CS4303 Geometry Wars Report

140011146

November 9, 2017

1 Introduction

In this practical we were tasked to implement a top-down shooter game, similar to games like *Robotron 2084* and *Geometry Wars* with the goal of implementing a variety of AI algorithms. In my game, I have implemented six different types of AI, including a flocking AI. There is also an AI director that controls the difficulty and pacing of the game. I have also implemented multiple different pickups and the ability to play over a network.

To build the game, run `ant` in the submission directory.
To run the server or a single player game, run `ant server`
To run the client, run `ant -Darg0="Name" client` where "Name" can be any name.

2 Game overview

In this game the player controls a character, running away and shooting at the enemies. The player has different weapons and can pickup different power-ups to help them.

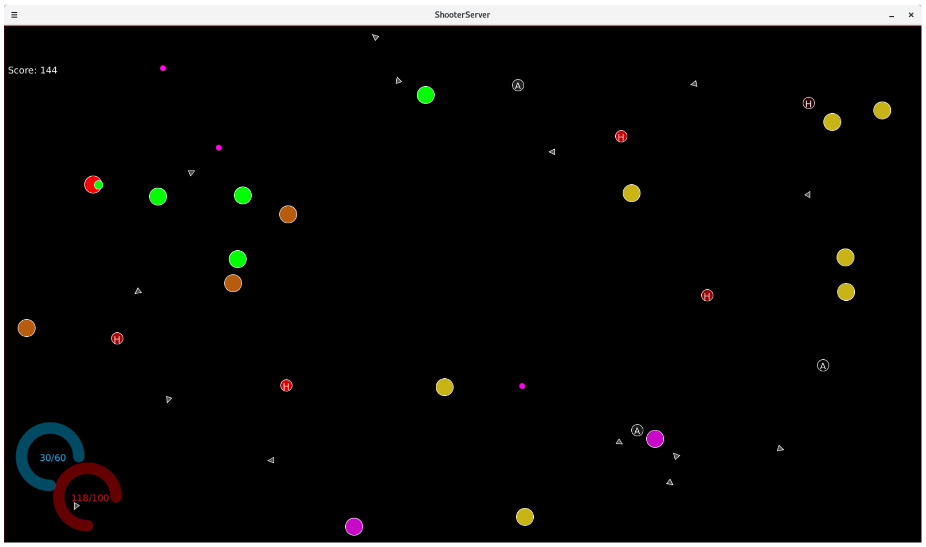

Figure 1: Screenshot of the game

The HUD on the bottom left shows the player's ammunition and health. Enemies who collide with the player deal damage to them and are destroyed. The player should try to avoid and shoot the enemies while picking up the powerups. It is important to get the pickups because they provide useful benefits like more health and damage, which is needed as the difficulty increases.

3 Game features and design

3.1 AI Enemies

3.1.1 Basic chase

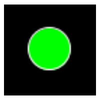

Figure 2: Basic Chase Enemy

This is the most basic AI that always goes towards the current position of the player. If the player is always moving, this AI will always be chasing behind the player, making it quite easy to run away from.

3.1.2 Circle

Figure 3: Circle Enemy

This AI is similar to the basic chase AI except they use acceleration in their movement rather than moving at a constant speed towards the player's position. This means that if the player moves past the quickly, they will take more time to reverse and follow the player again.

3.1.3 Ambush

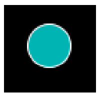

Figure 4: Ambush Enemy

This AI attempts to move towards where it predicts the player will be based on the direction the player is moving in. It will try to move to a position in front of the player.

I found when first creating this AI that if it implements this behaviour and there are a lot of them, they will all move synchronously when the player changes direction, making them easy to predict for a player. To deal with this, I added a random delay before they change their target position making each of them act more independent and seemingly somewhat randomly.

3.1.4 Patrol

Figure 5: Patrol Enemy

These AI spawn in with the generated pickups and will patrol in a square around the pickups. If the player comes near they will chase the player for a certain distance before moving back to their patrol locations.

This adds a dynamic to the game as the players can avoid these enemies or shoot them from far away without retaliation, but may be forced to go near them to get the pickup or from trying to avoid other enemies.

3.1.5 Shoot

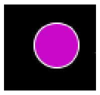

Figure 6: Shooting Enemy

This AI will shoot back at the player, making it more difficult. I chose to make them shoot directly at the player position rather than randomly near the player so it is easier to dodge if the player is always moving. These AI will move randomly around the screen while shooting.

3.1.6 Flocking

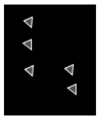

Figure 7: Flocking enemy

The flocking algorithm was implemented following the example on https://processing.org/examples/flocking.html [2].

Initially these enemies moved towards the player position if there were no nearby flockmates but this made the game very hard as they would all flock towards the player. Instead, their initial target position is the player's current position and there is no other target direction or velocity when there are no nearby flockmates. So although they initially move towards the player, they will ignore where the player goes afterwards and simply follow their flock.

These AI are also smaller and more are spawned in the game to show off their flocking behaviour without filling the whole screen. As a result, they have less health and deal less damage compared to all other enemies.

3.2 AI Director

3.2.1 Pacing

The AI director uses an adaptive pacing algorithm following the system used in Left 4 Dead [1]. The idea is to have bouts of high intensity and low intensity, the durations of which change dynamically. The "intensity" of players is roughly estimated:

- Increase intensity when the player takes damage, proportional to the damage.
- Increase intensity when the player kills an enemy, proportional to the score value of killed enemy.
- Decrease intensity over time.

If the intensity is too high, no shooting enemies are spawned to make the game a little easier for the players.

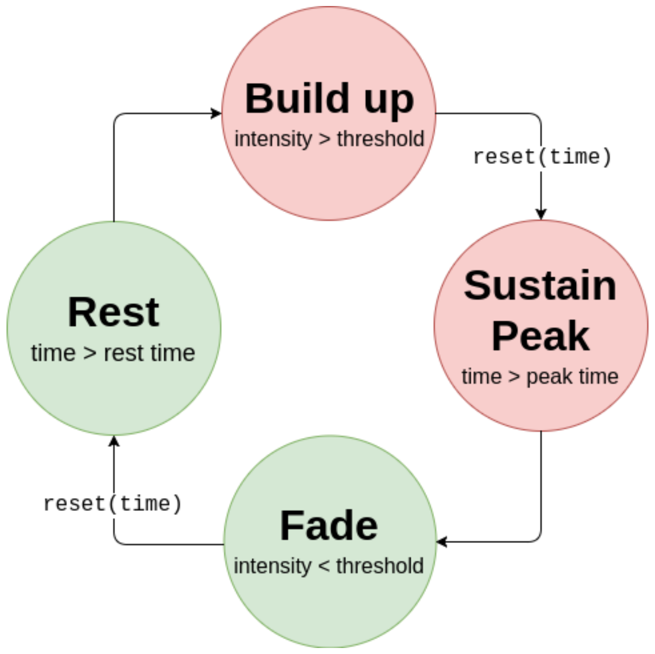

Figure 8: State machine of the AI director.

The **Build up** and **Sustain Peak** states have a higher spawn rate of enemies. The director moves from build up to sustain peak when the intensity is greater than the threshold. The sustain peak state is there to ensure there is some minimum time spent with high spawn rates of enemies.

Fade and **Rest** states are the opposite side to build up and peak. The director moves out of fade when the intensity is below the threshold and rest acts the same as peak, ensuring a minimum period of rest before the next build up.

3.2.2 Difficulty

At regular intervals, the state of the game is sampled to see how well the players are doing and adjust the difficulty accordingly. By default, the difficulty slowly increases at each interval to make the game progressively harder.

An increase in the difficulty value of the game affects the following parameters:

- Increased enemy health
- Increased enemy damage to players
- Increased enemy score value
- Increased enemy spawn rate
- Decreased pickup spawn rate
- Decreased pickup lifespan

The difficulty is adjusted based on the player health and the number of enemies on screen. If the players have high health or there are few enemies, it indicates they are doing well since they are healthy or killed lots of enemies recently so the difficulty is increased. The difficulty is decreased in the opposite case, as it indicates the player is struggling to kill enemies or stay alive.

3.3 Weapons

The player is equipped with three different weapons that can be quickly swapped at any point. Each weapon has a separate ammo and reload system, making it convenient if the player runs out of ammunition on one weapon to switch to another rather than wait for the reload time.

3.3.1 Pistol

Figure 9: Player with pistol equipped

The pistol is a simple weapon with low clip size and low fire rate and faster reload.

3.3.2 Rocket

Figure 10: Player with rocket equipped

The rocket is a powerful weapon that creates an explosion on impact, dealing lots of damage but it comes with extremely low clip size, low fire rate and slow reload.

3.3.3 Machine gun

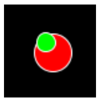

Figure 11: Player with machine gun equipped

The machine gun shoots bullets quickly with a larger clip size and high fire rate but takes longer to reload.

3.4 Pickups

Pickups in the game help the player by giving them either some kind of temporary or permanent boost. These pickups only last for a certain duration on the ground before they disappear, which is indicated by their reduced opacity.

3.4.1 Health pickup

Figure 12: Health pickup

The health pickup gives the player more health. A player is able to go over the maximum 100 health when picking up these pickups to act as a buffer for more health, but this will signal to the AI director that the player is doing well.

3.4.2 Ammo pickup

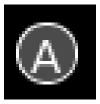

Figure 13: Ammunition pickup

The ammo pickup increases the ammo of each of the player's weapon by one clip. It is important to pick up these, otherwise the players will eventually run out of ammunition. Players have a maximum number of clips they can carry, so picking this up at that limit will not increase their ammo.

3.4.3 Speed pickup

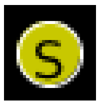

Figure 14: Speed pickup

This pickup temporarily increases the speed of the player who picked it up. This is useful to run away or reach a far pickup quickly, but it is a double-edged sword as the player can easily run head-on into a group of enemies.

3.4.4 Pierce pickup

Figure 15: Pierce pickup

The pierce pickup will allow the player's bullets to pierce through all enemies. This does not apply to the rocket weapon.

3.4.5 Damage pickup

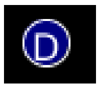

Figure 16: Damage pickup

This pickup permanently increases the player's damage. This helps the player deal with the increased health of enemies from increasing difficulty.

3.4.6 Bullet radius pickup

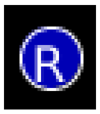

Figure 17: Bullet radius pickup

This pickup permanently increases the player's bullet's radius. This helps the player with easily aim as the larger bullets will catch enemies easier, especially when trying to shoot the flocking enemies. This increased radius also affects the rocket's explosion size, as it scales off the rocket bullet's radius.

3.4.7 Bomb pickup

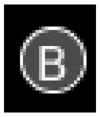

Figure 18: Bomb pickup

This is a powerful pickup that will clear the entire screen of enemies, giving players a brief respite.

3.5 Networking

The game can be played over the network, allowing multiple players to play cooperatively. Different players must have a unique name as this is how they are differentiated.

The server game should be launched and be in the playing state. From that point on, any client can drop-in connect to the same game. Multiple different games have not been implemented, as all servers and clients use the same multicast group to transmit messages.

4 Implementation details

4.1 Factories

The factory design pattern was used to spawn enemies and pickups. An abstract `SpawnFactory` is defined so factory implementations can take care of spawning game objects. The factories have a map of `<Function, Integer>` where the function is a function to spawn an exact type of game object and the integer is the spawn rate. The factories will take care of spawning entities into the game, scaling the spawn rate and object attributes with difficulty. The AI director can use the factories to spawn entities as it needs.

4.2 Pickups

Pickups are split in a `Pickup` class and `Effect` class. A `Pickup` is the actual pickup object displayed in the game while the `Effect` is stored on the player characters. Once the pickup is taken by the player, the effect is stored and applied on the integration step of the player. If an effect has a duration, it will also be un-applied at the end of its duration.

4.3 Network

To implement the networking in the game, I use a processing UDP library (https://ubaa.net/shared/processing/udp/).

UDP packets are sent on every frame of the game. The server will send the entire game state and let the client render the state. The Server uses multicast so every state packet reaches all connected clients. Clients only send the player input to the server to process. This means the server is always correct and would prevent client players from cheating their attributes such as position and health. The trade off is the server has to do a lot more processing and so the game cannot scale to having many players at once.

Because the game is relatively small, it was okay to send the entire game state over the network without much loss in performance and perform all calculations on the server side. For a larger scalable game, it would have been more appropriate to send the changes in the game state and let the client perform calculations based on those changes.

The network has only been tested over a multicast group on the lab machines and not across different networks.

4.4 Class Structure

4.4.1 Game states

The game states deal with most of the game logic and update steps. They contain the update function to update all game objects and the AI director to control the spawning of entities. Similar to the last practical, the game states form a state machine, but unlike the previous practical, this is only used for the start and end game states. There are not any different states of the game like pausing or end of wave.

4.4.2 Game objects and classes

All game objects such as enemies, pickups and bullets extend the `GameObject` abstract class. This allows the game state and controller to use polymorphic data structures over any game object. For example the spawn factories can be of any type that extends game object.

Classes such as `GameContext` and `GameInput` are wrapper classes that contain lots of information. For example the context contains all the game objects and the input contains all the various input.

5 Conclusion

In conclusion, I have implemented a 2D shooter game with a variety of AI enemies. To make the game more interesting, different powerups and weapons have also been implemented along with an AI director that controls the difficulty and pacing of the game. The ability to play cooperatively over the network is also implemented.

References

[1] Michael Booth. *The AI Systems of Left 4 Dead.* Valve Software, 2009.

[2] Daniel Shiffman. *Flocking.* `https://processing.org/examples/flocking.html`.