CS4096 AI For Games Project

2D Fighting Game

Click on the image link to watch the video

Click here to see the source code

Oscar Gutierrez Roman

23365978

Word Count: 2170

Introduction

This project centres around developing AI for a 2D fighting game or survival platformer, where the player character must evade a relentless boss, survive waves of spawned enemies, and navigate a procedurally generated map filled with platforms, obstacles, and bombs. The goal is to survive as long as possible, escaping the boss and fighting the enemies that attempt to attack the player.

In the game, the boss is a large, intelligent AI entity with the role of pursuing the player. The boss is designed to follow the player dynamically, expressing "thoughts" or emotional states through text displays or flashing its colour that respond to the player's actions, such as attacking, picking up bombs, or healing when killing an enemy. This AI feature not only heightens the sense of challenge but adds personality to the boss.

The enemy minions are smaller AI-controlled units that spawn continuously. Their primary behaviour is to detect, chase, and attack the player. This enemy AI adds pressure to the gameplay, requiring the player to constantly adapt and defend while evading the boss. Unlike the boss, these enemies rely on pathfinding techniques to chase the player efficiently through the map's obstacles.

So, the AI techniques used in our game are:

1. **Pathfinding**: Enables the minions to navigate the procedurally generated terrain, chasing the player without getting obstructed by the map's structure.
2. **Boss personification**: The boss's behaviour and decision-making are governed by a state-based system. The boss dynamically reacts to game events and changes its movement, colour, and speech based on context.
3. **Procedural map generation**: The map consists of dynamically generated tiles, creating varied and unpredictable terrains.
4. **Mini enemy reaction system**: Enemies use simple state-based mechanics to determine when to attack the player or flash upon taking damage.

Analysis

I will discuss each of the AI techniques we used by first briefly explaining its implementation, then its effectiveness and lastly its limitations.

1. Pathfinding for enemies (minions)

Implementation:

The Pathfinder method identifies the optimal path from the enemy to the player by using grid-based traversal. It evaluates neighbouring tiles and excludes invalid ones based on conditions like bounds, height differences, or tile occupation. It also considers vertical height differences and restricts movement to ensure logical behaviour (e.g., jumping when necessary).

The ReconstructPath function builds a path by backtracking from the target tile to the start tile, ensuring the path is efficient and directly connects the enemy to the player.

Movement is updated dynamically via the EnemyMovement coroutine, where the enemy iterates through the path tiles to reach the player. The system accounts for sprite flipping to keep visual consistency based on movement direction.

A stopping mechanism checks proximity to the player and stops movement once the enemy is close enough, adding realism to the chasing behaviour.

Rationale:

Using a grid-based pathfinding method is appropriate in this context due to its reliability in uniformly structured environments like grid-based maps. Research in game AI design suggests BFS is efficient for games where predictable pathfinding is required across procedural levels, especially when real-time adaptability is not critical. BFS also offers a performance advantage by focusing on shortest paths, making it well-suited for resource-limited scenarios like ours.

Effectiveness:

The use of BFS (or similar) ensures computational efficiency and accurate path generation even in complex, procedurally generated maps.

Restrictions on movement, such as vertical jumps and stopping within proximity, contribute to realistic enemy behaviour.

Limitations:

Jump mechanics might limit flexibility when enemies traverse uneven terrains with multiple height levels.

The lack of advanced algorithms like A* may result in less optimal paths in larger or more complex maps.

2. Boss Personification

Implementation:

Speed Adjustments: The boss dynamically calculates its movement speed through coroutines like DynamicSpeed, which measures the player's speed over time to adapt its own. There is also additional logic to slow down or speed up the boss based on proximity to the player and game context.

Reactions to Bombs: Using a switch statement, the boss reacts to the player's bomb count with custom animations, colour changes, and thought bubble messages. These reactions escalate with the threat level, providing a sense of progression in the boss's awareness and strategy.

Reactions to Health: a function tracks the player's health and adjusts the boss's actions, such as spawning minions or increasing movement speed. It only triggers new reactions when health changes, reducing unnecessary updates.

Thought Bubble System: The UpdateThoughtBubble function uses a queue to manage and sequentially display messages, ensuring that messages do not overlap or clutter the interface. The ProcessThoughtBubbleQueue coroutine handles timing and clearing of messages after display.

Colour Feedback: The FlashColor coroutine handles the boss's colour transitions, ensuring smooth visual feedback to the player. The boss's animator is temporarily disabled during flashes to avoid interference with the visual effects.

Rationale:

The use of predefined states and visual feedback was inspired by research on enhancing player engagement through "readable" AI behaviour. Studies suggest that visual and auditory feedback mechanisms, like changing colours or displaying messages, significantly enhance the player's immersion and understanding of AI intentions.

Effectiveness:

The boss feels more "alive" and responsive due to its thought bubble and dynamic visual feedback.

The combination of speech, colour changes, and adaptive movement creates a sense of unpredictability and challenge.

Limitations:

While the reactions are diverse, they rely heavily on predefined states. Incorporating machine learning or adaptive algorithms could improve variability and replayability.

The visual feedback (e.g., flashing colours) might not be sufficient in conveying the full range of the boss's emotional responses.

3. Procedural Map Generation

Implementation:

The map is constructed using tiles which are spawned dynamically as the player progresses. This ensures that the map continuously extends, creating the illusion of an endless or large game world.

Different shapes and configurations of tiles are generated programmatically, with specific rules ensuring navigability and gameplay fairness.

Tile placement considers the player's movement direction and position, ensuring that new areas appear seamlessly and logically. This adds depth to exploration without requiring predesigned levels.

The system integrates with the AI pathfinding, allowing enemies to navigate newly created sections without pre-existing navigation data.

Rationale:

Procedural generation was chosen for its ability to create variety in gameplay. This approach also aligns with game design for infinite games, where player retention often depends on continually fresh gameplay experiences without repetitive structures.

Effectiveness:

The dynamic generation ensures that no two playthroughs feel exactly the same.

It adds a layer of difficulty by requiring the AI (pathfinding) and the player to adapt to changing terrains.

Limitations:

Without detailed control over generation algorithms, there could be issues like inaccessible areas or overly challenging layouts.

If the map lacks aesthetic or logical cohesion, it may appear repetitive or unappealing to players.

4. Mini Enemy Reaction System

Implementation:

Enemies attack the player upon entering a specific range and flash upon receiving damage.

Enemies rely on proximity detection (to decide when to attack the player. This calculation uses distance thresholds in both X and Y axes, ensuring that enemies respond only when logically close enough.

When the enemies take damage, reactions are visually communicated through a flashing effect. We use Unity's sprite renderer to change colours temporarily, simulating a "hit" animation. The coroutine for flashing resets the enemy's colour to its original state after a brief interval,

The simplicity of the system allows for real-time updates without significant performance overhead, making it suitable for handling large numbers of enemies simultaneously.

Rationale:
A proximity-based detection approach was chosen due to its low computational cost and quick response times. Using a visual flash on hit also gives immediate visual feedback that enhances player engagement.

Effectiveness:

The simplicity of the system allows for quick and responsive reactions.

Enemy aggression increases the game's difficulty and keeps the player engaged.

Limitations:

The system does not incorporate complex strategies like coordinated attacks or defensive manoeuvres.

Enemy behaviours might feel predictable over time without further randomization or decision layers.

Reflection

Main Results and AI Behaviour

The implemented system of AI really works well, fitting the desired functionality and behaviours quite well. At first, the enemy AIs were using a very basic mechanism to walk directly to the player - not very realistic or challenging, whatsoever. Over iterations, this developed into a tile-based pathfinding system with the capability to make the enemies dynamically work their ways through a procedurally generated map. They could, for example, try to follow a player when he is avoiding obstacles.

The boss AI showed great success in addressing player events, such as gaining or losing health and bombs, and thought bubbles nicely help to express its “feelings”. Procedural map generation introduced some variation, and the behaviour of the mini enemies or logical in chasing the player, attacking when in range, and showing visual reactions for damage. Of course, the AI did struggle with consistency and adaptation with a few edge cases, and these did present several discussion points for improvement.

Challenges and Issues

Pathfinding was a challenge while implementing. The enemies would just update their position to the position of the player without considering any obstacles. Then on transitioning into a tile-based system, enemies were either getting into an infinite loop or not updating path. Jumping logic was also an issue as the enemies would jump into infinity or would never jump when needed. A flag variable was built in order to control jumping from happening.

Procedural map generation introduced its own challenges. While it added variety, it sometimes produced impassable structures or overly difficult sections, disrupting gameplay. Balancing the boss AI's speed with the player’s movements was also problematic, leading to unintended imbalances in the level of challenge.

The boss AI's preprogrammed reactions to certain player actions added some dynamism but didn't allow it to adapt to emergent gameplay.

Limitations

The pathfinding system, though improved, struggled with densely packed or complex map layouts, especially vertical sections. Enemies lacked the ability to dynamically adapt to failed paths or obstacles, limiting their effectiveness in some scenarios.

Procedural map generation also lacked validation to ensure navigability or balance, occasionally creating unfair or frustrating scenarios. These included isolated areas or overly challenging sections that disrupted gameplay flow.

Also, the boss AI was bound to a limited set of pre-defined responses, which became restrictive in innovating and surprising the player during longer gameplay sessions. Advanced approaches of reinforcement learning were not implemented, which restricted emergent behaviour by this AI.

Possible Improvements and Future Directions

These problems and limitations could be improved upon by applying an advanced pathfinding algorithm like A*; it will greatly improve efficiency and reliability, especially in complex layouts. The inclusion of adaptability features like marking of blocked tiles or dynamic adjustment of paths while navigating will significantly enhance navigation.

Procedural map generation could support a validation phase that would ensure the maps are fully navigable by both players and enemies. The inclusion of parameters required for difficulty scaling based on player progress would make for a more balanced experience.

Another such aspect Boss AI can be improved upon is reinforcement learning; it might learn from the player's strategy from one session to another. This will increase the depth for a more sophisticated emotion system and make the boss less predictable.

Mini-boss enemies could make use of communal tactics of ambushes or team attacks. Flocking algorithms could be implemented to better create improved dynamics between these creatures. Variable reactions-e.g., hesitations or evasive moves-would reduce predictability and enhance the challenge.

Advanced AI frameworks, such as utility-based systems, could raise the level of adaptability and decision-making of AI. Dynamically, these systems would prioritize actions by evaluating multiple factors at runtime, such as player health, bomb count, and distance, to enrich gameplay.

In a nutshell, though the AI system achieved both functional and engaging gameplay goals, this serves only to help in refining the critical issues at hand. Advanced techniques, dynamic adaptation, and strong design principles would easily enable the AI to develop into a more intelligent and immersive system that meets industry standards.

Conclusion

The following report described the implementation, reasoning, and evaluation of our project AI and gameplay systems, such as pathfinding, procedural map generation, boss behaviour, and enemy interaction. While these were successful in making gameplay engaging, there are areas of improvement.

Individually, I contributed to many core mechanics, from the pathfinding system that enabled enemies to navigate procedurally generated maps to a flashing feedback system that gave clear damage indications. Other contributions I made were in the player and enemy scripts and other interactive components, such as the thought bubble system, health bar, and kill counter. All of these systems needed careful integration and debugging; sometimes it would be painful, but the result was a polished and responsive feature set.

Overall, the project showed good teamwork and application of the principles of game AI. We might need to develop further skills in task delegation and efficient testing, but the systems I designed met the project goals and set a good foundation for refinement.

VIDEO

SOURCE CODE

References

https://theory.stanford.edu/~amitp/GameProgramming/

https://gamedev.stackexchange.com/

https://docs.unity3d.com/ScriptReference/Tilemaps.Tilemap.html

https://learn.unity.com/tutorial/introduction-to-tilemaps#

http://pcg.wikidot.com/