Timelock Trials

by Logan Dawes

High Concept:
A puzzle-filled world with turn-based battles where strategy and resource management will be your key to success. Every move you make matters, whether it be in fast-paced battles or in thought-provoking puzzles.

This game will be a top-down RPG-style game with turn-based combat elements. It will feature a strategic way to battle enemies, a world with puzzles to solve and places to explore. In battle, you will have to manage your health and other stats and make decisions on what moves to use on your opponent.

Controls: W(up), A(left), S(down), D(right), E(interact), Space(pause)

Platform: PC or Game Console
(Could be mobile given time for UI adaptation)

Target Demographic: Teens to Young adults who enjoy games with stat management, player progression, and tactical combat. These would be players who enjoy an interactive game world and skill-based battles with an adequate level of strategy.

Features:

- Game world with various puzzles
- Menu to view your character's statistics
- Turn-based Battle system
- Player Stats: HP, SP, DEF, ATK. Abbreviations for Health points, Special points, Defense, Attack.
- Pause functionality
- Player inventory

Unique points:

- Special points for managing special moves
- Puzzles in the form of riddles, memory puzzles, and exploration.

Game Play:

System: Overworld

Subsystem: Tile set

Overworld will be implemented using a tile set for quick assembling of levels and easy to define collisions for walls. Has a collision layer “Wall” and one without collision “Ground”, where the player moves around. This is done using a composite collider to combine colliders for each tile.

Subsystem: Player movement / collision
with PlayerMovement.cs

Top-down style perspective, which means gravity should be ignored. Collisions should be implemented with the walls of the level and to detect triggers such as enemies and intractable items. Every game object should be tagged appropriately to facilitate. Personal note to make diagonal movement an equal movement speed.

Subsystem: Camera
with CameraController.cs

The camera will follow the player smoothly, instead of being locked to the player as a child object. It does this using Lerp(), linear interpolation. The camera is made to be persistent across scenes. SetTarget() is used to move the camera’s target to a new anchor.

Subsystem: Enemy movement / collision
with EnemyAI.cs

Allows the enemies to patrol a certain area and engage battle with the player on contact. Composed of a roam and chase speed, detection range, 2D roam area, roam target (point within roam area), roam delay (how much time the enemy idles before moving again), and an ID and type.

Several methods for roaming, where the enemy picks a new target in the roam area and then picks a new target upon reaching it. The roaming can be interrupted by chasing, where the enemy uses the chase speed and approaches the player, as long as the player is in the detection range.

When colliding with the player, triggers LoadBattleScene() inside GameManager, passing the enemy type to determine the battle that starts.

The enemy ID is used to register when the enemy is destroyed so that it doesn’t get reloaded upon entering the scene again.

Subsystem: Doors
with Door.cs

has two colliders, one for trigger and one for colliding with the player. Designed to change sprite and disable physical collider when the player presses ‘E’ while in the trigger collider area.

can be locked, where a child object lock is enabled and the door can’t be opened without a key in the player’s inventory. Registers open/close state with GameManager to persist across scenes.

Subsystem: Keys
with Key.cs

has a collider that detects when in contact with the player, and if so adds a Key to player's inventory. Registers collection with GameManager so that it can't be collected again.

Subsystem: Barrels
with Barrel.cs

has two colliders, one for trigger and one for colliding with the player. Designed to play a small animation and grant the player associated item (string) when the player presses 'E' while in the trigger collider area. Registers emptied state with GameManager so that it can't be collected again.

Subsystem: Puzzles
with ButtonPuzzle.cs, ButtonScript.cs, ClockPuzzle.cs, Clock.cs

A puzzle manages several game objects and checks for the correct solution.
Button puzzle: manages a correct sequence of button presses, shows the player, and the player repeats it to go to the next phase or complete the puzzle.
Clock puzzle: manages four clocks and the player has to input the right times to complete the puzzle.
When a puzzle is completed, it opens a locked door marked as an event door (to prevent using keys on it)

Subsystem: Scene switching
with GameManager.cs

Handle transitions between different levels, game scenes such as battle, etc. HashSets are stored to register enemies, doors, keys, barrels, signs, and puzzles to maintain game state when switching between scenes.

Subsystem: Staircases
with StaircaseTeleport.cs, StaircaseTarget.cs

Attached to starcase objects that take the player to an associated staircase in a different scene. Uses staircaseID to define the staircase itself, and targetStaircaseID to define which staircase it links to. It passes these as arguments to LoadScene() inside GameManager. Functionality separated into StaircaseTeleport and StaircaseTarget in the case you want to make a one-way staircase, in that case one staircase would have the teleport script and the other has the target script.

Subsystem: Signs
with Sign.cs

A sign pops up a UI with text that can be closed with E again. Implemented to automatically pause the game, and pausing with a sign open should not change the paused state.

System: Menu

Subsystem: Main menu
with GameManager.cs, OptionsButton.cs, QuitButton.cs, StartButton.cs

Main menu scene with start game button, options button, and quit button. Start game puts you back in the game overworld where you left off, or at the default start location if you just started. Options button toggles the pause menu, while in game you open it with space key. Quit button takes you back to desktop.

Subsystem: Pause
with GameManager.cs, MenuButton.cs

Allows player to pause the game world and stop all actions, such as enemy movement or in battle. Should not always be allowed, ex: during battle. Pausing allows access to other menus / options such as inventory while in game world, volume, and back to main menu button.

Subsystem: Display player statistics
with GameManager.cs

Shows the player's current stats in pause menu, which will be Health points (HP), Special points (SP), Defense (DEF), and Attack (ATK).

Subsystem: Item Inventory
with GameManager.cs

Shows the player's collected items in pause menu. Planned items will be a key, which will be checked and removed to open certain doors, Armor which will change the DEF of player, and others.

Subsystem: Volume
with GameManager.cs and VolumeController.cs

Adjusts the volume of all sound sources by changing the audioMixer master volume. All audioSource components link to the audioMixer so that their volume can be altered

System: Battle

Subsystem: Battle UI
with BattleUIController.cs

Controls player selection in battle menu using WASD and E inside the BattleScene. Has menu and submenus within each menu element. Currently Attack, Defend, Stats, and Run.

Subsystem: Player Statistics
with BattleUIController.cs

Display and manage the player's stats during battle when the menu option is selected. HP/SP/DEF/ATK

Subsystem: Run
with BattleUIController.cs, EnemyAI.cs

When you run from an enemy, it exits the battle then temporarily disables the enemy before continuing to roam again. It does not register as destroyed.

Subsystem: Enemy field
with Enemy.cs

Display and manage each enemy's stats during battle. HP/DEF/ATK. Also manages the enemy's type, and initializes the position of the enemy, all based on the enemy type that is passed into the battle when it starts.

Subsystem: Turn order
with EnemyTurn.cs

Determines the order in which actions take place between the player and each enemy, based on pre-defined rules. A set rotation where the player starts and then each enemy from top to bottom of the field. Also determines which actions based on a random value to give chances for the enemies to perform special actions.

Subsystem: Game Over
with GameOverMenuManager.cs, BattleUIController.cs

Triggers as player's HP is <=0 and ends the battle scene. Opens a Game Over UI where the player can reset back to starting state, or quit.

Subsystem: Battle Victory
with BattleUIController.cs

Triggers after the player has defeated all the enemies, displays victory screen and returns to overworld scene

Subsystem: Player moves / Special moves
with BattleUIController.cs

Define which actions the player can perform on their turn, which includes normal attacks, special attacks, or defensive option to temporarily increase DEF to recover for the turn. Jab: attack one enemy with full attack, Slash: attack all enemies with ⅓ attack, Defend: temporarily gain defense, gain SP, Heal: use SP to heal HP.

Subsystem: Enemy moves
with EnemyTurn.cs

Determined by random variable (to calculate odds of attack) and the enemy type. Miss: can be done by any enemy, enemy doesn't do any damage, Lifesteal: can be done by bats, they heal the amount of damage they dealt, Spawn: can be done by insects, they summon a new insect into battle at the topmost available slot.

Game Content:

The two primary levels (scene components), would be the Overworld, the main world where the player moves and interacts with objects, and the Battle Scene, entered upon contact with an enemy in the Overworld and manages turn-based combat with the enemy. Although a much smaller aspect of the game, the menus would also be a notable scene component.

The Overworld is split into 3 different scenes, Level 1, Level 2, and Level 3. Level 1 introduces a lot of the components, and includes a hidden wall puzzle solved by deciphering a sign and looking at the shadows next to walls. Level 2 adds some of the different battle types and the button puzzle, solved by repeating the sequence provided. Level 3 includes the clock puzzle, where the player goes around to find out what times go on each clock.

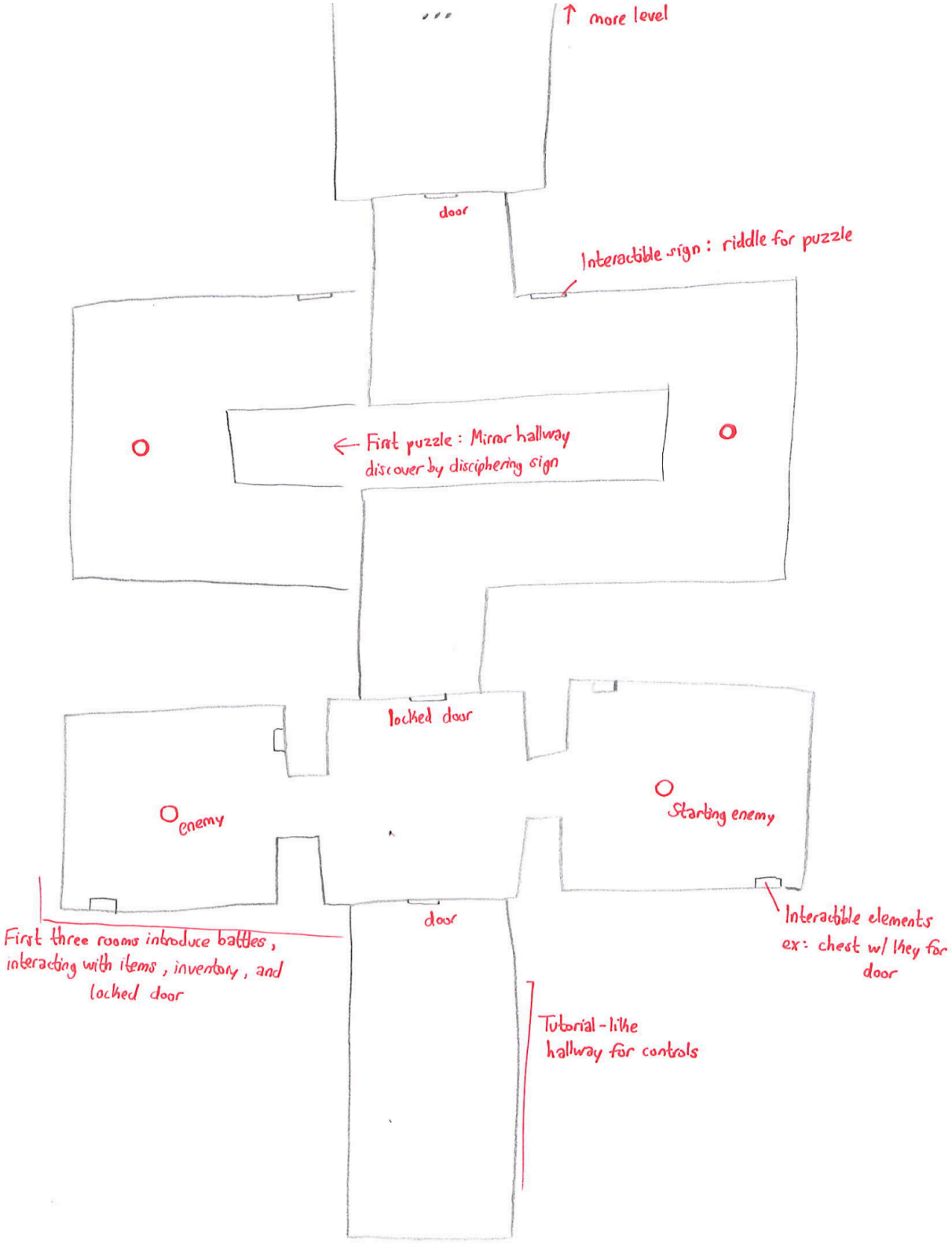
CREATED ASSETS:

Sign sprite
Clock sprite / Clock hands sprites
Sound effects

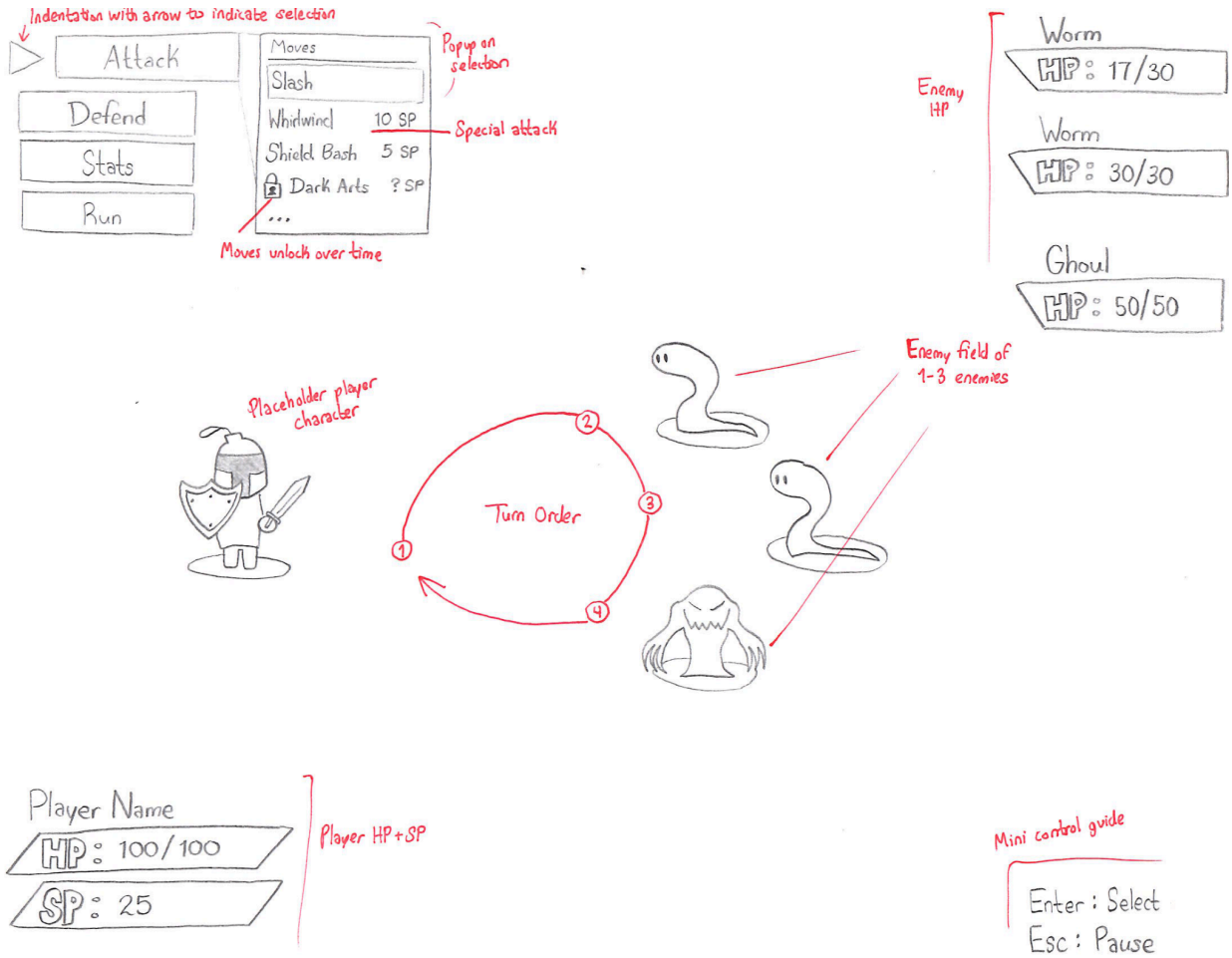
IMPORTED ASSETS:

Tilemap mini dungeon by aztharis
16x16 Knight by pixelfelix
Undead Asset pack by DeepDiveGameStudio
Key Items (16x16) by DantePixels
Clock ticking sound, Clock ringing sound

Overworld Sketch:



Battle Sketch:



Timeline:

10/06–10/12: Start initial pre-planning, get assets, game design plan

10/13–10/19: Implement overworld tile set, basic player movement, collision, enemy movement, camera behavior.

10/20–10/26: Implement basic triggers for interactions with items, make inventory, debug puzzle for puzzle reward mechanic, scene switching basics.

10/27–11/02: Implement Menus (pause, items, stats, volume, save&quit), debug battle scene

11/03–11/09: Implement logic for player turns in battle, turn order, enemy field, enemy attacks, Submit partial working prototype

11/10–11/16: Continue implementing battle systems

11/17–11/23: Turn debug elements such as battle and puzzles into real battles and puzzles, start on minigames.

11/24–11/30: Complete different moves and dodging minigames and finalize all systems. Make final overworld.

12/01–12/06: Polish out game bugs and submit final game code and review video

Appendix A - Implementation discussion:

- PlayerMovement.cs
  - Implements basic player movement
- CameraController.cs
  - Implements smooth camera movement and a method to set the target of the camera. Used for switching between scenes.
- EnemyAI.cs
  - Implements roaming and chasing of an overworld enemy to the player. Includes data passed to battle scene and ID to register as destroyed to game manager.
- StaircaseTarget.cs
  - Used to store a staircase ID on a staircase object.
- StaircaseTeleport.cs
  - When the player touches a staircase, it calls LoadScene() on game manager with the staircase data.
- Door.cs
  - Implements a door with two colliders. One for detecting E press and the other to stop from passing through it. Can be locked so that it checks for a key in player's inventory from game manager. Can be marked as an event door, so that it can only be unlocked from an external method call. Has a unique ID registered in game manager when opened.
- Key.cs
  - Collectible item, implemented with trigger collider. Has unique ID to register when collected to game manager.
- Barrel.cs
  - Implements with two colliders similar to door. On interacting it gives the specified contents to the player's inventory then is marked as empty, and ID is registered with game manager.
- GameManager.cs
  - Stores hash sets of all modifiable objects that need to maintain state between loading scenes, sets the object's state upon loading scenes.
  - Implements loading different scenes that require different things to be instantiated, moved, etc.
  - Stores a pause menu that can be opened with space, maintains when it should be open or closed. Also makes sure sign pausing and menu pausing do not interfere.
  - Keeps track of the player's stats and inventory and displays them in the pause menu.
  - Manages several player-oriented interactions in the battle scene.
  - Implements game over and reset game progress.

- StartButton.cs
    - Calls StartGame() from main menu
- OptionsButton.cs
    - Pauses game from main menu
- QuitButton.cs
    - Quits the application from main menu
- MenuButton.cs
    - Goes from pause menu in currently active scene to main menu and resumes.
- VolumeController.cs
    - Used to set the audioMixer component that all audio source's use to the value on a slider in the pause menu. Calls on slider changed.
- BattleUIController.cs
    - Allows player input to interact with the battle UI using WASD & E. Update keeps track of what menu the player is currently in (menu, submenu, targeting enemy). Mediates interaction between player's turn and enemy's turn, player is managed by game manager and enemies by enemy.cs and enemyturnmanager.cs
- Enemy.cs
    - Manages the stats of each enemy in battle and their hp bar. Methods are called by BattleUIController. Stats are initialized based on data passed into it from game manager.
- EnemyTurnManager.cs
    - Manages every enemy on the battle and goes through their turns. Decides the enemy action by getting a random number, then checks if the random value is within a certain range. Calls methods in game manager to damage player.
- GameOverMenuManager.cs
    - Implements buttons in the game over menu, one exits application and the other calls ResetGame() in game manager.
- Sign.cs
    - Detects player input 'E', if pressed opens sign menu in game manager, passing the sign message. Also keeps track of an outer glow sprite that flickers when not read. Registers ID with game manager.
- ButtonScript.cs
    - Attaches button to overall button puzzle so that it can be assigned a unique index and sound.
- ButtonPuzzle.cs
    - Manages an array of ButtonScript buttons and a door. When the activation area is entered, it will generate a button sequence based on the phase, show it by glowing the buttons and playing their sound, then allows input by the player and detects the input, If the player's sequence matches it, it increments the phase and repeats

for 3 phases. When the last phase is complete, is calls OpenEventDoor() on the door and registers the puzzle ID as completed with game manager.

- Clock.cs
  - Keeps track of the hour, minute, hour hand, and minute hand of a clock. Detects nearby E press to open clock menu in game manager. Has methods to change and set the time and hands of the clock.
- ClockPuzzle.cs
  - manages four clocks and four solution times. When one of the clock hands is moved, this script checks all the values and stores them with game manager. If they are all correct, it opens attached door and registers complete with game manager.
- ClockMenu.cs
  - Used to increment/decrement the active clock's hands with the clock menu in game manager.
- LevelJump.cs [PLACEHOLDER]
  - For debug purposes, adds methods to set the player's location directly to a certain scene. Does this by passing specific load scene methods. Methods attached to buttons in pause menu.
- TestLocalization.cs [PLACEHOLDER]
  - For debug purposes. Used to toggle the title on the main menu. Intended to display what localization would look like if applied to all text.
- Unimplemented: Saving to Desktop
  - Did not get around to save the game's state outside the application.
- Unimplemented: Attack minigame
  - Time constraints prevented implementation. Would have been corotuine between selecting enemy target and damaging target to determine damage.
- Unimplemented: Dodge minigame
  - Time constraints prevented implementation. Would have been coroutine between enemy attacking to allow the player to possibly dodge taking damage.

Appendix B - Advanced Topic:

TextMesh Pro: Localization to make a different language localization for the game. https://learn.unity.com/tutorial/textmesh-pro-localization

Attempted to implement a localization for the title in the main menu, can be tested by pressing the button next to it. I was able to import a katakana font into my assets and use it for the title. However, I was not able to find a font that properly utilized the unicode hex range. The biggest problem I encountered trying to implement this was getting the proper font assets to use, especially for the dynamic fallback, so it was not implemented.

Appendix C - Final debrief:

Development started with adding the basic player movement, camera controls, and tilemap in order to get basic overworld functionality. Then I added some basic components, such as a placeholder enemy, doors, and keys. A lot of development went into properly loading scenes with the help of a game manager that would not destroy on load. Added IDs to objects, to maintain state when loading. Then I implemented the pause menu and main menu and the buttons inside them. I added sound that could be controlled by the audio mixer. Next I implemented the Battle scene with test enemy, made the battle UI, and went on to make several player and enemy actions. Finally, I assembled the overworld and implemented the button puzzle and clock puzzle.