Missile Command

160020220

October 30, 2019

Contents

1 Introduction

In this practical I was asked to develop and implement in Processing a variant of the classic video game Robotron 2084. In my Project, I have implemented all the features in the basic practical specifications as well as many additional extensions.
Robotron 4303 is a single-player game in which the player controls a "superhuman" beset by a horde of robots, who have revolted against the humans. The superhuman has a projectile weapon with which to fight, and is also tasked with rescuing the last human family.
In my design, the game takes place in a haunted forest where ghosts and giant insects are the enemies and the player has to rescue humans. Since the Practical deadline is near Halloween, I wanted to go with that theme combines with a relaxing and light garden feel.
This practical focused on implementing some of the concepts from the AI and Procedural Generation components of the module.
In the practical directory, launch the game by running:
java -jar VGpratical2.jar

2 Quick Player Guide

HERE IS AN EXTREME PLAYER GUIDE TO BE A HARDCORE GAMER AND PWN ALL THE ENEMIES.
Move using WASD buttons. aim and shoot with the mouse. Use R to reload and 1-2-3 to switch between weapons. This is pretty basic stuff.
in the start screen, launch the game by pressing ENTER. NOW NOW NOW, you are in the game!
The HUD on the bottom left shows YOUR ammunition and health. Enemies who collide with YOU deal damage. YOU 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 ammo, which is needed as the difficulty increases.
TIP: GET THE PICKUPS. IT IS IMPORTANT!!!
At the end of the wave, Press F to open the SHOP. and R to return to end wave screen.
TIP: TRY TO SAVE POINTS TO GET THE DMG BOOST. IT IS VERY STRONG!!!
Kill the enemies to pass the waves. Earn points and buy more upgrades! GOOD LUCK AND HAVE FUN!

3 Design and Implementation

3.1 Basics

3.1.1 Player

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. The player moves with a set of movement keys (WASD) and aims and shoots with the mouse.

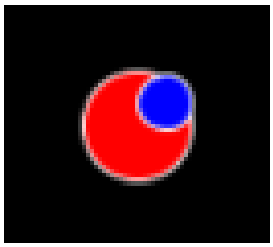

Figure 1: Screenshot of the Player Character

The player has a 100 Health points must kill enemies, collect humans and NOT DIE!!!

3.1.2 A*

What A* Search Algorithm does is that at each step it picks the node according to:
a value-'f' which is a parameter equal to the sum of two other parameters – 'g' and 'h'. At each step it picks the node/cell having the lowest 'f', and process that node/cell.

g = the movement cost to move from the starting point to a given square on the grid, following the path generated to get there.

h = the estimated movement cost to move from that given square on the grid to the final destination.

This Is referred as a Heuristic approach to the search and it makes this algorithm one of the best and popular used in path-finding and graph traversals.

3.1.3 Enemies

Humans And enemies implement an A* algorithm described above for the path finding.

Basic Chase Enemy

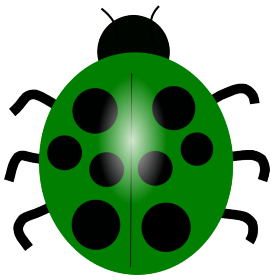

Figure 2: Screenshot of the Player Character

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. However, This enemy is not quiet dumb, It moves slowly and randomly until the player enter it's detection radius, then it runs very fast towards the player

Circle Enemy

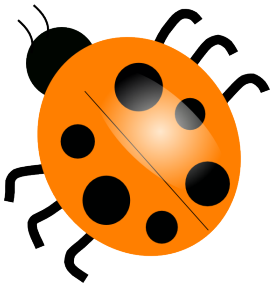

Figure 3: Screenshot of the Player Character

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. Plus, This enemy doesn't use a detection radius and will always chase the player. However, It is slower than the Basic Chase Enemy when chasing the player.

Ambush Enemy

Figure 4: Screenshot of the Player Character

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. // Furthermore, This enemy can go through walls! when implementing it, It was displayed over the walls which made it easy for the player to shoot them. To make things harder, the ambush enemy is displayed behind the trees but can still be seen when going between trees.

Patrol Enemy

Figure 5: Screenshot of the Player Character

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. // This enemy Can also go through wall to add a some difficulty.

3.1.4 Obstacles and Humans

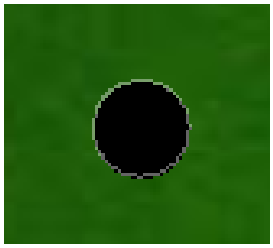

Figure 6: Screenshot of an obstacle

Obstacles extend enemies. they are a fixed enemy that deals damage to the player. They are represented as black balls as I want to confuse the player between pick ups and obstacles in later stages of the game.

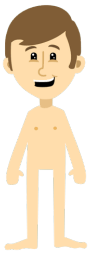

Figure 7: Screenshot of the Player Character

Humans on the other hand are really dumb objects that run towards the player when they see it. Humans use their detection radius and A* algorithm to move. Picking up a human add 5 points to the score. Humans can be killed by Enemies!

3.1.5 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!

Health pickup

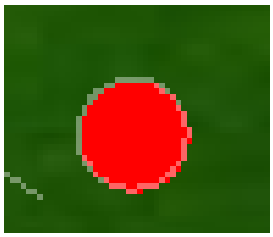

Figure 8: Screenshot of the Player Character

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!

Ammunition pickup

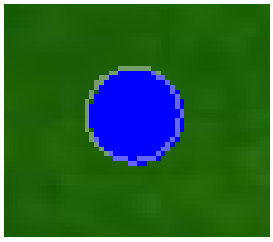

Figure 9: Screenshot of the Player Character

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.

Speed pickup

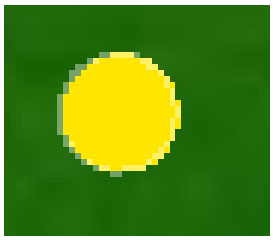

Figure 10: Screenshot of the Player Character

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.

3.1.6 Game states

To split the game up into different states such as start of game, end of wave, shop etc, I've implemented each state as its own separate class and made the game change states like a state machine.
The idea is that on each `update()` or `handleInput()` step, the current state will return what the next state should be. This allows each state to only keep track of themselves and handle everything that happens in the state within their own class.

The controller then does not care about what each state is or what it does, but simply calls the update and draw functions of the current state.
To complement the state machine, there are three classes that encapsulate many game properties. The `GameContext` class is used to encapsulate all information about the game, such as the lists of objects, the physics engine, the current level, the score, etc. `GameInput` represents all the input from the player and has fields such as the mouse position and the key that was pressed.

3.1.7 Waves

The game is organised into waves. At the beginning of each wave, the map is procedurally generated and GameObjects are spawned. Each wave has increasing difficulty as every wave an additional enemy is spawned which makes the game more challenging. Plus, every few waves, the player is rewarded with an additional human spawn but has to be careful because there will also be an additional obstacle!

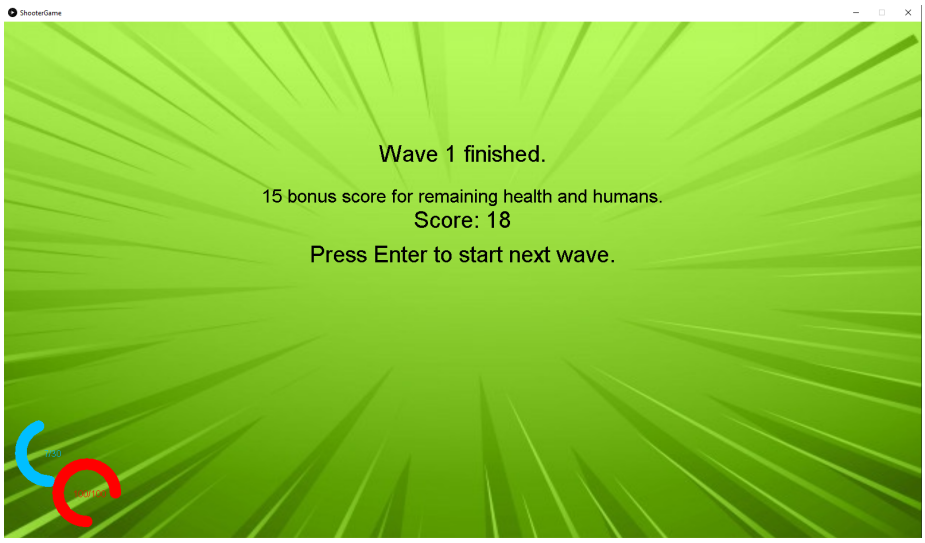

Figure 11: Screenshot of the Player Character

The number of waves is unlimited and the game is over when the player's health drops to zero.

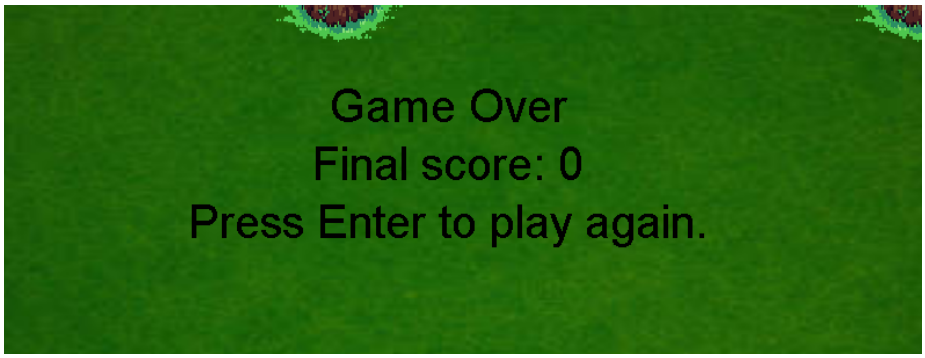

Figure 12: Screenshot of the Player Character

Many things add score to the scoring system:
Enemy Destruction gives some score during the wave. But the main scoring occurs at the end of a wave.
Indeed, the number of remaining Health and Humans are counted up and added to the player's score. Of course, Humans should be collected during the wave to prevent them from dying! This system rewards skilled players as they were able to keep more humans alive or even collect them and also use their positioning and shooting well enough to still have a lot of health remaining at the end of the round. Remaining Health points give a lot more score as it takes more skill

to keep them from dropping throughout the wave. This provides a trade off when deciding to spend score to regain health points as they can provide higher "income" of score at the end of every round, but the player must be able to keep them from plummeting. This income balance must be well managed by the players since the game gets more and more difficult throughout the waves:

- Increased number of Enemies every wave.
- new obstacles start to appear every few waves.

However, other things make the game easier as the waves pass:

- Pickups spawn rate is increased.
- The shop system allows the player to upgrade! (more on that later)

This increase in difficulty would automatically mean an increase in score since there are more enemies to deal with, and more humans to pick up. The idea of a score multiplier at the end of specified wave levels was also thought of. However, The multiplying wave scoring system Has been implemented but then deleted for balance issues. Indeed, Since I use a shopping system as on extension of the practical, The scoring multiplier would have made things too easy for the player in later stages of the game. The player would have been able to buy an absurd amount of upgrades and health/ammo packs.

3.1.8 Procedural Generation

To Procedurally generate my Map. I used a Random Walk Algorithm in combination with my walls objects.

Walls

Walls are circular obstacles that cannot be traversed by the player or the simple enemies. No GameObjects can spawn in walls except pickups. Indeed, I chose to make pickups spawn randomly all over the map because it would look like a real forest where a pickup is sometimes not accessible. Thus making the game more random, dynamic and fun.

Figure 13: Screenshot of the Player Character

The first thing we do is populate the entire map with walls using a while loop. Then we call a digger to remove walls and procedurally generate the map. This is where the Random Walk Algorithm is called.

Random Walk Algorithm

After making a grid-like map of walls, this algorithm starts from a random place on the map. It keeps making tunnels and taking random turns to complete its desired number of tunnels. It follows the following steps.

- Makes a two dimensional map of walls
- Chooses a random starting point on the map
- While the number of tunnels is not zero
- Chooses a random length from maximum allowed length
- Chooses a random direction to turn to (right, left, up, down)
- Draws a tunnel in that direction while avoiding the edges of the map
- Decrements the number of tunnels and repeats the while loop
- Returns the map with the changes

This loop continues until the number of tunnels is zero. I used the wiki page as my main documentation source:
https://en.wikipedia.org/wiki/Random_walker_algorithm
This codepen is what my digger algorithm does:
https://codepen.io/anon/pen/aLpORx

3.2 Extensions

3.2.1 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.

Pistol

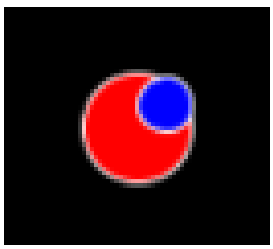

Figure 14: Screenshot of the Player Character

The pistol is a simple weapon with low clip size and low fire rate and faster reload. Initially, the player may choose to use the pistol when the game is easier to conserve ammunition on the other guns.

Machine gun

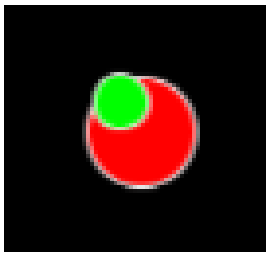

Figure 15: Screenshot of the Player Character

The machine gun shoots bullets quickly with a larger clip size but takes longer to reload and deals less damage.

Rocket

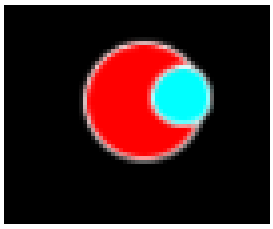

Figure 16: Screenshot of the Player Character

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.

3.2.2 Shop

A shop extension is added for players to spend score to purchase:

- HP Pack.
- AMMO pack.
- Damage boost for the pistol.

- Damage boost for the machine gun.

The player must choose between keeping their score high, or using it to ensure they don't lose in the next round.

Figure 17: Screenshot of the Player Character

3.2.3 Graphical improvements

I have Made a lot of graphical improvements by using images to represent objects.

3.2.4 Sound effects

To implement the Sound effects, I used a Minim Audioplayer. I would use this player to laod an audio file and play this audio file when I wanted to (this.player = minim.loadFile("data/audio.wav");).
In my game, I have 3 rudimentary sound effects:

- When the player shoots, the photon.wav sound file is played. This is done in the handleInput method that allows the file to be played when the player shoots from a Cannon.
- A death sound is played when an enemy dies.
- An ambiance music with birds and nature sound is looping to make the game more enjoyable

The sound effects might be a bit loud. Be careful when testing the game!

4 Conclusion

In conclusion, I've been able to implement many features into my game. Some features make the game more difficult, but others try to help the player be more successful.
I am very happy with the code quality and the overall deliverable!
Furthermore, most of the testing has been shown throughout the report, additional testing have been done but were not relevant enough to be mentioned in the report!