Missile Command

160020220

October 4, 2019

Contents

1 Introduction

In this practical I was asked to develop and implement in Processing a variant of the classic video game Missile Command. In my Project, I have implemented all the features in the basic practical specifications as well as some additional extensions. A flash version of the game is available here :
https://my.ign.com/atari/missile-command
In the practical directory, launch the game by running:
java -jar Pract1.jar
Missile Command is a single-player game in which the player controls three missile batteries, each with a set number of missiles. The player's goal is to protect the Bases from falling meteors using these Cannons.

2 Quick Player Guide

- press ENTER to start the wave.
- LEFT CLICK to shoot normal missiles.
- RIGHT CLICK to shoot a Black Hole.
- MIDDLE MOUSE BUTTON to use a Force Field.
- press F to open and close the SHOP.
- HAVE FUN

3 Design and Implementation

3.1 Basics

`Config` Is the central class where all the information about game data is stored.

The overall atmosphere for the game was really important to me. In this practical, I focused more on the Player experience rather than hard coding a pale copy of missile command. I also focused on the particles behaviours as well as forces handling.

I wanted my game to "feel" light and relaxing to play. This is why I went for a not so hard game, with light colors design and cartoony/cottony nuke post apocalyptic atmosphere.

3.1.1 Layout : bases and Missile batteries

The play area is arranged as per Missile Command with very few modifications explained throughout the Practical.

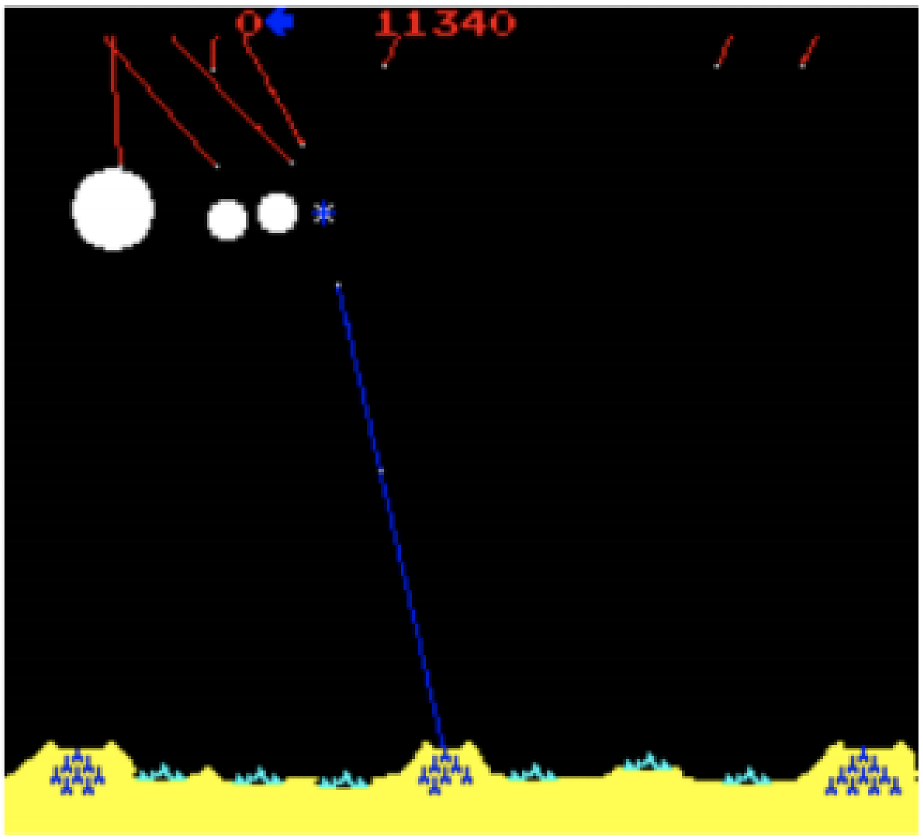


Figure 1: This is the original ground layout of the missile command game.

Now comparing the original game layout to our layout, we can see that the layouts are fairly similar and the placement of the bases and Missile batteries is quiet clear!

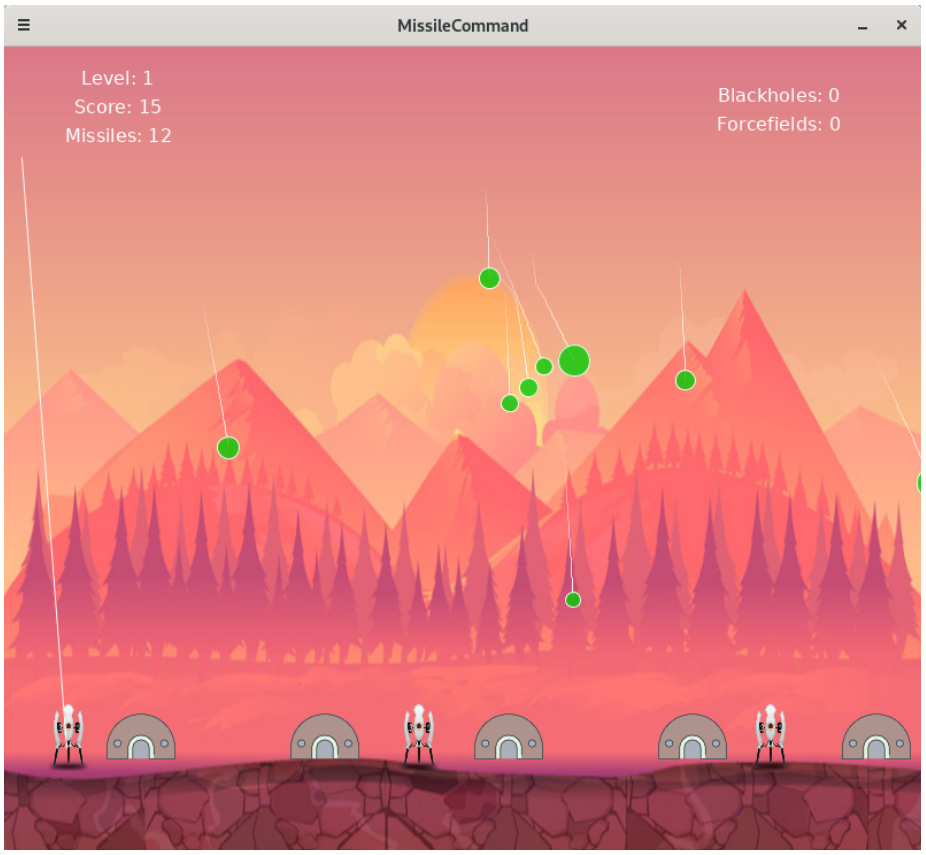


Figure 2: This is the Layout of my game.

3.1.2 Buildings: Bases and Cannons

Bases and Canons extend the `Building` class. The Buildings object's placement on the layout is fixed and their hit-box is represented by a circle. Buildings are destroyed when their hit-boxes are in range of an explosion.

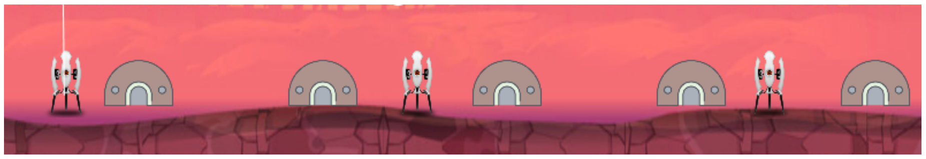


Figure 3: This is the Layout of my Bases and Cannons.

When a Base is destroyed, It is simply removed from the Layout and will not reappear unless enough points are made to rebuild the base. The Cannons However are disabled when hit and this is shown when a red exclamation mark circle replaces the Cannon. The disabled Cannons are repaired between each wave.

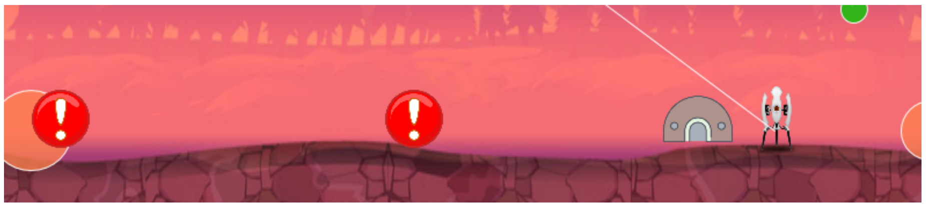


Figure 4: This Shows some destruction done to the bases and cannons.

The base arrangement is static and in evenly spread locations. This is so there aren't any balance issues with bases being too concentrated in certain locations.The number of bases can be changed and the game is able to place any number of them evenly.

Because the score given for remaining bases is high, I have chosen to have only five bases in the game by default. Having less bases also makes the game a little easier as there are less bases to defend and less to rebuild for the endgame

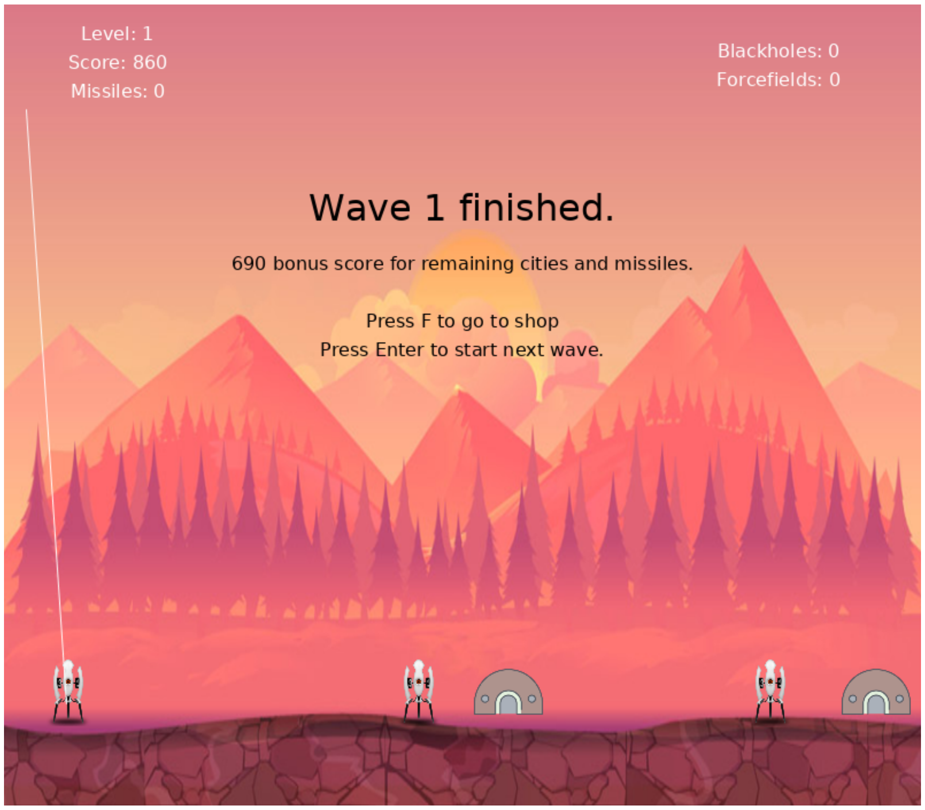
3.1.3 Game States

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 `Game` class is used to encapsulate all information about the game, such as the lists of objects, the physics engine, the current level etc. `Inventory` is for all information that is displayed to the player, for example the number of missiles left or the score. `Input` represents all the input from the player and has fields such as the mouse position and the key that was pressed.

3.1.4 Waves: Scoring System and Increasing Difficulty

The game is organised into waves. At the beginning of each wave, the number of missiles the player has is increased by a random scaling amount. Each wave has increasing difficulty as every few waves a bomber also comes which makes the game more challenging. However, every few waves, the player is rewarded with a black hole or force field to help them survive.



Wave 1 finished.

690 bonus score for remaining cities and missiles.

Press F to go to shop
Press Enter to start next wave.

Figure 5: At the end of every wave, This display shows up. From here, the player can access the shop

The number of waves is unlimited and the game is over when all bases have been destroyed.

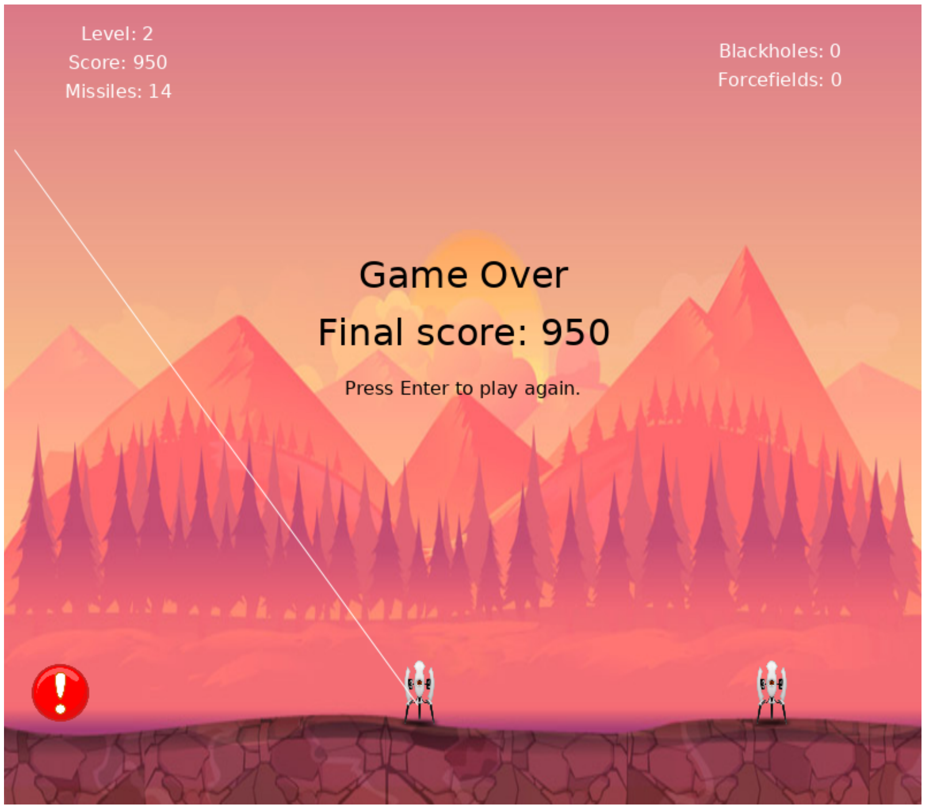


Figure 6: When all Bases are destroyed,It's GAME OVER.

Many things add score to the scoring system:
Meteor Destruction gives some score during the wave. But the main scoring occurs at the end of a wave.
Indeed, the number of remaining bases and missiles are counted up and added to the player's score. This is the same as in the original *Missile Command* game and rewards skilled players as they were able to keep more bases alive and also use their missiles strategically enough to still have some remaining at the end of the round. Remaining bases give a lot more score as it takes more skill to keep them alive. This provides a trade off when deciding to spend score to rebuild bases as they can provide higher "income" of score at the end of every round, but the player must be able to keep them alive. This income balance must be well managed by the players since the game gets more and more difficult throughout the waves:

- Increased number of meteors every wave.
- Bombers start to appear after a few waves and adds another bomber every few waves.

- Meteors have increased mass which causes them to have a higher chance of splitting into smaller meteors.
- Meteors go very slightly faster at each wave as specified in the specification sheet.
- Number of child meteors from a meteor splitting also increases with the level

This increase in difficulty would automatically mean an increase in score since there are more meteors to deal with. We also have the idea of a score multiplier at the end of specified wave levels.
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 missiles, force fields and black holes.

3.1.5 Particles: Meteor and Player missile

Almost all the game objects inherit from the abstract `Particle` class. This class contains all the fields that game objects need like position, velobase and the force accumulator. The `Particle` super class also implements the `IDrawable` interface, which means all particle sub classes must include a display function. This design allows each class to define its own draw function and change it accordingly based on its properties. For example an explosion has increasing radius as it is drawn, or a missile should stop displaying after reaching its destination.

The particles falling are implemented in the Meteor class. They spawn from a random location at the top of the screen and, as specified in the specification sheet, they have a random initial velobase. This initial random velobase goes in both the x and y direction. The random y velobase makes some particles come down faster than others to make the game a bit more difficult and the x velobase makes it so the particles don't all just fall straight down. This allows a truly random meteor generation throughout the waves to have a fun and entertaining game.

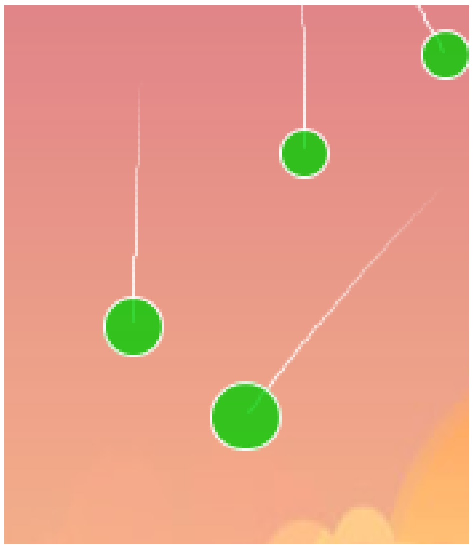


Figure 7: These lime Balls are meteors. We can see a trail following their paths

The meteors are also affected by a force of gravity and drag, implemented as `ForceGenerators`. `Gravity` is implemented just as a downwards force because it is much simpler compared to modelling the ground as Earth and doing gravitational attraction. The force of gravitational attraction is implemented, but not used for the "gravity" pushing the meteors down.
`Drag` is fully implemented following the formula with `k1` and `k2` constants of drag.

The player missiles fire from the center of the ground. Their velobase is normalised and not very fast so it takes a bit of planning to properly defend the bases that are further away. The missiles explode either when they reach their destination - the position of the mouse cursor when the missile was fired - or when they collide with a meteor during their trajectory.
To calculate where they have to go and when to explode, the missiles store their destination position vector which lets them calculate where to go and check if they are close to the destination before exploding.

3.1.6 Explosions

All particles can create explosions when they are destroyed as it is an abstract method all `Particle` sub-classes must implement. The explosion radius is based off of the particle's initial radius, increasing with each time step until they've reach the end of their lifespan. Any particles caught in the blast radius are destroyed.

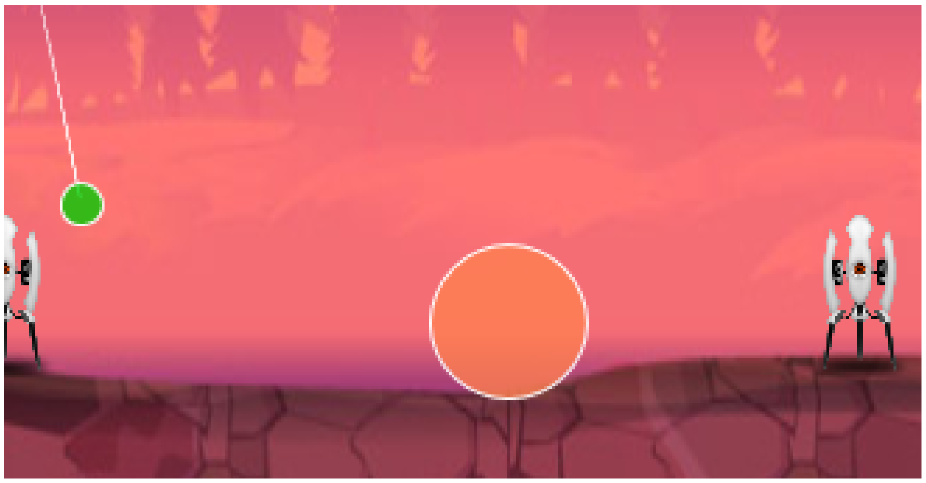


Figure 8: Normal explosions from player missiles.

Since all particles can explode, both player missile explosions and meteor explosions can destroy bases. However, players can purchase a permanent upgrade so missile explosions do not destroy the bases and Cannons. The Missile explosion then turns red after the upgrade is bought.

3.1.7 Physics and Collisions

Following the lectures and examples, I implemented the forces in physics as a `ForceGenerator`, with the game having a `ForceRegistry` to register all the particles and the forces acting on them. The registry and global forces are kept in the `PhysicsEngine`. Each new particle that is registered will register the two global gravity and drag forces.
I use this force handling to allow meteors to bounce on one another instead of just going through other meteors. It also allows me to create the Black Hole and Force Field items.

3.2 Extensions

3.2.1 Bombers

To make the game more difficult, bombers are also implemented in the game. These bombers fly across the screen and drop meteors that are much heavier than normal meteors. They are also given an initial downwards velobase to make them fly faster towards the player's bases. Bombers are not destructible since I found that the balance of the game was better that way.

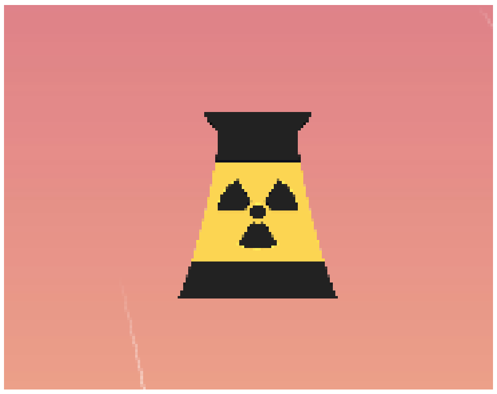


Figure 9: The Bomber

The bomber are represented by a nuke logo to stay in the game's vibe described in the introduction.

3.2.2 Split into child meteors

In addition to the bomber, the extension for splitting meteors into child meteors is also implemented. Meteors that are larger than a certain radius have a chance to split into 2 or more child meteors. As the player goes into higher levels, the number of child meteors that spawn is increased. Meteors only begin to split at after a certain level specified in the configuration class.

3.2.3 Shop: Black holes and Force Fields

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

- Black Holes.
- Force Fields.
- Extra Missiles.
- Base Rebuild.
- Friendly Missiles upgrade.

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

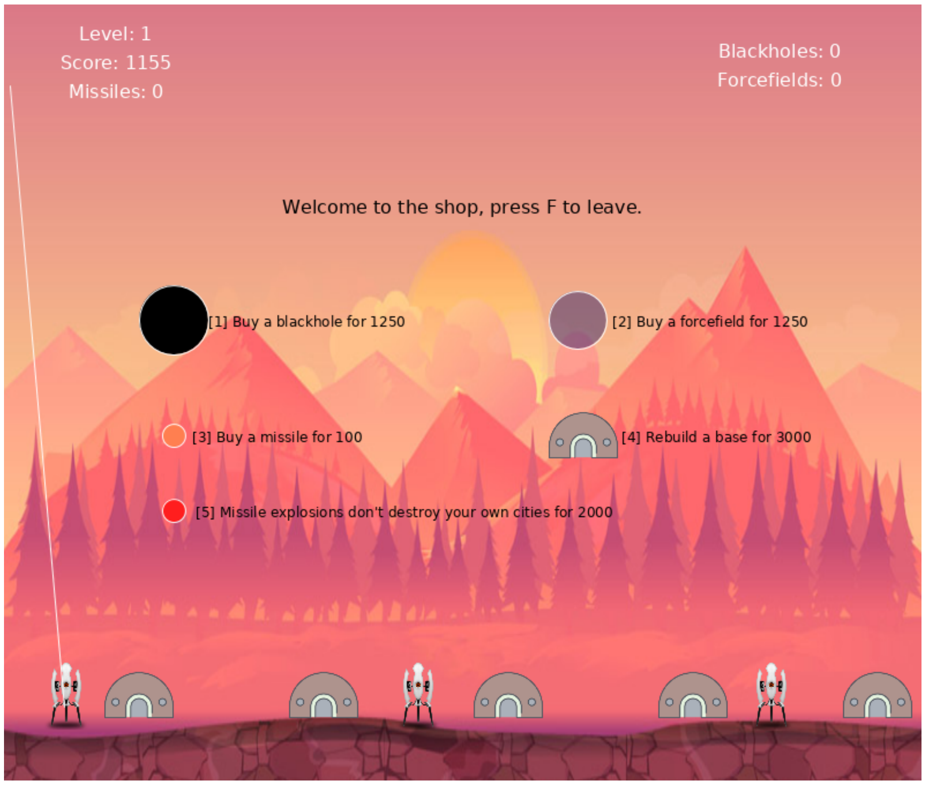
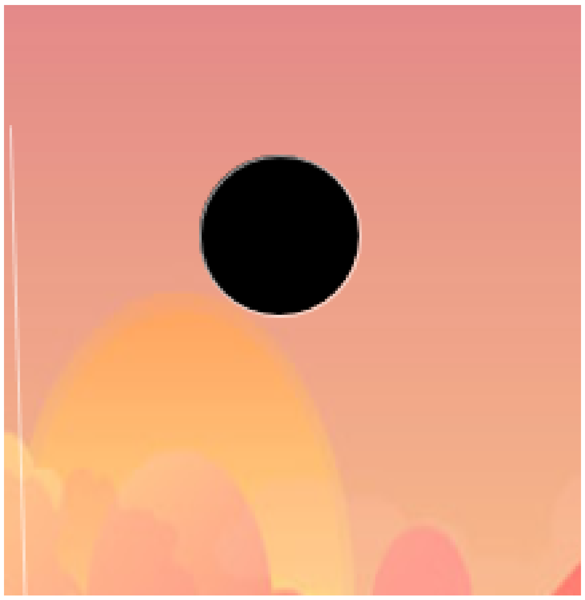

Level: 1
Score: 1155
Missiles: 0
Blackholes: 0
Forcefields: 0
Welcome to the shop, press F to leave.
[1] Buy a blackhole for 1250
[2] Buy a forcefield for 1250
[3] Buy a missile for 100
[4] Rebuild a base for 3000
[5] Missile explosions don't destroy your own cities for 2000

Figure 10: The shop Interface. The user can buy items here using the keypad.

The Black Hole is a special missile that player's can shoot. The black holes apply gravitational attraction to all meteors on screen and because they are intentionally much heavier than the meteors, most will be sucked into the black hole and destroyed. However, Further meteors might not be sucked in an their speed and trajectory towards the Player's bases would be modified and make the game harder. This is implemented to provide a bit of excitement and RNG. Indeed, if a meteor was being accelerated towards it but did not get sucked in, it will no longer have a force applied to it, but it will retain its fast velobase, only being slowed by drag.



Unlike explosions, force fields repel particles. This makes them a more global and powerful tool to defend against meteors compared to missiles. The meteors are repulsed away from the force field by multiplying the force of gravitational attraction by -1.

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 2 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.
- An explosion sound is played when An explosion is triggered.

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. The different features allow me to show off a few different forces and how they all interact with each other. The state machine allows each state to define its own functionality and again makes it easy to extend and add new states for different phases or parts of the game.
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!