Multi-Agents Pathfinding

Alexis Tan Bing Rui, Brandon Ho Jun Jie, Clifford Tan Eng Kiat, George Koh Jia Hao

1 Introduction

Exploration is a crucial element in video games that greatly enhances player engagement and augments the user experience to make it more realistic. Modern games often feature richly detailed worlds, designed to provide players with a sense of vast possibilities and rewarding experiences through exploration. Players are encouraged to explore these environments, with the promise of unique and interesting encounters that are neither repetitive nor trivial. This emphasis on exploration underscores its importance in game design.

Non-player characters (NPCs) play an essential role in the exploratory dynamics of games. NPCs are often tasked with navigating the game map, whether to ensure dynamic interactions with players, such as enemies naturally encountering them, or to fulfill roles like guards in stealth games. Designing NPCs to explore exhaustively helps simulate thorough searches, ensuring no area is overlooked. Additionally, NPCs can aid players by revealing hidden areas or by automating parts of the exploration process, making it less tedious for players to manually scour large game levels.

2 Problem Statement

In many games, particularly those in genres like roguelike and turn-based strategy, NPCs play a crucial role in aiding human players with exploration. However, the algorithms driving NPC exploration are often rudimentary and not well-documented. Observations from games such as Civilization V [1] and Dungeon Crawl Stone Soup [2] suggest that these games use simple greedy approaches for NPC exploration. Unfortunately, players frequently find these methods unsatisfactory, as they fail to provide the level of sophistication and efficiency expected in modern games.

A deeper analysis reveals that while these games successfully illustrate the necessity for effective NPC exploration techniques, they also highlight significant gaps in current approaches. NPCs often exhibit behaviors that, although appearing dynamic, are repetitive and limited in scope. For example, in The Elder Scrolls V: Skyrim, NPCs follow highly predictable routines that, while creating an illusion of a living world, ultimately serve the player's needs in a very deterministic manner. This results in a lack of genuine autonomy and variability in NPC behavior.

The challenge, therefore, lies in developing exploration strategies that enable NPCs to navigate complex, obstacle-rich, and procedurally generated environments more effectively. These strategies should ensure comprehensive coverage without excessive repetition, thereby enhancing the realism and engagement of the game world. Furthermore, such strategies need to account for the dynamic interactions between NPCs and players, ensuring that NPCs can perform their roles, such as guards in stealth games, or dynamic enemies that naturally encounter players, in a more nuanced and sophisticated manner.

To address these challenges, our research focuses on simulating NPC exploration using advanced algorithms and techniques. By implementing and testing various movement strategies within procedurally generated environments, we aim to identify solutions that improve the efficiency and effectiveness of NPC exploration. This, in turn, will contribute to creating richer and more immersive game experiences, where NPCs behave in ways that are both believable and beneficial to the overall gameplay.

This research seeks to bridge the gap in current NPC exploration methods by developing and evaluating strategies that enhance their ability to navigate and explore game environments thoroughly and efficiently. The goal is to elevate the standard of NPC behavior in games, making them more autonomous, engaging, and capable of supporting diverse and dynamic gameplay scenarios.

3 Proposed Solution

3.1 Engine Architecture

The simulation we developed uses the framework programmed by the team with the help of the popular SFML and ImGui graphics libraries. We have a Factory class that can spawn entities inherited from the Entity base class. This makes it more convenient to retrieve and destroy entities, as users can use a single API to do so regardless of the entity type. Moreover, the Factory class is responsible for drawing the entities onto the screen depending on their `shape` variable. Next, we have a Grid class that generates and maintains the map, flow field, and potential field. This class also draws the tiles onto the screen, including setting a tile's color based on its visibility for visualization purposes. As briefly touched on earlier, we use ImGui to display the interface (also known as the editor) the user can interact with to modify certain variables. For example, the Inspector in the editor allows users to view and edit almost all of the variables of each entity. The Control Panel allows users to change or toggle global variables that affect all entities. The Map Maker allows users to draw on the map and set the parameters for the procedural map generator. To save and load maps that we have drawn or generated with the engine, the Loader class serializes map data into text files and vice versa.

3.2 Fog of War

As we wanted our AI agents to explore the map instead of calculating a simple path to the goal, we implemented fog of war. Agents have a conic field of view of which its angle and radius can be adjusted. This fog of war is shared, meaning that agents do not need to inform other agents about newly discovered cells.

3.3 Vector Fields

To navigate the partially unknown map and locate the objective, our AI agents utilize vector fields. Our proposed solution integrates these fields with the fog of war, enabling entities to make informed decisions based on visibility and environmental context.

3.4 Methodology

Our simulation and timing tests are conducted in a 2D grid map environment. To enhance the robustness and reliability of our research, we utilize procedurally generated maps with varying parameters. This approach ensures a comprehensive comparison and analysis of the performance of different exploration techniques under diverse conditions.

3.5 Map Generation

While not technically part of our AI, we wanted a way to test its effectiveness without having to painstakingly handcraft tens of maps. Hence, to prove that our AI is robust and adaptable enough to navigate any kind of environment, we decided to create a procedural map generator. There are two main ways of generating terrain, noise functions and graph traversals, of which we chose the latter. While noise functions may generate more natural looking and smooth terrain, it is not only harder to implement but may also give worse results for our use case. How do we determine if one method is better than another?

For one, breadth first search (BFS) with each node having one parent guarantees that the map generated will always be a perfect maze. A perfect maze is a maze where there is exactly one way to get from one point to another. If we change the maximum number of parents each node in the BFS can have, the invariant of "there is at least one path from one cell to another" still holds true. This means that none of the map will be unusable or unreachable, though the agents would still be challenged to traverse through many obstacles in the form of walls to uncover the entire map.

Secondly, graph traversals tend to create more structured maps with neat and clear walls that more closely mimic the hills and rivers found in RTS games like Command & Conquer. These games often have an AI that the player can play against that will, like the player, need to explore the map to uncover important landmarks like resource grounds and enemy bases. By testing our agents with pseudo-random yet distinct maps, we can show that our AI can work within the constraints of a fictional RTS game and expand their fog of war efficiently using a variable number of units. Game developers can thus use our methods with a modified potential field to program an intelligent AI that players will want to play against.

Users can call the `Grid::generateMap()` function to generate a random map with modifiable global parameters. This function uses random walk with BFS to generate a perfect maze, then removes walls between cells that point to one another. At the start of the algorithm, all cells are set to walls and have their parents initialized to `nullptr`. Nodes are chosen a minimum of 2 cells apart so that walls can be removed to make a path between them. Wall removal is pseudo-randomized depending on the `noise` variable, with a 0% chance for walls to spawn beside non-wall cells at 0 (minimum) noise and 50% chance at 10 (maximum) noise. Lastly, the function will once again use BFS to find clusters of walls called islands and delete them if their size is less than the `minIslandSize` variable. There are two variables that control how "open" the map is, i.e. how many connections are there from one node to another. If these variables, i.e. the minimum or maximum connections, are not set to 1, the maze generated will not be a perfect maze, as some cells may randomly point to more than 1 neighbor. Users can also modify the wall and tunnel widths to generate mazes with wider or narrower pathways. Below are some screenshots of the generated maps and the corresponding input parameters (shown with a yellow arrow).

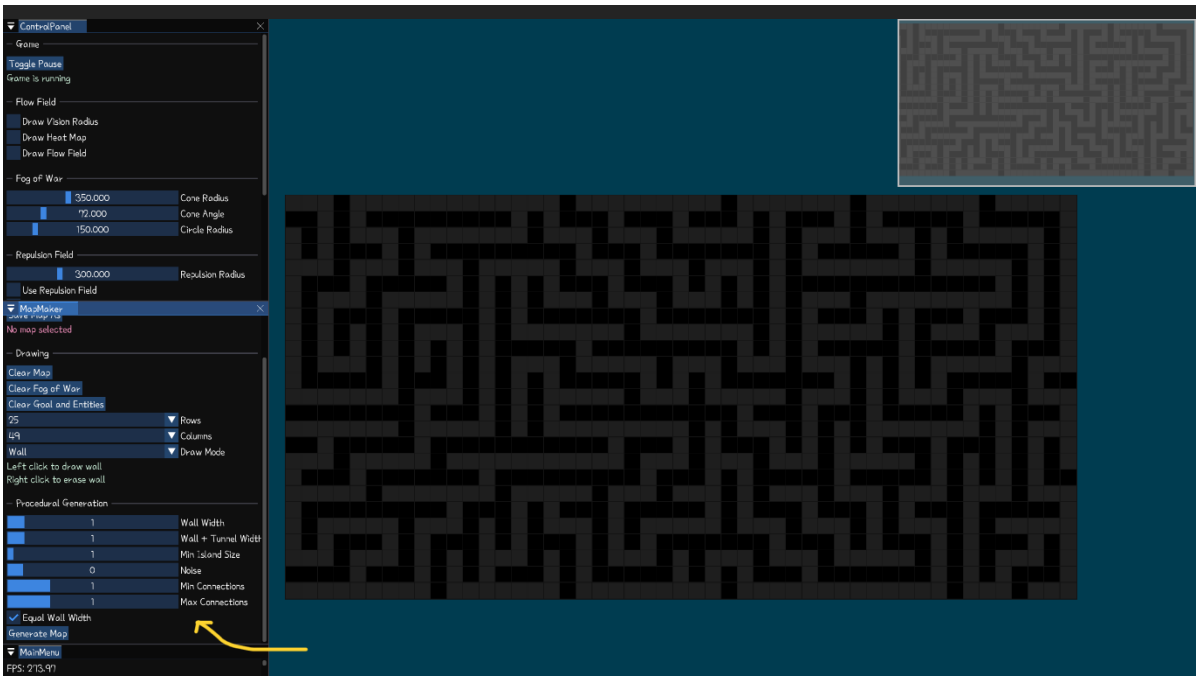

Figure 3.5.1, Generated Map with Default Parameters

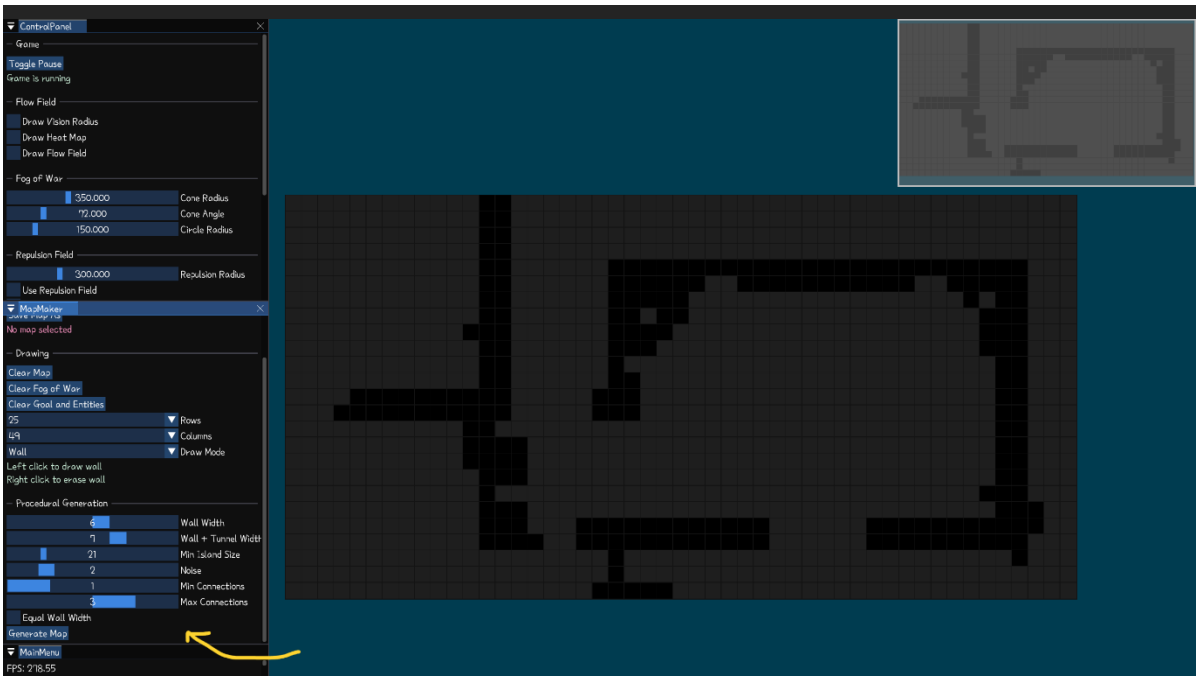

Figure 3.5.2, Generated Map with Custom Parameters

3.6 Heat Map

Heat maps play a critical role in 2D grid map exploration and pathfinding by visualizing the shortest distances from each cell to a target position, such as an exit. The heat map initialization starts by setting all cells to a maximum distance value, indicating their initial state as far from the target. The algorithm uses a BFS approach, starting from the target cell and calculating distances to each neighboring cell iteratively. This process continues until all cells are processed. Each cell's distance is updated based on its proximity to the target, with neighboring cells receiving incrementally higher values. The heat map data is then used to create a flow field, which is a vector field indicating the direction an entity should move to reach the target efficiently. Each cell in the flow field points to its neighbor with the shortest distance to the target, guiding entities smoothly through the grid. This process ensures that entities follow the shortest and most efficient paths, avoid obstacles, and adapt in real-time to changes in the grid, such as newly discovered obstacles or moving targets. The visual representation provided by heat maps also aids developers in debugging and analyzing the pathfinding logic, offering a clear picture of distance calculations and expected entity movement.

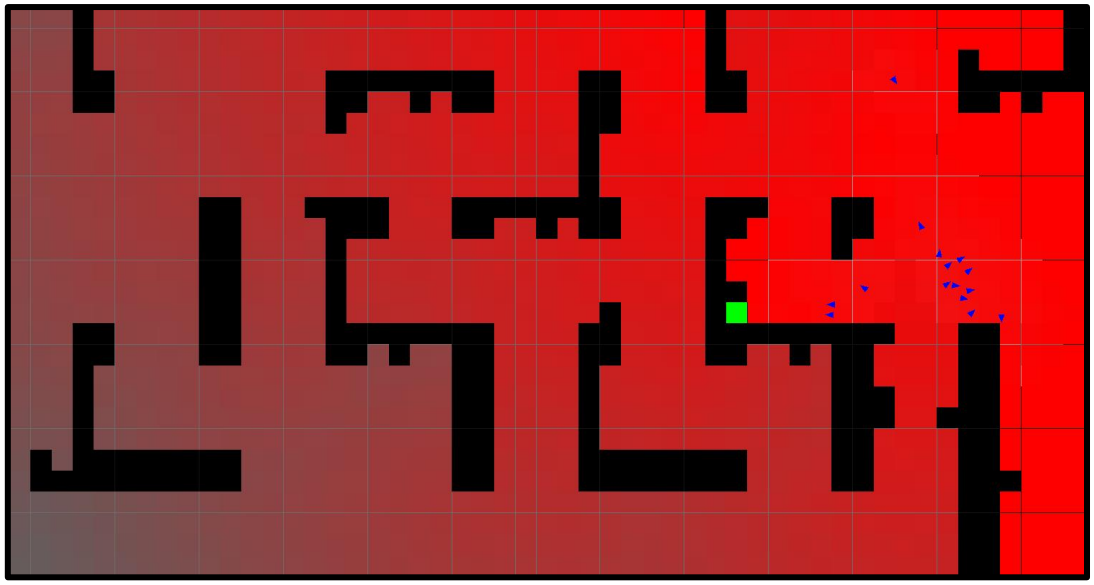

Figure 3.6.2, Heat Map

3.7 Potential Field

In the context of 2D grid map exploration and pathfinding, potential fields enhance the basic heat map by considering additional factors such as obstacles and goal locations, resulting in smoother and more natural paths for entities. The grid is initialized with cells assigned default potential values, and each cell has associated potential and direction values. The `Grid::updatePotentialField()` function iterates through the grid, calculating potential values for each cell based on attractive forces that pull entities towards the target and repulsive forces that push entities away from obstacles. This ensures that entities are guided smoothly towards their goals while avoiding collisions [3].

```
int md = std::abs(flowFieldCell.position.row -
blockCenter.row) + std::abs(flowFieldCell.position.col -
blockCenter.col);
if (md <= MAX_MD)
{
// Calculate potential
     float newPotential = MAX_POTENTIAL - ((float(md) *
MAX_POTENTIAL) / MAX_MD);

     if (newPotential > 0)
     {
          flowFieldCell.potential += newPotential;
          maxPotential = std::max(maxPotential,
flowFieldCell.potential);
     }
}
```

$$p_{unknown}(md) = \begin{cases} (0.25 - \frac{md}{8000}) & \text{if } md <= 2000 \\ 0 & \text{if } md > 2000 \end{cases}$$

Figure 3.7.1, Equation for Potential Fields

After calculating these potential values, the algorithm generates a flow field, providing a direction vector for each cell to guide entities along paths of least potential resistance. This combination of attractive and repulsive forces ensures entities follow efficient paths, adapt in real-time to changes, and avoid obstacles. The implementation involves initializing the grid, updating the potential field based on the proximity to targets and obstacles, and generating a flow field to guide entity movement. This approach improves pathfinding efficiency and provides a more dynamic and engaging simulation experience.

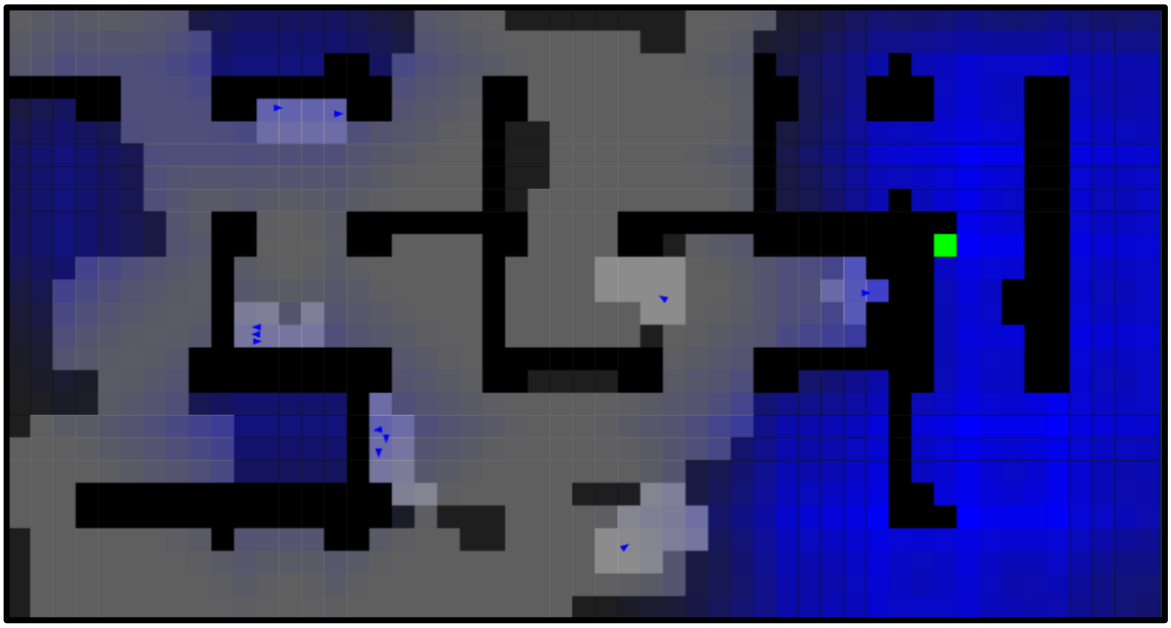

Figure 3.7.2, Potential Field

3.8 Flow Field

In our 2D grid map exploration and pathfinding project, the flow field is a vital component that guides entities efficiently towards their target. Each cell in the grid is initialized with default direction and distance values. The `Grid::generateFlowField()` function then calculates the direction vectors for each cell, determining the optimal path to the target by considering the shortest distances and avoiding obstacles. This involves checking neighboring cells and updating each cell's direction vector accordingly. The flow field ensures that entities follow smooth and efficient paths by guiding them in the direction of the least resistance. Additionally, the flow field can be visualized using arrows to indicate movement direction, aiding in debugging and understanding pathfinding behavior. This method enhances pathfinding by providing a clear and adaptive path, making the simulation more effective and responsive.

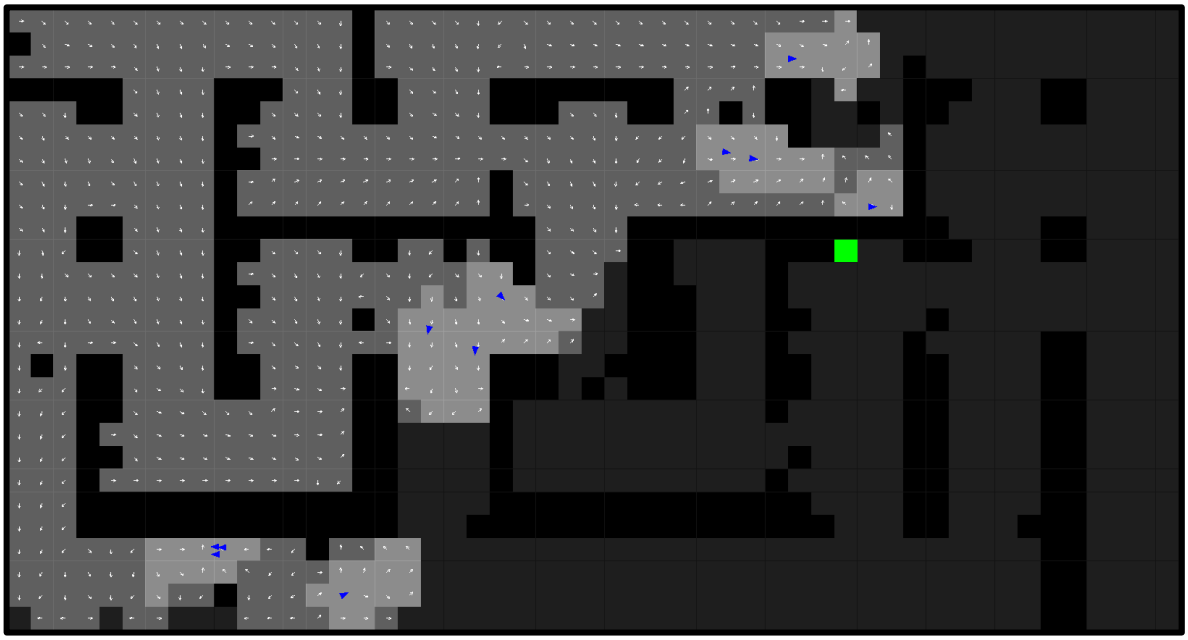

Figure 3.8.1, Flow Field

3.9 Repulsion Field

The repulsion field is a key component in 2D grid map exploration and pathfinding, enhancing navigation by incorporating avoidance behaviors around obstacles or hazardous areas. The repulsion field is generated using the `Grid::updateRepulsionMap()` function, which initializes parameters such as the central grid position, radius, and strength of the repulsion force. The algorithm calculates distances from the central point to surrounding cells within the defined radius, assigning repulsion values that decrease linearly with distance. This ensures that cells closer to the central point have higher repulsion values, guiding entities smoothly away from obstacles. The repulsion values are then integrated into the flow field, influencing the overall pathfinding and movement behavior of entities. Visualization of the repulsion field, using a color gradient, helps in understanding the distribution of repulsion forces across the grid. This method ensures efficient obstacle avoidance, smooth navigation, and dynamic adaptation to changing environments, complementing other components like the heat map and potential field to provide a robust pathfinding solution.

Figure 3.9.1, Repulsion Field

3.10 Exploration

Exploration of the map is achieved through heat maps, potential fields, and flow fields. The `updateHeatMap()`, `updatePotentialField()`, and `generateFlowField()` functions work together to guide entities through the grid environment efficiently. Initially, the heat map visualizes the shortest distances from each cell to a target, providing a foundational guide for entity movement. The potential field refines this by incorporating attractive and repulsive forces, ensuring smoother navigation around obstacles. The flow field then translates these distance and potential values into direction vectors, guiding entities along the most efficient paths. As entities move, their positions are continually updated based on the flow field, adapting to changes in the environment in real-time. This process not only optimizes pathfinding but also enhances the overall exploration strategy, ensuring comprehensive coverage and efficient navigation throughout the grid map.

4 Testing

Using our simulation, we performed tests by recording the time taken for entities to perform tasks such as fully exploring the map and finding the hidden goal.

Fixed parameters for all tests:

Entities' Field of View (FOV):

- FOV Cone Radius: 350 pixels
- FOV Cone Angle: 72 pixels
- Inner FOV Circle Radius: 150 pixels

This first test aims to determine the time taken for the entities to fully explore the map.

Test parameters:

Number of entities: 5

Map: Blank map (25 x 50 cells)

Results:

Table 1 Average Time Taken for Entities to Fully Explore the Entire Map

Vector Field methods	Average Time Taken to Explore Entire Map (seconds)
Default (Heat Map)	27.5
Potential Field with Default Settings	41.2
Repulsion Field	
Repulsion Radius: 300	30.0
Repulsion Radius: 600	33.6
Repulsion Radius: 900	37.1
Potential Field and Repulsion	
Repulsion Radius: 300	55.4
Repulsion Radius: 600	41.7
Repulsion Radius: 900	43.1

The tests revealed that the default method (Heat Map) took the shortest time to fully explore the map. In contrast, the potential field method with repulsion field took the longest time.

This second test aims to determine the time taken for one entity to find the hidden goal.

Test parameters:

Number of entities: 16

Map: Randomly generated map (50 x 50 cells)

Results:

Table 2 Average Time Taken For Entity To Find Hidden Goal

Vector Field methods	Average timing to Find Hidden Goal (seconds)
Default (Heat Map)	28.05
Potential Field with default settings	35.89
Repulsion Field	
Repulsion Radius: 300	30.42
Repulsion Radius: 600	34.66
Repulsion Radius: 900	39.65
Potential Field and Repulsion	
Repulsion Radius: 300	38.06
Repulsion Radius: 600	41.7
Repulsion Radius: 900	43.1

The default method (Heat Map) still resulted in the shortest time to find the hidden goal. The potential field and repulsion field methods, individually and together, took a longer time.

Despite the results obtained from the tests, we cannot conclusively determine which method is superior due to the numerous variables that can affect the outcome. For example, the spawn location of the entities significantly impacts the results for the repulsion field methods, as entities spawning in tighter areas tend to constantly repel each other. The goal location also heavily affects the potential field method as the algorithm prioritizes larger unexplored areas.

If the time taken is not the primary consideration, these methods can still be valuable as they offer unique exploration strategies and may be suited for different types of game environments. For instance, the repulsion field method might be ideal for scenarios requiring entities to maintain some distance from one another, while the potential field method could be useful in games where entities need to prioritize exploring new areas. Thus, each method has its own merits and can be utilized based on the specific requirements of the game.

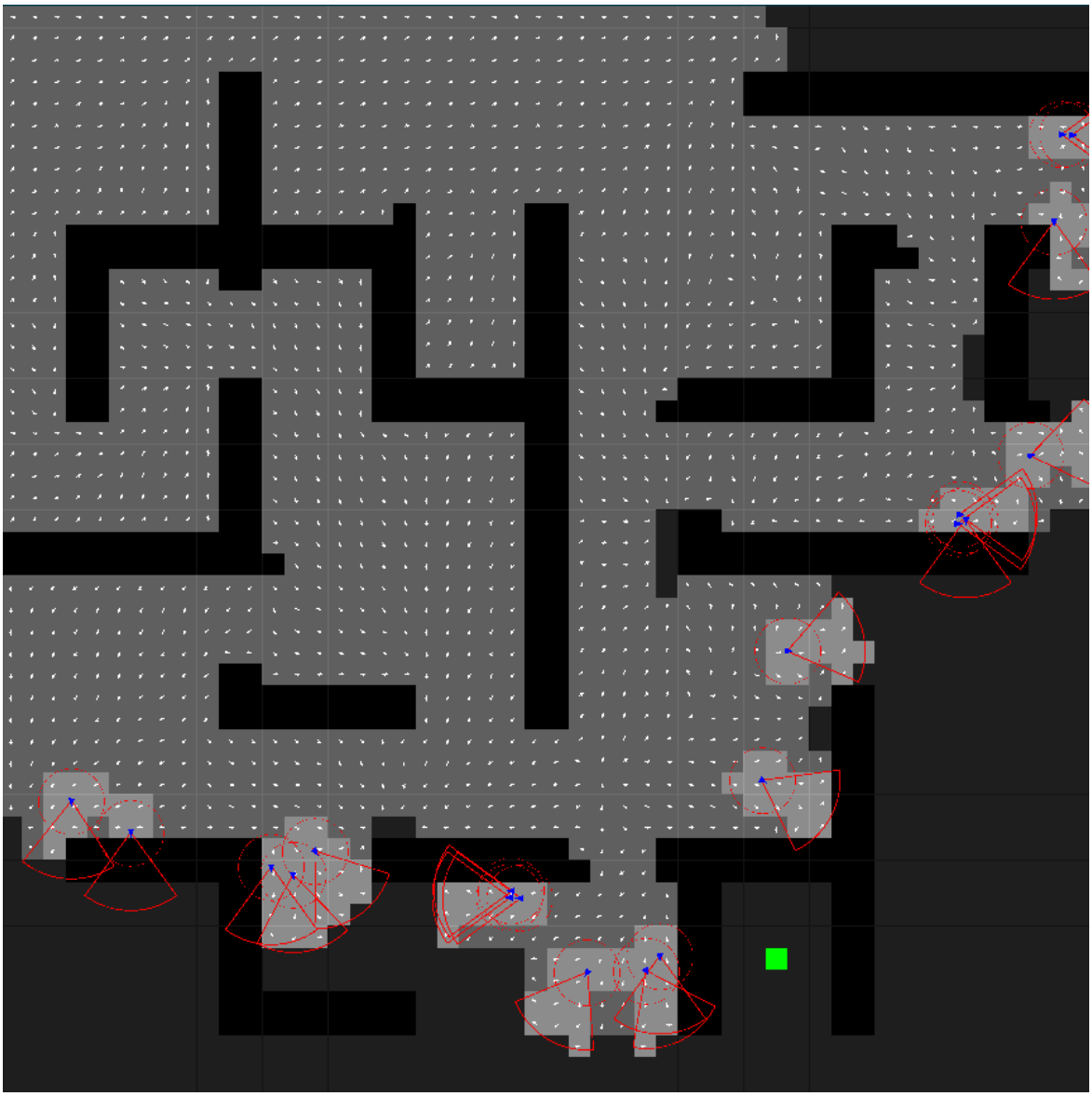

Figure 4.5.1, Map used for entity to find hidden goal

5 Conclusion

By layering a default heat map, repulsion map, and potential map into one collective flow field, we are able to simulate how we think a human would explore a randomly generated map. There is no data we can measure to determine if a search pattern is humanlike, but there are some factors that we can consider to compare how our agents explore to a human. The first is randomness: it seems that even under identical conditions (i.e., the same start position and wall placement), the path taken by the agents may differ, simulating the unpredictability found in real-world exploration. This randomness also means that agents might occasionally go down incorrect paths, adding to the authenticity of the exploration process. While our algorithm is technically deterministic, as there is no randomness baked into the pathfinding, it appears probabilistic as even the most minute of changes in the starting conditions or map layout will greatly affect the final search pattern. This is because the combined vector field depends on many parameters that are constantly being updated as more cells are revealed, such as the position of the other agents and the distance to the closest unexplored cell.

On the other hand, our AI agents exhibit semi-intelligent behavior that enhances their exploratory efficiency to some extent. They don’t need to make perfect decisions anyway as humans themselves are not perfect. Unlike traditional pathfinding algorithms that may send all units down the same route, our approach ensures that units will favor distributing themselves across the map to cover more ground. They share a collective fog of war that never fades, representing an indefinite shared memory that mimics a player in an RTS game. Since there is only one player controlling all the units in such a game, they essentially share one hive mind. The agents also prioritize larger unexplored areas over smaller ones, which is what a human might do to narrow down the places a target landmark, such as an enemy superweapon, might be at. As mentioned earlier, as more cells are explored, the target paths of the agents dynamically change, mimicking how a player would move their units in response to seeing enemy units revealed from the fog of war. This combination of randomness and intelligent decision-making provides a robust framework for realistic exploration in a myriad of environments.

6 Future Work

The simulation study of the potential field algorithm revealed several insights into its performance and areas for improvement. While this algorithm offers a reasonable solution time and can produce more natural and interesting pathfinding behaviors compared to the predictable and greedy Dijkstra's map, it also presents some challenges. Different scenarios yield varied results, indicating that the algorithm can be difficult to implement effectively in real-world situations. However, the ability of potential fields to dynamically adjust to changing goals by simply modifying the numerical values in the grid can be a significant advantage for rapidly shifting destination goals.

6.1. Refinement of Potential Field Algorithms

The potential field algorithm demonstrated non-optimal results in certain scenarios, particularly in narrow maps or environments with many walls. Future work should focus on refining these algorithms to improve their performance in such complex environments. This could involve developing more sophisticated methods for calculating potential values or integrating hybrid approaches that combine potential fields with other pathfinding techniques.

6.2 Adaptive Parameter Tuning

The results indicated that the efficiency of exploration can be significantly impacted by the parameters used in the repulsion fields. In narrow maps, repulsion fields sometimes slowed down the search or prevented agents from approaching the only explored cells in a region. Future research should explore adaptive parameter tuning methods that can dynamically adjust parameters based on the current map characteristics and agent distribution to optimize performance.

6.3 Application to Dynamic and Real-World Scenarios

Further testing and development are needed to implement the potential field algorithm effectively in real-world scenarios. This includes adapting the algorithm to dynamic environments where obstacles and goals can change over time. Techniques for real-time updating and recalibration of the potential fields will be crucial for practical applications in both gaming and robotics.

6.4 Exploration of Alternative Potential Sources

Other types of objects and environmental features can also be used to generate potential maps. Future research could explore the possibility of using various objects, such as dynamic obstacles, resource nodes, or environmental hazards, to influence the potential fields. This could lead to more nuanced and realistic behaviors in NPCs as they navigate complex environments.

6.5 Advanced Heuristics for Balancing Coverage and Distance

The need for effective exploration algorithms is critical in modern games to provide NPCs with realistic behaviors and reduce player frustration. While our proposed method offers strong guarantees of exhaustive exploration, future work should focus on developing advanced heuristics that better balance coverage and distance. These heuristics should be tested across multiple, realistic game maps to evaluate their effectiveness in diverse scenarios.

6.6 Integration with Machine Learning Techniques

Integrating machine learning techniques with the current exploration strategies presents a promising direction for future research. Reinforcement learning, for example, could be used to enable NPCs to learn optimal exploration strategies based on past experiences. This would allow NPCs to continuously improve their performance, adapting to new environments and scenarios more effectively.

6.7 Enhanced Cooperative Strategies

While the current study incorporates basic cooperative behavior through shared fog of war and repulsion fields, more sophisticated cooperative strategies should be investigated. This could include the implementation of task allocation algorithms that dynamically assign exploration goals to NPCs based on their current positions and statuses, enhancing overall exploration efficiency and reducing redundancy.

7 References

In-text Citation

[1] Various contributors. Civilization modding wiki. http://modiki.civfanatics.com/. A wiki for the Civilization source code.

[2] Various Contributors. Dungeon Crawl Stone Soup wiki. https://crawl.develz.org/

[3] The research by Hagelbäck and Johansson (2009) addresses strategies for dealing with the fog of war in real-time strategy game environments［Hagelbäck 09］

References

[Chowdhury 16] Chowdhury, M., & Verbrugge, C. 2016. Exhaustive Exploration Strategies for NPCs. In *Proceedings of the 1st International Joint Conference of DiGRA and FDG*, 7th Workshop on Procedural Content Generation, 1-15. Montréal: McGill University.

[Fathy 14] Fathy, H., Abd Elkader, H. A., & Fathy, H. A. A. 2014. Flocking Behaviour of Group Movement in Real Strategy Games. In *Proceedings of the 2014 Conference on Flocking Behavior*, 79-86. Menoufia: Menoufia University.

[Hagelbäck 09] Hagelbäck, J., & Johansson, S. J. 2009. Dealing with Fog of War in a Real Time Strategy Game Environment. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games (CIG)*, 55-62. Milano: IEEE.

[Julia 08] Juliá, M., Gil, A., Payá, L., & Reinoso, O. 2008. Local Minima Detection in Potential Field Based Cooperative Multi-robot Exploration. In Proceedings of the International Symposium on Experimental Robotics, 1-15. Elche: Miguel Hernández University.

[Johnson 09] Johnson, C. 2009. The Incredible Power of Dijkstra Maps. *Rogue Basin*. https://www.roguebasin.com/index.php?title=The_Incredible_Power_of_Dijkstra_Maps (accessed July 16, 2024).

[Maffei 20] Maffei, R., Souza, M. P., Mantelli, M., Pittol, D., Kolberg, M., & Jorge, V. A. M. 2020. Exploration of 3D Terrains Using Potential Fields with Elevation-Based Local Distortions. In *Proceedings of the 2020 IEEE International Conference on Robotics and Automation (ICRA)*, 4239-4244. Porto Alegre: IEEE.

Rahmania 21] Rahmania, V., & Pelechano, N. 2021. Towards a Human-Like Approach to Path Finding. *Computers & Graphics*. http://www.sciencedirect.com/science/article/pii/S0097849321001576 (accessed July 16, 2024).

[Siddiqui 24] Siddiqui, R. 2024. Path Planning Using Potential Field Algorithm. *Medium*. https://medium.com/@rymshasiddiqui/path-planning-using-potential-field-algorithm-a30ad12bdb08 (accessed July 16, 2024).

[Sullivan 24] Sullivan, A. 2024. Distance to Any. *Red Blob Games*. https://www.redblobgames.com/pathfinding/distance-to-any/ (accessed July 16, 2024).