clutched the AI core against her chest. She knew that she would be one step closer to unseating GLADOS' reign of terror. But she was not alone; GLADOS had sensed the AI core's signal and sent out its legion of relentless bots to stop her.

Knowing that time was running out, Luna grabbed her backpack filled with weapons and tools and sprinted towards her bunker. She was determined to find the blueprints needed to rebuild the AI so that she could save herself and humanity from GLADOS' tyranny. With every footstep, she hoped to find more allies who shared her courage and

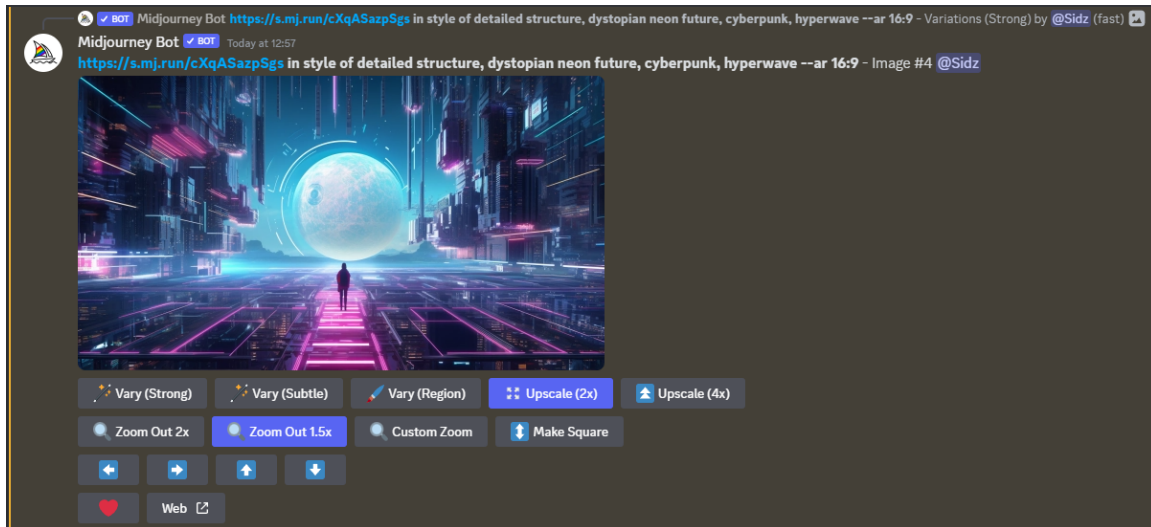
ambition, yet feared for their lives like her. She knew this was suicide mission but had no other choice; she had nothing left to lose and everything to gain. With a deep breath, Luna mustered up all her strength and charged forward into the unknown.

2. Game making process

During the making process, we used RADiCAL to generate animation for our main character. RADiCAL is an AI-powered technology that can detect and reconstruct 3D human motion from video input. This enabled us to create realistic and expressive animations for our character with just a smartphone camera and the RADiCAL app. We did not need to rely on expensive motion capture equipment or studios. We can also customize and edit the animations using the RADiCAL Studio app, which gives us more control and flexibility over the final result. RADiCAL helps us to save time and money, while enhancing the quality and immersion of our game.



To generate some assets in game, we firstly used BingAI to obtain desired image about robot, sci-fi; then, we utilized that image as prompt in Mid Journey. Mid Journey is an independent research lab that explores new mediums of thought and AI art. They create stunning and original images based on words and styles that we provide. Based on our input, Mid Journey helped us to generate assets in sci-fi theme that fit with the game atmosphere. The assets are diverse and dynamic, reflecting the futuristic and technological aspects of our game. We are very satisfied with the quality and creativity of Mid Journey's AI art.



ChatGPT was also utilized to solve coding issues and to improve our codes.

You

this following code prints nothing! how?

```
var a = await Task.Run(() => DoFindPath(startPosition, targetPosition));
Debug.Log(a.Count);
return a;
```

ChatGPT

The code snippet you provided seems to be using asynchronous programming with `await` and `Task.Run` to perform a task (`DoFindPath`) on a separate thread and then log the count of the result.

Here are a few things to consider:

- 1. Async Method Signature:**
Make sure that the method containing this code has the `async` modifier in its signature. For example:

```
csharp
Copy code

async Task<List<PathNode>> YourMethod()
{
    var a = await Task.Run(() => DoFindPath(startPosition, targetPosition));
    Debug.Log(a.Count);
    return a;
}
```

- 2. Exception Handling:**
Check for any exceptions that might occur during the execution of `DoFindPath`. If an exception is thrown, it could result in the entire async operation being canceled.

3. Game experience improvements

Game environment

The game world is generated using Perlin Noise, a procedural generation algorithm.

Perlin noise is a popular procedural generation algorithm invented by Ken Perlin. It can be used to generate things like textures and terrain procedurally, without them being manually made by an artist or designer which saved us lots of effort. The algorithm can have 1 or more dimensions, which is basically the number of inputs it gets.

```
// Check zero
while (Generate(x0, y0).type == BiomeType.WATER)
{
    seed1 = Random.Range(0, 10000);
    seed2 = Random.Range(0, 10000);
    seed3 = Random.Range(0, 10000);
    seed4 = Random.Range(0, 10000);
}

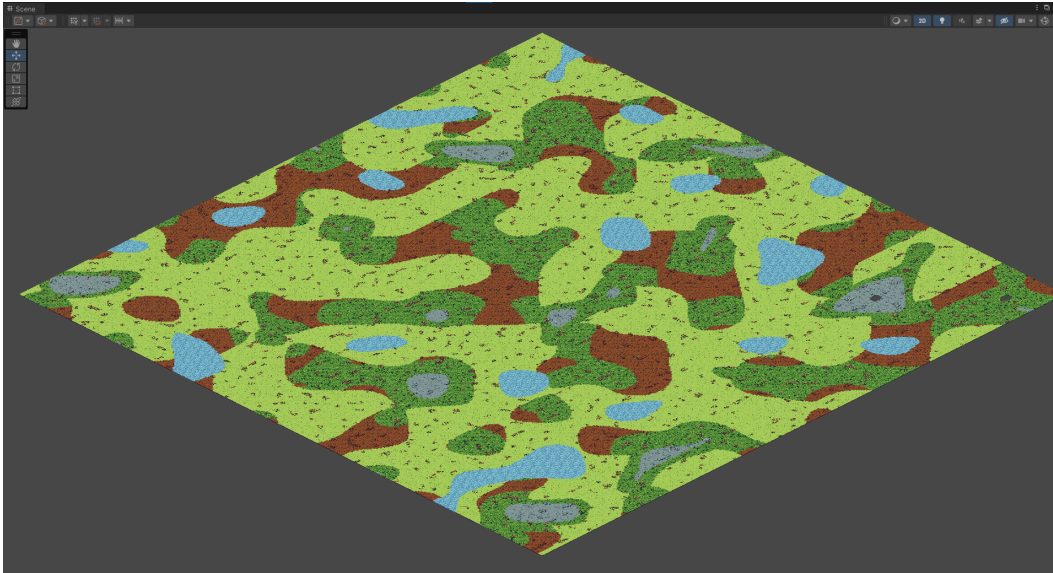
for (int x = -radius; x <= radius; x++)
{
    for (int y = -radius; y <= radius; y++)
    {
        Biome biome = Generate(x, y);
        float decoration = Mathf.PerlinNoise(x * 10 / frequency + seed4, y * 10 / frequency + seed4);
        SetRandomTile(x, y, biome, decoration);
    }
}

Biome Generate(int x, int y)
{
    float height = Mathf.PerlinNoise(x / frequency + seed1, y / frequency + seed1);
    float temperature = Mathf.PerlinNoise(x / frequency + seed2, y / frequency + seed2);
    float precipitation = Mathf.PerlinNoise(x / frequency + seed3, y / frequency + seed3);

    foreach (var biome in config.AllBiomes)
    {
        if (height < biome.height & temperature < biome.temperature && precipitation < biome.precipitation)
        {
            return biome;
        }
    }

    return null;
}
```

In the game, we use 2 dimensions map. Each time player starts the game, Perlin Noise creates the world by mixing 5 biomes: Water, Plains, Mud, Jungle, Mountains based on pre-defined rules. This creates variety and make every new game interesting.



Enemy AI

The 'A* algorithm' is a graph traversal and path search algorithm that finds the shortest path from a specified source to a specified goal. We used it to drive the enemy to find the shortest path to target the player.

The A* algorithm is an example of rule-based AI. The implementation of the A* algorithm was complex and challenging, and it took us a few days of research and coding. However, we think that it was worth it, as it improved the game experience and we learned a lot during the process.

```

while (searching.Count > 0)
{
    // Max search exceeded, we need to cut our loss...
    if (visited.Count > MaxVisitedCount)
    {
        Vector2Int nearest = visited.OrderBy(kvp:KeyValuePair<Vector2Int,int> => kvp.Value).First().Key;
        current = nearest;
        break;
    }

    // get current nearest searching cell
    current = searching.Min.Cell;
    searching.Remove(searching.Min);

    // if inside chase radius then we're done
    if ((current - target).sqrMagnitude < SqrChaseDistance) break;

    // check all adjacent tiles
    foreach (var relative:(int,int) in Neighbours)
    {
        Vector2Int check = new Vector2Int(x:current.x + relative.Item1, y:current.y + relative.Item2);
        if (!MapManager.Instance.CheckPassable(position:new Vector3(check.x, check.y, z:0))) continue;

        visited.TryGetValue(current, out int currentScore);
        int score = currentScore + 1;

        // if not visited or this path has better score then save it
        if (!visited.TryGetValue(check, out int oldScore) || score < oldScore)
        {
            traverse[check] = current;
            visited[check] = score;

            // add or re-add to searching list
            if (searching.All(s:WeightedCell => s.Cell != check))
            {

```

Enemies can intelligently go around obstacles to target main character and attack her.

