Ian McKibben, William McLaughlin
CSCI 3010 - 5 October 2021

Project Proposal

→ Section 1: Description ←

Our project is a 2D Roguelike game featuring procedural generation in which a player must attempt to traverse through the game to reach as far as possible, attempting to achieve a higher score by clearing areas and overcoming enemies and obstacles.

→ Section 2: Technologies ←

We are planning on using the Unity framework with C#. We both are quite familiar with the Unity framework and are planning on using the built in Unity framework in order to deal with physics and instantiation of our objects. We will use various image creation tools such as Inkscape, Piskel, and Photoshop to create artwork. Furthermore, we will use Jasmine for unit testing.

Tool Name	Usage	Familiarity - Ian	Familiarity - Will
Unity	Game engine, GUI, Physics	Experienced	Experienced
C#	For Unity	Familiar	Familiar
Inkscape / Piskel / Photoshop	Art	Familiar	Familiar
MSTest	Unit Testing	Unexperienced	Unexperienced

→ Section 3: Essential Parts ←

- Functional Moving Player
    - Our project will not work unless it has a moving player that is controlled by the user. This will include having a “WASD” / arrow key control scheme, collisions with enemies and objects around the player, and a viewport which reflects how the player moves.
- Hazards / Enemies
    - Our project will not work unless there are enemies and hazards which attempt to hinder the player from attaining a higher score. They will detect collisions with other objects such as the player and the game state will change accordingly.
- Procedurally Generated Map
    - Our project will not work unless there is a map that the player is placed in, as this will give the player bearings, and all actions that the player makes will be encompassed within the map, such as moving around and interacting with other entities.

- Score
    - Our project will not work unless there is a score metric that is associated with the player. This score will drive the player's decisions and will be a metric which increases when the player defeats an enemy or reaches a new area.
- Leveling
    - Our project will not work unless the player is able to level up and choose how they want their character to progress through leveling their abilities. This is essential to reward the progression of the character and also to provide aid as the enemies become more challenging over time.
- Lose Condition
    - Our project will not work unless the game has a lose condition, like running out of hitpoints. Otherwise, the game will last forever and will not meet our expectations for the experience of the player, where there will be an external engine which challenges the player.
- Menu
    - Our project will not work unless there is a way to stop, restart, and exit the game. This is necessary because without, the player will not easily be able to exit the game and the game will start on startup perhaps before the player is ready.

→ Section 4: Outside Resources ←

Because our game will be built from the ground up from the Unity framework, we will only really need artistic assets, such as sprites and potentially music. If we use external resources, we will cite our resources.

→ Section 5: Architecture ←

The game will be launched with an executable file, all menus and settings are run within the game and will appear on launch, all user interfaces will be made and run in Unity, with art done in inkscape and photoshop. The user interface will only be for traversing menus for the game, and therefore only has minimal interaction with objects which can be called within the game program. Everything is centralized and run through Unity so no interaction with other technologies will be necessary.

Classes:

- **Entity**
  - **Description:** All entities have collision and interaction functions that will be defined given the type of the “other” collision object
  - **Inheritance:**
    - Parent of: Player, Enemy, Trap
  - **Data:**
    - Vector2 position
    - int size
    - int speed
    - Sprite sprite
  - **Functions:**
    - void Move(Vector2 translate)
    - void Draw()
    - virtual void Collide(Entity &other)
  - **Design Patterns:**
    - Factory
- **Player**
  - **Description:** Entity controlled by user
  - **Inheritance:**
    - Child of: Entity
  - **Data:**
    - int health
    - int level
    - Skills skills[]
    - Weapon weapon
  - **Functions:**
    - bool IsAlive()
    - void Collide(Entity &other)
- **Enemy**
  - **Description:** Entity not controlled by user; different subclasses for enemy types, defining it’s behavior
  - **Inheritance:**
    - Parent of: enemy subclasses
    - Child of: Entity
  - **Data:**
    - int health
    - int level
  - **Functions:**
    - bool IsAlive()
    - void Destroy()
    - void Collide(Entity &other)
  - **Design Patterns:**
    - Factory

- **Trap**
  - **Description:** On collision with player or enemy have an overloaded behavior, such as damage taken; different subclasses for trap types, defining it's behavior
  - **Inheritance:**
    - Parent of: trap subclasses
    - Child of: Entity
  - **Data:**
    - int level
  - **Functions:**
    - void Collide(Entity &other)
  - **Design Patterns:**
    - Factory
- **Projectile**
  - **Description:** On collision with another entity will have overloaded behavior, as well as a lifetime and velocity
  - **Inheritance:**
    - Child of: Entity
  - **Data:**
    - Vector2 direction
    - int lifetime
    - Entity *parent
  - **Functions:**
    - void Collide(Entity &other)
  - **Design Patterns:**
    - Factory
- **Map**
  - **Description:** Contains the 2D structure for the map, the seed, as well as functions to generate parts of the map when lazy loading requires
  - Functions to update and get values to be called by entities
  - **Inheritance:**
    - Parent of: possible map subclasses for map types
  - **Data:**
    - Tile Map[][]
    - int seed
  - **Functions:**
    - void generateMap()
    - Tile getValue(Vector2 position)
    - VoidsetValue(Vector2 position, Tile tile)

- **Tile**
  - **Description:** Contains the sprite for a tile to be put into the map
  - **Inheritance:**
    - Parent of: possible tile subclass types for different tiles
  - **Data:**
    - Sprite sprite
  - **Functions:**
    - Tile()
    - Draw()
  - **Design Patterns:**
    - Factory
- **Weapon**
  - **Description:** Contains weapon statistics and behavior functions for how the weapon fire in game
  - **Inheritance:**
    - Parent of: Weapon subclasses
  - **Data:**
    - int damage
    - float attackSpeed
  - **Functions:**
    - void Fire()
  - **Design Patterns:**
    - Factory
- **Skill**
  - **Description:** Contains modifier for what skill does, to be attached to player when skill is being chosen
  - **Inheritance:** None
  - **Data:**
    - int id
  - **Functions:**
    -
- **GameState**
  - **Description:** Keeps track of active players, enemies, traps, and map data to effectively capture the game state
  - **Inheritance:** None
  - **Data:**
    - Player player
    - Enemies enemies[]
    - int score
  - **Functions:**
    - void Tick()
  - **Design Patterns:**
    - Prototype
    - Iterator

- **GameManager**
  - **Description:** Contains member which represents the current game state, as well as if the game is paused or not, or if the game is in the menu
  - **Inheritance:** None
  - **Data:**
    - GameState currentState
    - bool isPaused
  - **Functions:**
    - void startGame(int seed)
    - bool saveGame(int slot)
    - void switchState(state)
    - GameState loadState(int slot)
    - void Tick()
    - bool pauseGame()
    - bool resumeGame()
  - **Design Patterns:**
    - Singleton

→ **Section 6: UI Design** ←

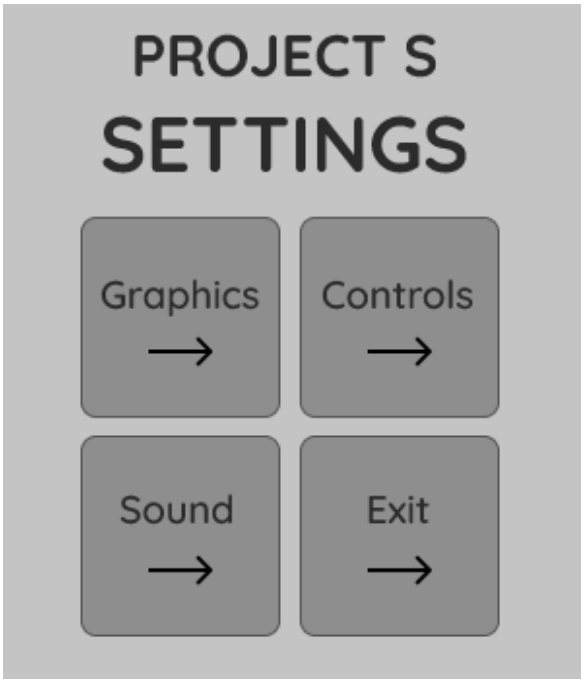
Menu Name	**Description**	**Options**
Main	Appears when the game is launched, or when the player exits to the main menu from the game.	> Start Game: Begin a new game > Load Game: Goes to the load game menu > Settings: Adjust the game's settings > About: Opens information on the game > Exit: Closes the game
Pause	Appears when the player pauses during the game	> Resume: Continues the game > Save Game: Goes to the save game menu > Load Game: Goes to the load game menu > Settings: Adjust the game's settings > Exit to Main Menu: Exit back to main menu > Exit to Desktop: Exit the game completely
Settings	Appears when the player selects settings from the main or pause menu.	> Graphics: Adjust graphic settings > Controls: Edit key mappings > Sound: Adjust game audio
Load Game	Appears when the player selects load game from the main or pause menu, allows the player to select a previously saved game to load.	> Save List: List of saved games to select from to load into the game > Start Game: Starts the game with the selected save file
Save Game	Appears when the player selects save game from the pause menu, allows the player to save their current game.	> Save List: Display the list of existing save games > Name Box: Box to enter the name of the save > Save Game: Saves the game with the given name

[Link to UI figma prototype]

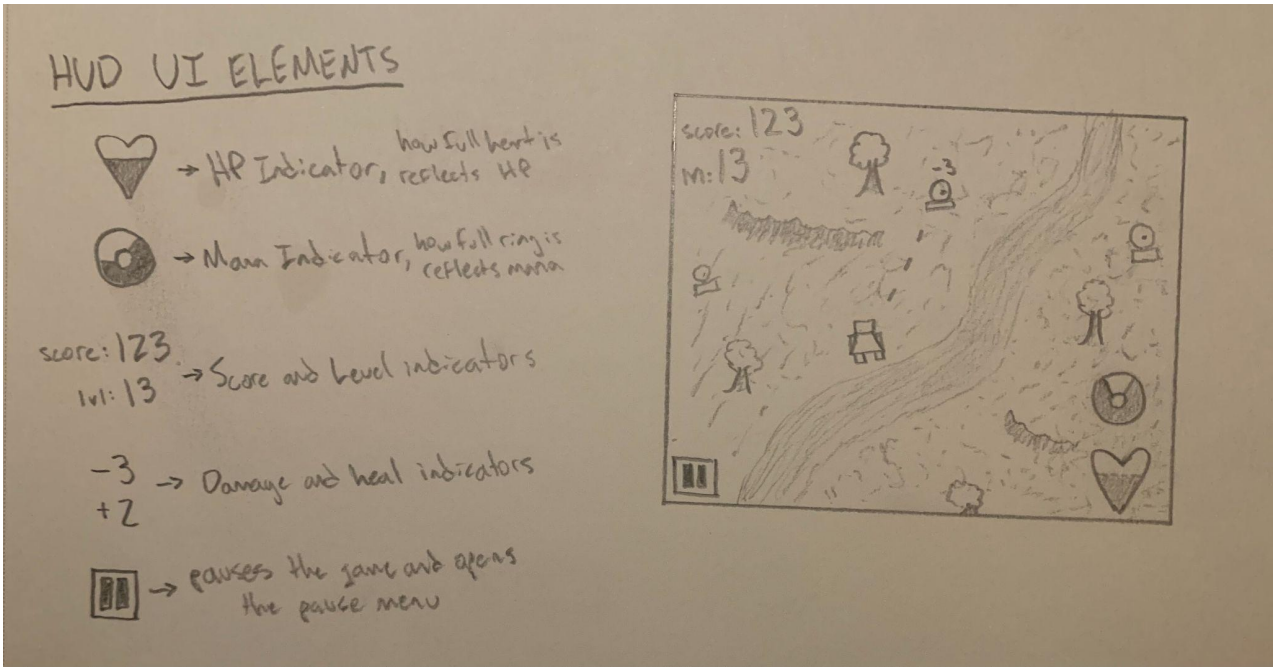
Main Menu	Pause Menu
	

Load Game Menu	Save Game Menu
	

Settings Menu



HUD UI



→ Section 7: Detailed Plan ←

	Due	Time	Complete	Knowledge
Homework 2	10 / 10	2-3 h	File setup/github - Ian Initial player obj - Both Player move - Both	How to handle objects and movement in Unity [comfortable]
Homework 3	10 / 22	12 d	Collision - Both Player hp - Both Point system - Both Basic menu gui - Both Art concepts - Both	Unity collision handling, Unity GUI, Variable handling with objects in Unity [familiar]
Homework 4	11 / 12	21 d	Procedural map - Will Moving enemies - Ian Basic functional art - Both	Procedural generation, Enemy behaviors, Pixel art [somewhat familiar]
HW 5 Check	11 / 30	18 d	Parallel dim mechanic - Both More map complexities - Both More enemies - Both	Abstracting game state, Managing classes, Instantiating prototypes [new]
Homework 5	12 / 9	9 d	Save/load files - Both Settings menu - Both Final art - Both Final UI - Both Weapon types - Both	File interaction with Unity, Menus [familiar]

→ Section 8: Course Engagement ←

We both believe we will be able to stay fully engaged in the course despite our project relating to a different (though similar) language. Since the languages are similar it will still be very valuable to translate what we learn in class to our work as we continue to learn new and better ways of programming. Beyond this, we are both enthusiastic about the project and have experience working in Unity, so though we will put much time and effort into the project we will still have the time and dedication to stay diligent on class work.