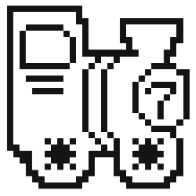
****

****

**Team 3 Design Document**

**Team members:** Ellison Hohmann, Rei Manning, Ryan Parris, John Slater, Danny Spear

**Project Title:** The Purdue Party Game

**Index:**

[**Purpose**](#_ljwe35eyd522) **3**

[**Functional Requirements**](#_ugqvws22hixn) 3

[**Non-Functional Requirements**](#_e72j5cjr1v2h) 2

[**Design Outline**](#_4sa0ktxhqppg) **2**

[**High-level Overview**](#_999zdfgz0lbq) 2

[**Sequence of Events Overview**](#_iwrcwltnnhfa) 2

[**Design Issues**](#_7uotnsqcgqtb) **2**

[**Functional Issues**](#_8rju75ia9rlf) 3

[**Non-Functional Issues**](#_ovtfpo2nmhyo) 3

[**Design Details**](#_d49j833s47m) **3**

[**Class Diagrams**](#_wx7ssucg5o5x) 3

[**Descriptions of Classes and Interaction between Classes**](#_sqycoofiexbi) 3

[**Sequence Diagrams**](#_7xyx5muy0zca) 3

[**Navigation Flow Map**](#_dx6wjljzbkpd) 4

**State Diagram** [4](#_dx6wjljzbkpd)

[**UI Mockup**](#_bj2bfofl3hle) 4

# 

# 

# Purpose:

Video games provide a great opportunity to connect with others, have fun, and destress. Players are able to interact with new, creative worlds with unique challenges and interesting mechanics, creating an enjoyable and immersive experience. Party games, designed to be played by multiple players, are a great way to foster connections through friendly competition. Party games are unique in their ability to bridge the gap between the real and virtual worlds.

Our goal is to design an offline, 2D party game based on the Purdue University campus and culture. Targeted mostly towards students, fans, and alumni, The Purdue Party Game will utilize the familiar sights (and people?) of Purdue to create an immersive world that further connects the players. Played on PC with up to four players, users can use keyboard or USB (and bluetooth?) controllers to traverse a virtual game board and play fun minigames. Players can earn ‘Boiler Bucks,’ a currency which can be exchanged for ‘Degrees’ that are used to determine the winner of the game.

In addition to in-person players, we will also offer CPU players to fill in any missing spots. CPUs will be able to move around the board, play minigames, and earn Boiler Bucks and Degrees just like a normal player. These CPUs will have a variable difficulty setting which determines their skill in minigames, offering options for a more or less challenging experience depending on the users’ preferences.

## Functional Requirements:

## Non-Functional Requirements:

# 

# 

# Design Outline:

Most functionality of our project takes place within the GameMaker framework and environment. Objects, event detection, functions, GUI updates, and during-execution game data storage can all be implemented within the GameMaker Application component.

In order to obtain user input, 1-4 Player Controller(s) are needed. These serve as devices that take in physical input from the player(s) and send that information into the GameMaker Application. With this information received from Player Controller(s), the GameMaker Application handles most of the system calculations and functions.

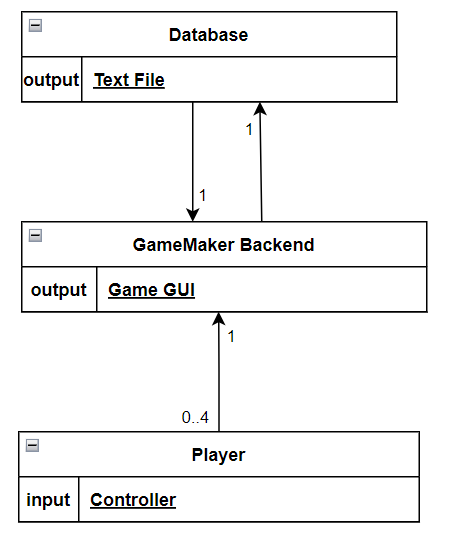
In order to locally save game data between executions of the program, a Database is needed. The conversion of game data into an encrypted file is supported within GameMaker. Whenever requested, the Database will convert and store current game data and statistics into an encrypted file format, storing said file locally on the device on which the application is running. Additionally, the Database will access this local file and send it back to the GameMaker Application when requested in order to restore the application to the state of the saved game.

1. Player Controller
   * Takes input from player’s actions
   * Requires one analog stick and four buttons for all user input
   * Communicates player actions to GameMaker Application
   * 1-4 player controllers supported
2. GameMaker Application
   * Receives data on player actions from Player Controller(s)
   * Receives data on previously saved game from Database when requested
   * Converts player actions to associated character movements/behavior
   * Algorithmically calculates CPU movements/behavior
   * Updates GUI based on player/CPU behavior
   * Displays graphics of game on monitor
   * Keeps track of player/CPU/game information and statistics
   * Sends data on game to be saved to Database when needed
3. Database
   * Outputs encrypted file
   * Receives data on game to be saved from GameMaker Application
   * Converts player/game information and statistics into encrypted file format
   * Stores file locally to maintain game state between executions of main application
   * Converts encrypted file into player/game information and statistics
   * Sends previously saved game data to GameMaker Application when requested

## High-level Overview:

The main component of our system is the GameMaker Application, which receives player action input from 1-4 Player Controllers.

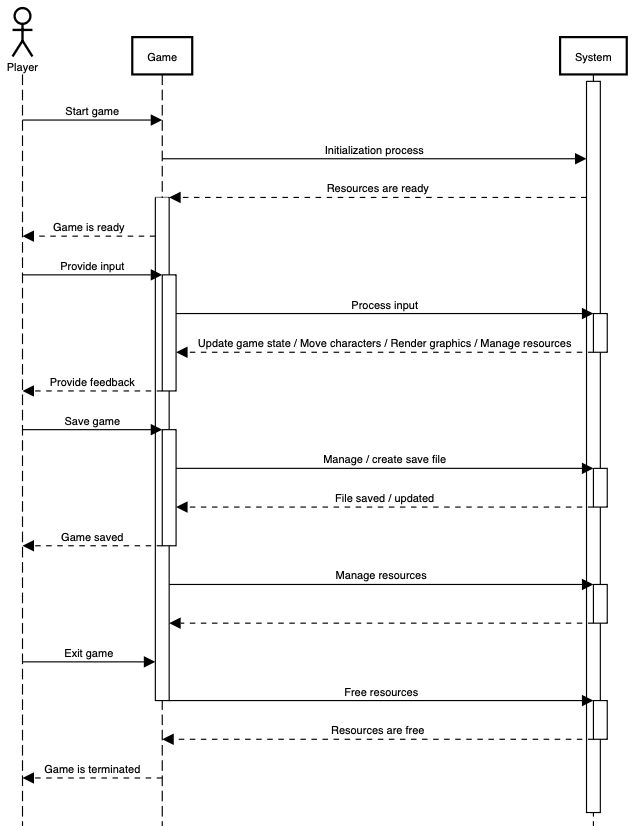
The GameMaker Application also sends data to and receives data from the local Database in order to support local saving of the game.



## 

## Sequence of Events Overview:

The sequence diagram below shows the typical interaction between the player, the game, and the computer it runs on. The sequence is initiated by a user starting the game. After starting the game, the player gives the video game input in multiple forms, such as pressing a button, selecting a specific option, etc. The game code then interprets the input, and sends it to the system if computer resources (memory, storage, etc.) or graphics need to be modified. The game will return the system’s results and its own results to the player. The game will also be sending the requests to the system if it needs more resources allocated for its own processes, and the system will respond accordingly. When the player saves the game, the game triggers the system to create a new file in the game’s filesystem in which to save game data. After the file is created (or modified if it already exists), the system gives a response (positive or negative depending on whether or not there was an error) to the game, which forwards it to the player.



# Design Issues:

## Functional Issues:

1. When do we want players to be able to interact with the game (i.e. when will player controller input be valid)?
   * Option 1: Only during their turn and during minigames, with the first player being the only one to also access title screen, settings, instruction pages
   * Option 2: Any player can interact at anytime
   * Option 3: Any player can interact at any time except when it is not their turn in the game

Choice: Option 1

Justification: The team decided option 1 was the best option to mitigate confusion and accidental selections in the title screen, settings, and instructions pages. Along with that, it enhances gameplay and allows the players to better focus on their turn without other players interrupting. We think having player one being the main person to decide the settings is the best choice so players can decide quickly and start their game as soon as possible.

1. When do we want each player’s current “Boiler Bucks” and “Degrees” to be available to view?
   * Option 1: A player is able to toggle during their turn to view all player’s scores
   * Option 2: After every minigame the score’s can be displayed for every player to see
   * Option 3: We can show each player’s respective “Boiler Bucks” and “Degrees” during the player’s turn

Choice: Options 1 and 2

Justification: The team decided option 1 and 2 were the best to integrate into our game so players were always aware at the start of each round where they stood with their opponents. We also thought it would be beneficial for each player during their turn to be able to view the leaderboard in the event that it influenced what their next steps were in the game and they could not remember where players were after the minigame. We thought these two were the best and that including option 3 would not be the most beneficial way to display the information compared to the two prior options listed.

1. How should the leaderboard be displayed?
   * Option 1: Have the players be sorted based on their current placements, based on who has the most “Degrees” first and who has the most “Boiler Bucks” second
   * Option 2: Have the players remain in the same constant location but mark who is first, second, third, and fourth

Choice: Option 1

Justification: The team decided option 1 was the best way to have the leaderboard be displayed because the player’s would be able to more quickly analyze where they stood compared to their opponents. Although option 2 would make it easier for an individual to find their “Boiler Bucks” and “Degrees”, it would be difficult for them to calculate what place they were in. We felt calculating it for the players and having them look for their sprite would be easier for them than them looking at the same spot but needing to calculate where they stood.

1. How should we handle ties in minigames?
   * Option 1: Have those who are tied share the placement they are in
   * Option 2: Have them roll a die to break a tie
   * Option 3: Have them play a minigame to break a tie

Choice: Option 1

Justification: The team came to the agreement of allowing people who do tie to share the placement they are in. We felt option 2 devalued what was done for them to result in last by having the results come down to luck in a tie. We felt option 3 would make the game take much longer than we felt was ideal. Option 1 maintains the integrity of the work done by players and it allows us to keep the game within a better length from start to finish.

1. What screen ratio do we want our game to be at
   * Option 1: 4:3
   * Option 2: 16:9

Choice: Option 2

Justification: The team decided on option 2 so we can utilize more of the screen for viewing the map and minigames. Although we heavily considered option 1 because it closely fit the aesthetic of a retro game, we found the space we would be missing out on too important. With having to share the screen with 4 players, maximizing how much of the screen is visible gameplay is crucial in creating a better experience for the players.

## Non-Functional Issues:

1. Where do we want to keep the game’s save file?
   * Option 1: Have it save to a server we host
   * Option 2: Have it save locally with the other game files in a standard text file
   * Option 3: Have it save locally with the other game files in a encrypted .sav file

Choice: Option 3

Justification: The team decided the best and most efficient decision was to implement option 3. We found option 1 to be unnecessary as it would require all systems who play our game to need an internet connection while playing. When we have the ability to make our game completely local to the system, we found saving the game locally was the best. Our team also believes that the integrity of the game is important so preventing the save file from being changed was what made us decide option 3 was the best for what we want our game to be.

1. How many frames per second should our game run?
   * Option 1: 30 frames per second
   * Option 2: 60 frames per second

Choice: Option 1

Justification: The team decided option 1 was the better option to fit the overall look and aesthetic of the game. The game is supposed to fit a 2D retro game aesthetic and feel for the player and after researching, doing 30 frames per second was the better choice. Although 60 frames per second is said to provide a “smoother” form of gameplay, it was unnecessary for the types of minigames in which the players would be competing against each other.

1. How many players should we support for our game?
   * Option 1: A maximum of 2
   * Option 2: A maximum of 4
   * Option 3: A maximum of 8

Choice: Option 2

Justification: The team decided supporting 4 players was the most ideal for the type of game we’re designing. Because we are running on computers, it would be unlikely for the user to have access to 8 USB ports, making a max of 8 players really unlikely. Because of that, we believe supporting that wouldn’t be the best use of our resources. We also decided against option 1 due to the lack of competition that occurs if one player is ahead of the second. By allowing 4, it is convenient for the user and allows a competitive and fun environment for the game.

1. What engine should we use for game development?
   * Option 1: GameMaker (GML)
   * Option 2: Unity (C#)

Choice: Option 1

Justification: The team decided to utilize the GameMaker engine for our game’s development. We decided on GameMaker due to its ease in building and designing 2D games. The team also had more experience with GameMaker compared to Unity and because of our comfortability, GameMaker would allow us to make more creative development decisions for our game.

1. How should our naming conventions be for the different coding events as well as sprites, objects, and other built in to GameMaker’s program
   * Option 1: For built in items in GameMaker, we follow AbbreviationForItem\_NameOfItem and we can have variables in coding events fit Pascalcase formatting. An example for an item would be SP\_Player where SP stands for sprite and an example in a coding event could be CharSpeed.
   * Option 2: We could utilize Snakecase for both items in gamemaker and coding events. An example for an item would be sprite\_player and an example for a variable in an event would be char\_speed
   * Option 3: We could utilize Camelcase for both items in gamemaker and coding events. An example for an item would be spritePlayer and an example for a variable in an event would be charSpeed

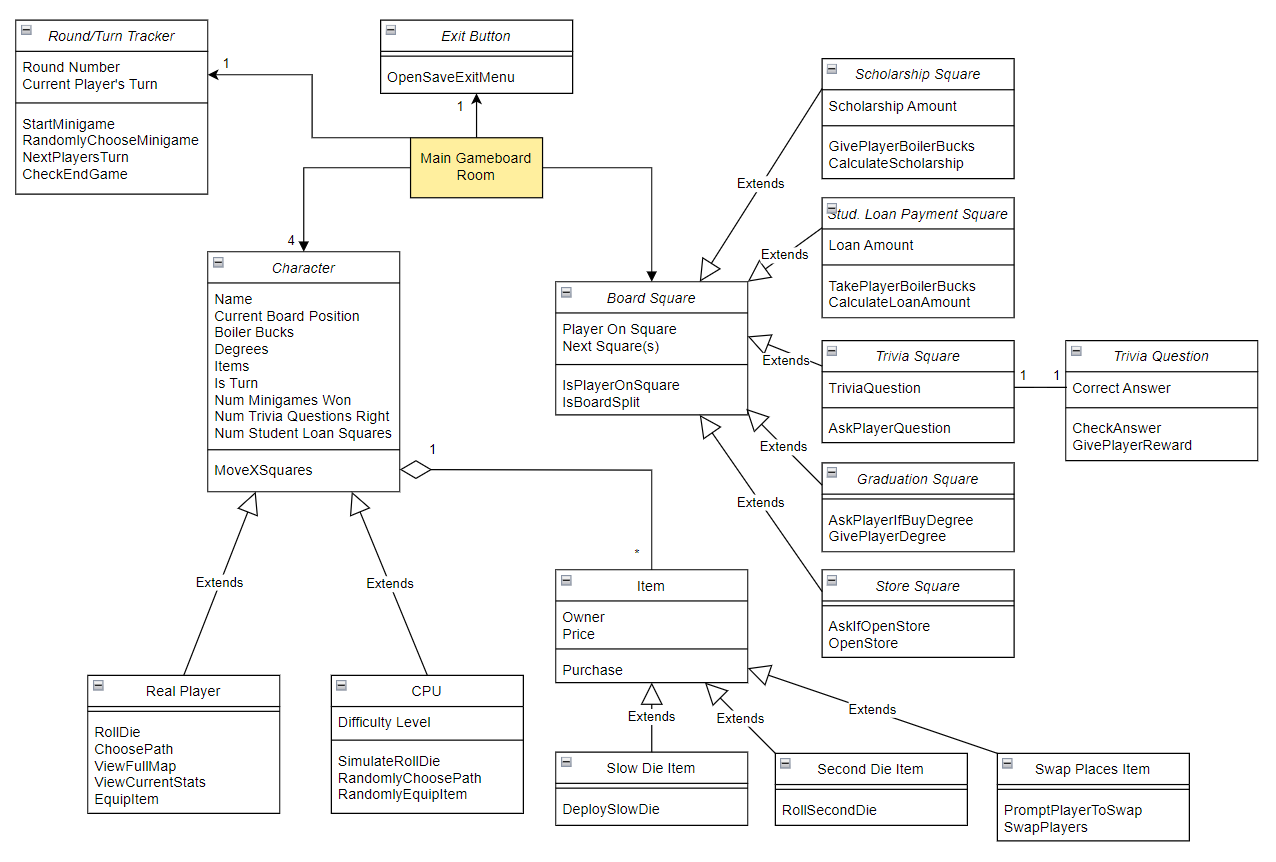
Justification: The team decided option 1 would be the best for our game as it would help keep track of the difference between items and variables contained within the different event functions. Option 2 and 3 are both good options but for the purposes of keeping items built into GameMaker and the functions and variables we build ourselves separate, option 1 would be the best fit.

**Design Details:**

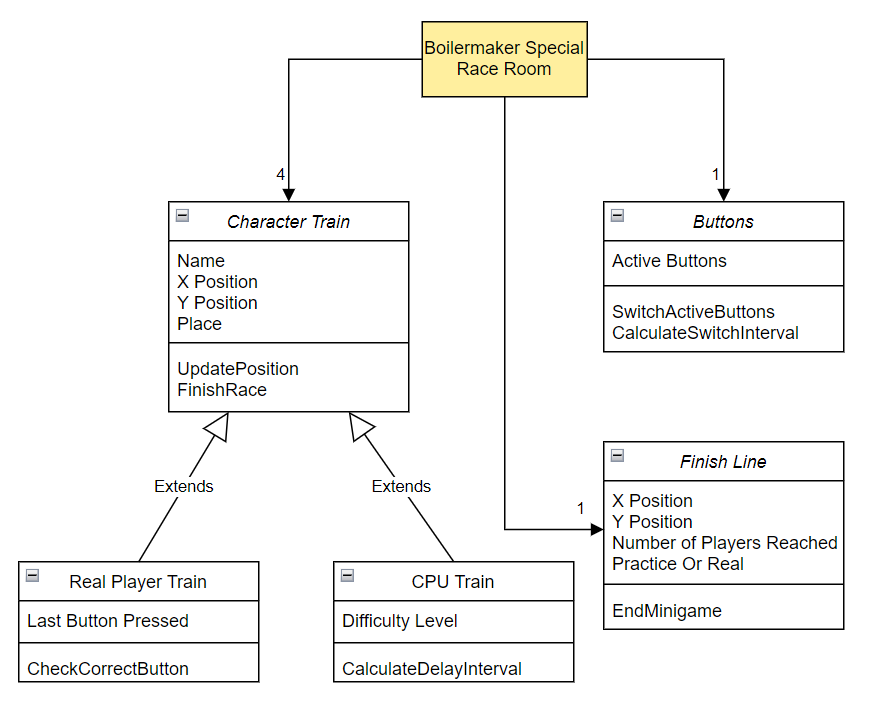
## Class Diagrams:

As our system is written in GameMaker Language, classes are a little different than normal object-oriented programming languages like Java. “Classes” in GameMaker Language are Objects, each of which has associated variables and methods. These Objects each belong to a certain Room in GameMaker, meaning that each of our “rooms”/screens have their own set of Objects and associations between those objects. Because of this, we have created a series of modified UML Class Diagrams where each diagram represents the class relationships for one Room. These Rooms are labeled in the diagrams as yellow-shaded boxes.

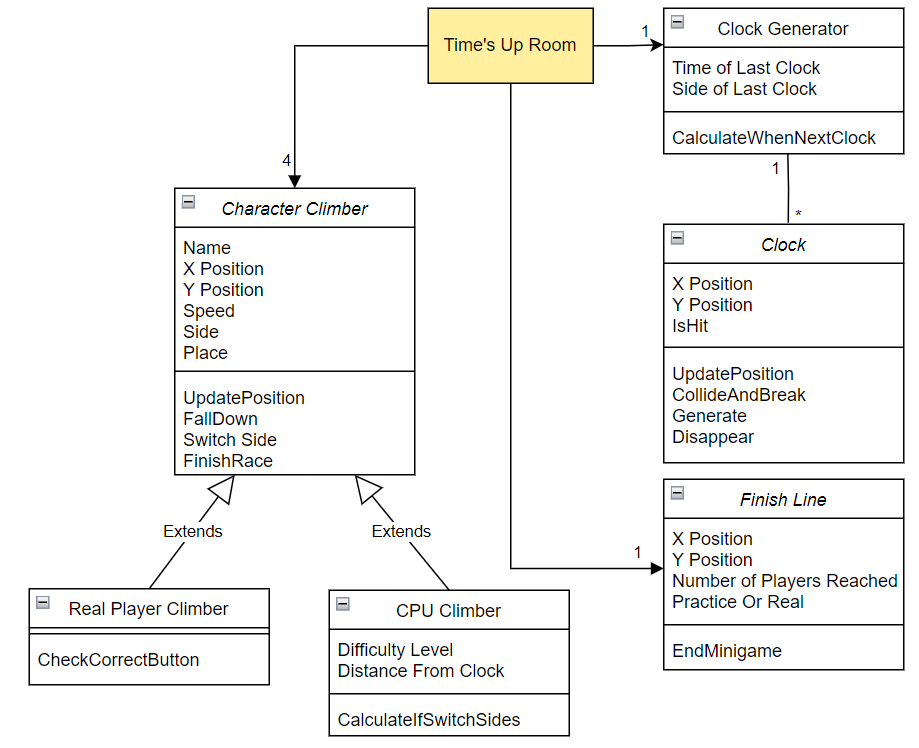
Main Gameboard Room:



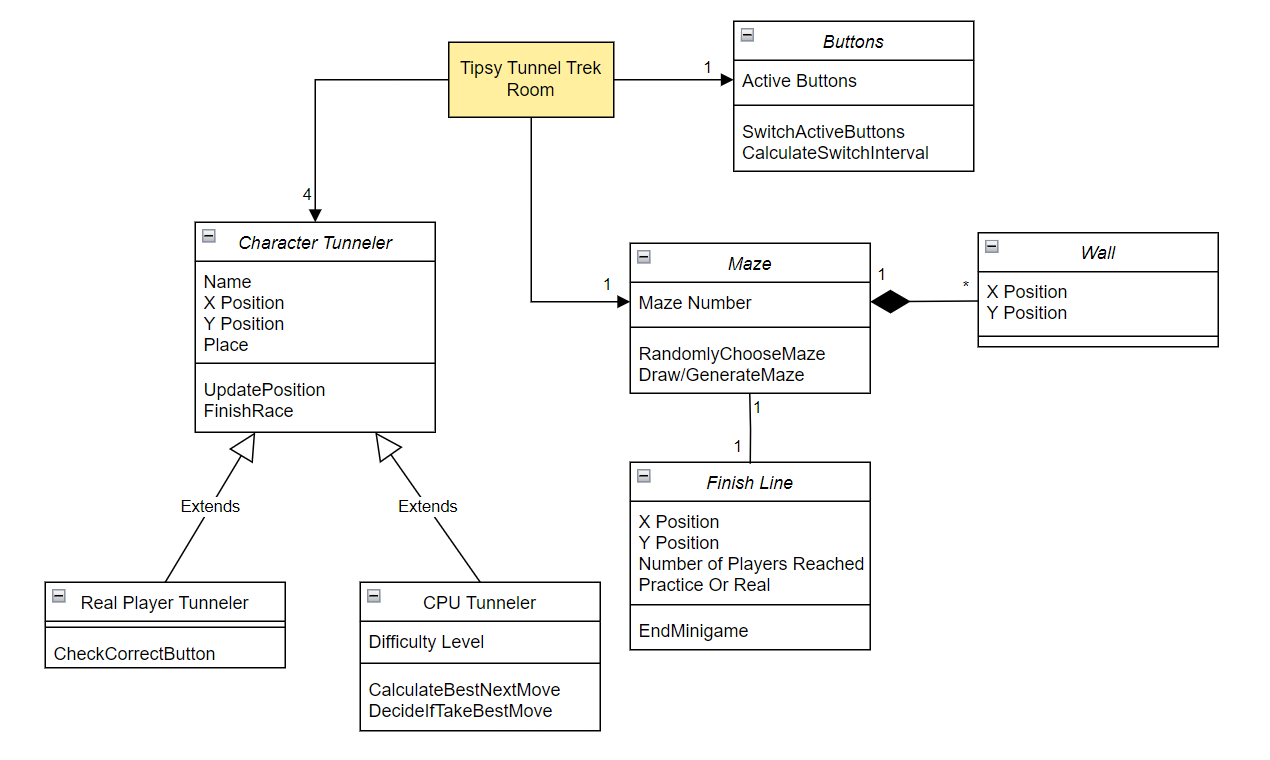
Boilermaker Express Race Room:



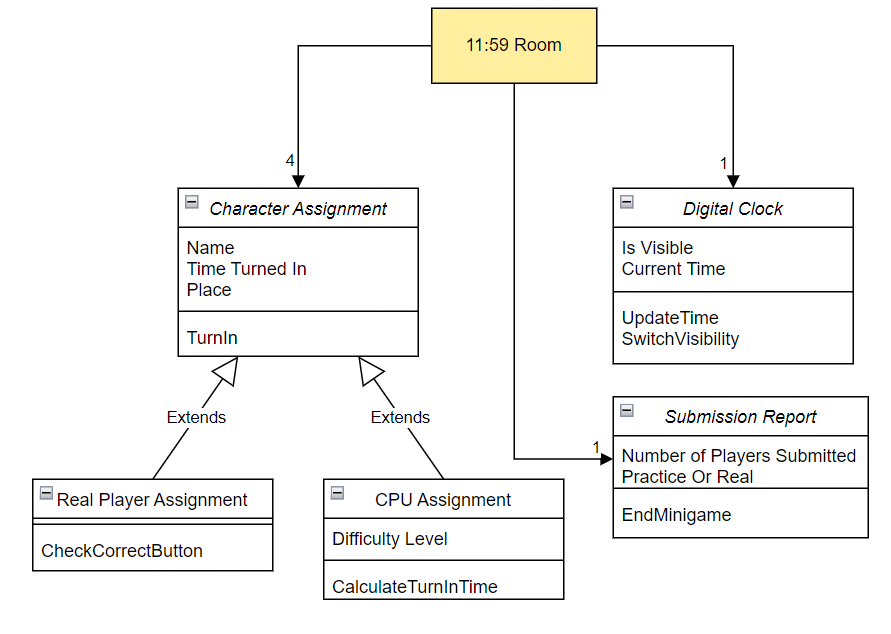
Time’s Up Room:



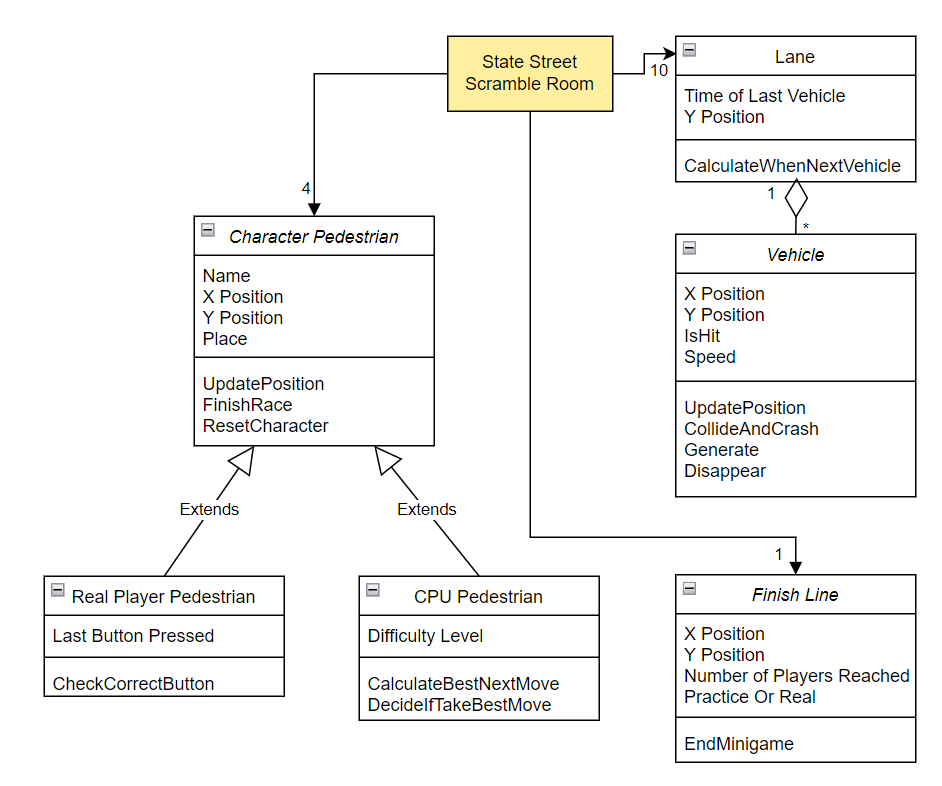
Tipsy Tunnel Trek Room:



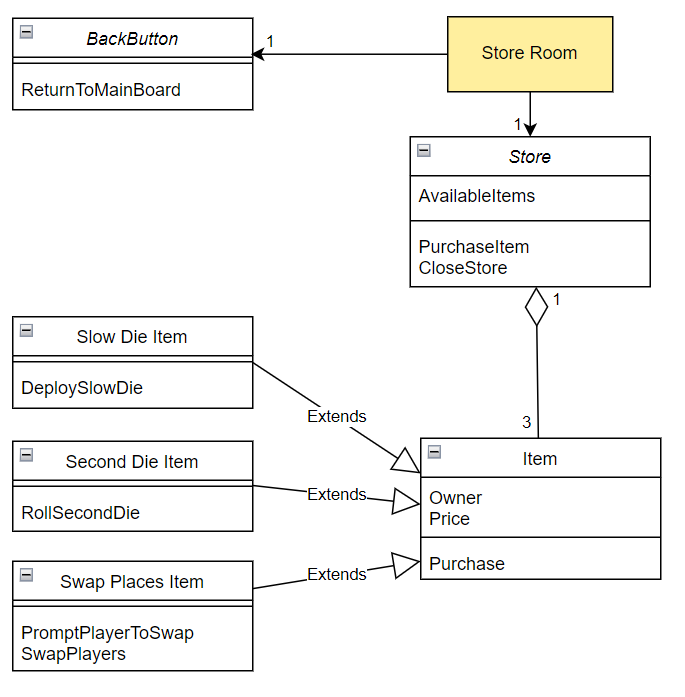
11:59 Room:



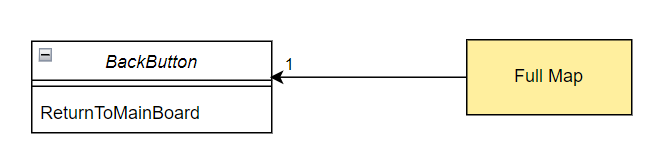
State Street Scramble Room:



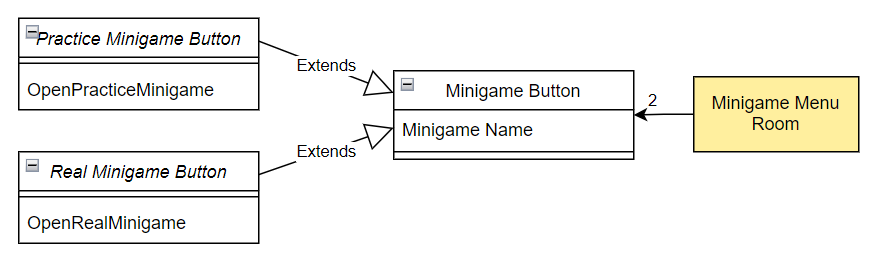
Store Room:



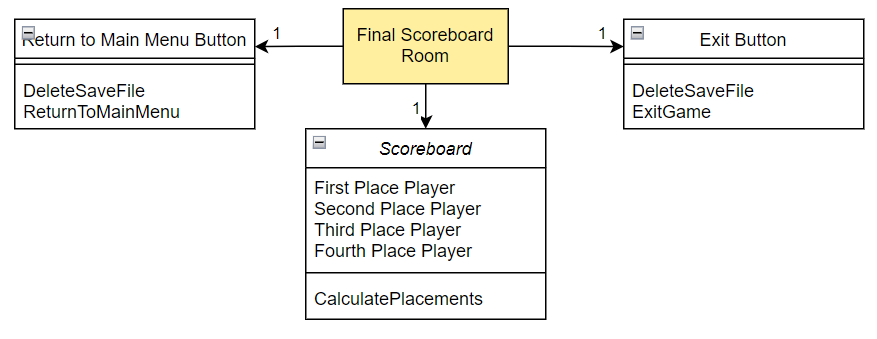
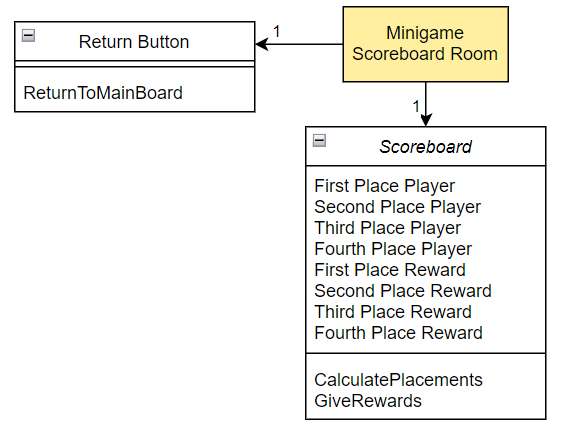
Full Map Room:



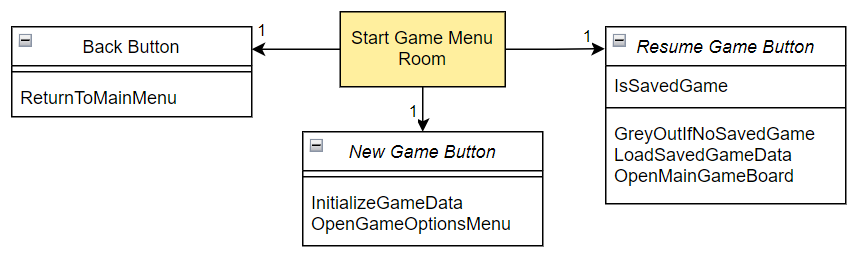
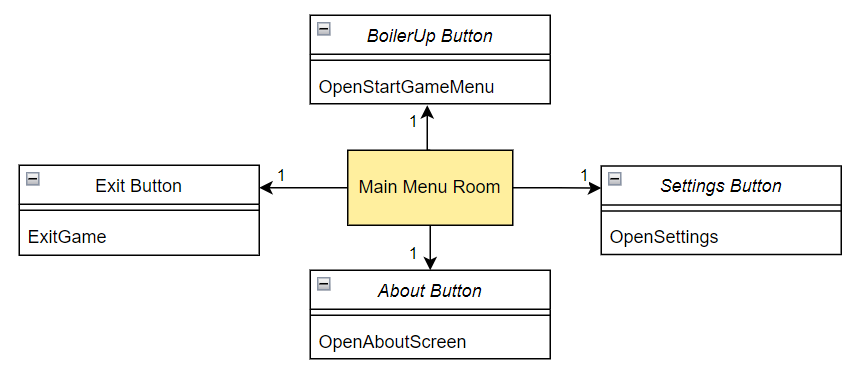
Minigame Menu Room:



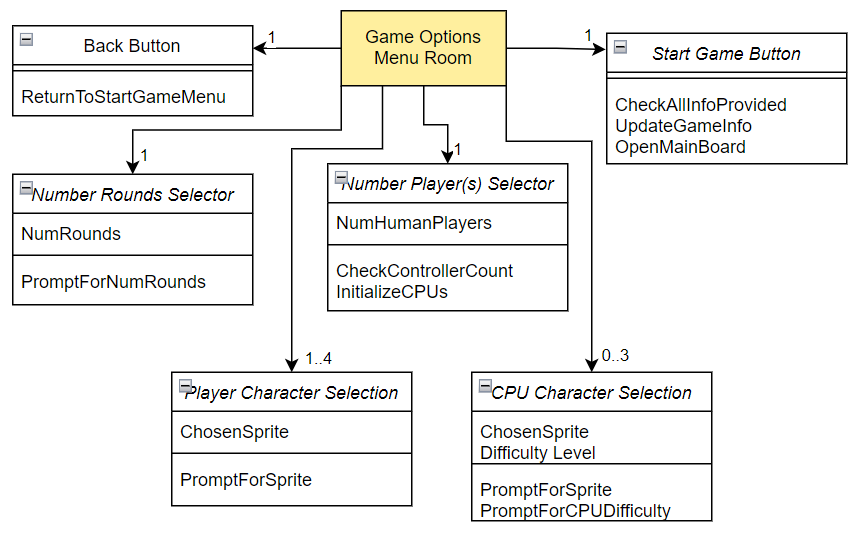
Minigame Scoreboard Room and Final Scoreboard Room:



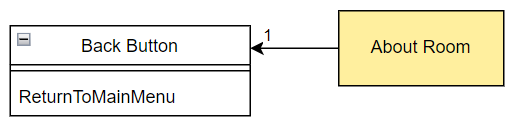
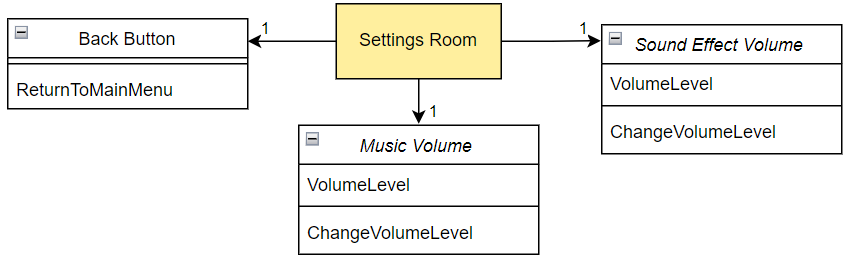
Main Menu Room and Start Game Room:



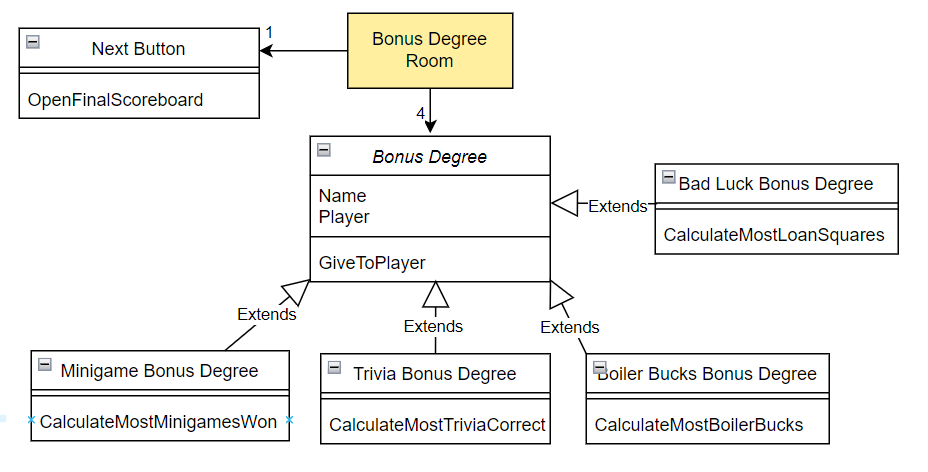
Game Options Menu Room:



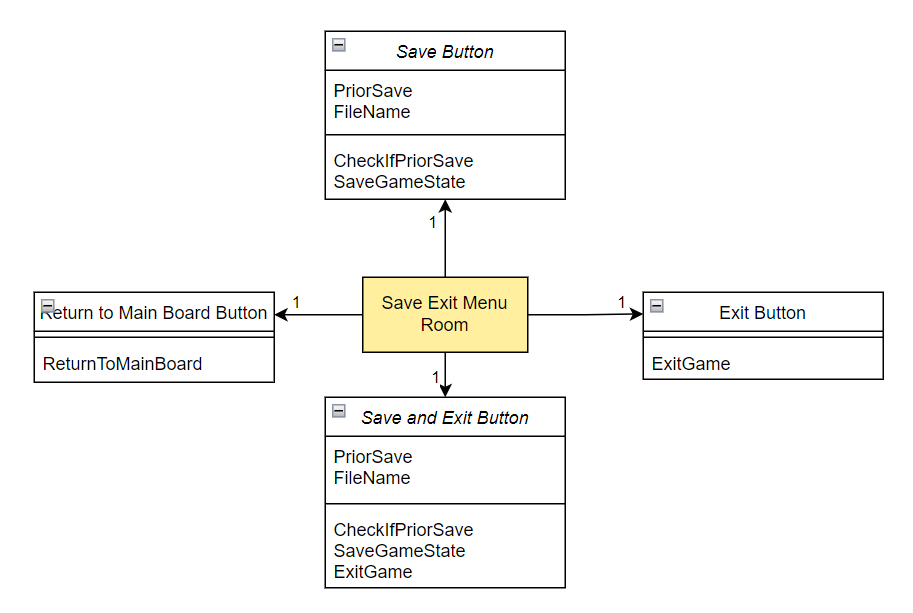
Settings Room and About Room:



Bonus Degree Room:



Save Exit Menu Room:



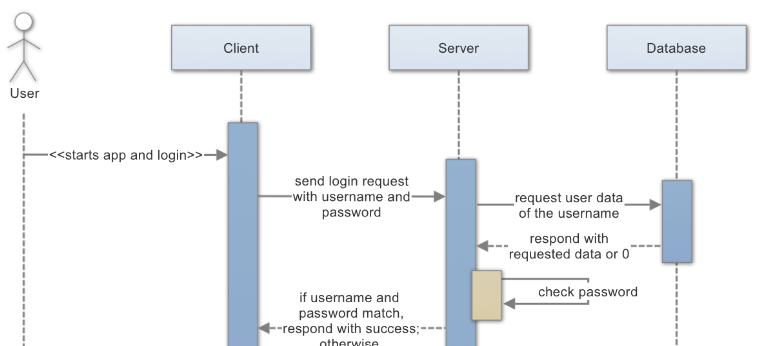
## 

* **Main Board**
  + Round/Turn Tracker
* Keeps track of current player’s turn and when round ends
* Generates random minigame at the end of each round
* Keeps track of number of rounds and when the game should end
  + Character
* Holds character’s stats including name, Degrees, Boiler Bucks, items, and statistics for Bonus Degrees
* Stores and updates character’s position on game board
  + Real Player
* Child of Character, allows human players to roll die, choose to view map, view score, or use items via controller input
* Allows human players to choose direction on board when there is an intersection
  + CPU
* Child of Character, allows CPU to roll die or use items randomly
* Allows CPU to randomly choose direction on board when there is an intersection
  + Item
  + Slow Die Item
  + Second Die Item
  + Swap Places Item
  + Board Square
  + Scholarship Square
  + Student Loan Payment Square
  + Trivia Square
  + Trivia Question
  + Graduation Square
  + Store Square
  + Exit Button
* **Boilermaker Express Race**
  + Character Train
  + Real Player Train
  + CPU Train
  + Finish Line
  + Buttons
* **Time’s Up**
  + Character Climber
  + Real Player Climber
  + CPU Climber
  + Finish Line
  + Clock Generator
  + Clock
* **Tipsy Tunnel Trek**
  + Character Tunneler
  + Real Player Tunneler
  + CPU Tunneler
  + Maze
  + Finish Line
  + Wall
  + Buttons
* **11:59**
  + Character Assignment
  + Real Player Assignment
  + CPU Assignment
  + Submission Report
  + Digital Clock
* **State Street Scramble**
  + Character Pedestrian
  + Real Player Pedestrian
  + CPU Pedestrian
  + Finish Line
  + Lane
  + Vehicle
* **Store**
  + Store
  + Item
  + Slow Die Item
  + Second Die Item
  + Swap Places Item
  + Back Button
* **Full Map**
  + Back Button
* **Minigame Menu**
  + Minigame Button
  + Practice Minigame Button
  + Real Minigame Button
* **Minigame Scoreboard**
  + Scoreboard
  + Return Button
* **Main Menu**
  + Boiler Up Button
  + Settings Button
  + About Button
  + Exit Button
* **Start Game Menu**
  + New Game Button
  + Resume Game Button
  + Back Button
* **Game Options Menu**
  + Number Rounds Selector
  + Number Players Selector
  + Player Character Selection
  + CPU Character Selection
  + Start Game Button
  + Back Button
* **Settings**
  + Music Volume
  + Sound Effect Volume
  + Back Button
* **About**
  + Back Button
* **Bonus Degree**
  + Bonus Degree
  + Minigame Bonus Degree
  + Trivia Bonus Degree
  + Boiler Bucks Bonus Degree
  + Bad Luck Bonus Degree
  + Next Button
* **Final Scoreboard**
  + Scoreboard
  + Exit Button
  + Return to Main Menu Button
* **Save Exit Menu**
  + Save Button
  + Exit Button
  + Save and Exit Button
  + Return to Main Board Button

## Sequence Diagrams:

Add for different activities in the system

1. Sequence of events when …

<- Example

1. Sequence of events when …

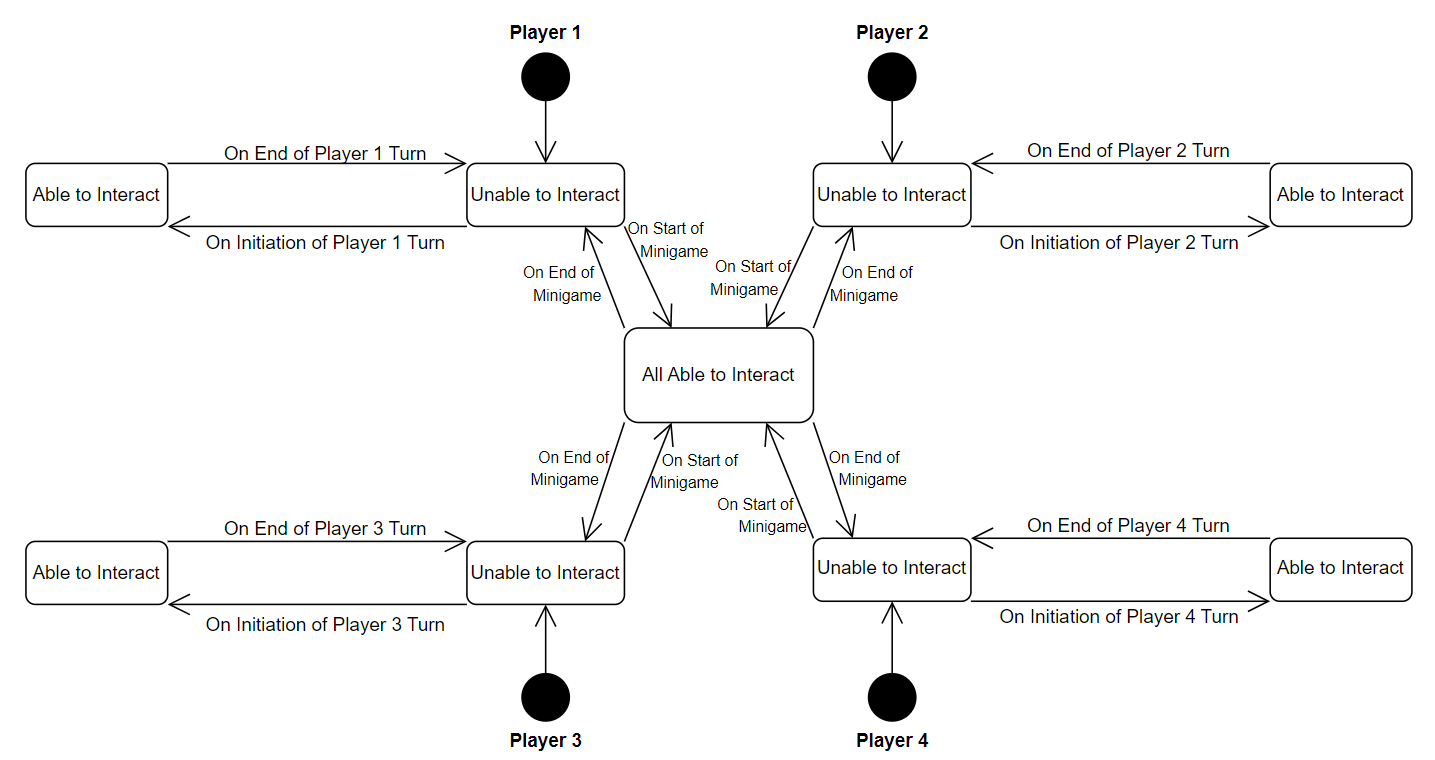
## 

## **Navigation Flow Map:**

## 

The Navigation Flow Map depicts how the screens of our application are navigated by the user. Each box represents a different screen that the user may see throughout the game. Arrows connecting these boxes represent the ways in which a user can transition from one screen to another, with labels depicting how the user initiates the screen transition. The gold box is the screen that the user starts on upon launching the application. The arrows pointing to **SYSTEM EXIT** represent ways in which the user can exit the game without interfering with the saving system.

## State Diagram:



This State Diagram depicts the interaction ability states of all four players during the main gameplay (Main Game Board, Store, Minigames) of our application. Each player can either be in a state in which they are Unable to Interact with the game, meaning that controller input will be ignored, or Able to Interact with the game, meaning that controller input will have an effect. All players will begin on initial entry to the Main Game Board in the Unable to Interact state. On the initiation of each player’s turn, the player will switch to the Able to Interact state until the end of their turn, when they will return to the Unable to Interact state. On the start of a minigame, all players will switch to an Able to Interact state until the end of the minigame, when they will return to the Unable to Interact state.

## 

## UI Mockup:

The below images are based off of the navigation flow map included above

* Title Screen
  + Boiler Up is a button to continue with setting up and starting the game
  + Settings is a button that will take you to the settings screen
  + Exit is a button that will exit the application
  + About is a button that will take you to the about screen



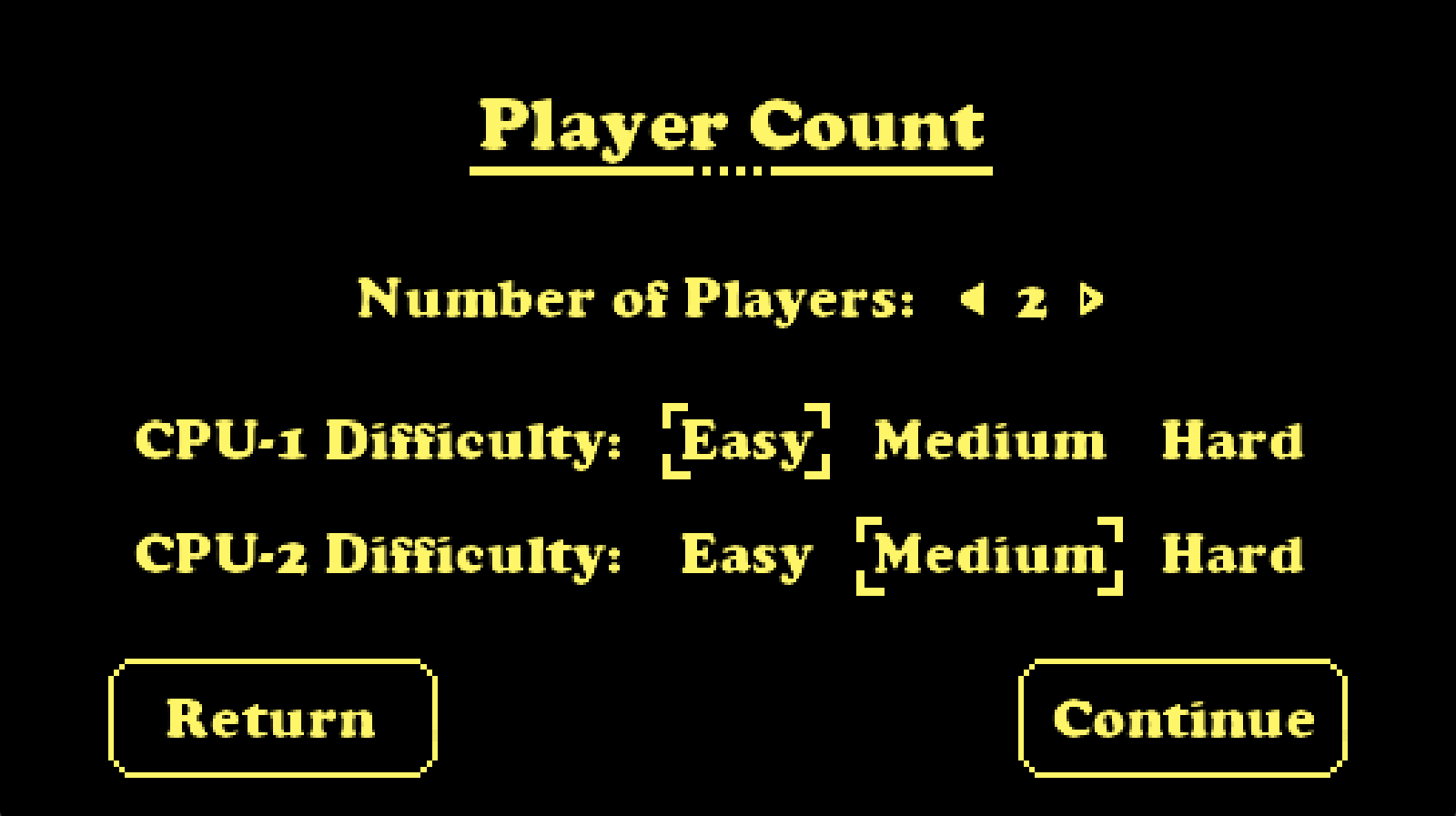
* Settings Screen
  + Return is a button that will take the players back to the title screen
  + The music and sfx settings allow the players to change the respective types of volumes from the game



* About Screen
  + Has a return button that will take the players back to the main title screen



* CPU Settings Screen
  + Return is a button that will take you back to the title screen
  + Continue is a button that will take you to the game setup settings
  + Number of players can be changed and based on that value, up to 4 CPUs will appear on the screen where their difficulty can be selected



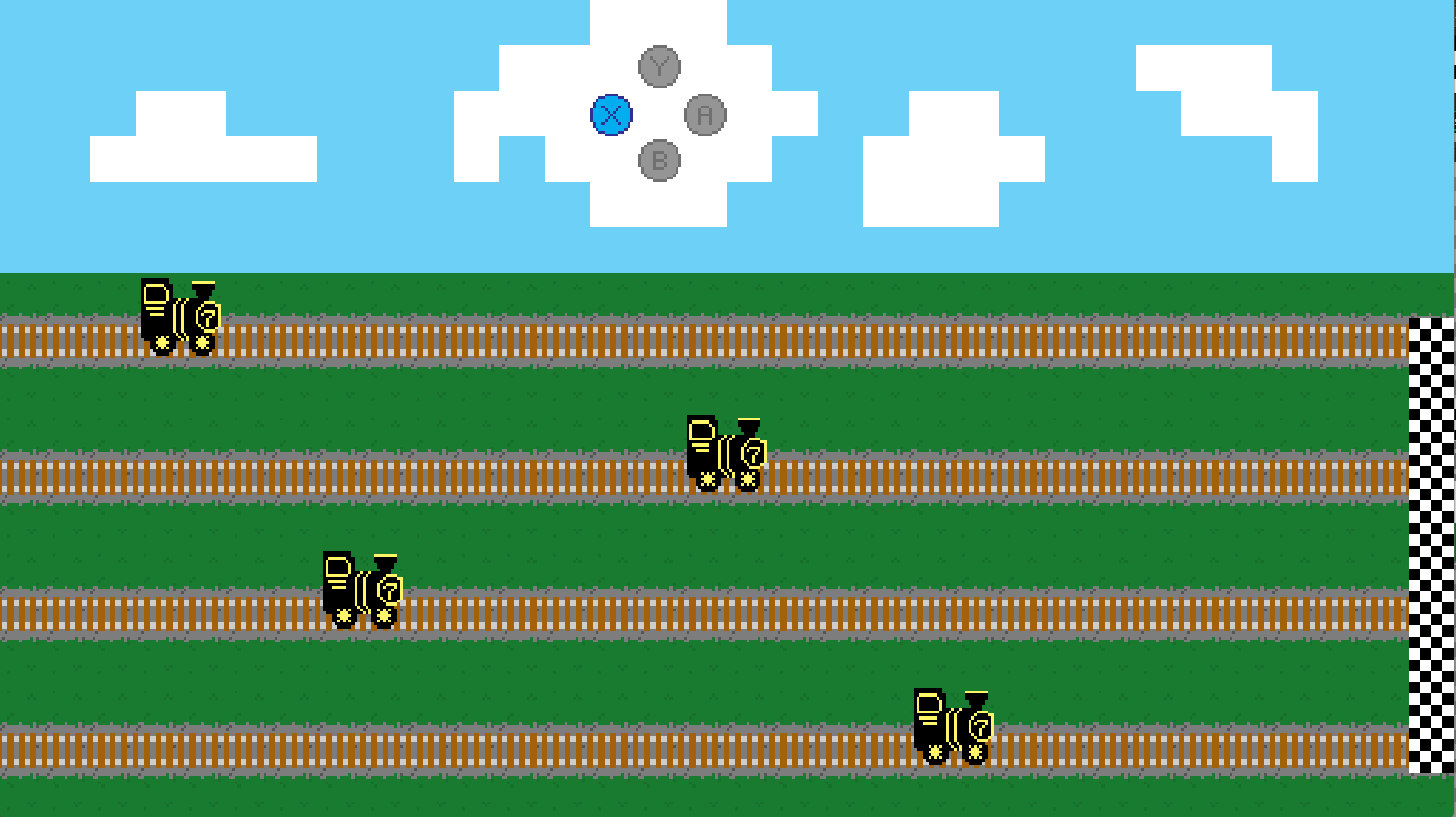
* Game Setup Screen
  + Return is a button that will take you back to the CPU settings screen
  + Continue is a button that will take you to the start of the game



* Sample Instructions Screen
  + Play game is a button that will take the players to the minigame the instructions are describing
  + Play demo is a button that will take the players to the minigame the instructions are describing but when completed, will have the minigame return to the instructions page and it will not contribute to the players’ score



* Boilermaker Express Minigame
  + Each train is assigned to a player in the game that moves at the rate they alternate between the two buttons currently in play
  + The buttons depicted in the clouds will flash to show which two buttons the players will need to alternate from in order to get their trains to move
  + When a player crosses the finish line at the end, we will know where the players are placed



* Store Screen



* Minigame Result Screen
  + Continue is a button that will allow players to return to the map and remain gameplay and start their individual turns for the round
  + Players will be displayed in the respective places based on how they did in the minigames. Their sprite, current “Boiler Bucks” and “Degrees” will be displayed in that space



* Final Result Screen
  + Main menu is a button that will take the players to the title screen so they may begin a new round if they would like to
  + Exit is a button that when pressed, will exit the application
  + Players will be displayed in the respective places based on how they did throughout the game. Their sprite, ending amount of “Boiler Bucks” and “Degrees” will be displayed in that space

