Game Design Document: Functional Programming

Pieter de Marez Oyens, 5720508

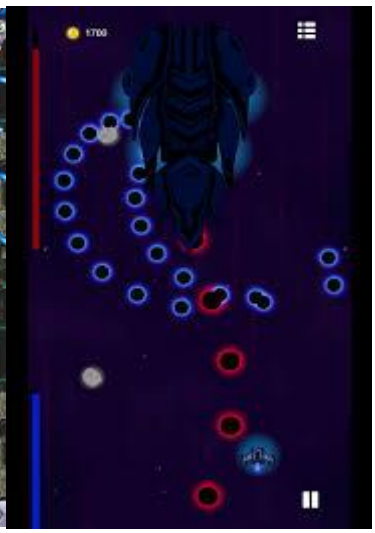
Bram Blaauwendraad, 5606918

Bullet = Spawn Bullet

Genre: Arcade
Type: Shoot ‘em Up/ Bullet-Hell/ Manic Shooters

Description of the Game

Our game will be inspired on the classic ‘Shoot ‘em Up’ games like *Space Invaders* and *DoDonPachi*. The player will control some sort of vehicle in a vertical screen, shooting bullet at enemies with at the end of each level a boss. Enemies will be shooting or flying toward the player, trying to do damage. To avoid the damage, the player should avoid the enemies and bullets by flying around the screen. See the pictures below to get a general idea of how such game look like (and here is a video to see a classic game in action: https://www.youtube.com/watch?v=CbPJhLpunXw).



Core Mechanics and Minimal Requirements:

Here we discuss how the core mechanics the game works.

The game plays on a vertical screen. It will has a black background colour and uses simple geometric shapes to display the player, enemies, bullets and particles.

Each object in the game will be represented by a gloss picture.
If the user is playing a level there will be the following objects:

- The player controlled sprite
- Enemy sprites
- Bullet sprites
- Particles
- Info like the player's lives, enemy wave number and score

A level will play in the following way:

The Player:

- Can destroy enemies by shooting bullets.
- Will control its sprite by using the WASD-keys, and able to move around in the 2d-space in four directions (north, south, west, east).
- Can shoot bullets by pressing down on the up arrow key. The rate of fire will be at a static rate about 2 bullets per second.
- Has a finite amount of lives. When the player loses all his lives, the player dies and the game is over.
- Can lose lives by being hit by an enemy bullet or directly by an enemy. When the player is hit, there will be a short period where the player is invulnerable (the player turns white when he is invulnerable).

The Enemies:

All enemies will spawn at the top of the screen and fly towards the player. Different enemies have different behaviour. An enemy cannot fly off the screen except the south side. Every enemy that flies of the south side of the screen is presumed "dead" and will be removed from the game. Every enemy except for a boss will fly off the screen at some point. Each enemy that dies at the hands of the player will show an explosion animation before disappearing.

- Basic enemy: enemies that just fly to the bottom of the screen in a straight line.
- Shooting enemy: enemies that can shoot bullets toward the player and fly downwards.

An enemy can also have a type of AI. The ones implemented are:

- No AI
- Chasing AI, where the enemy tries to follow and crash into the player by mimicking the player's x-position. This is done with some acceleration, so it takes some time to exactly be at the right position.

Bullets:

The player and enemies can shoot bullets. Bullets cannot interact or collide with anything except the player and enemies. Once a bullet disappears of the screen, it disappears from the game. When a bullet collides with an enemy or player, a life of either should be deducted and the bullet itself should disappear from the game. Enemies will not be able to shoot each other. Bullets have a set velocity which will be greater than the moving speed the player and enemies. Bullets can be shot in all directions in 2D space (so in 2π rad), however players and basic enemies can only shoot in straight line, while a boss may have shooting patterns that changes the direction of bullets.

Particles:

The game implement particles. Bullets and enemies can explode it different fashion and enemies can "collapse" (which kind of looks like an explosion still…). We implemented different kinds of explosions depending on whether something is a player bullet, enemy bullet, or enemy. Each explosion has its' own corresponding colour (the explosion is the same colour as the object that just exploded). Furthermore, a player bullet explosion is the biggest, follow by an enemy explosion and finally an enemy bullet explosion.
Each particle has a certain lifespan and disappears after this has ended.

Level Progression:

The level will start with the player and no enemies. The games progresses in waves. Whenever the player defeats all the enemies in a wave, or when all the enemies fly of the screen, a new wave is spawned. Which wave is spawned is chosen **randomly**, but it is more likely to spawn a harder waves after you have defeated more enemies. For the sake of demonstration, if the player has defeated 4 waves or more, a bunch of random enemies are spawned, to show that randomization and different enemy types are implemented. Since this game is not graded on game design etc. we thought this would make the grader's life a bit easier. (Instead of defeating 15 waves before you have seen every type of enemy).

Pause, Un-pause, High score, Reset:

A player can pause the level by pressing the 'p' key. This can only be done when the game is not over. Alternatively the player can also press 'h'. This will pause the game and show the highscores. Whenever the game is paused, 'h' can also be pressed to toggle between the "pause menu" and the "high score menu". Both will stop the basic game loop from updating. Pressing the 'p' key in both menus will un-pause the game.

Whenever the player loses all his lives, the game will be over, and a screen will be displayed. You can then press the 'r' key to reset the game and start playing again. Everything will be reset except for the high scores of course.

Interaction with the file system:

After the level is completed the player score will be calculated on how many enemies are destroyed by the player's bullets and how many lives are left. If the player has a high score (a score that is better than the 10 best scores so far) it will be saved in a text file. This is simply a small list of 10 numbers.

Description of Important Data types and Type Classes

Important Type Classes:

Ai

Describes behavioural functions for enemies

- Chase: when implemented should be a function that chases the player in a certain way.

BoundingBox

Describes the function

- getBox: to get the bounding box from an object.

Existance

Implements thee functions,

- Exist: to say that an object is alive
- Die: to make an object “dead” or not exist anymore. Mostly used to remove objects from the level.
- isAlive: give back a Boolean value saying whether or not an object still exists.

Moveable

Implements two function,

- Position: gives back the current position of an object
- Move: moves an object by giving it a certain velocity. This updates the objects position.

Renderable

Implements two functions,

- Render: says how an object should be displayed on the screen
- Shape: says what geometric shape an object should be. This function uses an object bounding box to determine the size of the geometric form.

Important Data Types:

Alive

Has two constructors: Alive | Dead. More descriptive than a Boolean, but serves the same purpose logically.

AiType

Has two constructors: None | Chase. Describes the type of AI and object has. None to say that an enemy doesn’t have “smart” behaviour. While Chase described that an object has player chasing behaviour. This way, we can create and differentiate diverse types of enemies by their AI type.

Box

Data constructors for an objects' bounding box. Can be either a Circle or Rectangle. The circle needs to be given a radius and the rectangle a width and height. This is used to check collisions and if objects go out of screen bounds. For convenience, this is also used to automatically make an objects sprite in the View. This may be incorrect use of the data, but since the game only used geometric shapes, it came quite in handy.

BulletType

Has two constructors: EnemyBullet | PlayerBullet. Used to describe a bullet. This is useful when checking collisions: a player cannot be hit by its own bullets and enemies cannot be hit be theirs. Also is used to give bullets different colours easily in the view function for bullets.

Bullet

Describes everything needed to know about a bullet in the game. Noteable about bullet: bulletLife. BulletLife is set to dead after it has hit an enemy for the player or enemy. This is done so that it cannot hit multiple targets.

EnemyType

Has two constructors: Basic | Shooting. Shooting enemies are different from basic enemies in that they are bigger (so easier to hit for the player) but can also shooting bullets. Basic enemies are smaller but cannot shoot however. This is also used to decide how to display enemies in the view functions for enemies.

Enemy

Described everything needed for enemies in the game. Noteable about enemy:

- enemyNextShot, enemyFireRate: help to let an enemyshoot in a controlled fashion. Next shot is used to determine when the enemy can shoot, and fireRate how often.
- enemyType, ai: both describe what an enemy can and cannot do. You can mix and match both to define new and interesting enemies.

LevelStatus

Has three constructors: Playing | Paused | GameOver. Hand datatypes for the game loop. Helps deciding what to draw and to update.

Level

Hold al the information in the level. This means whether the player in playing or that the game is paused/ game over. Furthermore hold all the data to play the game and what needs to be drawn. Some extra information such as the score, enemy wave to help with the game logic/functionality.

KeyStates

Has keystates for the following keys: w, a ,s ,d, up, mouseL(not used currently). They helps to let the player hold down a key and let the player move fluently throughout the game.

WhatToShow

Has four constructors: ShowLevel | ShowHighScores | ShowPauseMenu | ShowGameOver. Usefull for the view functions to choose what to display to the user.

WriteCheck

Has three constructors: Write | HaveToWrite | DontWrite. This is used to decide whether to write highscores to the highscores.txt file. This is more descriptive and less hassle than a Boolean value.

GameState

Hold the information for everything that is needed to play the game other than what is in a level. This is: the number generator, highscores, writing to the highscores file, the level itself, keystates and elapsed time (usefull for timing things in the game.

ParticleType

Has three constructors: BulletParticle | EnemyParticle | PlayerParticle. Usefull to discern particles this way for the view, to easily decide how to display a particle. Also usefull when deciding particle behaviour. For instance, we wanted different kinds of explosion for when an enemy, bullet or player.

Particle

Hold al information needed for a particle. Notable things:

- timeAlive, timeOfDeath: You can set how long you want a particle to be alive, and a time of death is calculated based on this. When the time of death has passed, a particle disappears. This way, you can have short/fleeting particles, or very long lasting ones.
- Particle Accelerations. Particles were implemented late in the game's life and we were more comfortable around the game and Haskell in general. So we wanted a nice acceleration. We were to late to implement it everywhere, but we could at least but in in particle movement. So now, acceleration is used.

Player

Has all things necessary to know about the player. Notable things:

- fireRate, nextFire: same system as the enemies have.
- invulnerableUpTo, invulnerable: Logic to let a player be invulnerable for some amount of time. Uses the same "time logic" used as when to fire next.