Game Development Project: Survivor Struggle

Link: https://youtu.be/D4jA6JH6odw

Source code Link: https://drive.google.com/file/d/1hOULuD6xvObICjr3jerni5nd-qEHO8Y7/view?usp=sharing

Name: Steven Gerard Mascarenhas

1. Introduction

Survivor's Struggle is a 2D platformer game where players navigate through the map with intelligent enemy AI, including spiders, archers, creepers, and a powerful boss with unique abilities. The player must strategically use health power-ups, melee attacks, and shoot bullets to survive and overcome challenges.

The player has basic movements like idle, running, attacking by shooting bullets, and using a sword to hit enemies. The player can also get hurt from attacks and dies when their health runs out. In Survivor Struggle, the main goal is to make the enemy AI more intelligent, ensuring the game is engaging and challenging for the player while remaining fair. Based on the player's behavior, enemies like the spider, archer, and creeper adapt to the environment and react to the player's presence in different ways.

To showcase complex enemy behavior, I worked with finite state machines [FSM's] for the Creeper and Boss AI and the A* pathfinding AI technique for the Bird AI. The key states that made the enemies feel complete included idle, patrol, look for player, player detected, charge, explode, and dead states for the Creeper, and run, attack, enrage, enrage run, and enrage attack states for the Boss AI. Using A* pathfinding, the Bird AI follows a predetermined path.

Examples of such technologies are seen in games like Left 4 Dead and Shadow of the Tomb Raider. In Left 4 Dead, FSMs control the behavior of various zombies, adapting to the player's actions to create a challenging experience. In Shadow of the Tomb Raider, FSMs are used for states like patrolling, attacking, and detecting players, combined with A* pathfinding to navigate dynamic environments, ensuring engaging gameplay.

2. Analysis

2.1 Boss Enemy

Fig 1. AI Boss Enemy

I implemented the Boss AI using a basic FSM because FSMs allow transitions between states based on conditions. In the case of the Boss enemy, I needed a structure that enabled the character to perform smart movements and execute special abilities and attacks.

The Boss AI FSM includes states such as movement and attack. What makes the Boss unique is its enraged run and enraged attack states. After the player hits the Boss multiple times, the enraged run state is triggered, causing the Boss to chase the player relentlessly. Once in proximity, the enraged attack state is triggered, delivering stronger attacks.

The normal attack targets the player based on position offset and uses colliders to detect proximity. By applying an attack mask, the Boss ensures it only interacts with specific objects like the player. If the player is detected, the Boss attacks, reducing the player's health. For the enraged attack, the damage dealt is increased, providing more intense gameplay. This logic ensures the Boss delivers a balanced but challenging experience.

2.2 Creeper Enemy

Fig 2. Creeper Enemy

For the Creeper, I used a reusable FSM with inheritance, allowing common behaviors to be shared across different enemies. The goal was to give the Creeper complex behaviors like moving, detecting the player, charging, exploding, and dying.

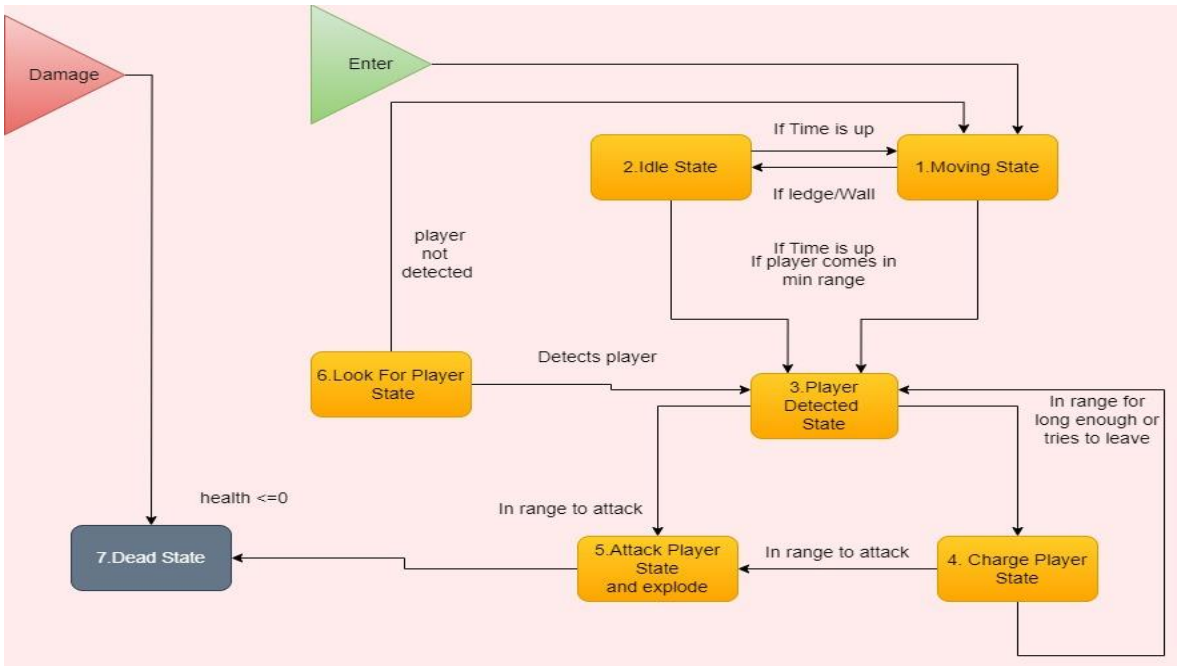

Fig 3. Finite State machine diagram

Based on conditions, the Creeper transitions between states:

1. Idle State – Keeps the Creeper stationary when a ledge or wall is detected.
2. Moving State – Allows the Creeper to patrol or transition to idle or player-detected states when conditions change.
3. Look for Player State – Lets the Creeper "search" for the player. If the player is not found, it returns to the moving state.
4. Player Detected State – Detects the player within range and prepares to attack.
5. Charge Player State – Enables the Creeper to charge at the player when detected.
6. Attack and Explode State – Causes the Creeper to explode, dealing damage to the player, and then transition to the dead state.
7. Dead State – Marks the Creeper as killed, cleaning up resources.

By using inheritance, I created a base class for shared states, allowing enemies to reuse and extend behaviors. This modular approach ensures scalability and reduces redundancy.

In general, the FSM framework consists of the state machine, states, and entities. The state machine handles transitions between states, the state class defines behaviors, and the entity class provides shared functionality, such as checking for ledges or detecting ground.

3. Reflection

3.1 Boss AI

The FSM implementation for the Boss AI successfully created dynamic and intelligent behavior, improving the gameplay's challenge and engagement. Key outcomes include:

1. Movement - The Boss actively navigates to the player, maintaining pressure.
2. Normal Attack - Targets and damages the player within a specific range using offset calculations.
3. Special Ability - Transitions to an enraged state after receiving significant damage, increasing the gameplay challenge.

In the enraged run state, the Boss's movement speed increases. In the enraged attack state, the damage dealt is significantly higher. These transitions create smooth and responsive AI behavior, keeping the gameplay engaging.

Initially, the enraged attack state relied on a time factor to activate, but I modified it to depend on the player's actions instead. This adjustment improved realism and flexibility, making the game more reactive to player behavior.

An alternative approach to FSMs could involve behavior trees. While FSMs are effective and easy to implement, behavior trees offer more scalability and flexibility, especially when adding more complex abilities in the future.

3.2 Creeper AI

The Creeper AI demonstrated successful implementation of intelligent and dynamic behaviors:

1. Idle State - Detects walls and ledges to avoid collisions or falling.
2. Moving State - Seamlessly transitions between patrolling and detecting the player.
3. Look for Player State - Allows the Creeper to "search" for the player and return to patrolling if none is found.
4. Player Detected and Charge State - Detects proximity to the player and charges intelligently.
5. Attack and Explode State - Deals explosive damage and transitions to the dead state.
6. Dead State - Ensures proper resource management for smoother gameplay.

Initially, the explosion attack triggered instantly, which felt unnatural. I introduced a brief delay to improve gameplay pacing. I also resolved an issue where the attack animation failed to trigger due to a missing animation event by explicitly linking the animation event to the attack function.

Future improvements could include enhancing environmental awareness, such as reacting to sounds like footsteps or weapon use, for more immersive AI behavior.

4. Conclusion

This report outlines the implementation of AI for enemies in Survivor's Struggle, focusing on FSMs and pathfinding techniques to create intelligent and dynamic behaviors. These approaches successfully enhanced the game's challenge and engagement.

Reflecting on my performance, I achieved my goal of designing adaptive AI while addressing challenges such as unnatural transitions and animation inconsistencies. However, exploring advanced AI techniques like behavior trees could further improve scalability and gameplay depth.

Overall, this project was a valuable learning experience, showcasing my ability to implement AI systems and balance gameplay for a more engaging experience.

References

https://learn.unity.com/project/finite-state-machines-1

https://www.youtube.com/watch?v=AD4JIXQDw0s&list=PLRRnET3ZAhEzkrCdxn9Ik1xvwuwrFpPDp

https://www.youtube.com/watch?v=Pux1GlFwKPs&list=PLy78FINcVmjA0zDBhLuLNL1Jo6xNMMq-W

https://assetstore.unity.com/packages/2d/characters/knight-sprite-sheet-free-93897

https://assetstore.unity.com/packages/2d/characters/pixel-adventure-1-155360

https://assetstore.unity.com/packages/2d/characters/2d-simple-character-swordman-133259?aid=1101lPGj