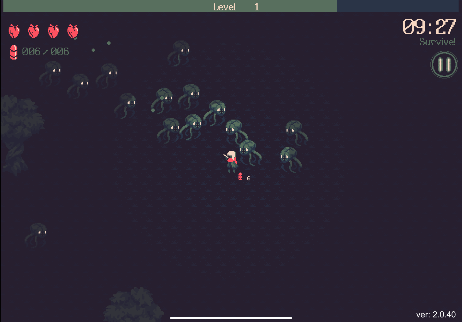
INFO 3225 Team Project

|  | Nathan Verma  100380776 |  |
| --- | --- | --- |

**Abstract**

One of my favorite genres of video games are rogue-lites. These games are run based meaning that once you die, you are put back to the beginning. The point is to try and get as far as you can in the run before you lose. My objective was to try and make my own rogue lite with some of the core aspects of the genre. I take inspiration from three rogue-lites in particular, these are; Vampire Survivors, 20 Minutes till Dawn, and Surivive.io.





My main goal was to implement a level up mechanic similar to these games but allow the user to choose which stats they want to increase. I also wanted to keep a similar camera view as these games, that being a high up bird’s eye view. The art style I was aiming for was also more playful and creative. I wanted to draw my own sprites making the games theme look like a sketch book. This would allow my sprites to blend right in and look as if they’re apart of the sketch. I wanted an arena like gameplay where the player is controlled and swarms of enemy’s spawn in all directions. Overall, I implemented many of the features I wanted. The game uses many concepts and mechanics such as; a full gameplay loop (Start - Play - End), finite states, search and sort algorithms, many loops and use of random numbers, many if, else, and switch statements, both static array and ArrayLists, push and pop stack structure and transformations, many classes some with grandparent - parent - child architecture, use of abstract classes and interfaces, many custom methods ,user keyboard and mouse interaction, and many UI elements.

**Target Audience**

The target audience is really quite broad. The casual nature as well as the easy to pick up controls allow it to be played by anyone. I would say that the true target audience would be those of a younger age, probably early teens and younger as they are the ones who game the most. They would also most likely appreciate the art style the most as it is more childish due to its scribbled and cartoonish drawings.

**Gameplay**

The overall flow of the game goes like this. The player spawns into the arena in the center where they are greeted with their first set of enemies. The player shoots projectiles at the enemies in order to defeat them all while doing their best to avoid getting hit by the enemies moving towards them. Once the player gains enough experience, they level up and are offered to choose 1 of 3 stats to increase. There are 6 stats in total that the player may get to upgrade randomly. The player continues to eliminate enemies. After the first set of enemies are defeated, more spawn in the next wave, this time being more difficult and plentiful due to the increasing difficulty. The player continues to get more powerful as they continue to level up and increase their stats. The enemies continue to spawn more and more each wave to combat the player. Once the player dies their score is recorded and they are given an option to play again.

**Design & Ease of Use**

I knew I wanted to make my own sprites for the game, but I am not much of an artist. My idea was to theme the game to resemble a sketch book, with a paper background and font that looks like it was written. This way I could draw my own enemies and they would blend into the art style as they would look like crude sketches on a page. The overall feel and usability of the game is fluid and seems to be easy to play, especially if you are familiar with video games. The game is also very re-playable as you are constantly trying to beat your best scores. The game should be able to run on Windows, MacOS and Linux thanks to the processing’s export feature.

**Screens**

There are 4 different UI screen that the user can navigate thoughA diagram of a computer process

Description automatically generated

*Main Screen* -this is the main menu that allows the user to either start a run, view the leaderboard, or quit.

*Leaderboard Screen* – I will go more in depth about how it works later, but this screen will show the leaderboard containing the player’s top 10 runs. The leaderboard is sorted using a sort algorithm specifically bubble sort. The leaderboard is saved into a .cvs file so even if the user closes the game and reopens it, their scores are still there.

A screenshot of a computer

Description automatically generated

*Game Screen* – I will also go more in depth about the mechanics after, but essentially this this is where the gameplay happens and a bulk of the code is used in, many updates regarding the players movements, the enemeies movements, collison and more happen here.

A screenshot of a video game

Description automatically generated

*Death Screen* –once the player dies by hitting 0 health, they are given the death screen which overlays overtop of the gameplay screen so the user can see how they died and get a final look of the scene. Here they are given options to start a new run, view the leaderboard, or return to menu.

*A screenshot of a video game

Description automatically generated*

**Classes & Heiharchy**

There are many classes in the program, some having much more code than others. There is also a main file called Simple Rogue Lite which does all the game scene upates and has the setup and draw methods. My structure goes like this:

*Entity Class (grandparent)* - the grandparent class for many of the games other classes this holds many of the key variables that all entities have such as coordinates, hitbox dimensions, health, speed, damage and more.

*Enemy, Player and Projectile (parent)* – the player class holds all the players stats as well as important methods. The enemy class holds values that all enmeis should have such as experience\_drop and score\_increase as well as the enemies movement update using a calcualted PVector that allows them to move towards the player. The players x and y coords are used in the direction PVector. The enemy moves a certain amont each frame towards the player depending on thier movment speed. The projetile class is used to mainly be a parent to the Player\_Projectile as well as to give me room to expand if I ever wanted to make more projectiles.

*Basic, Advanced, Speeder (child)* – these classes are different enemy types with different characteristics such as different health, speed, damage, experience drop, score drop and more. It allows me to add vareity by making unique enemeis.

*Boss (parent) and giant (child)* – the boss class is a parent class to the giant class, the giant is a boss type enemy.

*Effect (parent), Death effect and Hit Effect (childs)* – the Effect class is a parent class to both Death Effects and Hit Effects, this allowed me to add more effects if I wanted to

*Button* – a button class to allow me to easily make buttons for the UI so I can adjust a buttons, position, size, color, and give it an on hover change.

**Player Mechanics**

*Player interaction –* the player moves around the map using the WASD keys as well as the mouse to aim the player. Player speed is scaled depending on your speed stat. The player shoots by clicking or holding mouse 1. The player’s shooting speed is also dependent on the firerate stat. I use Boolean states to set the players movement so that they can be moving diagonally. Say the player is moving up and right, the current Boolean states are moving\_up = true and moving\_right = true, their position is then updated depending on their speed. These states are set back to false when the corresponding key is released. The player’s projectile is fired in the direction of the mouse cursor and the player’s sprite is also updated to always be facing the cursor as if they are aiming.

A screen shot of a computer code

Description automatically generated **A screenshot of a computer program

Description automatically generated**

*Player Stats -* there are 6 upgradeable stats in the game. These are max health, regen, speed, damage, firerate and projectile speed. The player on level up will allowed to upgrade a sinlge stat. They are given an option to choose between 3 stats that are radomly selected from the total 6. For example, on one level up I might get Damage, Speed, Firerate, and choose to upgrade Speed. On another I might get Health, Firerate, Regen, and choose to upgrade Regen. using the random\_stat() method, I have an array that holds the stats as strings such as “DAMAGE”, I then randomly shuffle the array and store them in a selected stats array which the user is then able to choose from using the select stat buttons on level up. The selected stat is then passed to the increase\_stat()method as a string in which the right stat is increased.

*Other Player Methods* – there are a variety of player methods in which I will go over some of the unique ones. The check\_regen() method is what allows the player to regenerate health depending on their regen stat, it has a timer to ensure the proper spacing between the regenration of health points. the check\_level\_up() method returens true if the player reaches the required experience amount for the next level. The increase\_stat() method is passed the chosen stat by the player to be upgraded. The reset\_player() method is used on each new game. The update() method is what allows the player to move by changing its positon depending on the moving boolean state and their current speed. The display\_stats()is what draws the UI, this includes the health bar, expericnec bar, the stats meter in the top left and the key legend in the top right

**Game States**

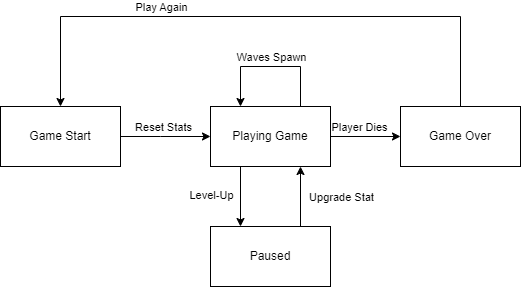
The game states are the following;

*Game Start* – resets the player, and game scene and starts first wave

*Playing Game* – the player plays the game by defeting enemies and completing waves

*Paused* – the player levels up and the game is paused to give them time to choose a stat. After a stat has been choosen the game unpauses. The player can also manualy pause if they wish

*Game Over* - the game ends on death and no more updates occur, player can choose to restart



**Collision Detection**

The way I check for if an enemy player or projectile hit one another, is by using collision detection. First I had to define hitboxes for each entitiy. This includes the player all enemies and projetciles. I do this in the Entity class by specifying hitbox\_length and hitbox\_width and giving it an offset so that its centered on the entity’s location. The logic used for the collision detection is the AABB collsion method. I am constantly checking for player, enemy and projectile collison in the game scene, and if the hitboxes intersect. Then collsion is returned true.

A screenshot of a computer code

Description automatically generated

**Game Updates**

The game is constatly updating due to many enemies, player movments, and collsions. In order for to properly update each aspect of the game I have multiple update methods and lines of code.

*Enemy updates* – there can be a mass amount of enemies on the screen at once, and updates must be done for all of them. Enemies are held in an object ArrayList called arena\_enemies. Here we loop throught the ArrayList and do updates for each enemy. These updates include their display, their movment towards the player, their collsion and death status. Inside the loop we check if the enemies have collided with any player projectiles which is held in the player\_projectiles ArrayList, we must loop through this ArrayList as well to check if an enemy has collided with any player projectile on the screen. If collsion is true then we proceed with actions such as taking damage, starting hit effects or marking for death. The player collsion against enemies also checked here. In order to avoid out of bound exceptions that I prevoiusly ran into when removing the enemy directly in the arena\_enemies loop, I mark the enemies who die by placing them in another ArrayList called enemies\_to\_remove. Once we move past the arena\_enemies loop we then remove the enemies from the arena\_enemies array by looping through the enemies\_to\_remove ArrayList as these enemies are marked for removal. We also give the player experience and score here depending on the enemey defeated as well as play start the death effect.

A screenshot of a computer program

Description automatically generated

*Projectile Updates* – the way we update the player projectils is similar to the enemies, by putting the objects in an ArrayList and looping through it. Here is where the projectiles display is updated and its location. The enemy collsion is all done in the arena\_enemeis loop so we don’t need to worry about that here, the only collsion that is looked for is with the screen width and height as if a border is hit, we want to remove those projectiles.

*Player Updates* – the player must be updated just like enemies. Here the player’s display is updated along with its location. It also has its unique updates such as regen so it can regenrate health based on regen rate and a timer. We also check for player level ups, as if the player reaches the required experience they should be prompted to upgrade a stat. the players UI is also updated here to show the user information about the player and run. we also check for player death here as well and save their score if needed. Effects are updated here as well.

**Leaderboard**

The leaderboard works by saving the players score after each death into the .cvs file. The program only allows the file to hold 10 rows maximum, and any other rows will be removed after the sorting process. If the file contains 11 rows due to the newly added player score after death. The program will use bubble sort to arrange each row into the right order. Now that the rows are in the right order, the bottom most row is removed, this being the 11th row. The file now is back to 10 rows. This ensures that the player’s scores are always the top 10 highest scores. There is also a search algorithm that will search each row and find the highest defeated enemies, which may be a different row from the highest score. The most defeated enemies are also displayed under the leaderboard after it has been identified.

A screenshot of a computer program

Description automatically generated

**Difficulty Scaling & Spawning**

Enemies spawn in waves, and the higher the difficulty the more enemies spawn each wave. Every 10 waves a boss also spawns to add some extra challenge and variety. The difficulty scaling is fairly simple and is meant to allow the player to stay powerful but also give them many enemies to shoot at as the game progresses. Difficulty is based off the player’s current level. The difficulty influences the spawn number of enemies by being a multiplier for when adding enemies to the arena\_enemies ArrayList. Enemies spawn randomly throughout the arena to always keep the player alert. The randomness of the enemy spawn helps add to the chaos of the game as before when I had enemies only spawn near the outside, they would clump up and be less appealing to fight.

**References**

McElroy, J. (2022, February 10). Vampire survivors might one day stand alongside Hades and the roguelike greats. Polygon. https://www.polygon.com/reviews/22927447/vampire-survivors-review-roguelike-bullet-hell-steam-pc-browser-8-bit-characters-power-ups

Madnani, M. (2022, September 20). *Vampire survivors-like 20 minutes till dawn has been updated to bring in controller support and offline play on IOS*. TouchArcade. https://toucharcade.com/2022/09/20/20-minutes-till-dawn-mobile-controller-support-offline-vampire-survivors-update/

Google. (n.d.). *Survivor.io - apps on Google Play*. Google Play. https://play.google.com/store/apps/details?id=com.dxx.firenow&hl=en\_US

Welcome to processing!. Processing. (n.d.). https://processing.org/

Cliffski. (n.d.). Crumpled looseleaf paper by Cliffski on DeviantArt. by Cliffski on DeviantArt. https://www.deviantart.com/cliffski/art/Crumpled-Looseleaf-Paper-105822312

Workman, K. (1AD). Collision detection on processing. Stack Overflow. https://stackoverflow.com/questions/36334442/collision-detection-on-processing

Ink, C. (2023, October 26). Piper ’s playroom font: Chequered Ink. fontspace. https://www.fontspace.com/piper-s-playroom-font-f105292

Pankaj. (2022, August 3). How to shuffle an array in Java. DigitalOcean. https://www.digitalocean.com/community/tutorials/shuffle-array-java

GfG. (2024, January 10). Java program for Bubble Sort. GeeksforGeeks. https://www.geeksforgeeks.org/java-program-for-bubble-sort/