Game Proposal: Amoebash

CPSC 427 – Video Game Programming

Team: Infection Games (Group 1)

Advay Rajguru - 85918902
Mercury Mcindoe - 85594505
Saurav Banna - 43442367
Shrey Gangwar - 76979327
Hazel Chen - 83988873
Dany Raihan - 53341608

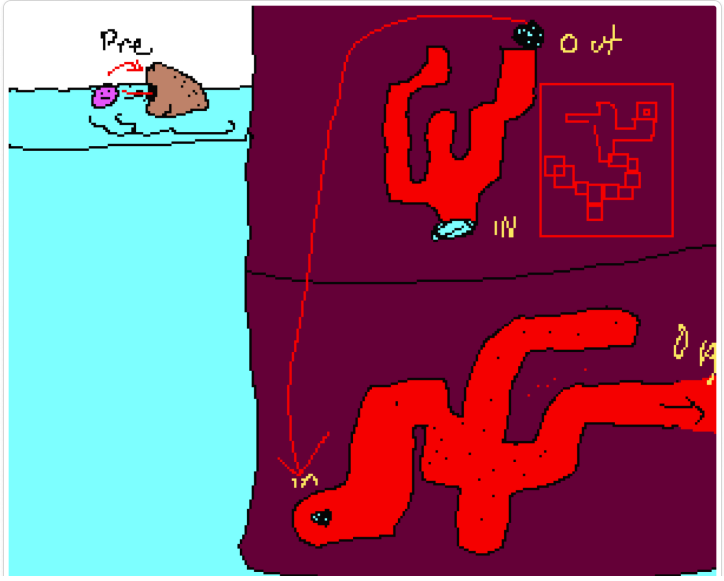
Story:

Amoebash is a roguelike game where you play as the nucleus of a brain-eating amoeba with one goal: to infect a human being. You start by going through a person's nose, and you'll need to swim, dash, and slash through your opponents while facing various challenges, with the final goal of reaching the brain. If you make it to the end, you face off against a multi-segment Neuron Dragon that uses Electric Attacks to stop you. Chip away at the dragon segment by segment and you might just get the glory of taking over a human!

Throughout your journey, you'll face off against hordes of enemies in procedurally generated dungeon floors, with each one getting harder and harder. Don't worry though; each enemy you defeat has a chance of dropping buffs to strengthen yourself during your run. You'll be able to get new weapons or power ups, which will help you later in the game.

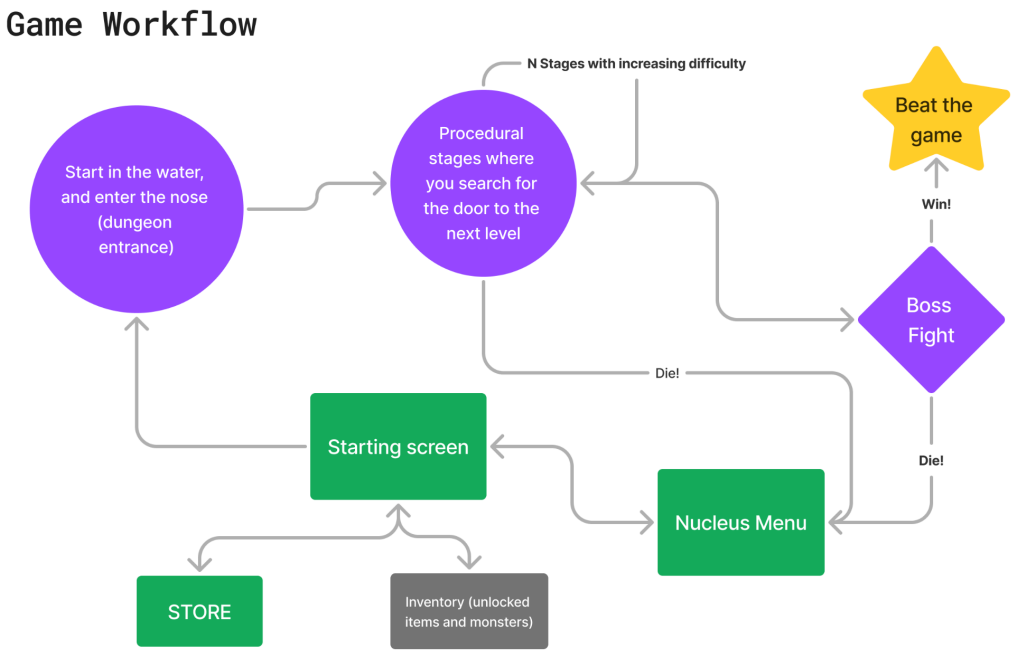
Don't sweat it if the Neuron Dragon or any other enemies get the best of you—just retreat and regroup! On your way out, you get to keep at least one buff you collected for the next run through a "Nucleus Menu". You might've stumbled upon a powerful buff, giving you an edge for your next daring attempt, which means a good run is never wasted!

Each enemy can also give you "Germoney", which you can use in the Shop to buy new buffs, or increase how many buffs you can keep when you die.

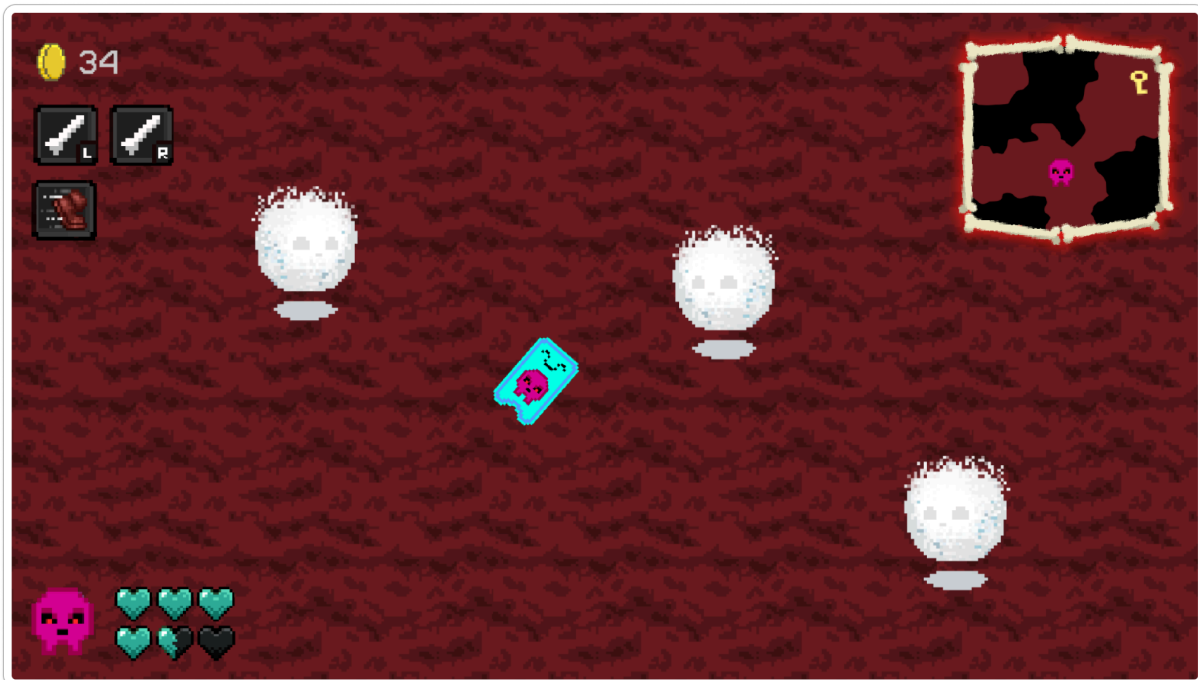


Scenes:

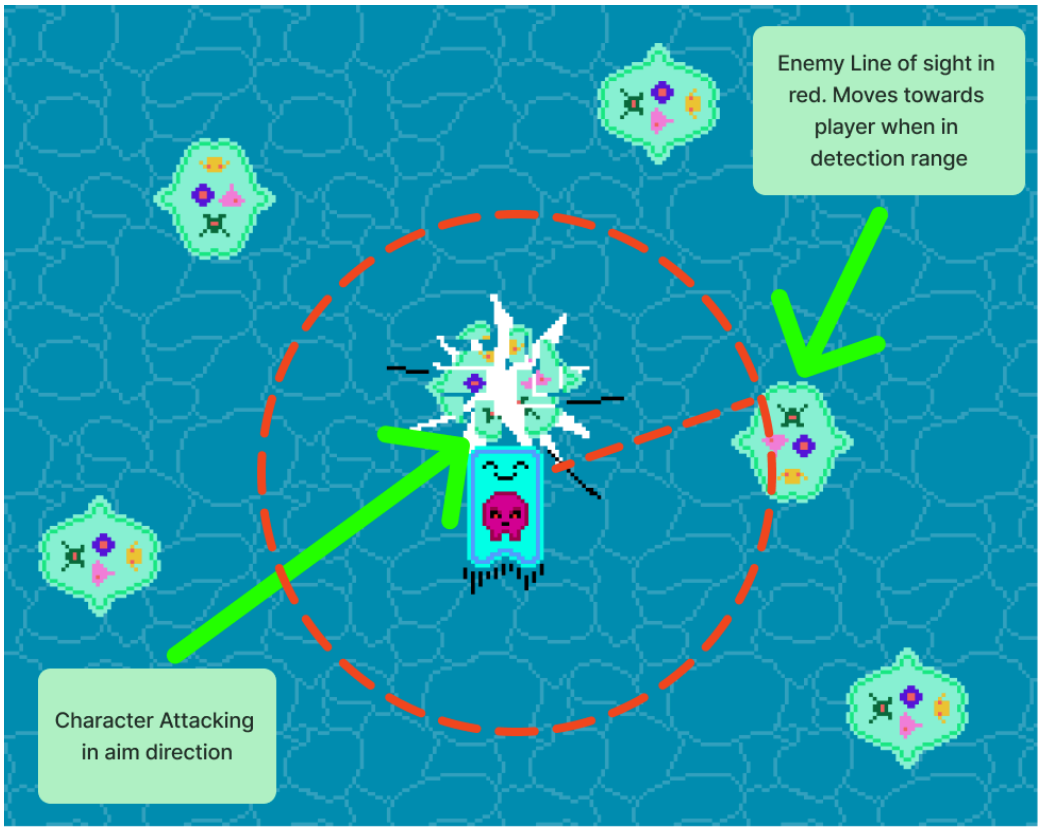
The game is a roguelike that follows the graph below.



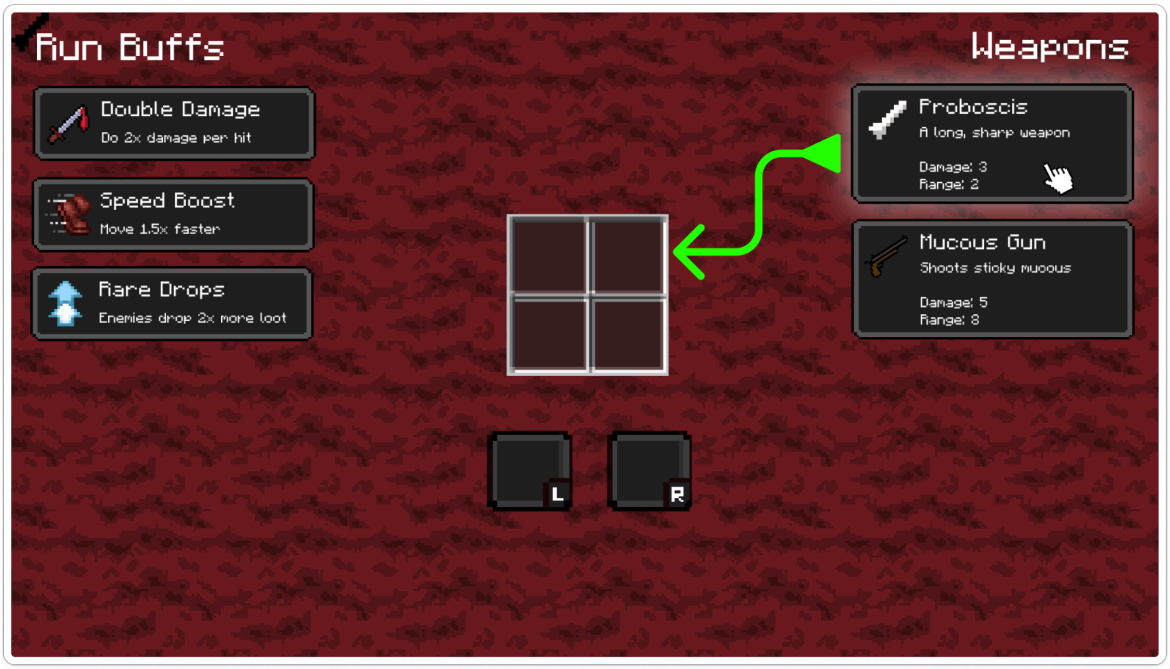
The general UI should depict the weapons, buffs, player health, in-game currency, and mini map. Any current buffs appear on the top left (speed boost pictured below) and disappear when they expire.



Enemies move towards the player when within a certain detection range. Furthermore, the player attacks in the direction it is oriented towards. The image below shows a dash attack.

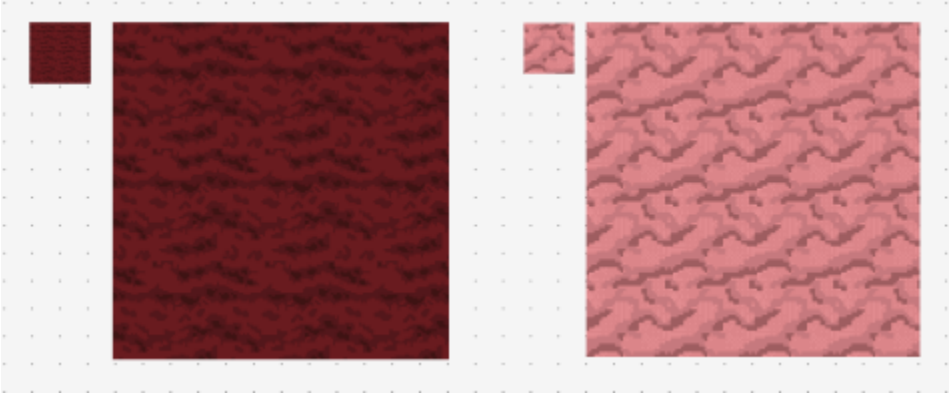


Upon death in a run the nucleus menu will come up where the player determines which buffs (including weapons) to “save” for the next run.



Furthermore we will implement a shop menu, a pause menu and unlock section. These sections will be interactive: for example, the pause menu lets you learn more about the different characters on screen by hovering over them, while the shop menu displays tooltips for each item.

All screens will use pixel art as it's easy to standardize, and make consistent. Pixel art will allow us to reuse similar tiles for the environment with simple palette swaps if need be. Here are some samples of the tiles for the map that we have created so far.



Using tiled textures also makes it easier for us to procedurally generate maps and make them look cohesive.

Character animations and boss animations will have 8 frame loops, where weaker enemies will have 4 frame loops to reduce workload.

Technical Elements:

Identify how the game satisfies the core technical requirements:

Rendering:

In addition to the basics of rendering the character, map, enemies, etc, we will also incorporate simple rendering effects that utilize the fragment shader. Some things we want to flash a little bit of red color on an enemy sprite when the character hits and damages it, and the same thing for when the character is hit.

Additionally, when the user dashes, we want to add slight colouring to the sprite to signify the dash. This will also include some particle effects and a “trail” off the sprite, using instanced rendering to render the particles.

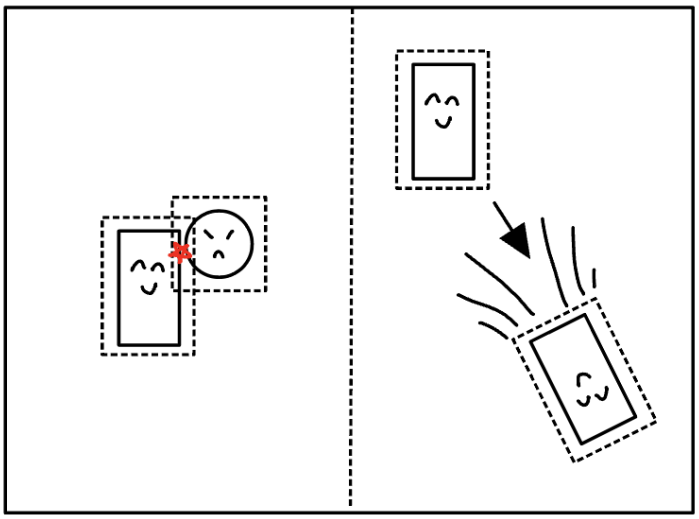
Sprite Assets:

The game will use pixel art sprites that we created to depict animation and the intractable UI elements. The sizes we will primarily work with are 32x32 pixels with some sprites at 64x64 pixels and our overall game window following the aspect ratio 270x216.

2D Geometry Manipulation:

2D geometry manipulation will primarily be used to handle collision detection and related interactions. Objects such as enemies, ranged attacks and the player will be represented using bounding boxes, which will be used to detect when an attack inflicts.

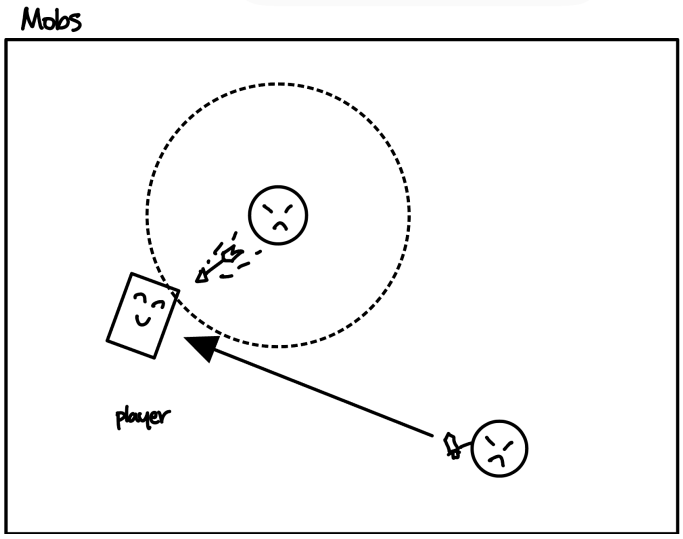
Additionally, during actions like dashing in different directions, the player's hitbox along with the visual representation will rotate accordingly to match the character's direction. The player character rotates as the mouse moves around, so rotations will be a big part of the gameplay.



Gameplay Logic/AI:

The player explores the map, battling enemies and searching for the door to progress to the next stage. Along the way, the player will have the chance of picking up or buying status buffs / weapons that enhance their abilities. Once reaching the last stage, the player will have to kill the final boss to achieve victory. If the player is defeated, they respawn to the first stage, with a chance of bringing one piece of loot during its previous run and re-enter the dungeon with the ultimate goal of defeating the final boss. The player can also collect money during the runs to purchase upgrades or items to improve the chance of living. The flowchart provided above depicts gameplay.

We are aiming to incorporate different AI models for the type of enemies. Basic mobs will use simple path-finding (e.g. BFS) to track the player. For ranged enemies, they will maintain distance and only attack when the player is within their line of sight. Whereas melee enemies will approach closely to the player and make physical attacks. Bosses and mini-bosses will have AI that will be described in the "Advanced Technical Elements" section.



Physics:

The dash and movement will have slight physics applied to add a drift as well as some particle effects. Basic physics such as inertia, acceleration and deceleration, will be applied to add drifting (a continued movement after a dash) to give the player a sense of momentum.

Similarly, ranged attacks by both the player and enemies (eg. bullets) will move according to kinematics.

Advanced Technical Elements:

Additional technical elements prioritized on likelihood of inclusion:

Audio Assets:

Our group has already prepared and will add audio assets that will change during each step of the game. Audio is aimed to be changed in/out of combat and different audio will be played for boss fights. We also want to add sound effects as feedback for the different player actions, such as dashing or attacking. If time permits, we also want to have the background music change in mood (eg. get more intense) as the game progressed to further immerse the player. If this feature is not implemented, a single looping background track will be used, which may risk reducing player engagement over time.

Procedural Map:

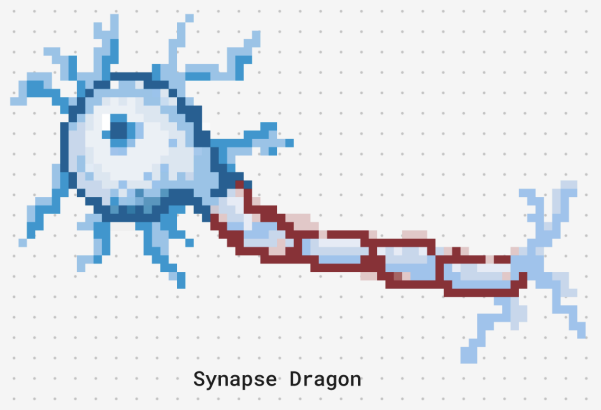
The game will use a procedural map and thus be rendered as the user enters the stage. We have conducted research on multiple map algorithms such as Binary Space Partitioning, Random Walk, and Cellular Automata, and are yet to pick one of them. Regardless of the specific algorithm, for each stage we will randomly pre-generate a map. On top of this map, we will randomly generate enemies, obstacles, and other things as the user progresses through it.

Additionally, we will utilize the pre-generated map to render the minimap, that will show to the user a small image preview of their location on the map, and will move as the player moves as well. The minimap would also highlight important locations or items that the player would want to get.

The alternative to if we can't get this feature done is 1-5 preset maps for each stage that we create and set up by hand, instead of a randomly generated one. The main impact on gameplay will come from replayability for the user, as if the game is replayed the user is likely to come across a map that they have already played (depending on how many presets we do per stage).

Skinned Motion:

The final boss model will use bones and kinematics to curve its body as it travels around the map. This will ensure an engaging and challenging boss fight where the player will have to predict where to attack and how to catch up to the vulnerable middle section of the boss. The alternative would involve the boss having the same motion as a basic mob, making it a simple sprite floating around with no dynamics and significantly simplifying gameplay. Here is a sample design of the boss from one of our groupmates.



Storytelling:

We plan on implementing a few different story components:

- Popups from the nucleus (player character) to indicate information to the player when they first engage a new enemy or mechanic.
- Tooltips on the Pause screen when a character is hovered over. This should give a description of the character, along with its stats (for example, its health and damage for an enemy)
- A simple cutscene at the start of the game explaining the goal of the amoeba and how it ends up in the human
- If time permits, an animation of the nucleus "running away" upon death to respawn
- If time permits, adding small story / explanatory elements to the item tooltips in the Shop or Nucleus Menu, to give players a bit more information or lore

If we can’t get all or some of these features done, the game is still fairly self-explanatory and isn’t heavily story based. It may be less immersive for the player, and they may be less invested in finishing the game, but the main draw of the game is the challenging gameplay itself.

Specialized shaders:

We plan to implement specialized shaders to enhance the microscopic theme of the game:

- Add a vignetting effect that mimics the circular view of a microscope
- Add a blurring effect to simulate the distortion caused by focusing through different "depths" within the environment, such as when transitioning between levels or entering boss arenas
- If time permits, add a fluid distortion shader to add subtle ripple effect to areas of the map that resemble liquid environments

As an alternative, we plan on sticking to a normal pixel art style for the game. This may make the game look less distinctive, but is also a familiar and classic look for many games.

Particle effects:

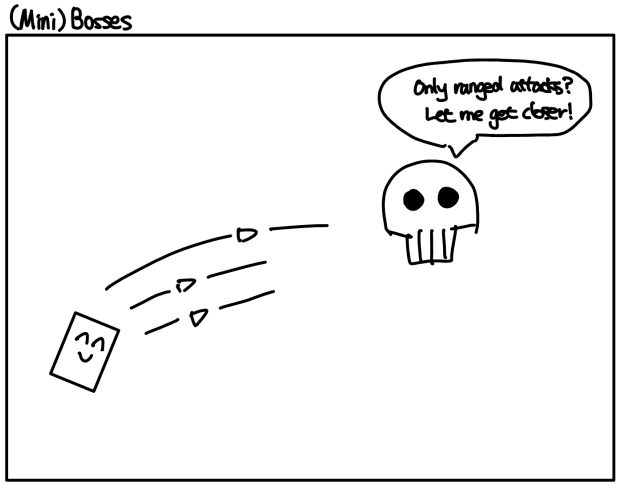
Additional particle effects for dashing, enemy death and weapons.

We will incorporate particle effects for a variety of events in the game. This includes when the player dashes, leaving a trail of particles as they move. Other instances where we want particle effects include when the player shoots a weapon, or an enemy dies. For all of these particle effects, we hope to use instanced rendering so we can render the same particle object multiple times, in an efficient manner. We especially want to use particle effects during the final boss fight, as it will make heavy use of electricity.

The main impact of not including particle effects on gameplay is purely visual. Not having these effects may make it less clear when an enemy dies or the player loses health, especially during more intense moments, which can affect gameplay. So we plan on prioritizing those effects over others.

Advanced AI for Special Mobs:

We will aim to incorporate Advanced Decision-Making for (Mini-) Bosses, these mobs will analyze the behaviour of the player and adjust attack patterns accordingly. For instance, if the player relies more on ranged attacks during gameplay, the boss will approach the player. For melee focused gameplay, we can apply the opposite. If this feature is skipped, we will incorporate the main simple path finding algorithm just like the basic mobs. It would still be somewhat challenging as the bosses are stronger than basic mobs, but it could result in less engaging gameplay as the dynamic difficulty scaling is lost.



2.5D Lighting:

We plan to add lighting to certain enemies, such as glowing bacteria or electric cells as they dynamically illuminate nearby areas as they move. Also, adding ambient light sources in dark environments such as nasal pathways will create an engaging atmosphere.

As a part of the lighting we plan to use normal maps for the pixel art sprites, allowing the game to incorporate dynamic lighting effects that enhance the visual depth and atmosphere. This will make enemies, player sprites, and environmental objects appear more three-dimensional even within the pixel art style.

Not having this feature would be a purely visual change, so gameplay should be the same.

Parallax effects:

If time permits we will replace our base tiling with sets of parallax tiles. Multiple layers of background and foreground elements (e.g., swirling fluids, drifting particles, or neuron-like structures, ..) will move at different speeds relative to the player's motion. This will reinforce the feeling of being inside a microscopic environment and increase visual dynamics.

The effects of not having this feature are purely visual as well, so it wouldn't affect gameplay.

Devices:

We plan on using the keyboard (X, Z and the spacebar) as well as a mouse and left click.

The X and Z keys are mapped to primary and utility weapons, and the spacebar triggers the pause menu. For direction, and dashing we can use the mouse and left click respectively. Moving the mouse around affects the direction of the character dynamically.

Tools:

Our main coding environment will be **Visual Studio Code** or **Visual Studio**, depending on individual preferences. We will use **CMake** for cross-platform builds and **GitHub** for version control to facilitate collaboration and version tracking.

For libraries, we are planning to use the following as well as defaults learned in class:

• **stb_image**: To load sprites and other 2D visual assets.

• **Box2D**: For physics and collision detection, essential for gameplay mechanics like dashing and interactions with enemies.

• **GLM**: For mathematical operations, such as parallax effects, 2D transformations, and custom shaders.

• **Freetype**: For rendering text in the UI, such as player health, buffs, descriptions, and menu options.

• **RenderDoc**: For debugging and optimizing the rendering of pixel art assets, procedural maps, and shaders.

While implementing new mechanics, we sometimes implement them in a game engine like Unity first as a proof of concept, and to play around to identify potential issues before implementing it in our code.

For asset design, we will be using a combination of Aseprite and Procreate for making the assets, and Figma for organizing everyone's ideas. For audio design, we will be using FL Studio.

Team management:

Tasks will be divided based on complexity, time commitment, and individual strengths. Advay, Mercury and Shrey will focus on the development of Game Engine and Gameplay Mechanics, Dany and Hazel will handle Game Engine and UI, while Saurav will be in charge of Gameplay Mechanics and UI. While each teammate has a specialized area, they are encouraged to collaborate across different fields if they express interest. Tasks will be tracked via a shared worksheet that outlines the task, its requirements, and its assignees. Internal deadlines for these tasks will be decided during weekly group meetings while referring to the development plan. If a team member feels their workload is unfair, they are encouraged to bring it up during the weekly meeting. The team will collaboratively review and adjust task assignments to maintain fairness.

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).

We will mark plans with an '*' to indicate those that may not be completed. For such plans, we will provide an alternative below each Milestone.

Milestone 1: Skeletal Game

Week 1

- Set up an OpenGL template for rendering.
- Implement basic player input handling (e.g. movement via keyboard, mouse).
- Render placeholder assets, such as player and basic environment, mobs.
- Implement basic Game loop (e.g. mobs and player only environment).
- Implement advanced motion (e.g. drifting from dash).

Week 2

- Integrate audio assets.
- * Design first draft assets of 5 enemies, 2 bosses and 2 weapons with 4 animation frames.
- Implement basic collision handling (e.g. player vs. mob).
- Introduce random event-driven response (e.g. obstacles, mob spawns).
- Ensure basic UI, functional elements (e.g. shop / pause menu).

Alternative Plan: As high quality assets require time, these can be built upon progressively, all other things in this week are essential.

Milestone 2: Minimal Playability

Week 1

- Implement basic enemy AI (e.g. enemies detects and attacks player).
- Develop procedural map generation using preset tiles.
- Expand player mechanics (e.g. attacks).
- Add additional assets (e.g. enemy sprites, map tiles).

Week 2

- Decide enemy/obstacle placement on the generated map.
    - * Replace dummy maps with procedurally generated maps.
- Implement basic health systems for enemies, player.
- Refine the game loop to fully integrate the game's storyline.

- Introduce/Implement buff system for gameplay.
    - Finalize buff system mechanics.

Alternative Plan: In the case that procedural mapping algorithm fails, we will move on with the basic map generation with template and evaluate alternative map generation algorithms to reach a robust solution.

Milestone 3: Playability

Week 1

- Refine player, enemy gameplay mechanics.
    - Range attacks for both enemy and player.
    - Miniboss, Mainboss specific AI.
- Improve game UI (e.g. shop menu, pause menu, enemy sprites, weapons).
- Develop shop mechanisms.

Week 2

- Integrate particle effects (e.g. enemy sprites, weapons).
- Add challenges for maps (e.g. variations, map quests).
- Implement situation-wise audio adapting (e.g. in/out combat, bosses).

Alternative Plan: As the week 1 features take time we will allow them to fall into week 2 if need be, keeping it intentionally empty. This also allows us to revisit any objectives we are behind on.

Milestone 4: Final Game

Week 1

- * Add advanced gameplay features.
    - Specialized shaders for vignetting effect.
    - Lighting effects for enemies etc.
    - Add parallax effects.
- Cross-platform testing.
- Final bug/error check.

Week 2

- Play testing / Game Balancing.
    - Buffs.
    - Mob stats.
    - Enemy stats/pattern.

Alternative Plan: Advanced features may be dropped if time is required to fix/implement features from the earlier weeks.