Jarquin Danny

COMP 465 - Mon/Wed

Professor Karamian

Unity Battleship Game

**Table of Contents**

**1 INTRODUCTION 3**

**2 DESIGN GOALS AND OBJECTIVES 4-8**

**3 SYSTEM BEHAVIOR 8-10**

**4 LOGICAL VIEW 11**

**4.1 Unity Class Hierarchy 12**

**4.2 Detailed Class Design 13**

**5 USER INTERACTION 14**

**6 USER INTERFACE 14**

**7 SCREENSHOTS 15-18**

**8 LESSONS LEARNED 19**

# Introduction

The project emphasizes the development of the game battleship. It is a game that has been around since World War I. The main concept for the game revolves around two players taking turns to try and sink each other’s naval ships. Each player is given five ships and a 10x10 grid board for players to set their ships. The ships have various sizes which may take more than two squares up to five on the grid. There have been various iterations of the game and this one in particular is developed through Unity’s game engine and it’s meant to be a single player experience against an ai player.

The first phase of development is to set the foundation on which the player and the ai player will be playing on. This involves the creation of the player boards and allows the user to select any of the ships and place it either horizontally or vertically on the board. The ai player on the other hand sets its ships randomly on their board before the user is allowed. Additionally, each ship has a determined size which restricts how many tiles a ship occupies. There are rules for both the player and the ai player to follow when it comes to ship placement. These rules are ships are not allowed to occupy the same tiles on the grid, ships are not allowed to be hovering outside of the board, and lastly both players are limited to only five ships with disregard to ship orientation.

The second phase of the project can be divided into two sections, the first part handles core gameplay mechanics after the play board has been set and part 2 is user interface work to handle all the user interactions with the game. For the core gameplay mechanics after the board has been set, the game can begin initially player one gets their turn first. Player one must select a tile on player 2 board and the ai player must choose a tile on the player one board when it is their turn. The change of player boards is done through a transition of the camera when each player gets their turn. When a tile is clicked by either player if it is occupied by a ship the board tile changes to a red color, then an explosion sound is made, and a smoke animation starts playing on the tile. If the player chooses an occupied tile, they get to attack the other player again. Furthermore, when the player picks a tile that is not occupied by a ship the color of the tile is set to black and a water splash sound is played. This signifies that the player did not hit a ship and signals that their turn is over. The second part of phase 2 involves the development of the user interface and sound for the game. It involves the creation/acquisition of assets for textures, sounds, back grounds used for the game. These are used for the main menu, the menu on the game scene, buttons, sliders in the game.

The document explains the various parts designed for the development of the battleship game. This attains aspects within the game such as sound, user interface and gameplay mechanics.

**Change History**

**Version:** 0.5 – Battleship Part 2

**Date:** 12/15/2022

**Description of Change:**

Added a visual effects graph to simulate smoke when a hit is registered on an occupied tile and added in a cheat mode for the user.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Version:** 0.4 – Battleship Part 2

**Date:** 11/20/2022

**Description of Change:**

Added gameplay logic for the non-player to randomly select a tile on the board and assign it as attacked or missed if it is occupied. Updated the camera to move to the designated board after a delay once a hit is registered.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Version:** 0.3 – Battleship Part 2

**Date:** 11/7/2022

**Description of Change:**

Added gameplay logic for the user to select a tile on the board and assign it as attacked. Additionally added a camera transition through the use of cinemachine.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Version:** 0.2 – Battleship Part 1

**Date:** 10/15/2022

**Description of Change:**

Create the functions for the placement of ships for the user and non-player user. The non-player user ships are set randomly based on coordinate, and the player also has the option to choose how the ships are placed through a temporary user interface.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Version:** 0.1 – Battleship Part 1

**Date:** 10/10/2022

**Description of Change:**

Develop the creation of the game board by instantiation of game objects that are used as tiles to create a grid.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Design Goals and Objectives

The goals for the game were to take the concept of the battleship game with the imposed requirements for the project while giving it our own personal touch when it came to the design of assets and overall aesthetics for the game.

* Users –.Create a simple interface for users to navigate and interact with through the use of the mouse and keyboard.
* Developer – Create code that is easy to understand and follow. Apply proper methodology of functions, methods associated with the classes used for development of the game. Proper comments for each class, methods to portray clarity of its function and complexity.

A simple approach was taken to ensure the reduction of making the implemented tasks less complex by avoiding advanced techniques or structures for the project.

The scripts within the program are as follows:

* **MainMenuUIManager –** handles a few things related to the main menu gui and is the simplest of the classes within the project.
  + **Change Scene**- when the button is clicked to start the game. It takes the name of the next scene to load in unity and passes it to Unity’s library for scene management to load the next scene.
  + **Quit-** when the user clicks on the exit button the application quit function is called to exit the game.
* **SoundManager-** this class handles all audio aspects of the game. This ranges from sound effects, the music used in the main menu and during gameplay.
  + **SoundFileAttributes-** a helper class for the sound manager. Its role is to define the attributes for audio files. These are the names displayed as in unity’s inspector, the audio clip file, the mixer, volume, pitch, song loop and play as soon as the object is attached to becomes active. It is heavily used to control the volume of sound effects.
  + **Awake-** it takes the attributes from the helper class and creates audio sources based on array of audio files for unity to reference and use with later functions in Sound Manager.

* + **Start-** It loads the sound values from the player preferences.

* + **MasterVolumeSlider-**This is the master control for sound within the game. It takes the value of a slider button its attached to and saves it to the player preferences and loads the new values.
  + **MusicSlider-** This controls the volume of the music played in the scene. It takes the value of the slider it’s attached to, saves it to the player preferences and updates the volume level of music playing.
  + **SFXSlider-** It takes the value of a sound effect slider, saves it to the player preferences and then loops through all the available sound effects and adjusts the volume for them accordingly.
  + **Play-** it takes a integer as parameter which is used for finding the audio file to play. This is mostly used with the sound effects
  + **LoadPrefs-** this handles loading the volume values from player preference and adjust the sliders to reflect that on the menu. It is used on every slider function to load the values.
* **BoardUIManager-** this UI manager handles everything related to the games ui from keeping track of scores, starting the game and selecting the ships, sending the sip size to the board manager, the ship orientation, change of scene, pause menu, resume the game, and last but more importantly cheat mode.
  + **Start-** is used to reset the player scores once it’s active on the scene.
  + **Update-** is used to call the other functions within the boarduimanager such as pauseIt, StartInter, and scoreupdater.
  + **Scoreupdater-**score updater keeps track of players scores throughout the game. It loads the values from the player preferences and finds the labels used to keep the scores. Whenever either player reaches 18 points a win or lose screen is displayed.
  + **StartInter-**is used to check whether the player placed all of their ships. If there is still a ship to be placed the button will not be interactable but if all ships are placed on the board the player can start the game.
  + **StartIt-** sends the value the game is about to start to the playmanager and boardmanager to adjust their functions accordingly.
  + **setShipSize , setShipId and sendShipInfo-**this three methods work hand in hand. When the user selects a ship the id and size of it are set the sent to the boardmanager to know what ship model will be loaded when the ship is placed on the board.
  + **ChangeOrientation-** when the orientation button is pressed, the variable value is sent to the boardmanager to tell it to rotate the ship’s orientations to either vertical or horizontal.
  + **PauseIt-** this pauses the game whenever it is called. It detects the player input for the escape key and does a series of checks to prevent menu spamming. If the ship menu is still needed it will activate it or deactivate it accordingly.
  + **Resume-** is used for when the user clicks on the resume button on the pause menu and returns the game to its normal state before the pause menu was called.
  + **Cheat Mode-** it is exactly what it sounds like it calls the opponent board and checks for ships placed on their tiles if a ship is occupying a tile in the main game the tile is turn into a green color.
* **BoardUnitInfo-** this class is used for setting attributes on the tiles that will be created on the player boards for the game, whether the tile is occupied by a ship or if it has been attacked.
  + **Awake-** Sets the label for the tile to be of index [0,0] for starts
  + **Update-** sets the cube to occupied or unoccupied. It was mostly used for debugging purposes.
  + **Update Unit display-**it is used to update the text label within the tile and set the row and column of the tile within the grid for the board.
* **BoardManager-**this handles the creation of tiles to generate the player boards and sets the ships for the players on the board.
  + **OnEnable and OnDisable-** this two help send data across two of the scripts when a new event is triggered. These are used to get the changes on ship information such as the id, the size and orientation.
  + **Start-** generates the boards for the two player by instantiating the tile object created with the attributes from the board unit info. The two boards once generated are placed next to one another with a small distance between them.
  + **Update-**calls the userboard and the aiplace methods which are used for the placement of ships.
  + **AIPlace-**this is used for the placement of ships for the ai player, it starts by generating two random numbers, then proceeds to randomly get the orientation of the ship. Once that is done the tile is selected based on those two numbers. After two checks for the orientation are performed based on the orientation selected. It checks if the ship has enough room to be placed without colliding with another ship or going out of bounds. The process is then repeated until all the ships are placed on board.
  + **UserBoard-**the ship placement for the user is done through this one. It is similar to the aiplace with the only difference is that it keeps track of where the user last clicked on the board. Once it finds which tile the user has selected it performs the checks for orientation and out of bounds or collision between ships. It then proceeds to call the function to place the ship model on the board according to the coordinates selected if allowed to be placed.
  + **ShipTypePlaced-** takes the coordinates, the id and array of ships which are going to be placed. It then goes through a switch case statement to find the correct ship based on the id. If the ship is found it then does checks for which orientation the ship must be placed and proceeds to instantiate the model then rotates it around based on the orientation. Once that is done the ship is then flagged as being placed.
* **PlayManager-** it handles the gameplay for the game whether that is taking turns between players or updating the score count points and managing the camera movement for the players.
  + **OnEnable and OnDisable-** this two help send data across two of the scripts when a new event is triggered. This to tell play manager whether the game has started once the start button ahs been clicked. It then updates the start variable through the onstartchange function accordingly.
  + **Update-** this is used to start the game and makes a call to the gamestart function.
  + **gameStart-** first sets the camera for player one to take their turn then lets them take their turn and if the state of turn variable is changed then the ai gets to take their turn.
  + **AITurn-** the function generates two random numbers then as long as it is within bounds proceed to get the corresponding tile. If the tile is unoccupied then the tile is turned into a darker color , the turn state is changed to the other player and a splash sound is activated to play. Furthermore, if the ai lands a hit o the ship the tile is turned red, and an explosion sound is played then the ai can take their turn again and select a new tile on the board.
  + **playerTurn-**for the player turn first it find what tile the user clicked on, then if it was a hit the color is changed to red, and an explosion sound is played and can proceed to click on another tile. If it was a miss, then a splash sound is played, and the turn state is switch to the aiturn.
  + **CamControl-** is used to transition between the two player boards when the turn state is changed. A delay is always added between the turns to give the player enough time to check the tile the ai selected on their turn.

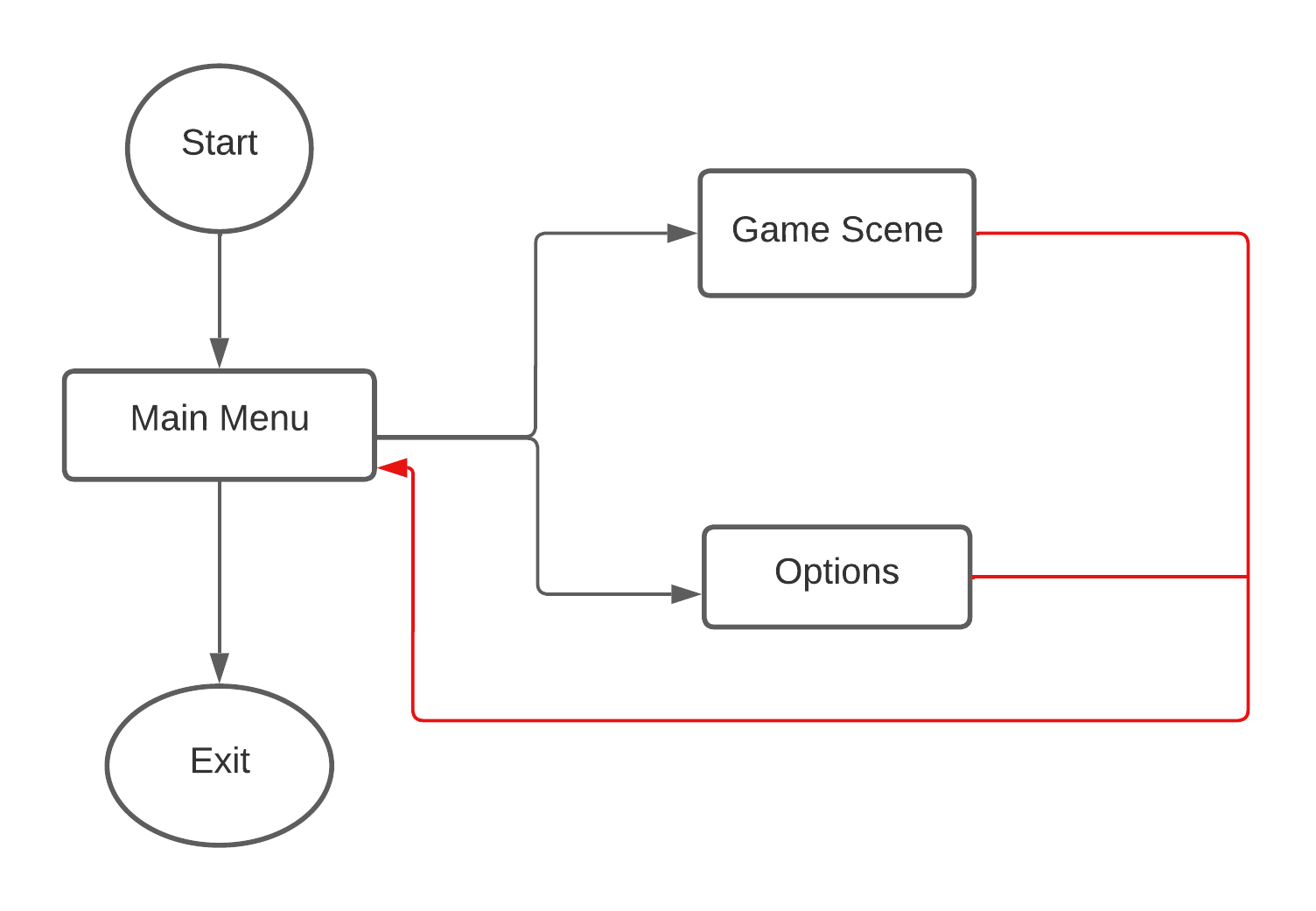
# System Behavior

## Scene Manager

The system behavior is handled by the user’s input. In this case the scene manager handles the transition from the main menu scene to the game scene where the user can start playing the game.

1. **MainMenu Scene**:

This is the start scene where the user is taken to when the game is loaded. The user is given a menu with buttons to select a few options to start the game, change settings for sound and exit the game.



1. **Game Scene**:

Here the user is given a ship placing menu which has all the ships available. There are five buttons in it, which four are used for the selection of the ships and one to control the orientation of the ship to be placed. There is a top bar menu which is used to start the game and also keep track of player scores once the game begins.

Diagram

Description automatically generated

# Logical View

The logical view explains each step of the system when the game launched and ready to play. It is then broken down to show the game’s architecture detailing the classes and methods used in the development of the game.

• User starts on the main menu screen and has the options to select from the menu.

o Player can choose to start the game and be loaded into the game scene.

o The user has access to the options menu and adjusts the volume controls for the music and sound effects in the game.

• When the user starts the game, there is a menu for them to select a ship and change its orientation.

• The player can then move the mouse on to the board and receive visual feedback of the orientation of the ship, the tiles it will be placed on and whether the ship of that type has been placed before.

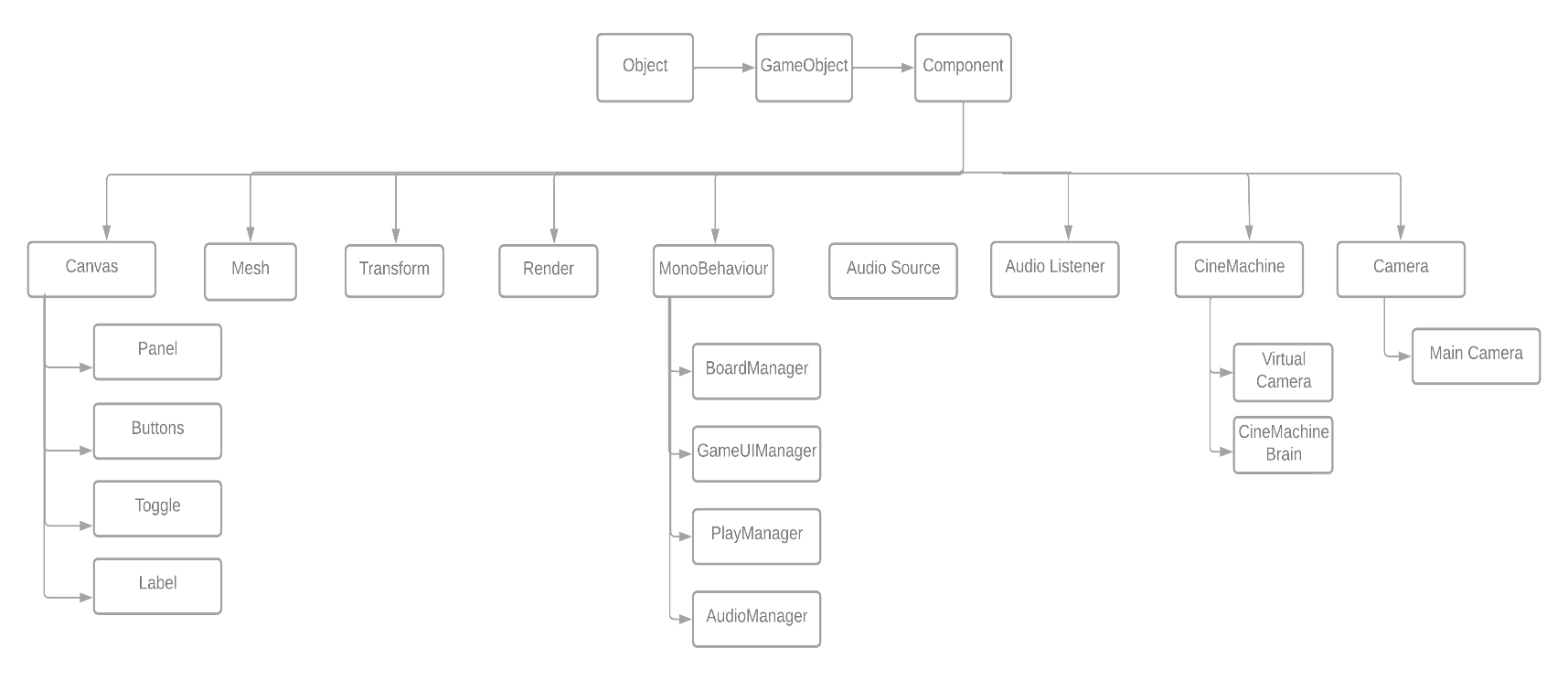
• the user at any stage of the game is able to access the pause menu.

o In it the user can select to resume the game, go into the options menu to adjust volume control and activate cheat mode.

o It also grants access to return back to the main menu if the user wishes to exit the game screen.

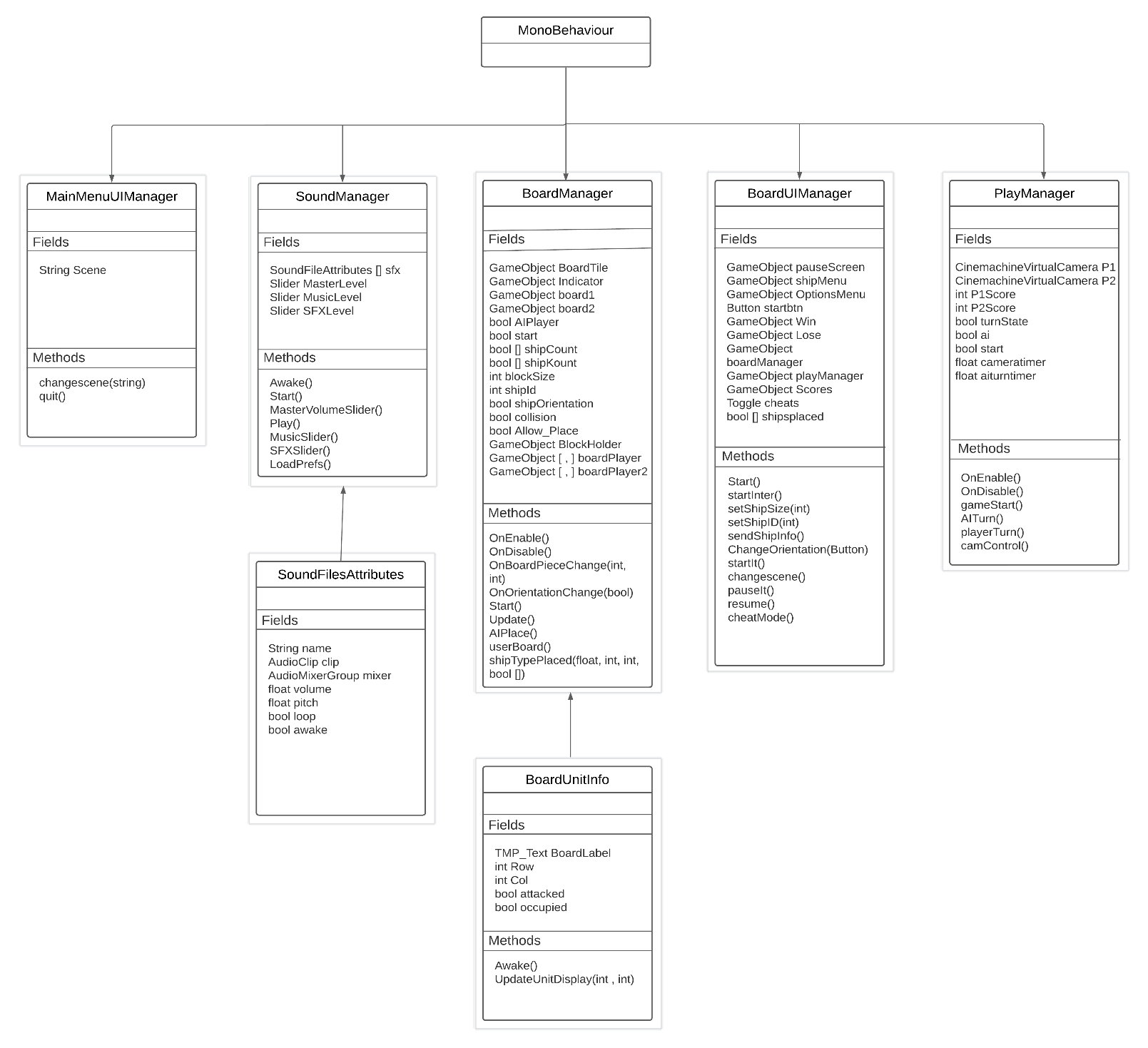
## Unity Class Hierarchy

The unity class hierarchy below demonstrates the components used to create many of the assets and its related functions within Unity’s game engine. The creation of GameObjects to sustain many of the components listed provide a foundation for the game.



## Detailed Class Design

A detailed diagram of the scripts and their methods.



# User Interaction

1. Clickable Buttons
   * Start: Used to load the “Game” scene
   * Options: Used to access Volume sliders and cheat mode.
   * Exit: Used to quit the game
   * Replay: Used to load the game scene again to reset the board state and be able to play again.
   * Main Menu: Used to return back to the “Main Menu” scene
   * Start: Used to start the game once all ships are placed otherwise the button is not interactable.
   * Orientation: Used to change the ship to be place orientation to either vertical or horizontal.
   * Resume: Used to return back to the game from the pause menu.
2. Sliders
   * Volume: Used to control the volume for the entire game, this includes music and sound effects.
   * Music: Used to the music volume level for the game.
   * SFX:
3. Toggle
   * Cheat Mode: enables player one to know enemy ships positions

# User Interface

The user interface for the game was kept simple to not overwhelm the user with too many animations or sounds. The menus have a golden border to it and a fading gold to grey as the textures reach the center of the menus. The buttons are a dark grey when the user isn’t hovering over them and become white and plays a sound once the user moves the cursor on top of it and a sound is played when the button is clicked. The menu titles are bolded and colored white to help remind the user what menu they are on.

# Screenshots

1. The Main Menu scene

A picture containing graphical user interface

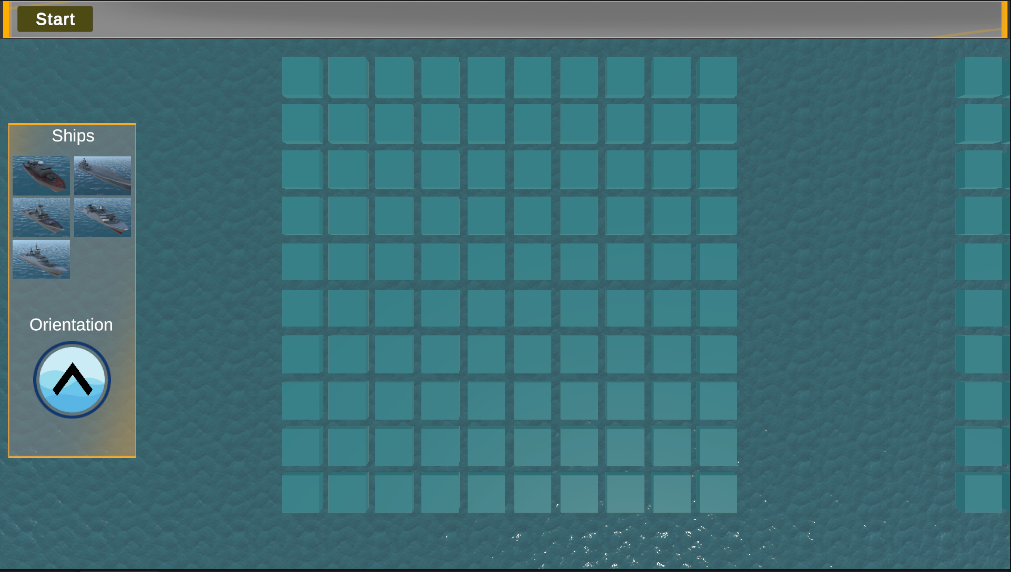
Description automatically generated

1. The options menu on main menu scene.

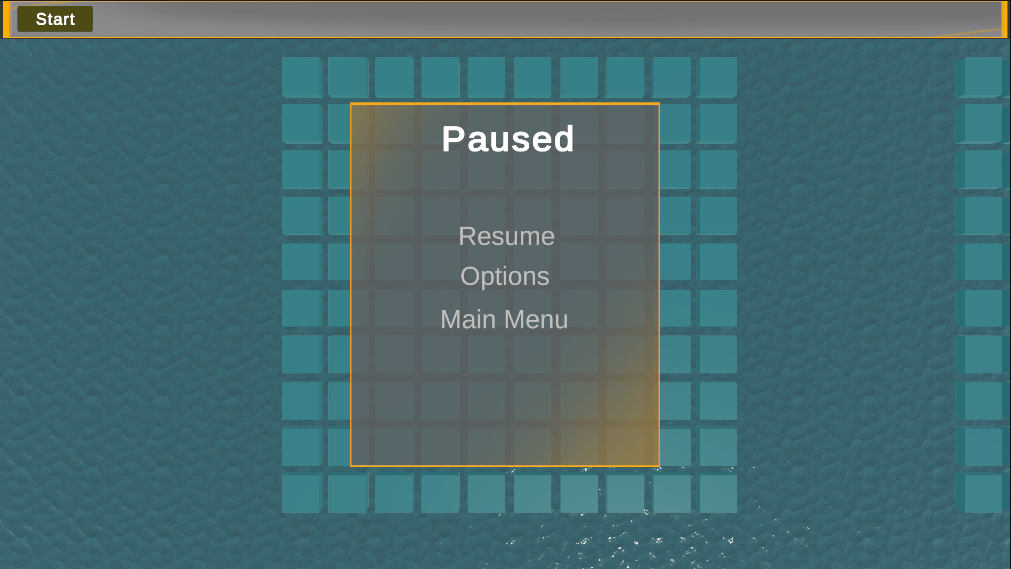
Graphical user interface

Description automatically generated

1. The ship menu for ship placement and top menu bar to start the game on the Game scene.



1. The pause menu screen.



1. Options menu again the game scene with the sliders set to the same level from the main scene.

Graphical user interface

Description automatically generated

1. Ships placed and orientation button set to horizontal and start button becomes interactable.



1. User attacking the other player board, it shows a red tile where a ship was hit and a darker tile where the user missed earlier.

A screenshot of a computer

Description automatically generated with medium confidence

# Lessons Learned

Battleship Part 1

The first part of the project involved using a lot of what we learned in class at the time. The material itself was dense and could sometimes prove difficult to understand due to the nature of how involve it is working with objects in a 3D space and changes its attributes. It is why the use of Unity help solidify some of the harder concepts with the use of the provided documentation and interface the unity editor comes with. Unity and the use of C# scripts were new to me as I never used them before, but C# was actually a surprise of how similar it was to C and java in general. Furthermore, I ended up creating prefabs, editing their attributes, instantiating them to create the board and the models used for the ships on the game. It was a different experience than I’m usually accustomed to seeing code do something visually instead of being used to handle databases, algorithms, back end, or front end, etc.

Battleship Par 2

The second part of the project really amped up the challenge. By a previous assignment I learned about the universal rendering pipeline to create a water asset which later on was implemented onto the battleship project. This made things a bit more difficult since there weren’t many resources to reference about URP in general. It broke quite a few of my textures used for the menus, ships, and a shader I was using to blur elements on my menus which I was unsuccessful at porting over to URP. Later on in class we briefly went over particles effects such as smoke, explosions, fire which we could later implement onto our project. The unity interface for setting up particle effects was daunting and felt overwhelming to try and create anything in it. I also face similar issues to textures not loading properly through the use URP with particle effects hence why later on I moved to use the visual effects graph addition for unity. This made the interface for creating effects a tad more friendly and similar to state machines on unity. Additionally, for the project I struggle to come up with a way to display my player boards to the user. This is when it dawned on me to use a camera transition to move from one board to another depending on whose turn it was. In order to do just that I used the cinemachine library to ease handling the transition of the camera movement from one board to another. Overall the project was an interesting mix of what we learned during class and putting it into practice. It made me appreciate how far things have come from the difficulty of setting up an environment to do something similar in OpenGL as opposed to Unity.