**Domain:** Games

**Problem:** Optimizing pathfinding for non-player characters (NPC) to find player character in a grid based RPG style game.

**Goal:** To identify an optimal AI pathfinding technique to implement into an NPC for tracking the player character in a grid-based RPG. The objective is to achieve the shortest distance traveled, highest efficiency, and minimal traversal cost, accounting for various terrains with distinct difficulty levels or weights.

**Introduction**

All good grid-based RPG games employ specific pathfinding algorithms for their NPCs, to make the NPC movements look planned and deliberate giving a sense of realism. Pathfinding, also known as path planning, has many different approaches on how it can be tackled and used. For the domain of games, the author chooses to explore 3 of the most iconic pathfinding techniques and decide on the most optimal one as a solution for his grid based RPG game. The 3 Pathfinding techniques are: Depth-First Search (DFS), Dijkstra's Algorithm and A\* (A-star) Algorithm. Additionally, for a more comprehensive exploration of AI pathfinding, the author will discuss another commonly used AI technique in level design, touching upon

terrain generation and incorporating the use of Perlin noise.

**Depth-First Search (DFS)**

Depth-First Search (DFS)……

**Strengths and Advantages of DFS**

1. **Simplicity:** DFS is straightforward and easy to implement, making it suitable for quick prototyping and testing phases in game development.
2. **Minimal Memory Requirements:** DFS has minimal memory requirements as it only needs to store information about the current path being explored.
3. **Versatility:** DFS is versatile and applicable to various types of grids, making it suitable for different game environments.

**Weaknesses and Disadvantages of DFS (Depth-First Search):**

1. **Lack of Optimality:** DFS does not guarantee the optimality of the path, leading to suboptimal solutions. In a grid-based RPG, this can result in NPCs taking longer routes to reach the player.
2. **Potential for Infinite Loops:** DFS may get stuck in infinite loops, especially in grids with cycles, causing NPCs to repetitively explore the same areas without progress.
3. **Non-Adaptability:** DFS might struggle in dynamic RPG environments where the grid changes frequently, as it does not adapt well to alterations.

**Examples of Application**

In a grid-based RPG where an NPC is tasked with tracking a player, Depth-First Search (DFS) can offer a straightforward approach. The NPC, employing DFS, explores the grid systematically, moving through adjacent cells until it locates the player. As the NPC traverses paths, it discovers various areas of the game world. However, due to DFS's non-optimality, the NPC might not consistently choose the shortest path, resulting in a less efficient route to track the player.

**Input Data and Expected Output**

*Input Data:* For Depth-First Search (DFS) in a grid-based RPG, the algorithm primarily requires information about the connectivity of cells in the grid. This data could be represented as an adjacency matrix or list, indicating which cells are adjacent to each other.

*Expected Output:* The expected output from DFS is the path taken by the NPC in the grid. However, due to DFS's non-optimality, the path may not be the shortest, and it represents the first path discovered by the algorithm during exploration.

**Greedy Best-First Search**

Greedy Best-First Search is a heuristic-based algorithm designed to efficiently navigate through complex environments. Unlike traditional search algorithms, Greedy Best-First Search prioritizes exploration based on heuristic information, emphasizing speed over optimality. This algorithm is particularly well-suited for scenarios where quickly finding an approximate solution is more crucial than ensuring the absolute shortest path.

**Strengths and Advantages of** **Greedy Best-First Search:**

1. **Speed:** Greedy Best-First Search excels in speed, quickly navigating through the grid based on heuristic information. This makes it suitable for real-time scenarios in games.
2. **Ease of Implementation:** Similar to DFS, Greedy Best-First Search is relatively simple to implement, providing a quick and accessible solution for game developers.
3. **Adaptability:** The algorithm is adaptable to different grid-based RPG environments, providing flexibility in NPC tracking scenarios.

**Weaknesses and Disadvantages** of **Greedy Best-First Search:**

1. **Lack of Optimality:** Similar to DFS, Greedy Best-First Search prioritizes speed over optimality, potentially leading NPCs to choose suboptimal paths in a grid-based RPG.
2. **Heuristic Dependency:** The algorithm heavily relies on heuristic information, and if the heuristic is poorly designed, it can result in biased decision-making and less effective NPC tracking.
3. **Limited Exploration:** Greedy Best-First Search may focus too narrowly on promising paths based on initial estimates, limiting exploration of other potentially optimal routes.

**Examples of Application**

For the task of NPC tracking in a grid-based RPG, Greedy Best-First Search proves valuable for its speed-oriented approach. The NPC, utilizing this algorithm, quickly navigates through the grid based on heuristic estimates, prioritizing paths that seem most promising for locating the player. The algorithm's emphasis on rapid decision-making enables the NPC to respond swiftly to the player's movements. However, the selected path may not always be the shortest, highlighting the trade-off between speed and optimality in the context of player tracking.

**Input Data and Expected Output**

*Input Data:* Greedy Best-First Search relies heavily on heuristic information. In a grid-based RPG, this could be data representing estimated distances from each cell to the player's location. The heuristic guides the algorithm in selecting the most promising paths.

*Expected Output:* The expected output is the path taken by the NPC. Since Greedy Best-First Search prioritizes speed over optimality, the chosen path might not be the shortest but is determined by the heuristic's initial estimates.

**Dijkstra's Algorithm**

The Dijkstra's Algorithm is another widely used pathfinding algorithm with distinct strengths and weaknesses in the context of game development. Let's delve into its characteristics, advantages, disadvantages, and applications.

**Strengths and Advantages of Dijkstra's Algorithm**

1. **Optimality:** Dijkstra's Algorithm guarantees the optimality of the solution, ensuring that the NPC chooses the shortest path to reach the player in a grid-based RPG.
2. **Versatility:** Dijkstra's Algorithm is versatile and applicable to various grid scenarios, providing accurate and optimal solutions for NPC tracking.
3. **Consideration of Costs:** By considering the costs associated with traversing between cells, Dijkstra's Algorithm provides a realistic approach to pathfinding in the RPG environment.

**Weaknesses and Disadvantages of Dijkstra's Algorithm**

1. **Computational Complexity:** Dijkstra's Algorithm can be computationally expensive, especially in large grids, making it less suitable for real-time applications in grid-based RPGs.
2. **Memory Requirements:** The algorithm requires storing information about all explored paths, leading to higher memory requirements compared to simpler algorithms like DFS and Greedy Best-First Search.
3. **Non-Adaptability to Heuristics:** Dijkstra's Algorithm does not incorporate heuristic information, potentially resulting in suboptimal paths compared to algorithms like A\*.

**Examples of Application:**

In the grid-based RPG environment, Dijkstra's Algorithm becomes a precise tool for NPC tracking. The NPC, employing Dijkstra's Algorithm, meticulously calculates the costs associated with traversing between grid cells. This comprehensive evaluation ensures that the NPC consistently chooses the optimal path to reach the player. While computationally more intensive, Dijkstra's Algorithm excels in scenarios where the NPC's precision in tracking the player's movements is critical for an immersive gameplay experience.

**Input Data and Expected Output:**

*Input Data:* Dijkstra's Algorithm requires information about the weights or costs associated with traversing between cells in the grid. In a grid-based RPG, this data could represent the actual distances or obstacles that affect movement.

*Expected Output:* The expected output from Dijkstra's Algorithm is the optimal path taken by the NPC to reach the player. The algorithm guarantees optimality by considering all possible paths and selecting the one with the minimum cumulative cost.

**A\* (A-star) Algorithm**

A\* (A-star) algorithm is a widely used and highly efficient pathfinding algorithm that combines the benefits of both Depth-First Search (DFS) and Dijkstra's Algorithm while mitigating their drawbacks. Let's explore the strengths, weaknesses, advantages, and disadvantages of A\* in the context of game development for the authors RPG game.

**Strengths and Advantages of A\* Algorithm**

1. **Optimality and Efficiency: A\* combines the optimality of Dijkstra's Algorithm with the efficiency of heuristic-guided search. It ensures both speed and precision in finding the optimal path for NPC tracking.**
2. **Adaptability: A\* is adaptable to different heuristic functions and grid scenarios, allowing game developers to tailor the algorithm to specific RPG environments.**
3. **Precision in NPC Tracking: The algorithm's ability to consider both actual costs and heuristic estimates results in precise and optimal NPC tracking in a grid-based RPG setting.**

**Weaknesses and Disadvantages of A\* Algorithm**

1. **Computational Complexity: While more efficient than Dijkstra's Algorithm, A\* can still have higher computational complexity compared to simpler algorithms like DFS and Greedy Best-First Search.**
2. **Heuristic Quality Dependency: A\* performance heavily depends on the quality of the heuristic function. Poorly designed heuristics may lead to suboptimal paths, affecting NPC tracking in grid-based RPGs.**
3. **Implementation Complexity: A\* requires implementing both the cost calculation and heuristic functions, making it more complex to set up than simpler algorithms.**

**Examples of Application**

In the context of a grid-based RPG where an NPC is assigned the task of tracking a player, the A\* algorithm offers a robust solution. The NPC, employing A\*, dynamically adjusts its path by considering both the actual costs of traversing grid cells and heuristic estimates of remaining distances to the player's location. This dynamic adaptation ensures that the NPC responds promptly to the player's movements, while the algorithm consistently guides the NPC on paths that balance efficiency and optimality in the grid. The A\* algorithm proves effective in enhancing the overall tracking experience, providing a pragmatic approach to NPC tracking in the RPG environment.

**Input Data and Expected Output:**

*Input Data:* Similar to Dijkstra's Algorithm, A\* needs data about the costs associated with traversing between cells. Additionally, A\* requires heuristic information, estimating the remaining cost from each cell to the player's location.

*Expected Output:* The expected output is the optimal path taken by the NPC. A\* combines the efficiency of heuristic-guided search with the optimality guarantee of Dijkstra's Algorithm, ensuring both speed and precision in NPC tracking in the grid-based RPG.

**Manhattan Distance as a Heuristic:** One commonly used heuristic in A\* is the Manhattan distance, especially in grid-based games. This heuristic calculates the distance between two points on a grid by summing the absolute differences of their coordinates. Using the Manhattan distance as a heuristic in A\* is advantageous in scenarios where movement is constrained to grid positions, providing a good balance between accuracy and computational efficiency.

*Proving A Superiority:*

After comparing and analyzing the 3 pathfinding techniques the author feels A\* can be considered superior to DFS and Dijkstra's Algorithm in scenarios where optimality and computational efficiency are crucial. By incorporating a heuristic like the Manhattan distance, A\* significantly reduces the search space, focusing on the most promising paths. In comparison to DFS, A\* guarantees optimality, and compared to Dijkstra's Algorithm, it often achieves the same optimality with lower computational cost.

In conclusion, A\* stands out as a powerful and widely applicable pathfinding algorithm for the authors RPG game. Its ability to find optimal paths efficiently, guided by a well-designed heuristic like the Manhattan distance, makes it the perfect choice for the player tracking NPC in a grid-based RPG.

**Terrain Generation Using Perlin Noise**

Terrain generation using Perlin noise is a widely employed technique in game development to create realistic and visually appealing landscapes. Let's explore the strengths, weaknesses, advantages, and disadvantages of using Perlin noise for terrain generation.

**Strengths and Weaknesses of Perlin noise for Terrain Generation**

Perlin noise is renowned for its ability to generate smooth, continuous, and natural-looking patterns. This quality makes it particularly well-suited for terrain generation, as it mimics the organic variations found in natural landscapes. The algorithm is relatively simple to implement, providing a good balance between computational efficiency and visual quality.

While Perlin noise excels in creating organic patterns, its weakness lies in the potential for uniformity (this can be observed during Part C Implementation section). Without additional modifications or layers, Perlin noise might generate terrains that appear too smooth or lack the desired level of detail. To address this, developers often combine multiple layers of Perlin noise or apply additional algorithms for more intricate terrains.

**Advantages and Disadvantages of Perlin noise for Terrain Generation**

One of the primary advantages of Perlin noise is its versatility. It can be adjusted to generate a variety of terrains, from rolling hills to rugged mountains. The simplicity of the algorithm makes it accessible for developers at various skill levels. Additionally, Perlin noise provides a level of randomness that contributes to the uniqueness of each generated terrain, enhancing the overall gaming experience.

The main disadvantage is the potential for monotony without proper adjustments. While Perlin noise offers a good starting point, it might not capture all the nuances of complex natural landscapes. Developers often need to complement Perlin noise with additional algorithms or tweak parameters to achieve the desired level of realism and variety in terrain generation.

**Examples of Application:** Perlin noise is commonly used in open-world games to generate realistic landscapes. For instance, in a role-playing game (RPG), Perlin noise could be employed to create diverse terrains, including plains, forests, and mountains. The algorithm's adaptability allows developers to tailor generated terrains to suit the thematic requirements of their games.

**Input Data and Expected Output:** Perlin noise typically requires input parameters such as scale, octaves, and persistence to control the characteristics of the generated terrain. The expected output is a heightmap representing the elevations of the terrain, which can then be translated into 3D landscapes or 2D maps for game environments.

In summary, Perlin noise is a valuable tool for terrain generation in game development due to its simplicity, versatility, and ability to produce visually appealing landscapes. While it may have limitations in generating highly detailed terrains, developers can overcome these challenges by combining Perlin noise with other techniques or tweaking parameters to achieve the desired level of complexity and realism.

The Author chose to include terrain generation using Perlin noise in this section to further enhance the profoundness of pathfinding techniques. The next section will showcase different implementations of A\* algorithm for various situations and also how it stays optimal even in a procedurally generated terrain.