Game Proposal: Touch Of The Void

CPSC 427 – Video Game Programming

Team 10:

Adrian Lee 75841320

Alex Choi 43949767

Andrew Cheah 92349745

Celvin Koesasih 22195788

Nathan Dar 97147391

Shafquat ul Bari 91437913

Story:

The three most common elements in the known universe are hydrogen, helium, and oxygen. In the unknown universe: hydrogen, helium, and hellspawn!

Touch of the Void is a procedurally generated sci-fi dungeon crawler. On each run, you play as a new space pilot snatched and coerced by a demonic entity held captive in a torturous experiment. The player is led to believe that they are fighting to free a trapped prisoner. However, in actuality, the player is being led by the said demonic entity to take its place in the unending trap and thus unknowingly, you fight to free the demon by trapping yourself. Overall, the implication is that the demonic entity enticing you is no one else but the former player (from a previous run), broken by the experience, now setting up the same trap that they once fell into.

The game itself consists of the player making their way through an abandoned space station called Planck-4. The player will navigate through the defunct mega-structure infested with dangers leftover from the science experiments gone awry and must demonstrate their skill with their spacecraft to survive. Ranging from infested eyeball drones to amorphous blobs of flesh to more stereo-typical time-delayed laser traps, the player must navigate through increasingly difficult and chaotic rooms of enemies and obstacles.

The player will fight against these enemies using a variety of weapons. All of the weapons used by the player will have a finite amount of ammo available with the exclusion of the player's first weapon, their ship's Gatling gun, which will have infinite ammo reserves.

All of the player's and enemies ' weapons will vary in accuracy, range, speed (fire rate and reloading), and effect. For example, we plan to include melee weapons (ex. overcharged shields, ramming attacks, blades), and weapons with short, medium, and long ranges (ex. shotgun blast, vs laser bolts). As well as ammunition that falls with gravity, or explodes on impact (ex. bombs), ammunition that is self-propelled and aimed (homing missiles), or disables the player's ability to fire or move when hit. The enemies that use these differing ammunitions will each behave as you may expect (i.e. a melee enemy tries to stay close to the player, while a ranged keeps its distance). Later in this document, we will clarify our priority for implementing the different enemy and weapon types.

The player will be able to collect ammunition for these weapons once first finding them natively within the structure, after which they can purchase additional ammo from ammo vending machines that have the chance to spawn during a level. This should force players to conserve ammunition and be smart about their engagements, it may also encourage certain weapon's ammunition to be more coveted.

Occasionally, a "cleared" room (one free of hostiles) may spawn an upgrade chest upon "clearing", where the player will be given a new weapon, or separately, to upgrade their character using technology found on the station from a choice of three random options at a time. At first, these upgrades will be mundane (i.e. "faulty calculator" for boosting multiplier rates, "tampered shield capacitor" to temporarily turn your shield into a melee weapon), but as the player collects more upgrades they will gain access to more invasive yet game-changing upgrades (i.e. "implanted cranial padding" increases acceleration factor, or "supplementary appendage" allowing for faster reloading); hence, the names of the upgrades are themed to attempt to force the player to feel as if they are increasingly changing into something inhuman to gain strength.

Our game has two distinct levels with their own visual styles, in each level after a pre-determined amount of rooms have been cleared, a value meant to scale with difficulty, a boss level may be spawned. After a boss level has been spawned, the map generation will become capped and eventually no more new rooms will spawn, forcing the player to progress. A boss level when entered will be a point of no-return. And there the player fights a more difficult fight against a larger enemy.

In the first level, the environment will be consistent with the standard sci-fi space station, although empty and perhaps a bit rusty. After the first level's boss is beaten, you progress onwards deeper into a more infested part of the station, where all recognition of being inside of a space station vanishes. Beyond the aesthetic change and wider enemy variation within the second level, the gameplay loop is largely the same as the first. After reaching and beating the second level's boss, you have beaten the game and may restart. However, if you manage to make your way to the end of the game without sacrificing your humanity (accepting no upgrades), a secret ending in which you escape the trap and end the cycle is unlocked, and as you do not become the villain at the end, on your next run you are no longer allowed to play the game.

Scenes:

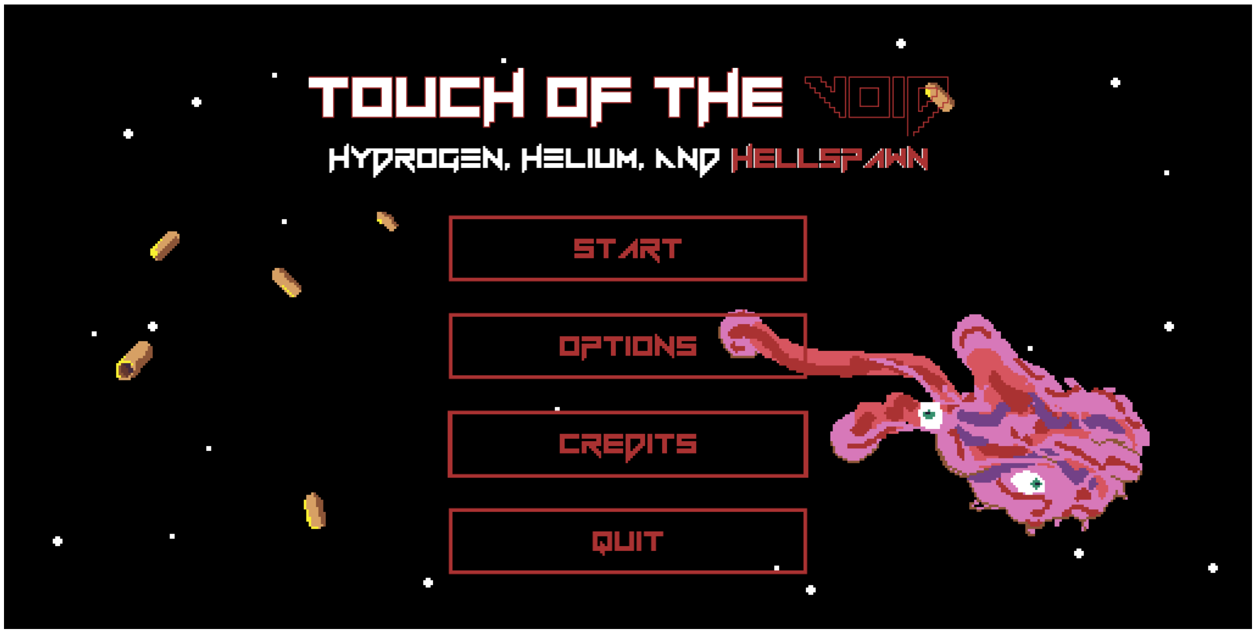

Figure 1: Start/Menu Screen

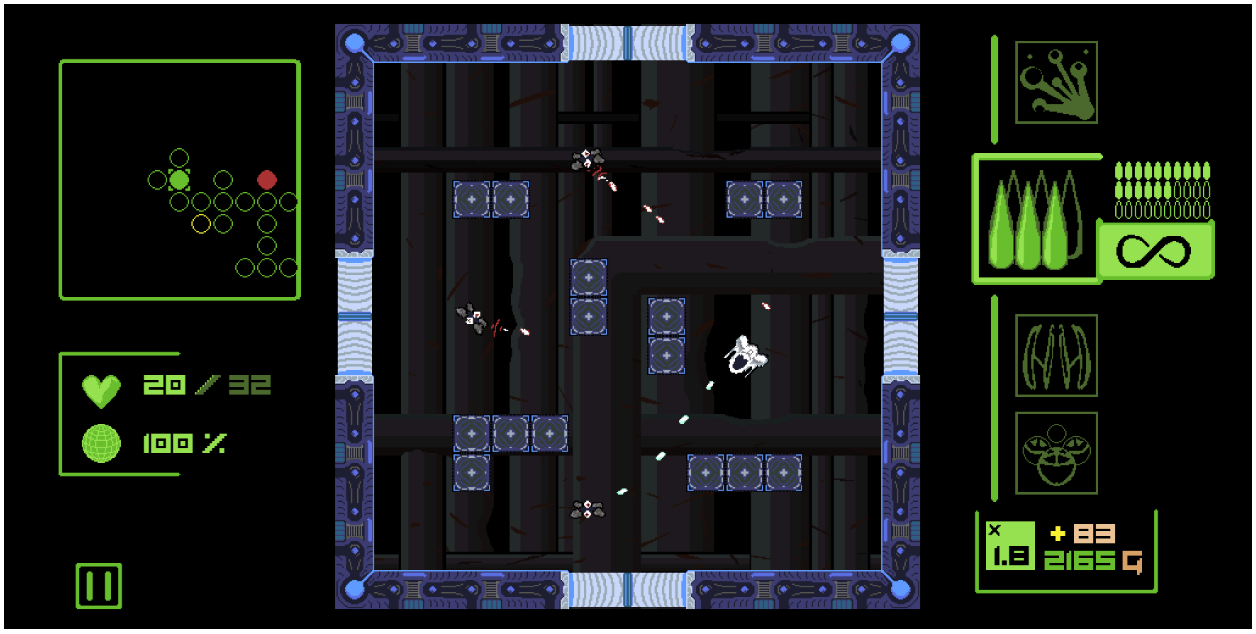

Figure 2: Level 1 Gameplay Screen

Our main gameplay screen. In the centre is the main game window, in which you can see a 4-door room with floating obstacles and the player engaging in combat with some simple enemies.

On the right, you can see the currently equipped weapon and its associated ammo (ammo left before needing to reload, and total ammo reserves). This information is displayed as part of a scrolling select wheel, demonstrating the other weapon choices above and below.

Further below the weapon bar is the gold counter. This section displays the current gold multiplier, the total sum, and any amount currently being added.

On the top left of the screen is the map. In which the current room is displayed with a solid green circle whose bounding box's edges have been bordered in green. Already cleared rooms are shown by a black circle bordered with green. Rooms that contain upgrades/ammo interactives are shown by a black circle border with yellow. Boss rooms are shown in red. Overall the legend follows the rules that normal rooms are green, pickup rooms are yellow, and boss rooms are red; as well any room previously explored (meaning no enemies remain inside) is only coloured along the border.

Below the map, is the player's status HUD showing their current and max health. As well as the percentage of their shield that is charged.

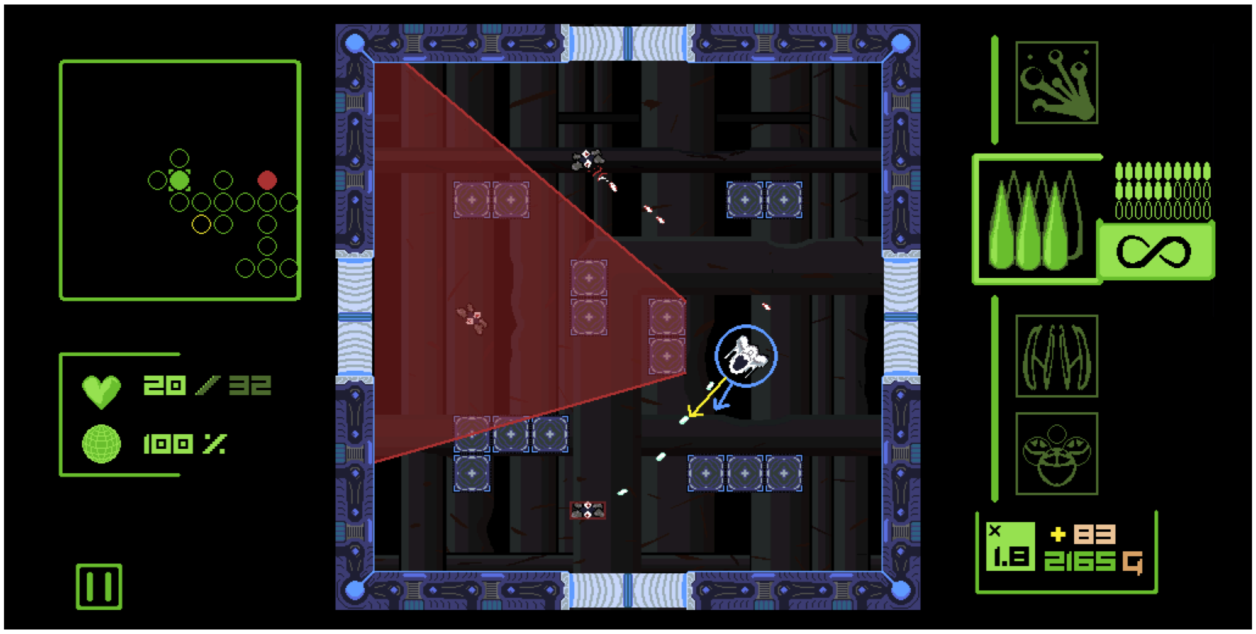

Figure 3: Gameplay Screen (Player POV)

In the above image, you can see the Player circled in blue with their direction vector also pointed in blue. The yellow arrow indicates the direction that they are aiming (i.e. firing). Please note that the direction the player moves is independent from the direction they are aiming. As well the red portion shows how the floating obstacle is blocking the player from firing upon the enemy within.

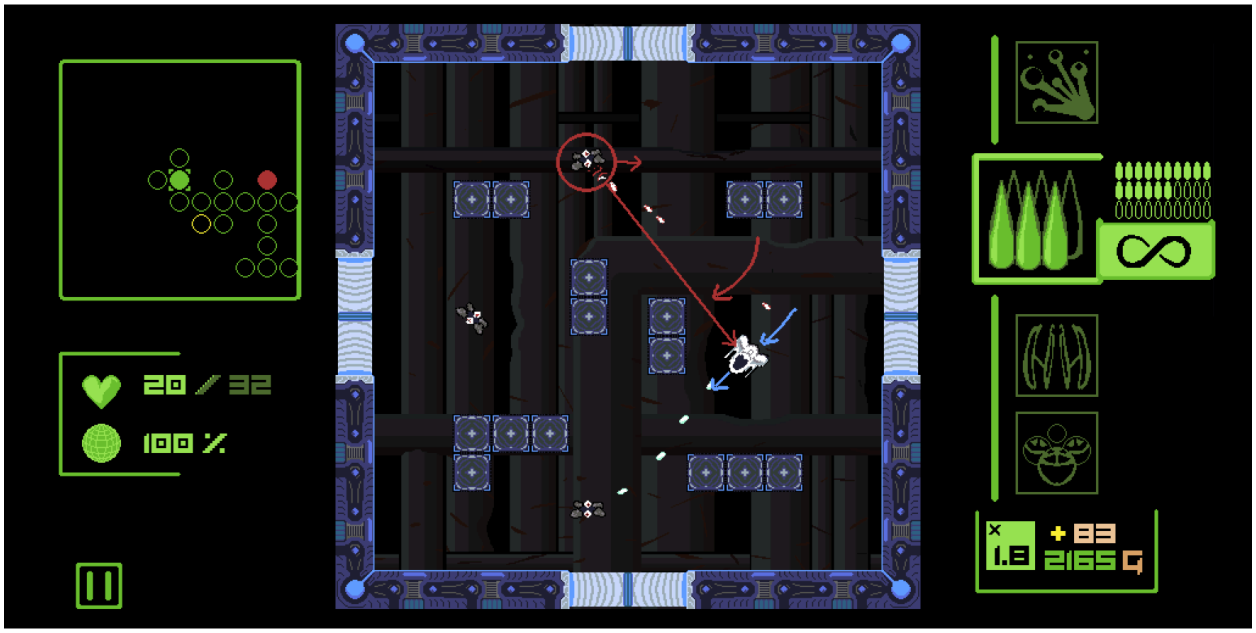

Figure 4: Gameplay Screen (Enemy POV can see Player)

Similar to the above but from the enemy's perspective. As you can see the enemy is moving to the right to keep visual on the player (short red arrow), as well as is firing upon the player and rotating their aim (long red arrow) with the player's movement.

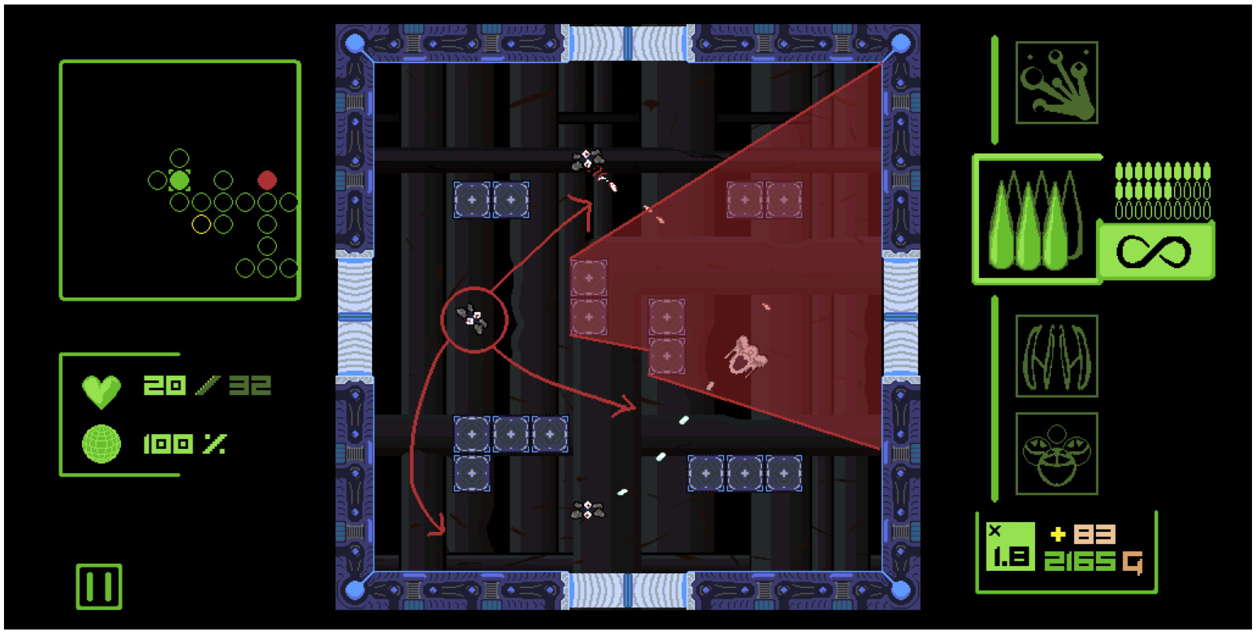

Figure 5: Gameplay Screen (Enemy POV cannot see Player)

Here the enemy cannot see the player and must move to get a visual before firing.

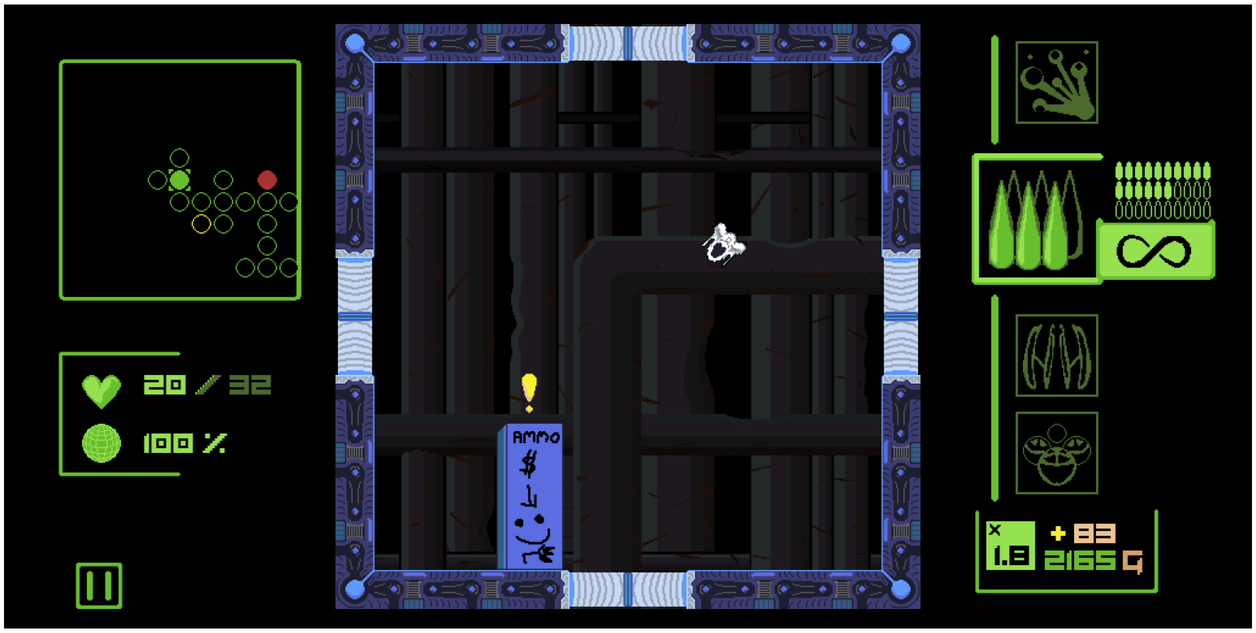

Figure 6: Pre-ammo Buy Screen

The player has found an ammo vending machine and may approach and enter the menu by accepting a button press prompt.

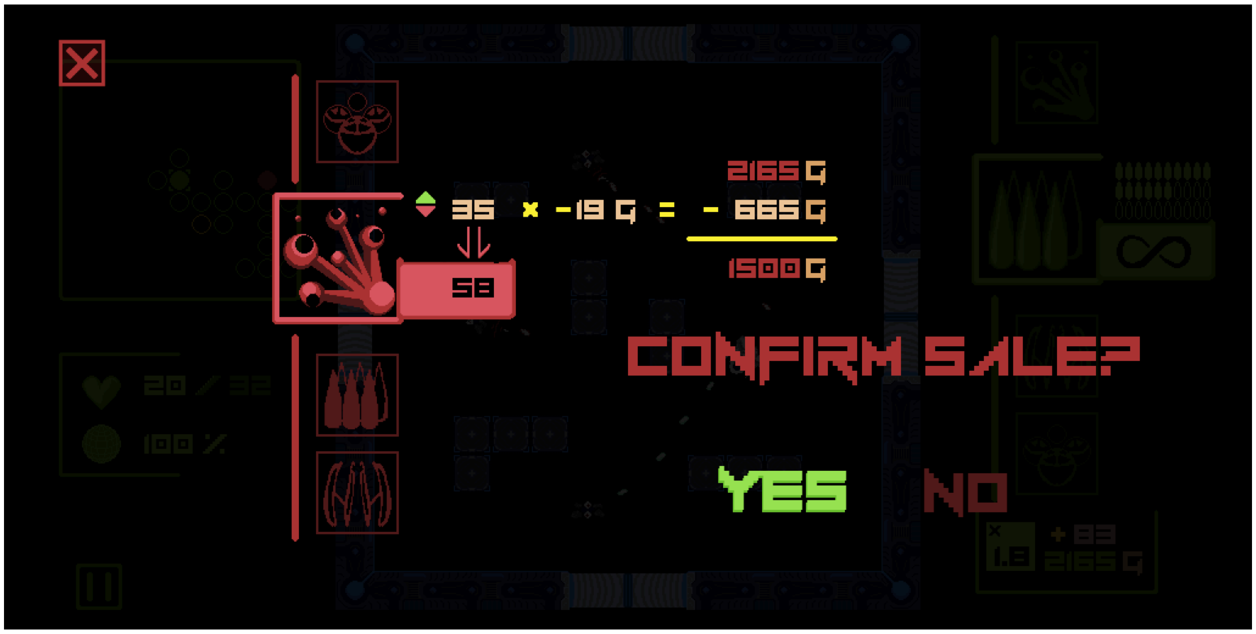

Figure 7: Ammo Vendor Screen

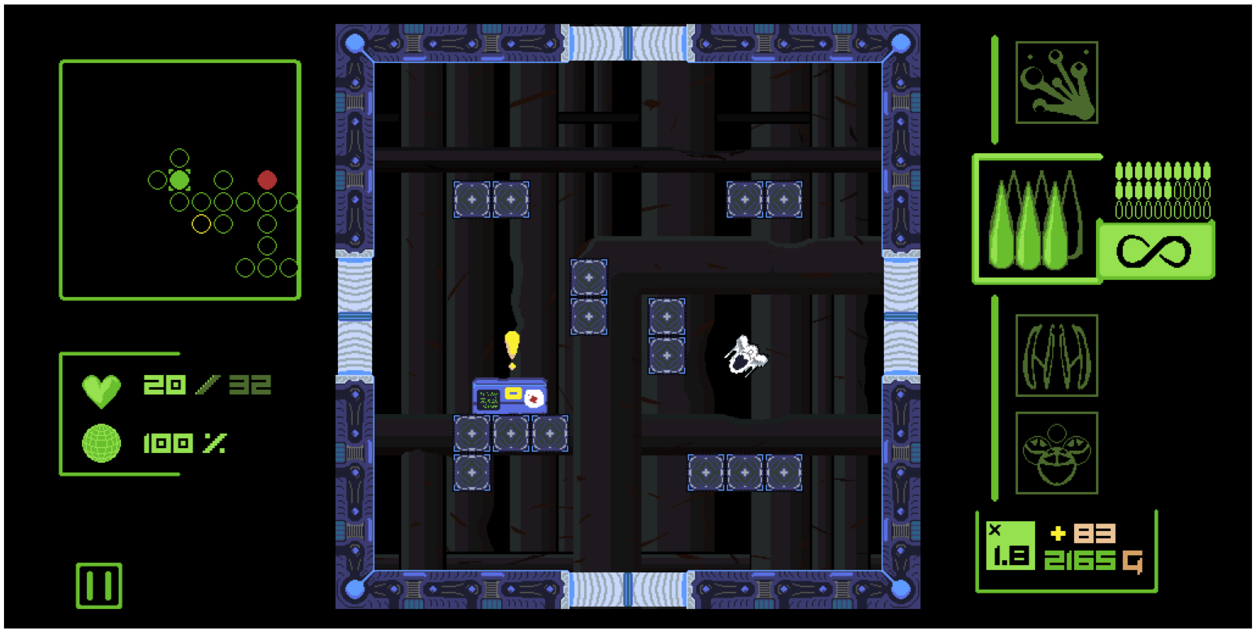

Figure 8: Pre-upgrade Pickup Screen

The player has found an upgrade/loot chest and may approach and enter the menu by accepting a button press prompt.

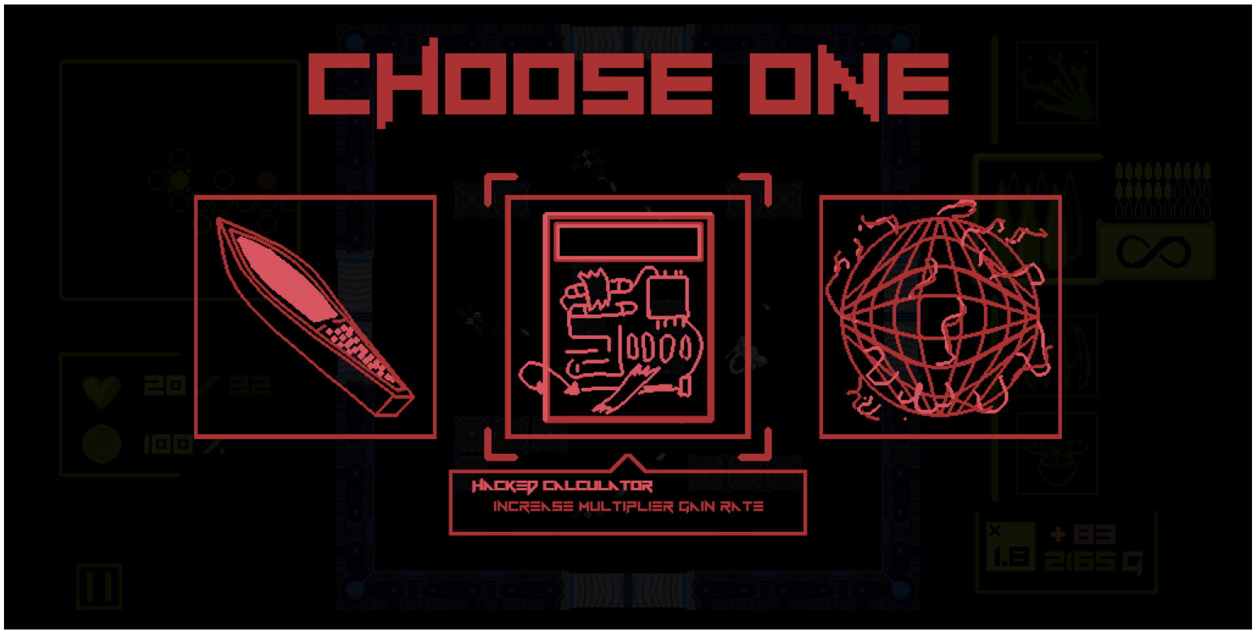

Figure 9: Choose Upgrade Screen

The player is picking from one of three possible options (which options are offered will be randomized)

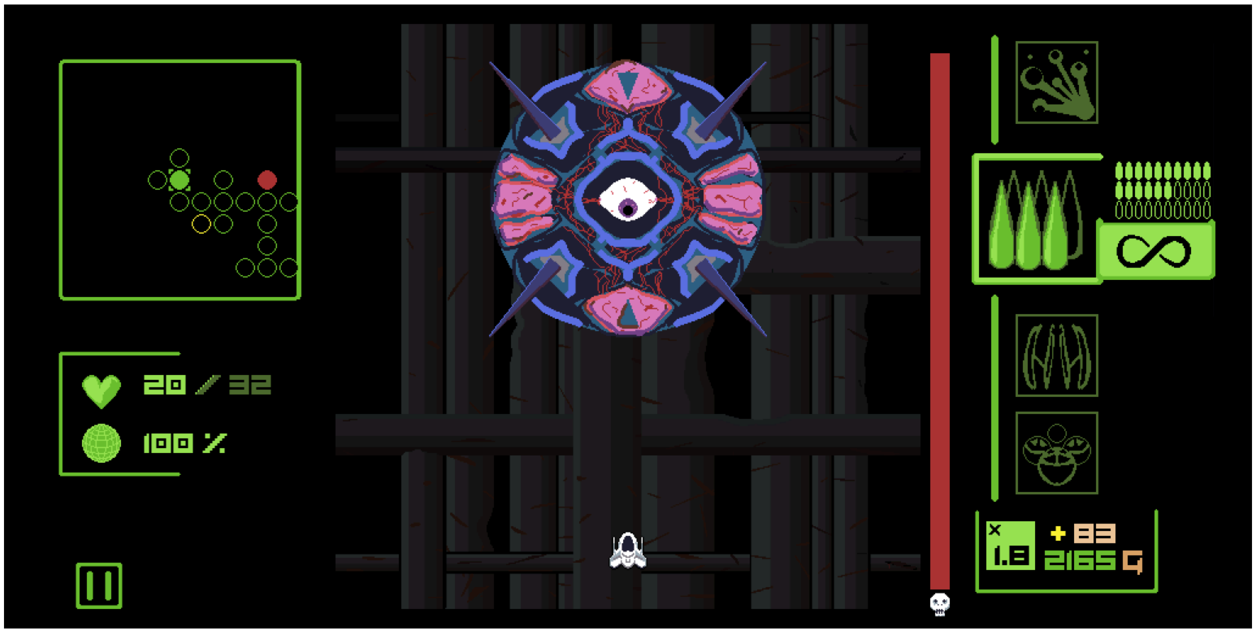

Figure 10: Level 1 Boss Fight Scene

The player has entered a boss room and a boss has appeared. Additionally, the monster's health bar has appeared on the right of the game window.

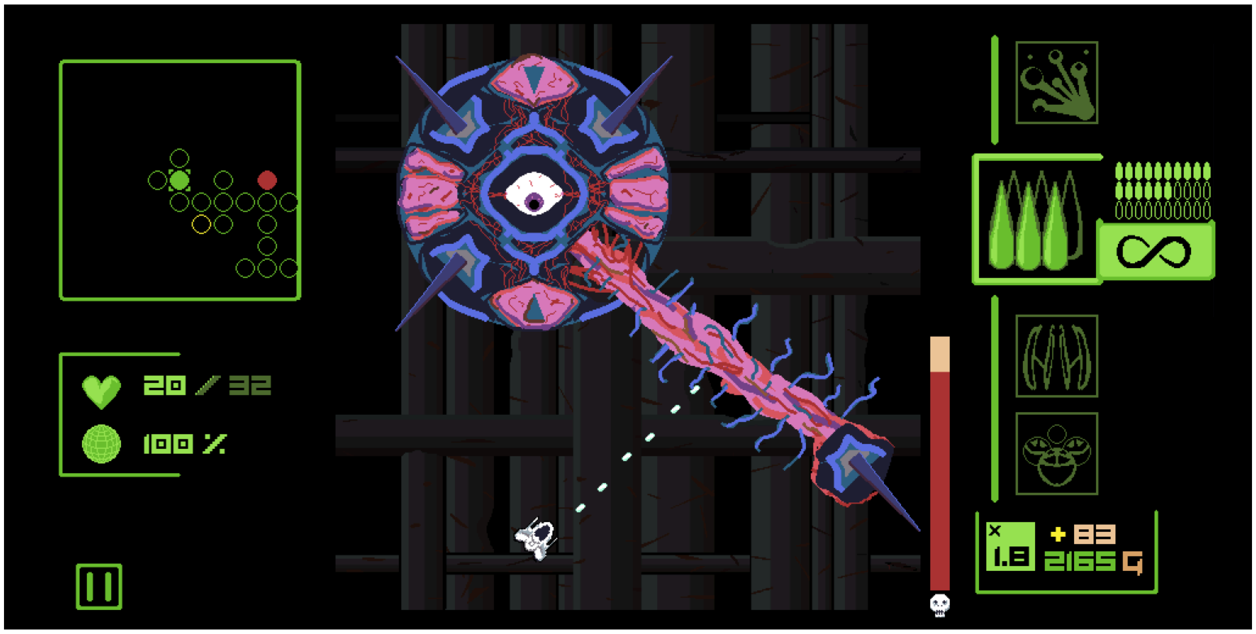

Figure 11: Boss Fight Action Scene

The boss has begun attacking the player, as both the boss and player are flying about the screen and attacking with their move-set. The player is firing at the boss's vulnerable points and damaging them.

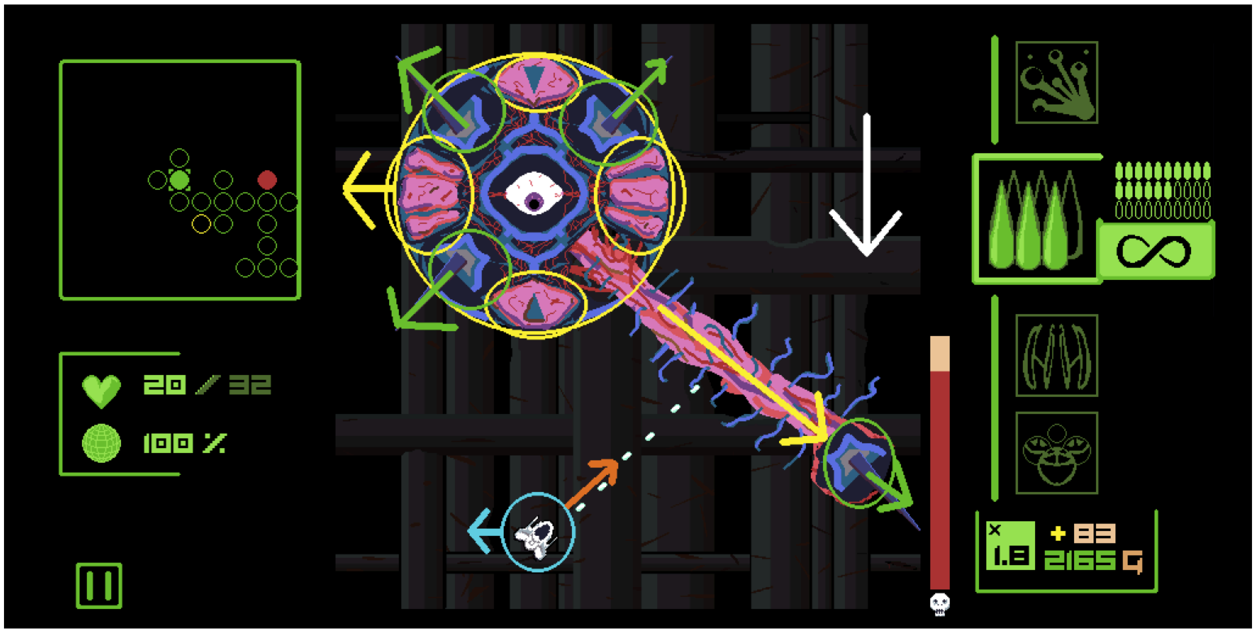

Figure 12: Boss Fight Explain Scene

Here the green circles and arrows represent the areas and direction the pre-determined attack moves can come from. The yellow circles represent the vulnerable areas of the boss, where being attacked will subtract from their HP. The yellow arrow indicates the movement of the boss during this frame. As well, the white arrow indicates that the background would be parallax scrolling to give a sense of constant movement. The blue arrow indicates the player's movement, and the orange arrow is the direction of their fire. Note: please ignore the map as it is inaccurate to where the player is.

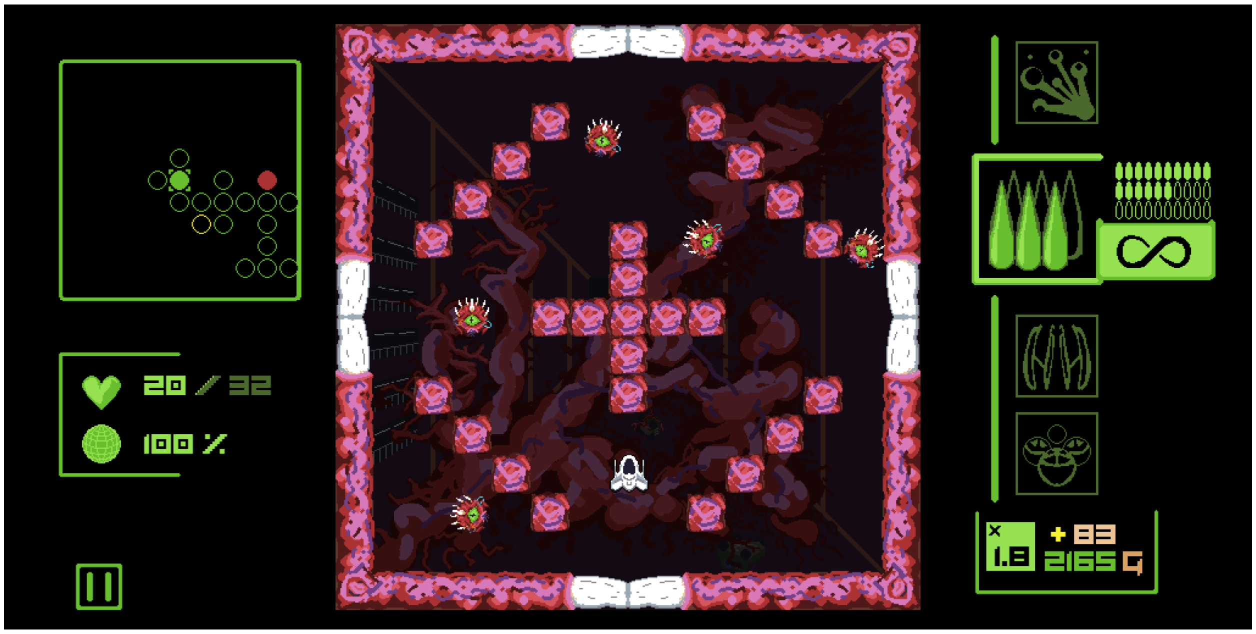

Figure 13: Level 2 Gameplay Scene

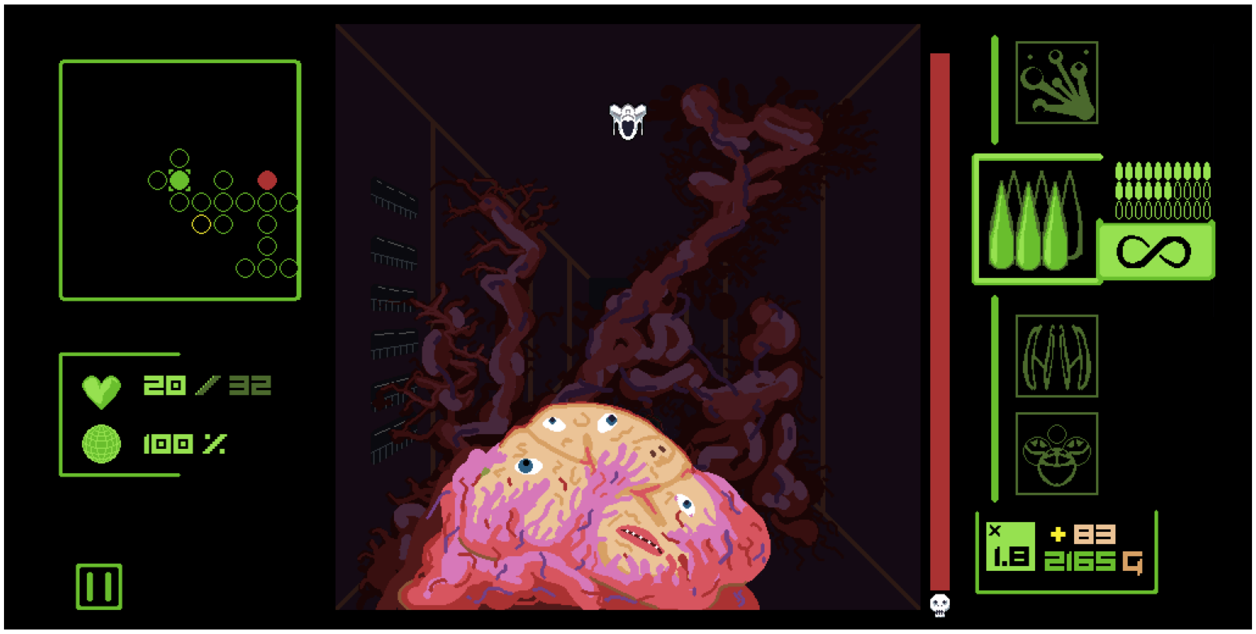

Figure 14: Level 2 Boss Fight

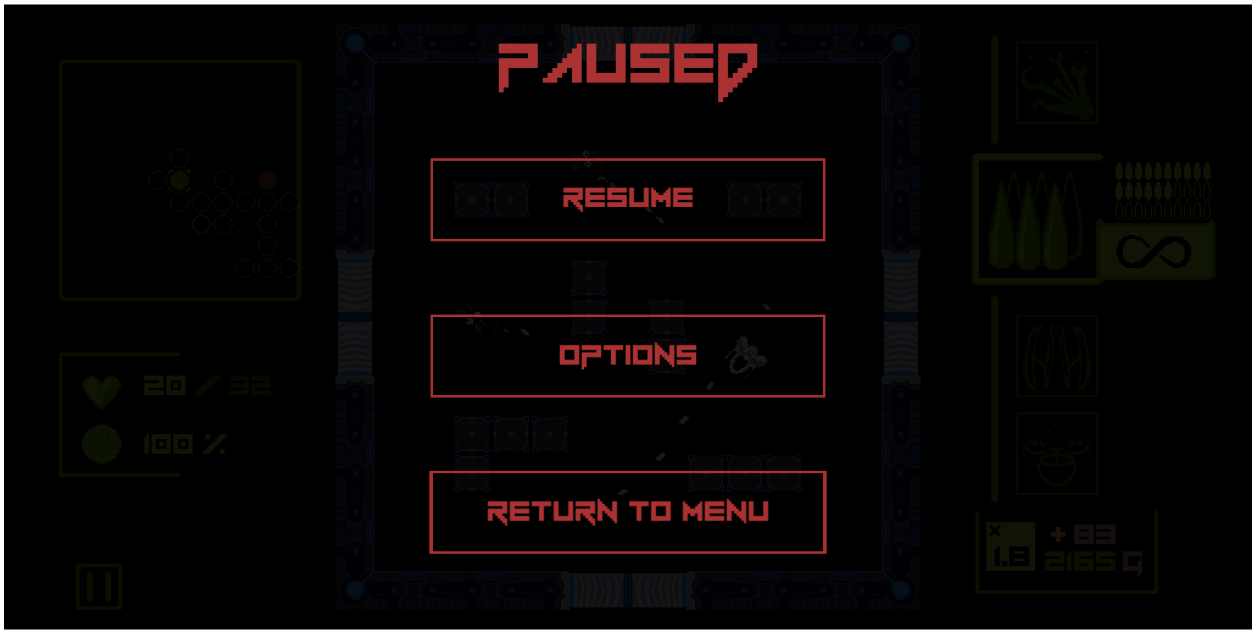

Figure 13: Pause Screen

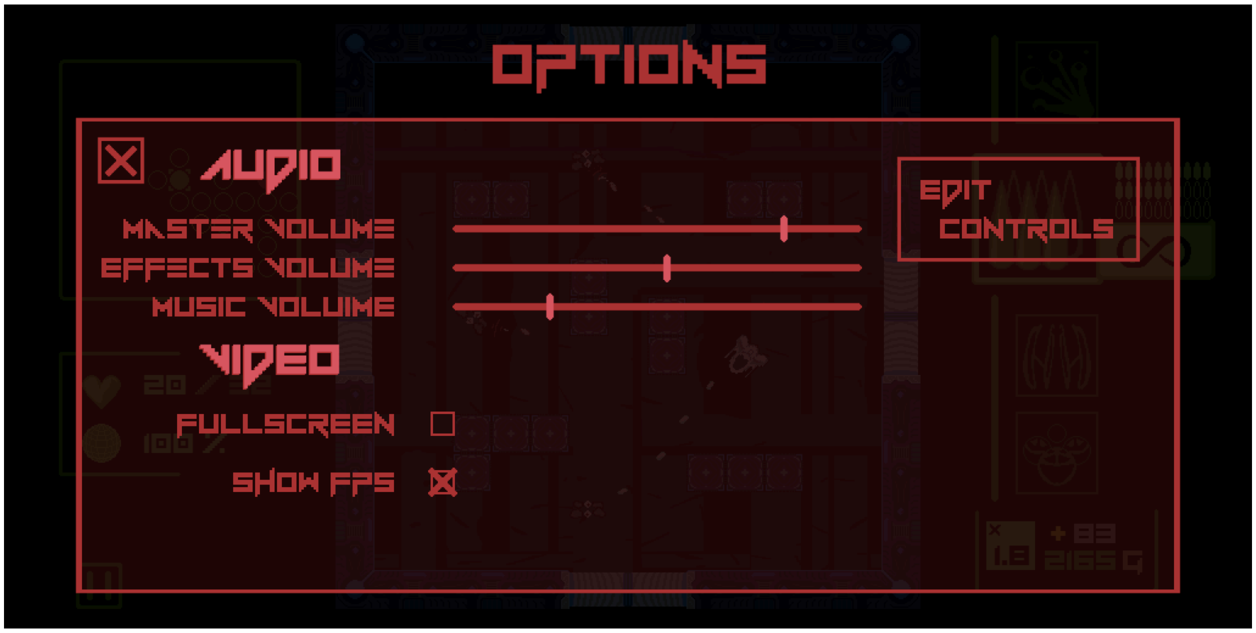

Figure 14: Options Screen

Note: the customizable options here are just for example. We do not yet promise to support control remapping or fullscreen. These options are provided here as examples of what could be here.

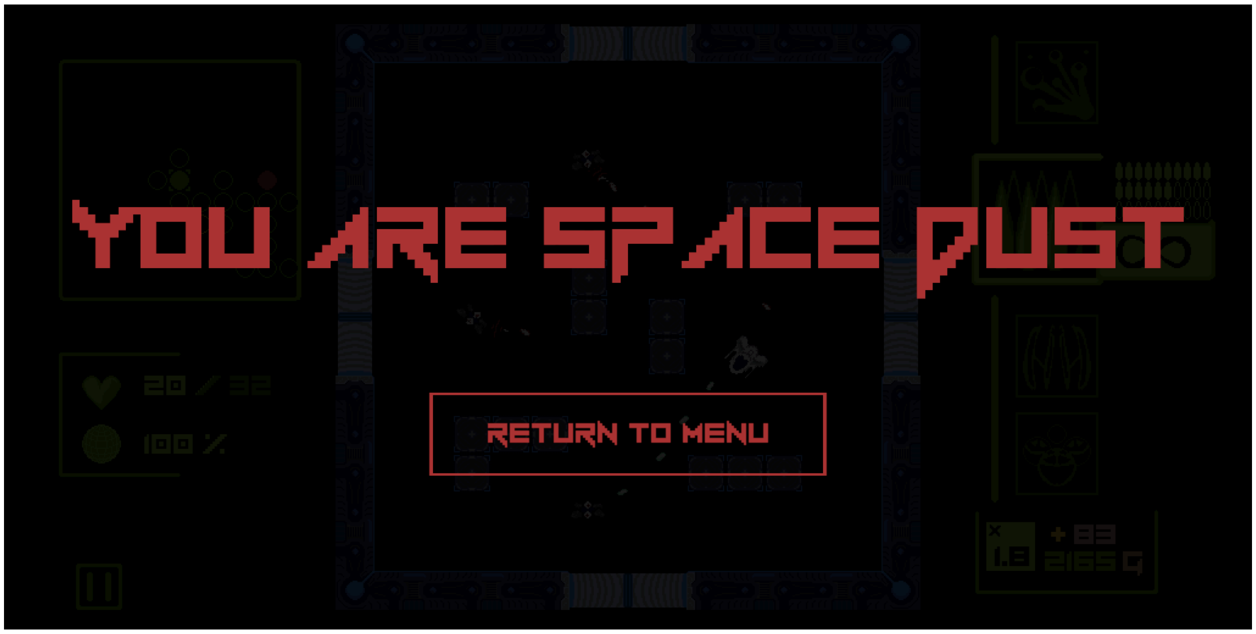

Figure 15: Death Screen

Figure 16: Brainwashing Scenes

A .gif example of what the non-gameplay story elements may look like.

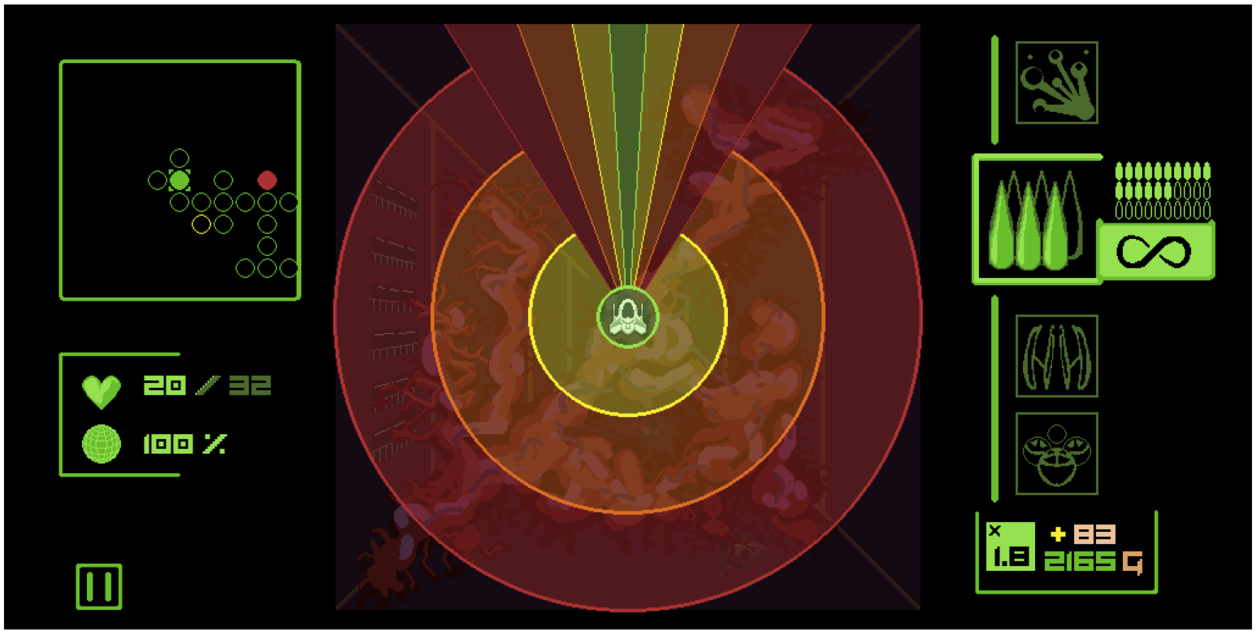

Figure 17: Range and Spread Explanation

Above is a visual demonstration of what distance certain ranges of weapons may fire. As well the conical vision slices indicate the spread/accuracy of certain weaponry. The circles indicate the field of range for weapon fire (melee => green, low => yellow, medium => orange, long => red), once being fired by the player after this distance has been travelled the bullet will disappear. The cones indicate the spread or accuracy of weaponry when the player is firing (green => high accuracy, yellow => medium accuracy, orange => low accuracy, red => terrible accuracy).

Technical Elements:

Rendering:

- Utilize OpenGL for efficient 2D rendering of gameplay scenes and animations.
- Implement fragment shaders for the colour-changing effects of a sprite such as when
    - an enemy is hit with a bullet, when a character is hit, or when the character collides with a wall, enemy or enemy projectiles (ex. invincibility frames from Mario).
    - a bullet that glows is fired (i.e. lasers), shaders can be used to “illuminate” the surroundings.
- HUD features that indicate helpful information like a map, gun clip counters, and other player statuses.
    - In the map, room data is displayed as follows: normal rooms are green, pickup rooms are yellow, and boss rooms are red; as well any room previously explored (meaning no enemies remain inside) is only coloured along the border.

Geometric/Sprite/Other assets

- Sprites and assets will be generated in-house using Aseprite, however, when applicable (ex. bullets, flame animations) assets will be sourced from free vendors.

Physics & Simulation:

- Kinematic physics for realistic interactions for character collisions, as well certain ammunitions a character may shoot and hit an enemy and for rotations when the character moves the direction it is facing.

Artificial Intelligence:

- Simple pathfinding implementation using A* or BFS algorithms for the enemy movement; enemies, depending on whether they are normal, elite or boss, will have different movement patterns (as well as different shooting styles with different weapons) but they will all find a path to move towards the character.

Software Engineering:

- Basic reloadability through serialization.
- Usage of external tools or libraries such as PhysX and bullet for physics simulation.

User Interface (UI) & Input/Output (IO):

- Mouse gestures for aiming which will rotate the character, allowing it to shoot in different directions; scrolling wheel input for scrolling through different weapon choices.

- WASD keyboard inputs for character movement which will allow the character to move to any location within the map and reload their current weapon.
- Audio feedback for interactions which will include audio feedback when a bullet hits an enemy, the character shoots bullet(s) and uses abilities, the character gets hit, picks up ammunition, power-ups and gold; background music reflecting the game's atmosphere.
- Implement the Options page which we plan to include a bar that you can drag left or right to change master, effects and music volume as well as video settings to enable full screen or fps.

Quality & User Experience (UX):

- Basic integrated asset creation includes new sprites for item drops such as ammunition supplies, power-ups and gold.
- Game balance for enjoyable yet challenging experience including normal, elite and boss enemies where each enemy will behave differently and have different maximum health, making it difficult to defeat higher tier enemies than lower tier enemies.
- Incorporate a compelling narrative with character development and interesting story elements.
- A variety of room layouts and enemies encourage a replayable experience.

Advanced Technical Elements:

Listed below is a list of advanced technical elements we plan on implementing into our game, sorted with (1) being the highest priority:

(1) Precise Collisions

- *Impact:* The implementation of precise collision using advanced techniques is crucial for our game's core mechanics and contributes to the overall fluidity and satisfaction of our game. In a game heavily reliant on player interactions, particularly shooting enemies, having a consistent and accurate collision system is integral to gameplay. If we opt for basic collision detection instead, the accuracy and reliability of interactions between players and enemies would be compromised. This could lead to frustrating gameplay experiences where shots may not register correctly or enemies might exhibit erratic behaviour due to imprecise collision responses.
- *Alternative:* Basic Collision System

(2) Procedural Level Generation

- *Impact:* Procedural Level Generation significantly contributes to the game's replayability by introducing diversity and unpredictability in the level design. If we choose hard-coded levels as an alternative, the impact would be a substantial reduction in replayability. The levels would remain the same on every playthrough leading to a more linear and predictable structure, diminishing excitement with consecutive playthroughs. We aim for these levels to be completely procedural generated and random on each playthrough, but will need to see whether that is feasible for performance reasons (as we would need

to frequently generate rooms). If unfeasible for performance, we plan to switch to pre-generated, and then random selection.
- *Alternative:* Hard-coded, Predefined Levels

(3) Swarm Behaviour

- *Impact:* We plan on supporting many enemy types with varying AIs. One of these we would like to support is Swarm Behaviour which adds a layer of sophistication to enemy AI, making their actions appear more natural and unpredictable. If we opt for basic AI instead, enemies may lack dynamic movement patterns. Additionally, basic AI would exhibit predictable movements reducing the challenge and excitement for players.
- *Alternative:* Basic Enemy AI

(4) Parallax Scrolling (Boss Fights)

- *Impact:* Parallax scrolling in boss fights enhances the visual spectacle and intensity of these crucial gameplay moments. The dynamic movement of background layers creates a sense of depth and immersion, making the boss encounters more visually engaging. If we opt for static backgrounds as an alternative, the impact would be a significant reduction in the overall visual appeal and cinematic quality of boss fights. The absence of parallax scrolling could make these encounters feel less epic and immersive, potentially diminishing the emotional impact on players.
- *Alternative:* Static Background

Devices:

We will use the keyboard, specifically WASD for movement, ESC for opening the options screen, and R to reload. The Mouse will be used where the player will left-click for shooting and clicking on-screen elements such as the pause button. We will potentially add support for Xbox controllers as a stretch goal.

Tools:

We plan to use the following libraries:

- **Aseprite**: Art creation tool used to build gameplay mockups
- **GLFW**: Create and manage windows, keyboard, and mouse input.
- **GLSL (OpenGL Shading Language)**: Used to define our vertex and fragment shaders
- **OpenAL**: Play audio files such as .wav in our game

Team management:

Our team management plan is as follows:

1. Split the team into three pairs, each focusing on specific aspects of game development.
2. Assigning tasks and deadlines, and monitoring progress using platforms like Asana.
3. Communicate through Discord for daily coordination and quick queries.

4. Employ Github for code management, ensuring each pair works on separate branches and follows a review process before merging.
5. Set bi-weekly milestones for tracking progress, establish a code review policy for quality assurance, and enforce a “no late merges” rule near milestone deadlines to support build stability.

Team 1 (Adrian, Nathan): *Focuses on developing core gameplay mechanics like health, score, and ammo, and implementing dynamic gameplay elements such as power-ups.*

- **Adrian Lee:** I have object-oriented programming experience and acquired debugging skills through software engineering experience.
- **Nathan Dar:** I have general experience in Software Engineering through co-ops. I am interested in developing features related to the core and dynamic gameplay features.

Team 2 (Andrew, Celvin)*: Leads in game engine development, including physics, game design, rendering, and procedural generation.*

- **Andrew Cheah:** I have game development experience with C# and Unity, also through taking CPSC 314 and co-op, I have experience with WebGL and Graphics. As well, I have a personal interest in procedural generation and game design.
- **Celvin:** I have experience programming simple games with the Pygame library and also in Javascript utilizing the canvas HTML element and also elementary experience in designing user interfaces with Figma.

Team 3 (Shafquat and Alex)*: Concentrates on enemy AI, feedback systems (gives players real-time responses to their actions), and designing the game over and upgrade scenes.*

- ***Alex Choi:*** I have some Software Engineering and machine learning/AI experience from the CPSC courses and internships. My general interest is in game development; this is my first time developing a game using C++ and OpenGL.
- ***Shafquat ul Bari:*** I have developed a foundation in gameplay AI and feedback systems through my experience working on game projects using C# and Unity engine. I worked on single-player elements in racing games and 3D FPS games which relies very heavily on sophisticated AI.

All teams - Focus on bug fixing and final testing.

Development Plan:

Provide a list of tasks that your team will work on for each of the weekly deadlines. Account for some testing time and potential delays, as well as describing alternative options (plan B). Include all the major features you plan on implementing (no code).

Our development plans will revolve around organizing the list of tasks to be completed before the end of each week of each milestone. The following are the goals that we aim to complete at the end of each week.

Milestone 1: Skeletal Game

Week 1: Focus on establishing the UI, gameplay mechanics, and a simple game engine. Begin the creation of simple art and AI.

In the early stage of the development, we will focus on establishing the technical specifications of the game and the development of the **essential features** of the game. The main goal of this week is to create some working prototype of the **menu screen** and **make the player visible and movable on the screen** and to have a small portion of the code that handles AI and physics to be written. The tasks to be completed before the end of this week are as follows.

1. **Implement the rendering engine**, specifically the part that handles rendering simple 2D shapes on the screen given the position and dimension of the object.
2. **Create a prototype of the main menu screen**, with clickable buttons for transitioning into the game loop and an exit button. The list of things to be implemented includes
    - The button class or function, which will render a button given the attributes of a button (width, height, text, colour - more to be added if necessary)
    - The text class or button, which renders a piece of the string given its attributes (string to display, font type and size, colour)
3. **Define the entity class and basic components that are focused on the player entity**. The list of classes to be implemented includes:
    - Entities: Player, Bullet, Enemy
    - Components: Position, Velocity, Weapon
4. **Implement a portion of the enemy AI and procedural level generation algorithm**. This includes the components that the enemies need, which are information about the player (visibility to the enemy, last known location, weapons possessed) and also helper functions for both the enemy's decision-making AI and procedural generation of levels.
5. **Write the movement systems portion of the physics engine**, which handles the movement of the entities - currently, the only entity we have is the player - according to their position, velocity, and orientation.
6. **Integrate the systems into the game loop and play tests** to find bugs that appear in our game.

Week 2: Develop menus, initiate level generation, and implement simple rendering effects.

At this stage of development, we aim to include enemies in the game and to have some prototypes of the levels. We will write code to represent the **enemy entities and components**, together with enemy actions such as shooting and moving, but we are not aiming for the enemies to be able to move yet at this stage. At the end of this week, we aim to have a space-invader-like prototype where the player can move and shoot (statically positioned) enemies and also **collide** into walls and bullets. The list of tasks includes the following.

1. **Fix the bugs found from the previous week**.
2. **Implement the projectile system, which handles the movement, damage calculation and the removal of projectiles** on the screen.

3. **To create the prototype for the pause and game over menu screens**, we will use the widget class or functions that have been implemented in the previous week.
4. **Define the components and systems for collisions**. the Collision component, which defines the entities colliding with the owner of said component instance and also implements the collision detection system, which will loop through every entity and check if it has collided with any other entities in the vicinity and update its corresponding component.
5. **Integrate new systems into our game loop and play tests** to find bugs that appear during this iteration of the game.

Milestone 2: Minimal Playability

Week 1: Enhance the UI with feedback elements like the health bar, score, armour, and bullet count. Start integrating different enemies and refining AI.

We will start introducing the major mechanics in this milestone: **Health and armour, scores, player upgrades, gold, and player inventory (ammunition for different weapon types)**. We will also aim to start introducing the **enemy's AI** into the game. At the end of this week, we will introduce the previously mentioned mechanics into the game together with enemies that will move and shoot according to the AI. The specific goals of this week are as follows.

1. **Fix the bugs that are found from the previous week**.
2. **Complete the enemy AI system for movement and shooting and integrate it into the game loop**.
3. **Implement a large portion of the algorithm for the procedural generation of the levels**, which includes the helper functions and roughly over half of the code that handles the generation of the levels.
4. **Implement the components and systems that make up the health and armour, score, gold, and player inventory mechanics**. The components and systems involved are as follows.

    - Components:
        - **Health and Armor:** defines the health and armour value of the various entities
        - **Gold:** defines the gold count that enemies will give out.
        - **Player inventory:** defines the weapons and ammunition count of the player (we may also include the player's gold count in this component)
    - Systems:
        - **Health and armour management system:** handles how each entity gets affected by damage and health/armour pickups.
        - **Currency system:** handles the increments and decrements of the player's currency count, increments may change based on certain conditions such as the currency multiplier power-up. This system also handles any action involving the shop.
        - **Player inventory system:** handles the increments and decrements of the player's ammunition count for different weapon types.

5. **Start development on the texture mapping portion of the renderer** for rendering the game sprite on the shapes on the screen.
6. **Integrate the major systems into the game loop and run a playtest** to find any existing bugs.

At this stage, we may scrap some major mechanics from the game if we can't make it.

Week 2: Address bug fixes, and introduce collision and environmental effects.

We will introduce the **procedural generation of levels** during this week. We will also fix the bugs that we found in the previous week. We aim to have the basic and major game mechanics working during this week and for the game to resemble our final product without all of the game sprites. We will also start to introduce **different enemy types** in this iteration of the game and also the **power-ups** that the player can obtain. We will prioritize the top three enemy/weapon types (medium-range machine gun, melee ram, and short-range shotgun) the top two tier-one power-ups and the top two tier-two power-ups from the priority list. The priority list of the enemy/weapon types and the power-ups are defined as the following.

Priority of Implementation Order for Enemies/Weapon Types

1. Basic medium-range machine gun
2. Melee range ram attack
3. Short-range shotgun type
4. Long-range laser type
5. Mine hazard
6. Homing missile
7. Movement disabling gun

Priority of Implementation Order for Player upgrades

- Tier 1: (Level 1 upgrades, non-invasive, mediocre)
    a. Multiplier for collecting gold
    b. Overcharged shield (temporarily turns shield into electricity ram weapon)
    c. More health
    d. Better shield recharge
    e. Increased overall bullet speed
    f. Decrease time to charge for any charge-up weapons
    g. Improve accuracy (decrease bullet spread)
- Tier 2: (Level 2 upgrade)
    a. Increased acceleration
    b. Ignore death once
    c. Regeneration
    d. Fast Reload
    e. Dash ability

In addition, we will start to utilize our game sprites in the game to start giving shape to the final game. The list of tasks that we will tackle this week is outlined as the following.

1. **Fix the bug found in the previous week**.
2. **Fully implement the level generation system**, which handles the generation of levels and integrate it into the game loop to generate levels at demand. We will also have to implement a system for moving the player from one level to another.
3. **Define and implement the entities, components and systems for the power-up mechanic and enemy/weapon types**. The enemy-type entities will be subtypes of the base enemy entity that we defined in milestone 1, the power-up will be a new entity that we will implement. The components that we will create include a Weapon component, which defines the type of weapons that the player and enemies are holding; a component for each of the power-ups which defines what other components should be updated.
4. **Update the projectile system** to handle the creation, movement, damage calculation, and removal of new types of projectiles.
5. **Integrate the new systems into the game loop**, render the game sprite for the player, walls, and basic enemy and play test to find bugs.

Alternatively at this point, if the enemy pathfinding and shooting algorithm hasn't been completed, we will opt for basic enemy behavior to substitute for the AI. If any part of the essential mechanics hasn't been finished, we will scrap the plan to implement a few of the major mechanics to work on the essential mechanics.

Milestone 3: Playability

Week 1: Add more enemies that were not introduced in Milestone 2, and introduce dynamic gameplay elements like power-ups. Introduce advanced visual effects for damage, weapons and collisions.

By this point, we hope to have completed at least half of our game, which includes the essential and main mechanics of our games. If we haven't gotten most of the mechanics from the previous milestone working, we will continue to make bug fixes and improvements on it. If we think that it is sufficiently working, we will **start working on a boss-level** enemy in the game, which includes its AI and attack mechanics. In addition, we will **polish the user interface** for the main and pause menu, game HUD, and game over screen. In addition, if we have time, we **may also add additional power-ups and weapon/enemy types** from the priority list. The list of tasks is shown in the following.

1. If most mechanics from Milestone 2 Week 2 haven't been working properly, **fix bugs and improve code**, before moving on to the following items.
2. **Design and create the components and systems for the boss-level enemy**, which includes the boss' AI and attack options.
3. **Define the components and systems for the boss' mechanics**. The components that are required include the boss' unique weapons and the systems we require for the boss include the boss' behavior AI system.

4. **Polish the UI** for the main menu, pause menu, game-over screen, and the game HUD. In addition, add in the background for each of the screen and the game levels.
5. **Play-test the game** to check for any bugs happening during rendering and also if there are any bug residues from Milestone 2 Week 2.

Week 2: Develop the game-over screen, upgrade screen, polish the UI, and introduce game assets like audio.

We will **fully develop the boss-level enemy** and integrate it into the game and develop the player upgrade and end game screen. We will also start **adding audio into the game** during this phase. We will mainly prioritize on integrating the audio into the game since at this point the development of the boss enemy might be insufficient after the start of week 2. The list of tasks are listed as follows.

1. **Fix the bugs that were found in the previous week**.
2. **Create the upgrade screen and the functionalities of choosing upgrades**. This part may reuse the components used for the power-up mechanic.
3. **Create a sound manager** which will be responsible for the following.
   a. Stores the data from a music file in memory.
   b. Play stored music for a specified amount of time or iteration (single loop, infinite loop, etc.)
   c. Pause or Stop the music from playing.
4. **Finish the implementation of the boss and integrate it into our game**.
5. **Hook up the end game screen with the end game event** where the boss is defeated.
6. **Conduct a play test** to find bugs.

Milestone 4: Final Game

Week 1: Focus on bug fixing, final testing of rendering, AI, and physics

For this milestone, we will be **making the finishing touches on the game** on this milestone, specifically, we will be **making sure that the entire playthrough of the game runs smoothly**. We will mainly **conduct bug testing** focused on the game rendering, game AI, and bugs that might be hiding deeply in our game. The list of tasks includes.

1. **Finish the integration of the boss** into the game, if otherwise not finished from the previous week.
2. **Conduct a play-through** of the game and check for any problems or bugs and fix them.

Week 2: Continue bug fixing and final testing, ensuring all game elements are polished and work seamlessly.

If there are too many bugs found in the previous week, we will again be **focused on locating bugs and troubleshooting**. If there are no or a few bugs that don't affect gameplay too much, **we might consider adding extra features** and mechanics into the game, such as the options screen, controller support, or additional enemy/weapon types into the game. The goal of this week is as follows.

1. **Fix the bugs remaining** from the previous week.
2. **Conduct a final play-test** of the game and fix any found bugs.
3. If there is extra time, implement the extra features such as additional enemy/weapon types or power-ups.