MegaStorm Games

Ross McDade & Alasdair Hendry

Micro Mayhem

Games Design Document

Table of Contents

[Game Overview 2](#_Toc497823381)

[Game Proposal 2](#_Toc497823382)

[Summary 2](#_Toc497823383)

[Gameplay 2](#_Toc497823384)

[Navigation 2](#_Toc497823385)

[Influences 3](#_Toc497823386)

[Mood board 4](#_Toc497823387)

[Approaches 6](#_Toc497823388)

[Chosen Approach 6](#_Toc497823389)

[Rejected Approaches 6](#_Toc497823390)

[List of Resources 6](#_Toc497823391)

[Project Plan 6](#_Toc497823392)

[Milestones & Deliverables 6](#_Toc497823393)

[Requirements 6](#_Toc497823394)

[Functional 7](#_Toc497823395)

[Non-Functional 7](#_Toc497823396)

[Design 7](#_Toc497823397)

[Artwork 7](#_Toc497823398)

[Sketches/Concept Art 8](#_Toc497823399)

[Level Design 8](#_Toc497823400)

[Robopill 8](#_Toc497823401)

[Enemies 9](#_Toc497823402)

[Flowcharts 9](#_Toc497823403)

[Overall Game Flowchart 9](#_Toc497823404)

[Menu Flowchart 10](#_Toc497823405)

[Level Flowchart 11](#_Toc497823406)

[Flowboards 12](#_Toc497823407)

[UI Design 12](#_Toc497823408)

[Evolution & Infection Ratio 13](#_Toc497823409)

[Implementation 14](#_Toc497823410)

[Pseudocode 14](#_Toc497823411)

[Damage Pseudocode 14](#_Toc497823412)

[Points Pseudocode 14](#_Toc497823413)

[Player Pseudocode 15](#_Toc497823414)

[Enemy Pseudocode 15](#_Toc497823415)

[Friendly Cells Pseudocode 16](#_Toc497823416)

[Power-Ups/Weapons Pickups Pseudocode 16](#_Toc497823417)

[Use Case Diagrams 16](#_Toc497823418)

[Use Case Descriptions 16](#_Toc497823419)

[Test Plans 16](#_Toc497823420)

[Technical Testing 16](#_Toc497823421)

[White Box Testing 17](#_Toc497823422)

[User Testing 17](#_Toc497823423)

[Black Box Testing 17](#_Toc497823424)

# Game Overview

## Game Proposal

Title: Micro Mayhem

Platform: PC – Windows.

Genre: Twin Stick Shooter.

Theme/Setting: Inside the body of a human, levels may possibly take place around vital organs and such.

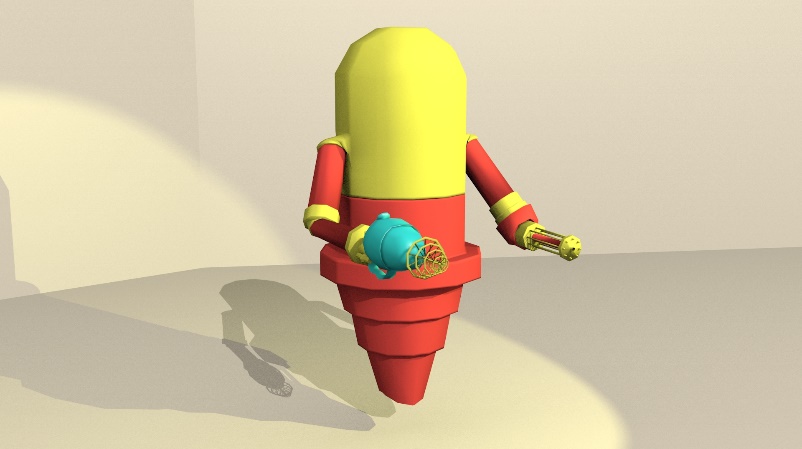
## Summary

“Micro Mayhem” puts the player in control of a technologically advanced form of medication, wittingly named “RoboPill”. RoboPill is a new form of medication that was able to be developed due to recent medical advancements. The player’s job, as RoboPill, is to enter the patients system, and destroy all harmful bacteria whilst saving as many healthy cells as possible. The player will progress through the game in a level-based fashion, earning evolution points at the end of each level depending on how well he was able to help the patient.

## Gameplay

### Navigation

The player will control RoboPill from an almost orthographic angled camera, whilst retaining full 3D perspective view.

As can be seen in the reference image, RoboPill’s body is section into two parts; Upper & Lower. This allows for more in-depth control of the character in regards to movement and aiming.

Using WASD or the arrow keys, the player will be able to manipulate RoboPill’s lower-body thruster to move him around the environment using a fixed axis; meaning they will thrust in the same direction regardless of the rotation of RoboPill’s upper-body.

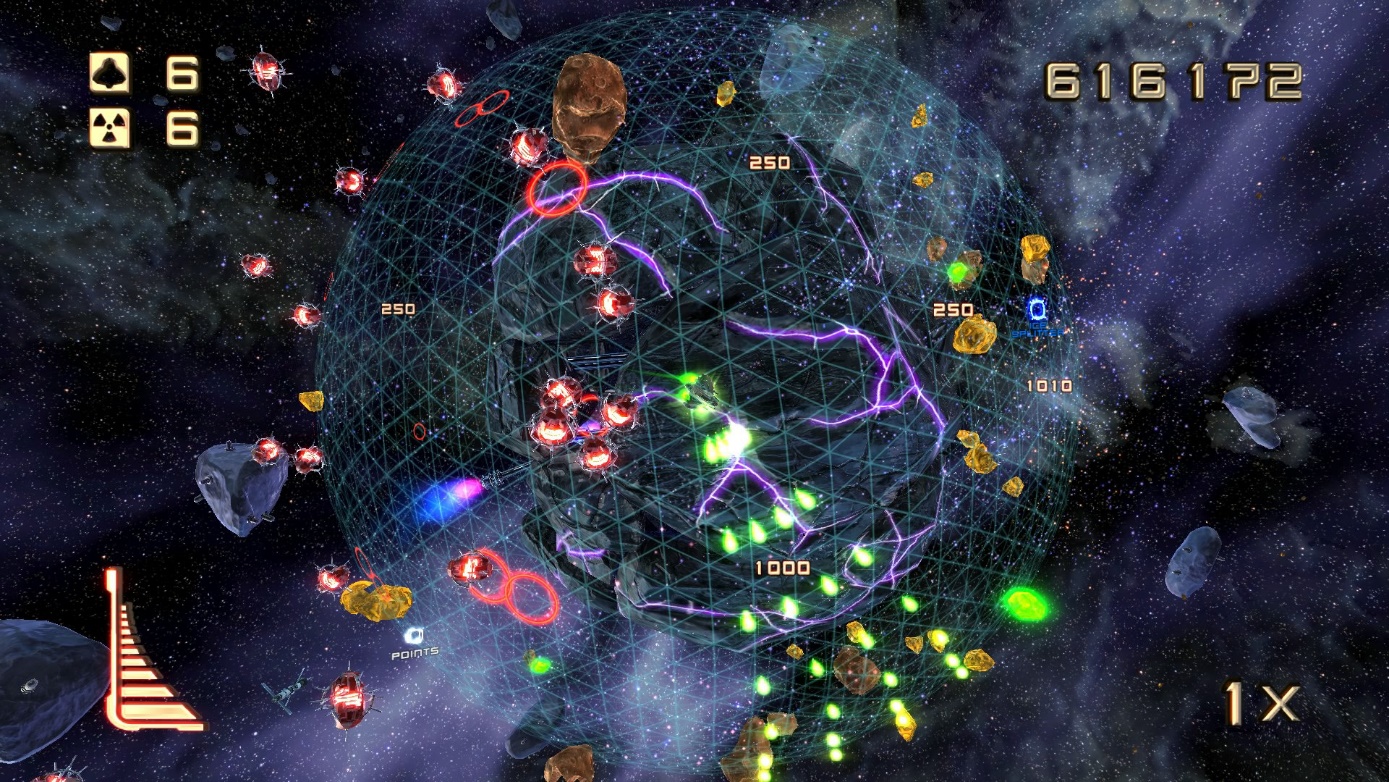
The mouse pointer will be used to rotate the upper-half of RoboPill’s body, giving a full 360 degree range of motion to fire weapons. The upper-half will be rotated depending on the angle between the mouse pointer and the character’s world position.

# Influences

The gameplay itself will take influence from other very successful twin stick shooters such as Assault Android Cactus, Helldivers and Super Stardust Ultra. Our game will stay true to its genre as it will be incredibly hectic and fast-paced, not giving the player time to think and allow them to be immersed in the action of each level.

The game’s art style has been influenced by things such as Super Stardust Ultra, Overcooked, and Hotline Miami where there is a lot going on the screen at once and the models and level will all be done in 3D. The game will also be influenced from these games in the sense that the more enemies the player kills in rapid succession, the more points they will be awarded. However the player will only have to worry about moving on one axis. Images can be found in the Mood Board, the Sketches and Concept Art sections as well as the Appendix Document to give a clearer idea on how the game will look.

# Mood board





# Approaches

## Chosen Approach

We decided to go with the Agile Software Development Life Cycle. It was agreed that due to the short amount of time we have to create the game this approach will work best for our team as it allows for lots of very quick iteration and is flexible when it comes to making changes to the project, this will work well as we have ideas for the project that can be implemented later if the time is available for us. It also fits very well with our test plans as we decided that we would test as we implement which Agile encourages. However, what we did choose to amend were the scrum meetings, instead we will merge them with our weekly minutes meetings and use that to address any and all problems while we are developing the project. Instead of releasing the project to any form of client or outside tester we will be handling the majority of it internally.

## Rejected Approaches

The Waterfall Approach was discussed as a possibility, but ultimately was rejected due to it not being as flexible for later implementation as it has a lengthy, yet rigorous, iterative nature. Had we been given access to a longer timescale to plan, design and develop the project we may have been able to use the Waterfall Approach. The V-Shaped Model had also been suggested, but it was decided that this approach was too similar to the Waterfall Model and we would face too many of the same problems.

# List of Resources

Resources that will be required in order to make the game, e.g. sound effects, models, textures, etc.

|  |  |  |
| --- | --- | --- |
| **Models** | **Textures** | **Sound Effects** |
| Player Character | Player Texture | Player Firing |
| Melee Enemy Type | Power-Ups Texture | Player Taking Damage |
| Martyr Enemy Type | Player Main Weapon Texture | Enemy Firing |
| Shooter Enemy Type | Melee Enemy Texture | Enemy Taking Damage |
| Brute Enemy Type | Martyr Enemy Texture | Friendly Firing |
| Level Environments | Shooter Enemy Texture | Friendly Taking Damage |
| Power-Ups | Brute Enemy Texture | Player Dying |
| Friendly Characters | Friendly Character Textures | Enemy Dying |
|  | Level Textures | Friendly Dying |
|  |  | Game Over: Fail |
|  |  | Game Over: Win |
|  |  | Menu Sounds |
|  |  | Menu Background Music |
|  |  | In-Game Music |
|  |  | Game Over Menu Music |

# Project Plan

## Milestones & Deliverables

Talk about Gantt chart and stuff. Link to Appendix document. Quickly list milestones and deliverables.

## Requirements

### Functional

Functional requirements are a collection of constraints that the game must have implemented, or be capable of, the fully function as specified in the design process. Listed below, these are split into technical requirements and in-game requirements.

#### Environments and Technical details

1. The game will be developed using the C# programming language.
2. Unity 2017.1 will be used as a development environment.
3. Blender v2.79 will be the modelling environment used to develop 3D assets.
4. Final build must run on any modern Windows PC.

#### Gameplay

1. The player’s Mini-Gun (Primary Weapon) will fire when the Right Mouse Button is pressed.
2. The player’s secondary weapon/ability will be fired/used when the Left Mouse Button is pressed.
3. Secondary weapons/abilities must be consumed when ammunition runs out.
4. RoboPill’s upper-body must rotate based off of the mouse position.
5. RoboPill must move on a fixed axis, ignoring the rotation of his upper-body.
6. All damageable items in the scene must be able to take damage and die.
7. Infection Meter must be updated each time a cell is destroyed.
8. Enemies must follow their assigned AI type.
9. Player must die when health runs out.
10. Explosions must cause damage in a defined radius.

### Non-Functional

Non-Functional Requirements are not necessarily required for the game to completely function, however, they may add a drop of immersion and aesthetic that may not have been present before-hand. Such things include high quality sound effects or fine-tuned global lighting. The requirements for this game have been listed below.

#### Audio and Visual

1. The game should include a diverse soundtrack of at least 3 musical pieces.
2. Sound effects should be use to notify the player when events have triggered (i.e. Enemies spawning, enemies dying, ammo running out).
3. Subtle global lighting should reflect the current health of the subject’s body, regarding how many healthy/unhealthy cells are currently alive.
4. Particle effects should be used frequently to display the carnage being caused.
5. Cartoon-like effects and textures should be used to fit the theme of the game.

#### Interface

1. User Interface should be kept to a minimum to avoid clutter.
2. Displays should be easy to navigate, and avoid showing unnecessary information.
3. A general theme should be followed to reduce player confusion.
4. Interfaces should scale with the player’s screen size to ensure consistent gameplay on different hardware.

# Design

## Artwork

### Sketches/Concept Art

Talk about what we wanted the game to look like. Link to actual sketches and drawings in appendix document.

#### Characters

When designing the characters that would be present in the game we had originally decided that we would model them around different types of bacteria. However, it was later decided that creating actual humanoid looking enemies gave the game more character and charm. These sketches can be found in the appendix document.

##### Robopill design

As you can see from the sketches of Robopill, in the early stages of the character’s design it was originally meant to have legs and run around to navigate the levels. After team meetings it was decided to use the second design as the pill was being marketed as the pill of the future and the name Robopill had already been chosen, the robotic jet booster for navigation seemed more fitting in terms of looking more robotic and more futuristic.

##### Friendly Cell Design

## Level Design approach

Each level will take place either inside or on top of an organ inside the subject’s body. Some may use the same organ as an environment, however, the following locations of interest should be unique to each level;

1. Spawn Points
   1. Enemy spawn points
   2. Friendly spawn points
2. Environment
   1. Interactive props
   2. Obstacles
3. Pick-ups
   1. Ability pick-up locations
   2. Secondary weapon pick-up locations

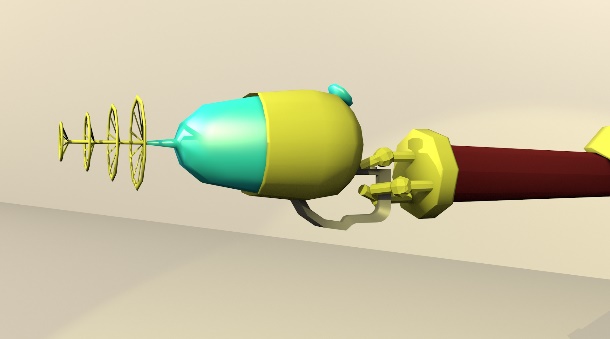
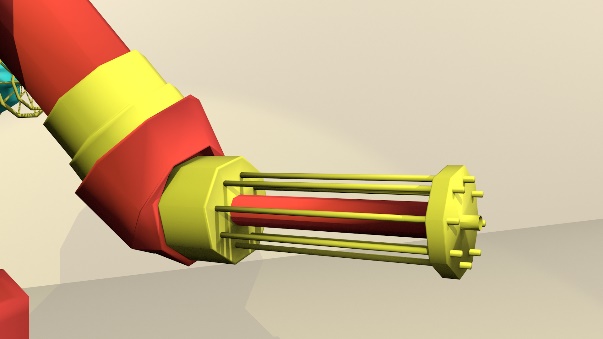
Depending on the environment, the player may be constrained to the bounds of the level. If the level takes place inside an organ, the organ walls should stop the player from straying too far from the action, however, as the player progresses through the game, environments may begin to take place on the outer surface of an organ, increasing complexity for the player as the issue of “falling off the edge” may become a factor.

## Robopill

RoboPill is the result of thousands of hours of medical research and micro-bacterial evolution. With the intended result being an extremely potent and multi-purpose medical capsule, RoboPill is able to enter a subject’s system and combat 90% of the world’s most fatal diseases.

RoboPill’s body is divided into two parts; the upper body, and the lower body. The lower body is made up of a rotational thruster, allowing for a full range of motion throughout a subject’s system. The upper part is a more combat ready segment, which can rotate around its centre in a 360 degree motion.

An improvement upon previous capsules, RoboPill’s left arm has been replaced by a mini-gun based weapon, which is always readily available. His right arm, however, features 3 opposable fingers which are able to grip secondary weapons. Shown below is a comparison between RoboPill’s Mini Gun arm and his wielding arm.



RoboPill's wielding arm, shown with The Cleanser equipped

RoboPill’s Built-In MiniGun

## Enemies

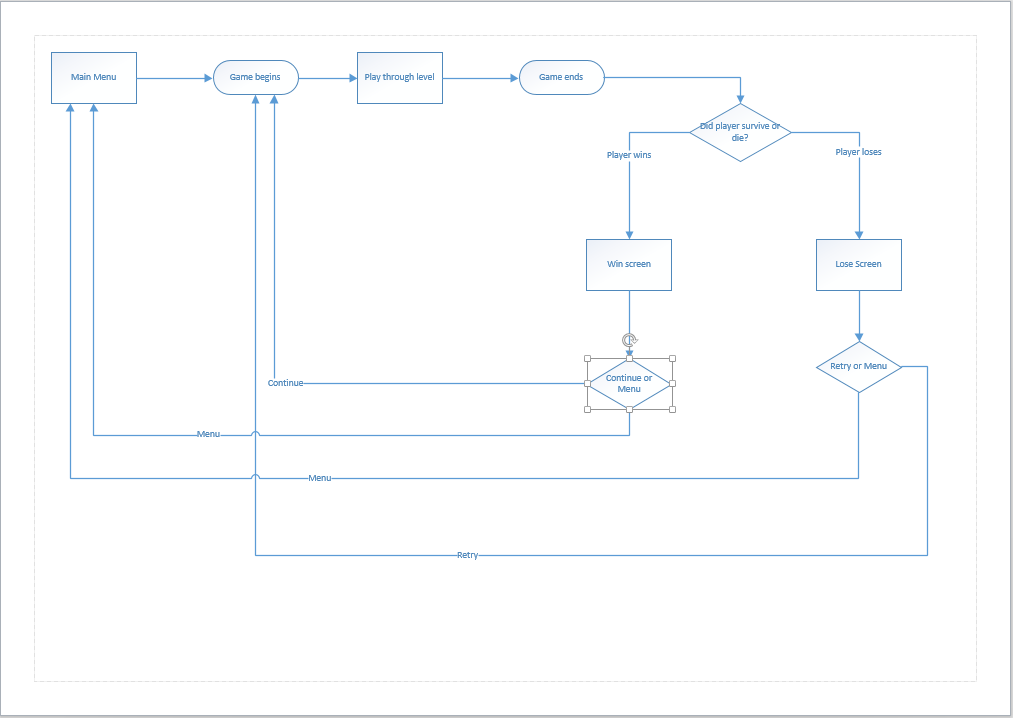
Enemies within the game will be mainly based on different variations of real-life bacteria that can be found. Each level, a new enemy will be unlocked, which the user will need to logically understand and defeat. Varying AI logic will be applied to each of the enemies, forcing the player to fight strategically whilst at the same time causing as much carnage as possible.

A wave-based spawning system will be utilised in the game, allowing many variations of enemies, adding replay value and unique gameplay which should further the complexity and immersion of the play-through. Although the wave-based spawning system will increase complexity and the need for multi-tasking, having a basic understanding of the game mechanics and enemy AI should allow the player to make it through each level.

## Flowcharts

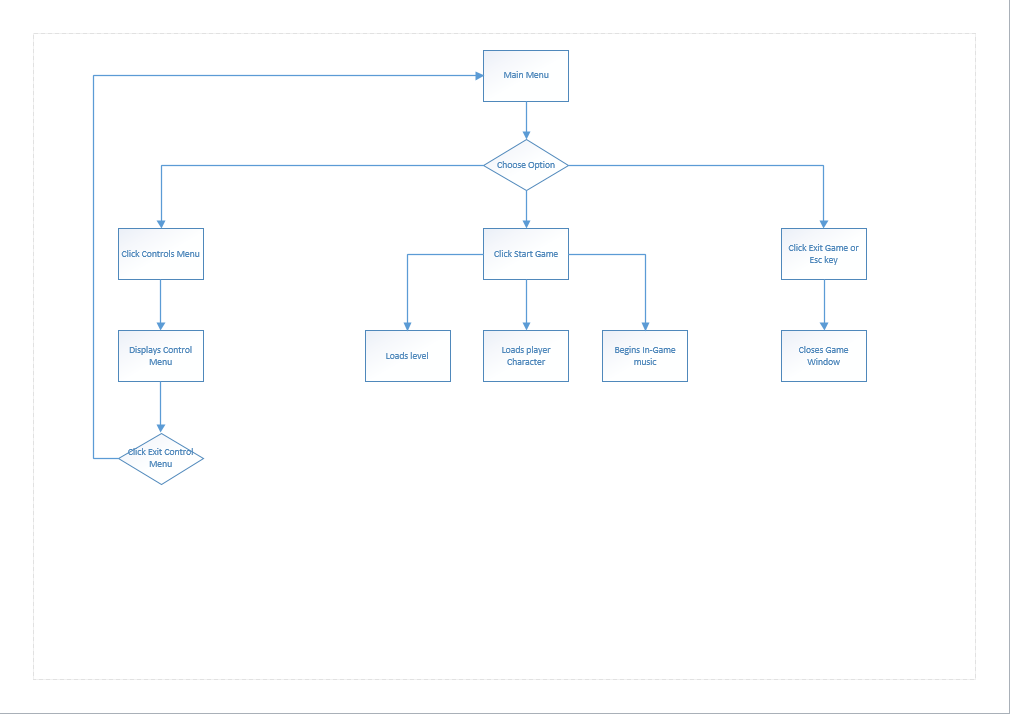
### Overall Game Flowchart

This flowchart shows a basic run-through of the game process from the menu to the player completing or failing a level.



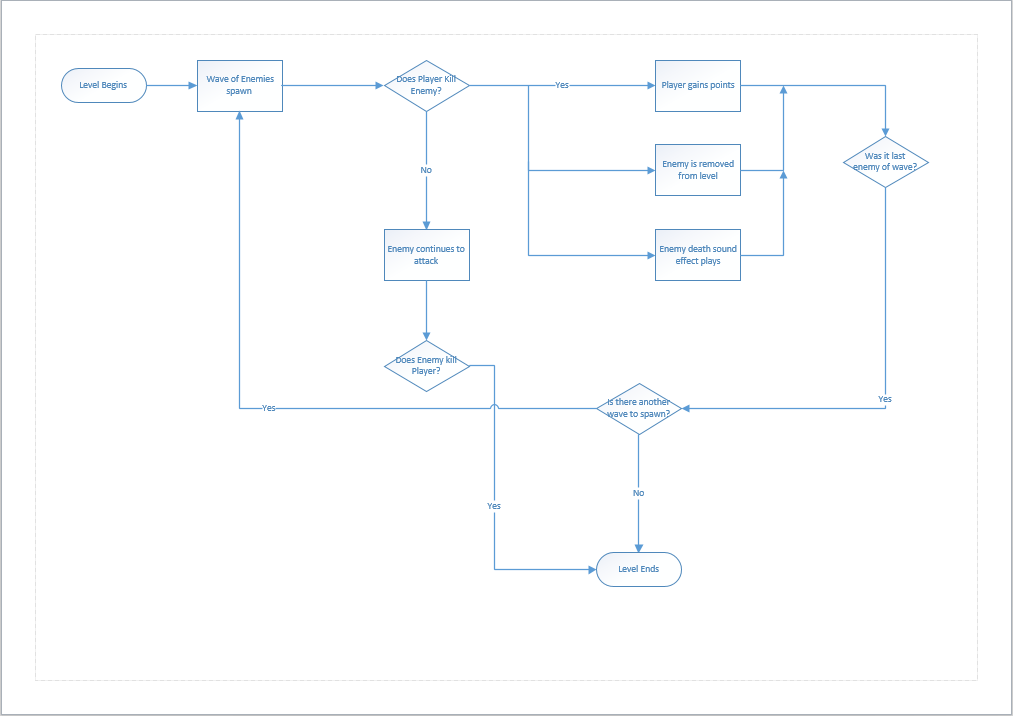
### Menu Flowchart

This flowchart gives a more detailed look at how the main menu will operate and what will occur when each option is selected.



### Level Flowchart

The basic level structure will run-through similar to this flowchart.

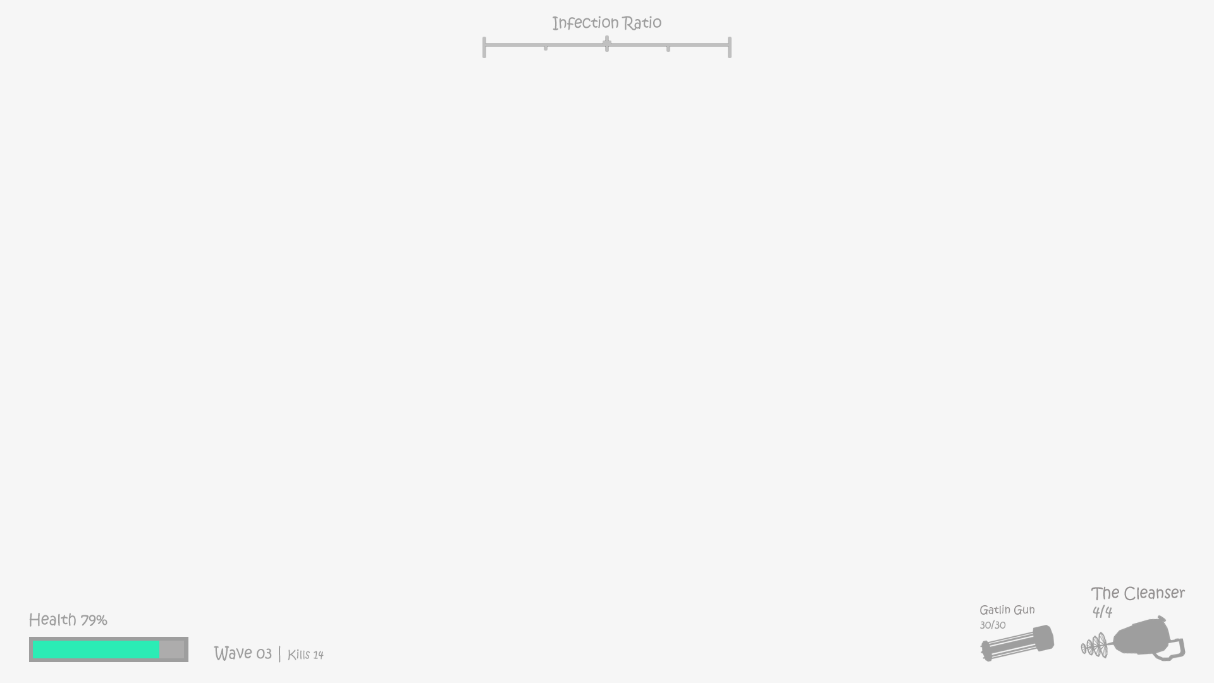


## Flowboards

Eventually add in flowboards.

## UI Design

Aiming for a sleek, minimalistic style, the UI Design will be bare-bones and will only show the player what they need to know. This avoids any clutter on the screen which can be off-putting and distracting, especially in such a high-paced game. Seen below, the user interface will consist of an indicator to show how well the player is doing, their current health and which weapon is equipped. As this is just a sketch, the in-game interface will likely be more aesthetically pleasing.



## Evolution & Infection Ratio

Throughout the game, the player will accumulate evolution points which can be used for progression. These points can be used to increase the effectiveness of the player, by increasing damage of certain weapons, unlocking new weapons and increasing the player’s defensive values.

Each level will provide a set value of evolution points to ensure the player is able to unlock certain features during their play-through. Furthermore, bonus evolution points will be provided depending on how well the player performed during the previous level.

The “Infection Meter” (That can be seen at the UI image above) is a GUI element used to display the ratio of good and bad cells that are present in the subject’s body. When the meter is closer to the left, the subject is in a stable condition, as more healthy cells are present in the body, however, if it travels to the right, this show the subject is unstable, and is near death. If the meter stays here for too long, the subject will die and the player will fail the level.

The status of the Infection Meter will also be relayed to the player using in-game global illumination, allowing the player to stay focused instead of staring at GUI elements.

At regular intervals, the status of the Infection Meter will be recorded, and used as an average throughout the level to calculate the player’s performance, and subsequently, how many bonus evolution points they are awarded.

Aiming for a sleek, minimalistic style, the UI Design will be bare-bones and will only show the player what they need to know. This avoids any clutter on the screen which can be off-putting and distracting, especially in such a high-paced game. Seen below, the user interface will consist of an indicator to show how well the player is doing, their current health and which weapon is equipped. As this is just a sketch, the in-game interface will likely be more aesthetically pleasing.

# Implementation

## Pseudocode

We decided to create some Pseudocode for the main functions of the game in order to help better understand what skills would be required to achieve our vision. It will also help outside parties to understand the source code along with the comments that are contained within the code itself. Please see ‘Pseudocode’ section of the appendices document for pseudocode that was created when planning for implementing additional features but was removed.

### Damage Pseudocode

#### Player Gets Hit

If player is hit by enemy attack  
 corresponding value of damage is removed from health bar   
else if value of damage is greater than value of health bar  
 player dies

#### Player Dies

If player dies   
 game is over

If game is over  
 game over screen pops up  
 gives player option to use regeneration ability if purchased  
 allows player to restart

#### Enemy Gets Hit

If player hits enemy with attack  
 enemy takes damage

If damage value is greater than enemy health  
 enemy dies  
 player gets points

### Points Pseudocode

#### Level Completion Points

If player completes level  
 Certain amount of points are awarded

#### Bonus Points

If player gets combos  
 points are awarded based on multipliers

Points are awarded based on the average position of infection meter

### Player Pseudocode

#### Player Movement

If w key is pressed   
 player character moves up to top of screen/back of map

If a key is pressed  
 player character moves to left of screen

If s key is pressed  
 player character moves to bottom of screen/front of map

If d key is pressed  
 player character moves to right of screen

#### Player Combat

If right mouse button is clicked  
 minigun fires bullets

If left mouse button is clicked  
 special weapon is fired/pickup ability is used

If r key is pressed   
 minigun will reload

#### Player Direction

If mouse is moved left   
 player character will rotate to the left

If mouse is moved to the right  
 player character will rotate to the right

### Enemy Pseudocode

#### Enemy Type 1(Melee) Behaviour

If player is not in the vicinity of enemy   
 charge towards player’s position  
else if player is in vicinity  
 use melee attack to cause damage to player

#### Enemy Type 2 (Martyr) Behaviour

If player is not in the vicinity of enemy   
 slowly make way towards player’s position  
else if player is in vicinity  
 explode to cause damage to player

If friendly cells have spawned  
 slowly make way to friendly cells’ positions  
else if friendly cells are in enemy vicinity  
 explode to cause damage

#### Enemy Type 3 (Shooter) Behaviour

If player is not in vicinity of enemy  
 begin firing bullets at player’s position  
else if player is in vicinity  
 retreat away from player and begin firing at player’s position once far enough away

#### Enemy Type 4 (Brute) Behaviour

If player is not in vicinity of enemy  
 slowly make way to player’s position  
else if player is in vicinity  
 use ground pound attack to cause damage

### Friendly Cells Pseudocode

#### Friendly Cells Behaviour

If no enemies in vicinity of friendly cell  
 remain idle  
else if enemies in vicinity of player character  
 attack closest enemies to player  
else if enemies in vicinity of friendly cell  
 attack closest enemies

### Power-Ups/Weapons Pickups Pseudocode

#### Picking Up power-Ups/Weapons

If f key is pressed near power-up/weapon  
 player picks up and stores power-up/weapon

If player has power-up/weapon stored and f is pressed over new power-up  
 power-up/weapon will not be switched

## Use Case Diagrams

Talk about UCDs and link to Appendix Document for diagrams themselves.

## Use Case Descriptions

Talk about Use Case Descriptions and link to Appendix Document

# Test Plans

## Technical Testing

### White Box Testing

White Box Testing shall be used for the majority of testing the game due to it being more rigorous and giving us a better idea from a coding viewpoint how the game is going.

#### Test As You Go

As the game is built the developers shall be testing as they go, using a ‘Test As You Go’ document that will look somewhat like the prototype below. In essence whenever the developer tests a section of code they will log each section they test and if there are any errors they shall document what the error was, how it was fixed and approx. how long the debugging time was.



#### Safe/Extreme Testing

The game shall also be tested after the implementation has come to an end using safe/extreme testing methods. Components of the game shall be testing within their limits and tested based on how they were designed to work, taking in the correct data. However, the game shall also be tested using extreme testing, pushing the game to its limits and even beyond the limits in order to gauge how robust the code is.

## User Testing

### Black Box Testing

Black Box Testing shall also be used as it is also very useful in giving us an idea from an outside perspective. For example, there may be bugs that we, the developers, notice a lot but a player may not notice at all. It shall also help in terms of UI and user experience as we will know how to play the game inside out and may forget to include tutorials or control instructions that a player may require. In order to get feedback we shall create a ‘User Testing Document’ like the one below.

#### User Testing Document Prototype

User name: **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

☐Component Testing/☐General Testing

If Component Testing, what was tested: **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Comments

|  |
| --- |
|  |

Observed by: **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Date: **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**