Game Proposal: Cyber Yaga: Vindicta

CPSC 427 – Video Game Programming

Team: Neon Grid Studios

Jeff Kim (70668132)
Matias Chomali (69516334)
Mana Longhenry (43629526)
Parsa Saeedi (84982438)
Clive Yong (34877712)

Story:

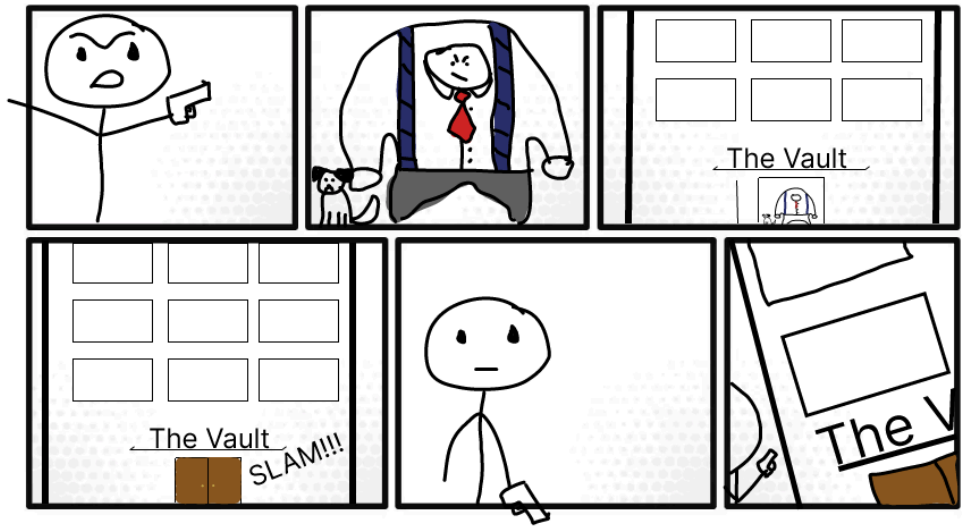
The game unfolds in The Vault, a towering neon-soaked skyscraper in a crumbling cyberpunk dystopia ruled by mega-corporations and syndicates. The Vault, a sanctuary for assassins and criminals, thrives under the enigmatic Arbiter, a ruthless ruler who wields power through fear and cunning. The story follows Juan Brick, a retired assassin whose legendary reputation once struck fear into the underworld. After defying the Arbiter during a pivotal mission, Juan became a marked man. Seeking solace with his dog Daisy, his peace was shattered when the Arbiter's assassins destroyed his home, stole his car, and took Daisy, leaving him broken but alive. Driven by vengeance and love for Daisy, Juan returns to the underworld, battling waves of assassins as he ascends The Vault to save Daisy. Each fight brings him closer to the Arbiter, culminating in a climactic showdown where Juan defeats his nemesis, rescues Daisy, and escapes. Battered but victorious, Juan walks into the neon-lit streets with Daisy, his fight finally over.

Scenes:

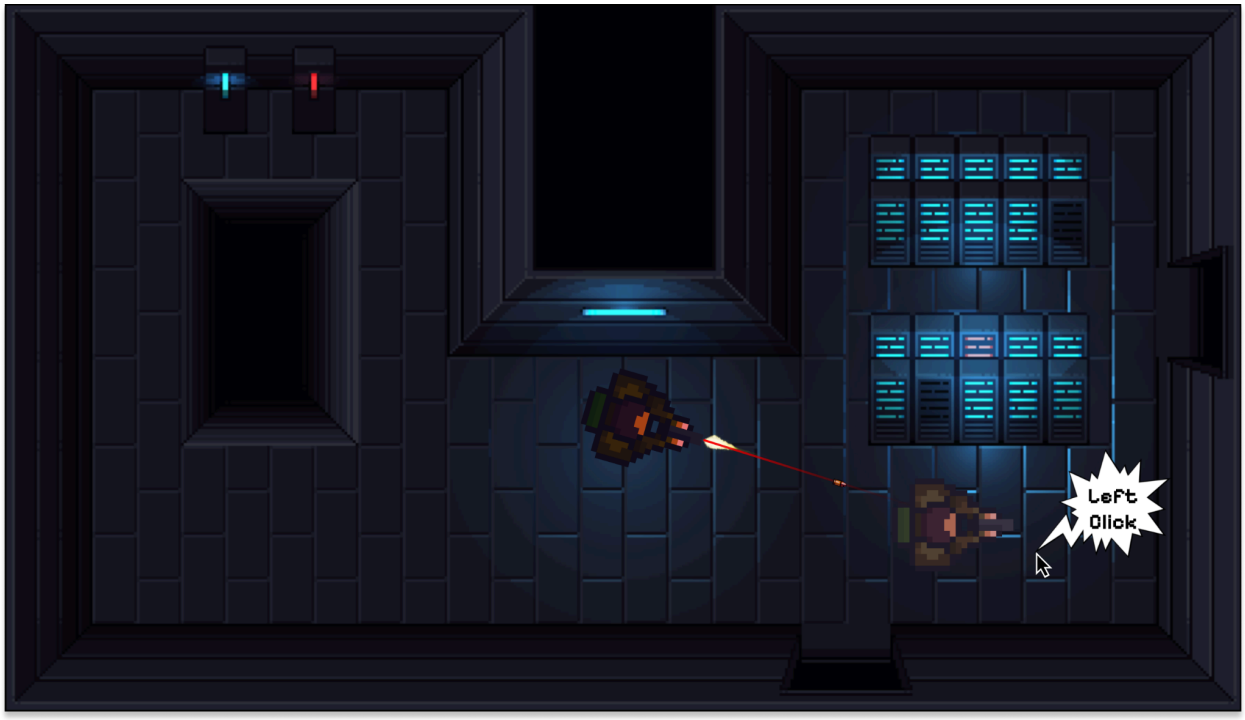
Title Screen:

Background image made by parigrafpix (Gave permission for use)

Intro screen: Gives the player a brief introduction to the story.



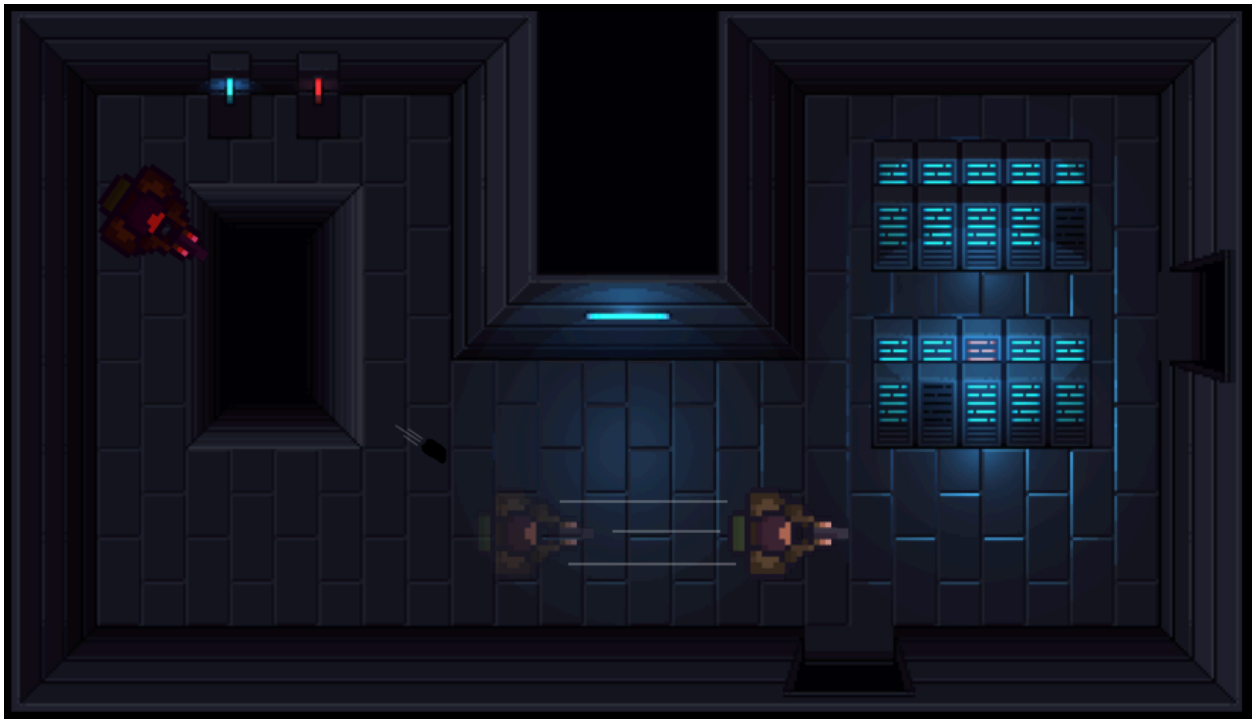
Gameplay 1 (Player Shooting):



Gameplay 2 (Player Melee):

Enemy death:

Gameplay 3 (Player Dodging):



Player dodge initiated with the press of the space bar.

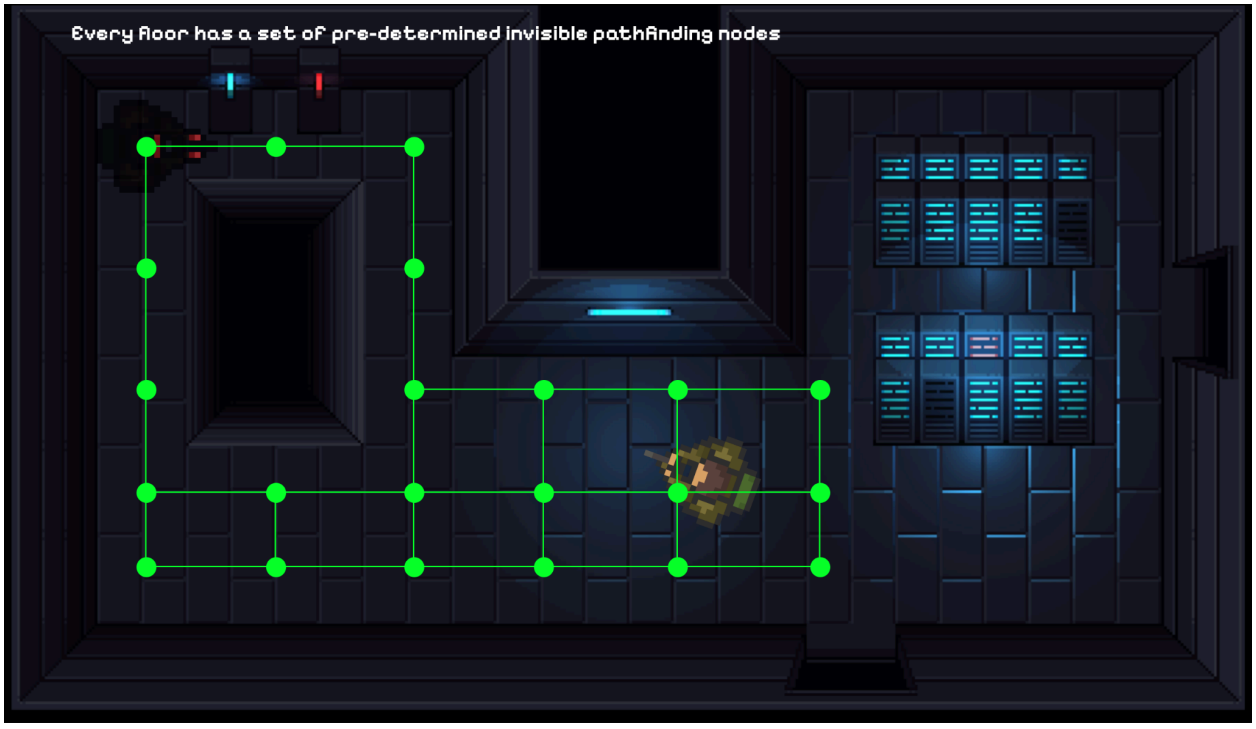
Gameplay 5 (Enemy Alert):

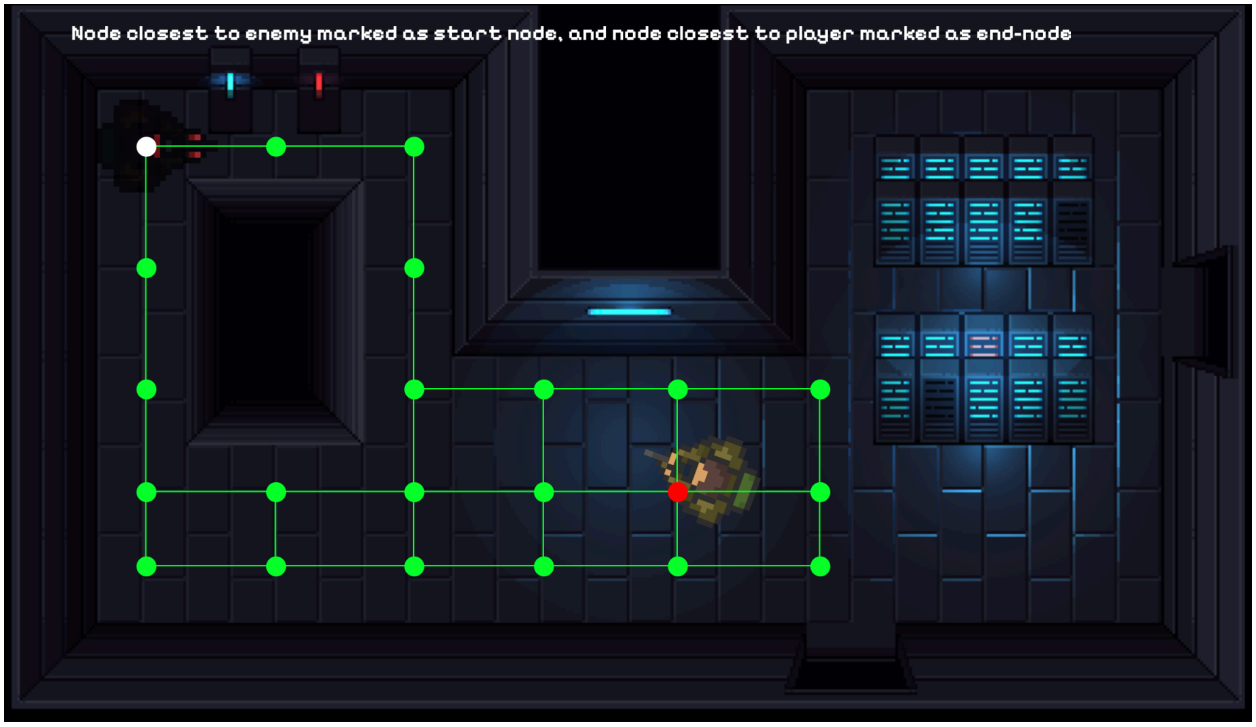
Gameplay 6 (Enemy Shooting):

Gameplay 4 (Player getting hit - health go down):

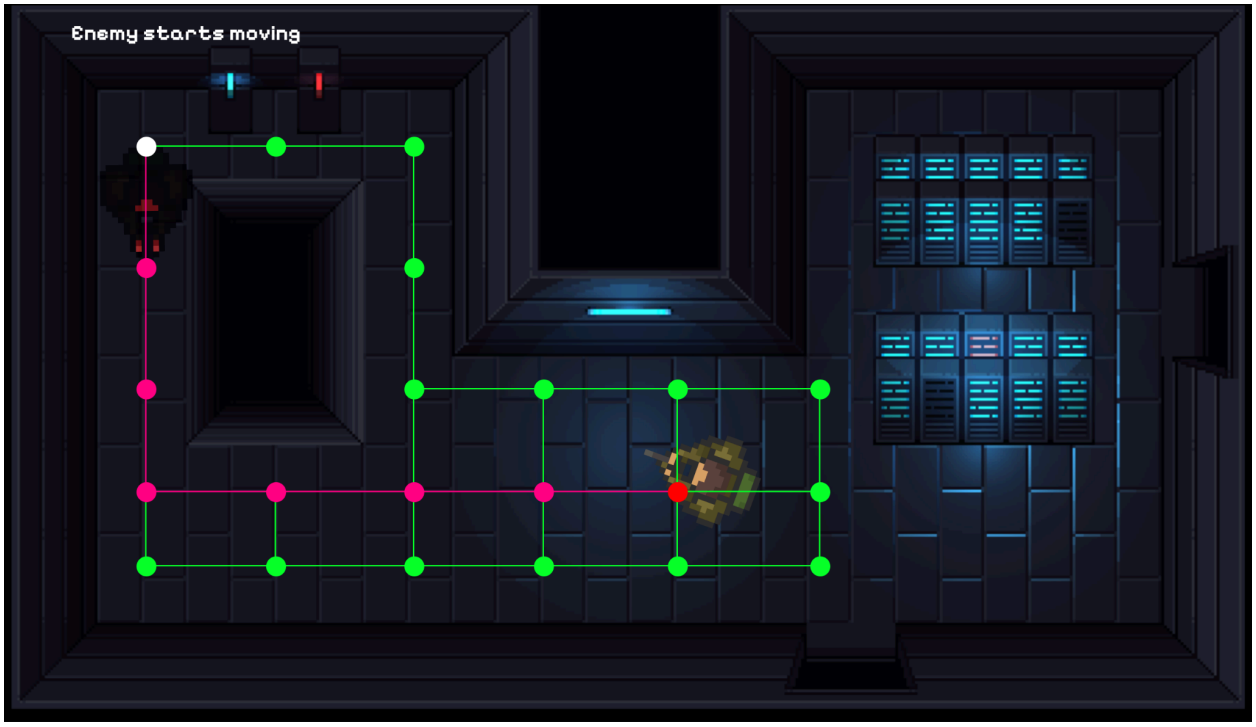
Gameplay 7 (Enemy Pathfinding):


Enemy switches status to Aggro'd
(The exclamation mark is not part of the game)

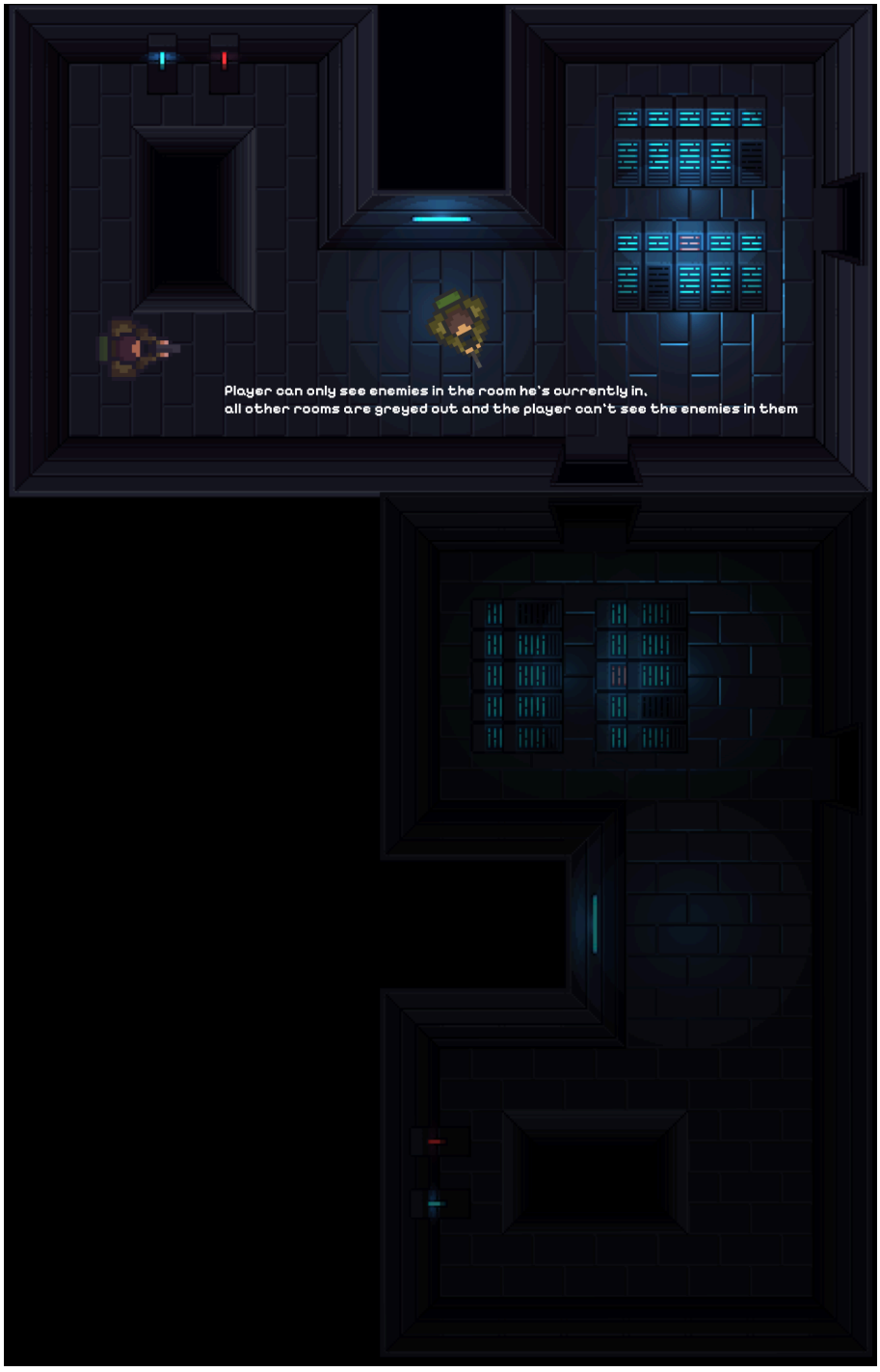

Every floor has a set of pre-determined invisible pathfinding nodes

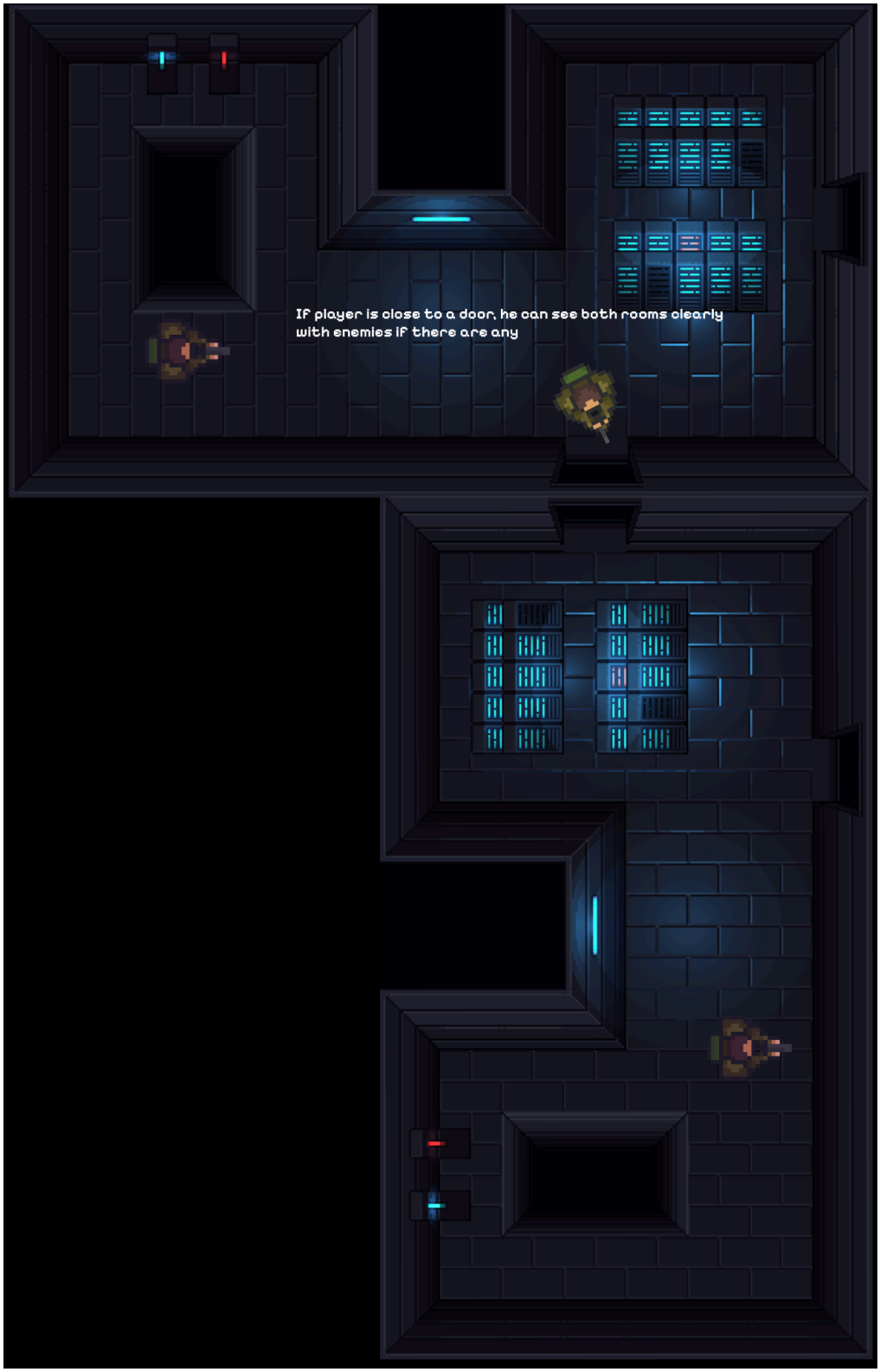

Node closest to enemy marked as start node, and node closest to player marked as end-node

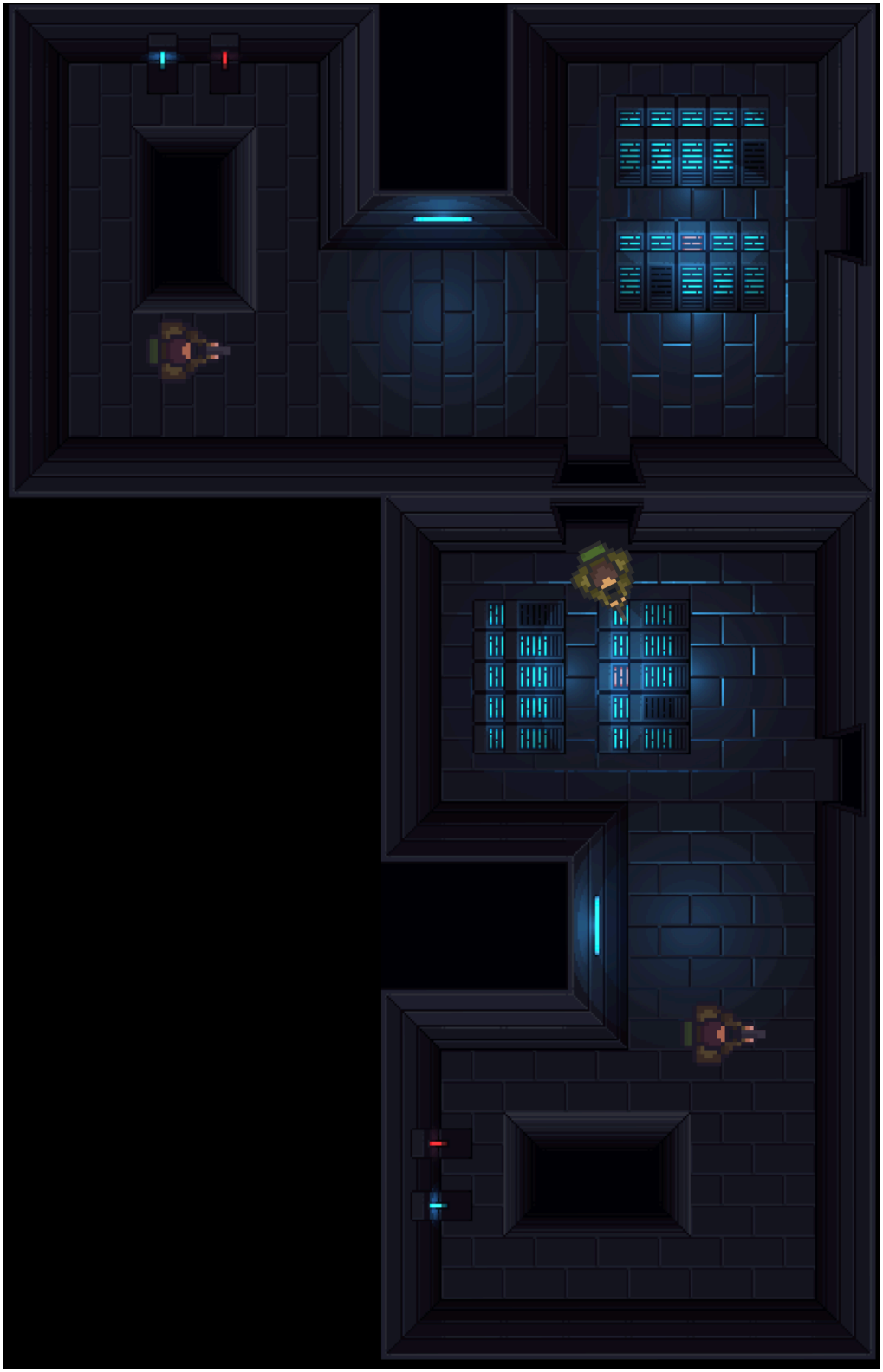

PathAnding algorithm Ands optimal path

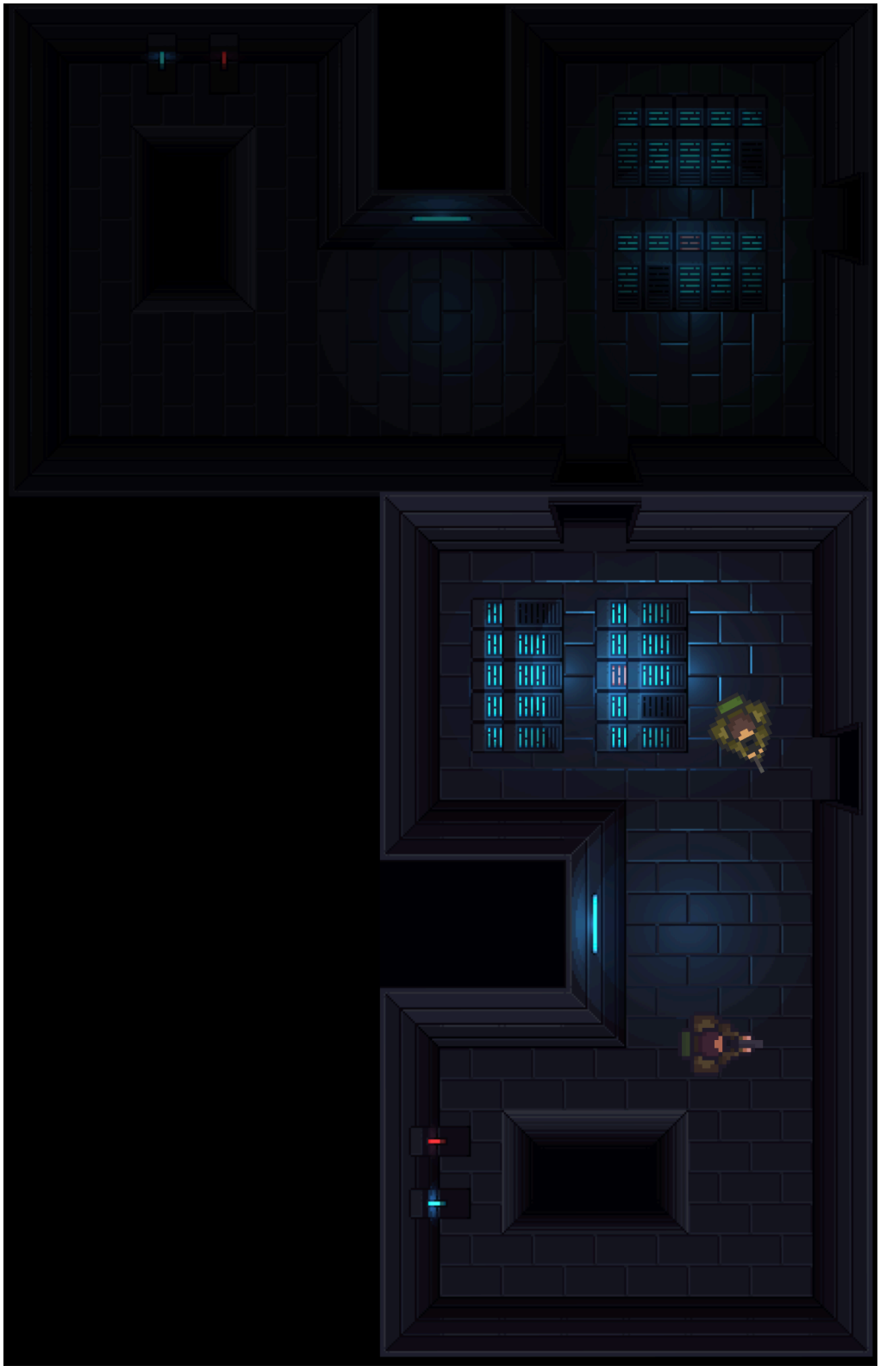

Enemy starts moving

Gameplay 8 (Room visibility): Matt



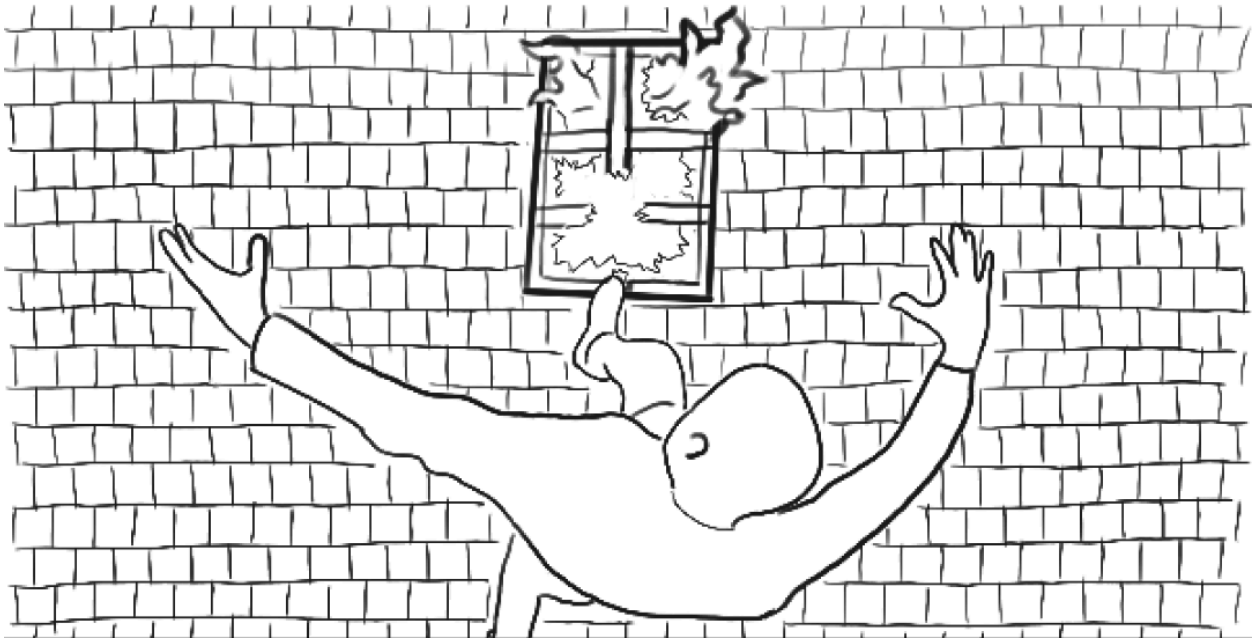

If player is close to a door, he can see both rooms clearly
with enemies if there are any



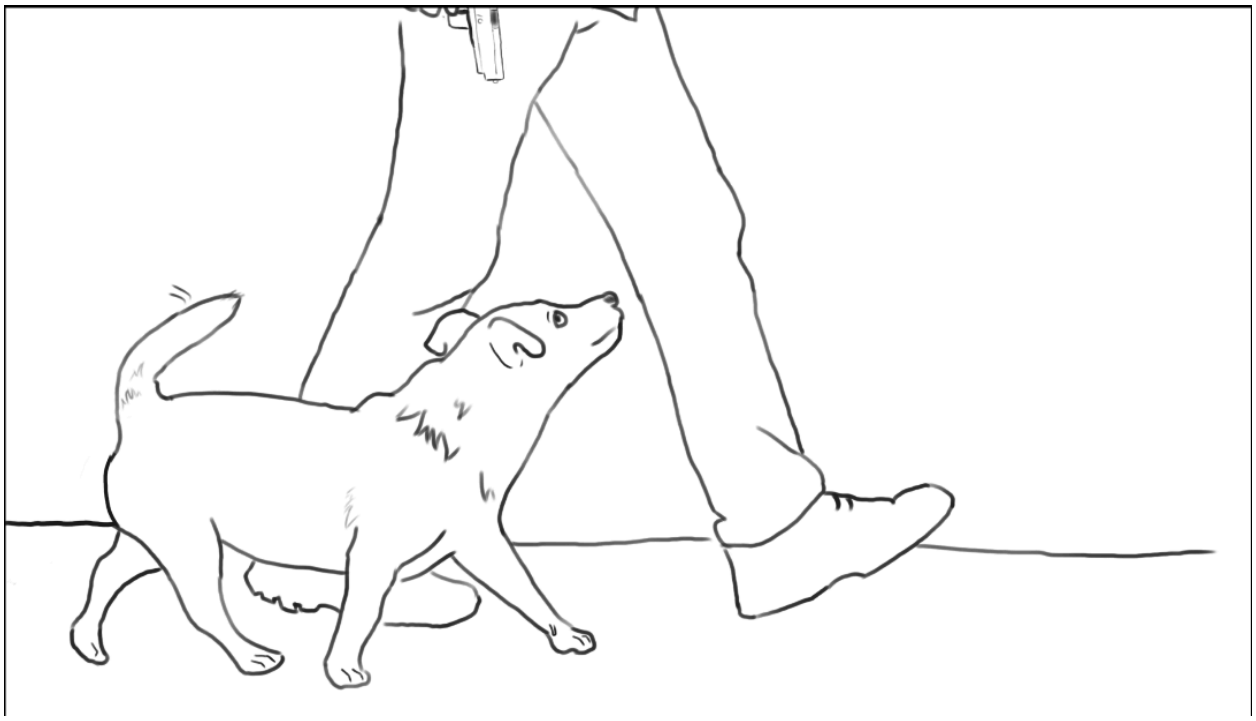


Gameplay 9 (Boss Level):

Game Over Screen



Victory Screen:



Technical Elements:

Rendering:

We will have a rendering loop, iterating over all components with the "Render" component and drawing them to the window. We also plan on using a variety of vertex and fragment shaders for special effects when hitting enemies, when the player gets hit, etc. The rendering will happen all in one system, so even though the "Projectiles" or "Guns" systems might update a bullet's position component or animation frame component, it will only get rendered during the "Rendering" system.

Sprites:

Most of the elements we render will have sprite textures, we plan on abstracting this in a function that allows us to pass a sprite location in memory and the coordinates of where we want to draw it on screen. This will take place during the rendering system. Entities that have sprites will have a "Sprite" component saying the name of the sprite, but the sprite itself will be stored in memory. This way we don't have to store a copy of each sprite for each entity that has that sprite.

2D Collisions:

Every "collidable" entity will have a collision boundary separate from its texture/sprite. To keep calculations fast and implementation simple, the collision shapes will be limited (at least initially) to just rectangles and circles. In the future, we can also implement more complex collision boundaries as long as their shape is convex and not concave.

Gameplay logic:

We will keep the logic for each major gameplay in its own system. We will keep rendering logic in its own system. This will all happen in a main "Game loop", the game loop will iterate over all the systems, with "rendering" being last or one of the last to make sure that the other systems have updated all the animations, textures and positions necessary. The game loop will run as fast and as often as the CPU/GPU allows it, but the frame rate of the game (controlled by the rendering system) will always be locked to a fixed number (probably 60 fps). This keeps the physics engine separate from the frame rate, and also makes sure that our sprite animations look normal and not sped up/slowed down.

AI:

Each enemy type has to have its own AI behavior, this involves aggro range, accuracy, turn speed and pathfinding. We will have to implement different pathfinding algorithms, and make sure that the AI plays to their strengths, for example, an enemy holding a shotgun shouldn't try

to maintain a distance from far away, while an enemy holding a rifle shouldn't try to rush right next to the player.

Physics:

Most of our physics technical requirements will come from collisions as well as projectiles. We plan on making projectile bullets that can collide and potentially ricochet off walls with some guns. In order to implement projectiles, we will also have some sort of hitscan function that can detect if a ray has touched a "collidable" element. Enemies and the player will also collide with the environment when moving, not allowing them to go through walls and other obstacles.

Advanced Technical Elements:

Procedural level generation:

- We plan to use a constraint satisfaction problem approach to creating entire floor layouts on the fly.
- Initially, we want to create manually made maps, and spawn obstacles and enemies procedurally in the room.
- After, we plan to provide to the algorithm a fixed floor layout, and allow it to place rooms as it sees fit.
- The final implementation would create the entire floor procedurally, first creating the floor layout, including room locations and shapes, the main hallways, etc. And then also generating decorations, obstacles and enemies inside each room.
- If we don't finish this mechanic, the game will be less replayable, but it won't be too impactful as the main appeal of our game is the satisfaction the player gets from using the unique guns, as well as the challenge of beating the levels.
- The alternative would be to do everything manually, including enemy spawn locations.

"Level of Detail" sprites:

- In order to make sprites look good on all resolutions, we would like to have the same sprite in different resolutions, so it doesn't look "upscaled" in high resolution screens.
- Alternative: Leave it as it is and have the game look more pixelated in high resolution or big screens.

Devices:

Supported input device: keyboard and mouse.
Mappings:

- Mouse Movement: Controls the aim of the weapon
- Mouse Left Click: Fire weapon
- Mouse Right Click: Melee attack
- Keyboard W: Moves the player up

- Keyboard A: Moves the player left
- Keyboard S: Moves the player down
- Keyboard D: Moves the player right
- Keyboard E: Pick up weapon and Throw weapon
- Keyboard R: Reload weapon
- Keyboard SpaceBar: Activates the players Dodge
- Keyboard Esc: Pauses the game

Tools:

We do not currently have in mind any libraries we will use. But we will most likely find some useful libraries in the future, and will document them here.

Team management:

Assigning and tracking tasks:

Group will use Jira to create a backlog of all tasks that need to be done to complete the project. Every week, we will meet at the beginning and discuss which tasks are the priority to complete the next milestone. Among these "priority" tasks, team members will be able to pick up those that they are interested in, and we can then distribute the rest that weren't picked up among team members with a lower workload. Of course, team members are free to pick up other tasks from the backlog if they finished their priority task.

Description of internal deadlines and policies used to meet the goals of each milestone: Every week, the group will meet to allocate tasks and create specific deadlines based on progress and expected time to complete tasks. Initially, each team member will get to choose which part they want to work on, and then the team will ensure that all tasks are being assigned.

If the team is failing to meet the deadlines, the team will meet and plan what needs to be done in order to ensure that milestone goals can be met. If a certain individual misses consecutive deadlines, and communication is falling apart, the group will consult with the TA how to proceed.

Development Plan:

Milestone 1: Skeletal Game

Week 1 (Due Feb 2)

- Be able to render objects with sprite textures
- Simple level box
- Collision detection
- Player input

Week 2 (Due Feb 9)

- Player: Movement, dodging, shooting
- Player health and enemy health. Player death and enemy death.
- Simple enemy. alert and shoot in player's direction

Milestone 2: Minimal Playability

Week 1 (Due Feb 16)

- Multiple rooms with randomly spawning enemies
- Implement 2 floors
- AI behavior: alert, movement, pathfinding

Week 2 (Due Feb 23)

- Room visibility basic, just grey out rooms player is not in
- Simple death and game win cutscenes
- Basic boss level and game completion

Week 3 (Due March 2)

- Different weapons, useable by both player and enemy
- Higher detailed sprites and animations

Milestone 3: Playability

Week 1 (Due March 9)

- Procedurally generated floors
- Fully implemented AI behavior

Week 2 (Due March 16)

- More particle and hit effects for weapons
- Refine boss level fight gameplay, add phases and more attacks

Week 3 (Due March 23)

- Shop and modifiers

Milestone 4: Final Game

Week 1 (March 30)

- New game+ boss sprite change
- Refine shop and modifiers

Week 2 (April 6)

- Pacing, boss fight difficulty
- Refined cutscenes