Design Document

CSCE 361 - Fall 2017



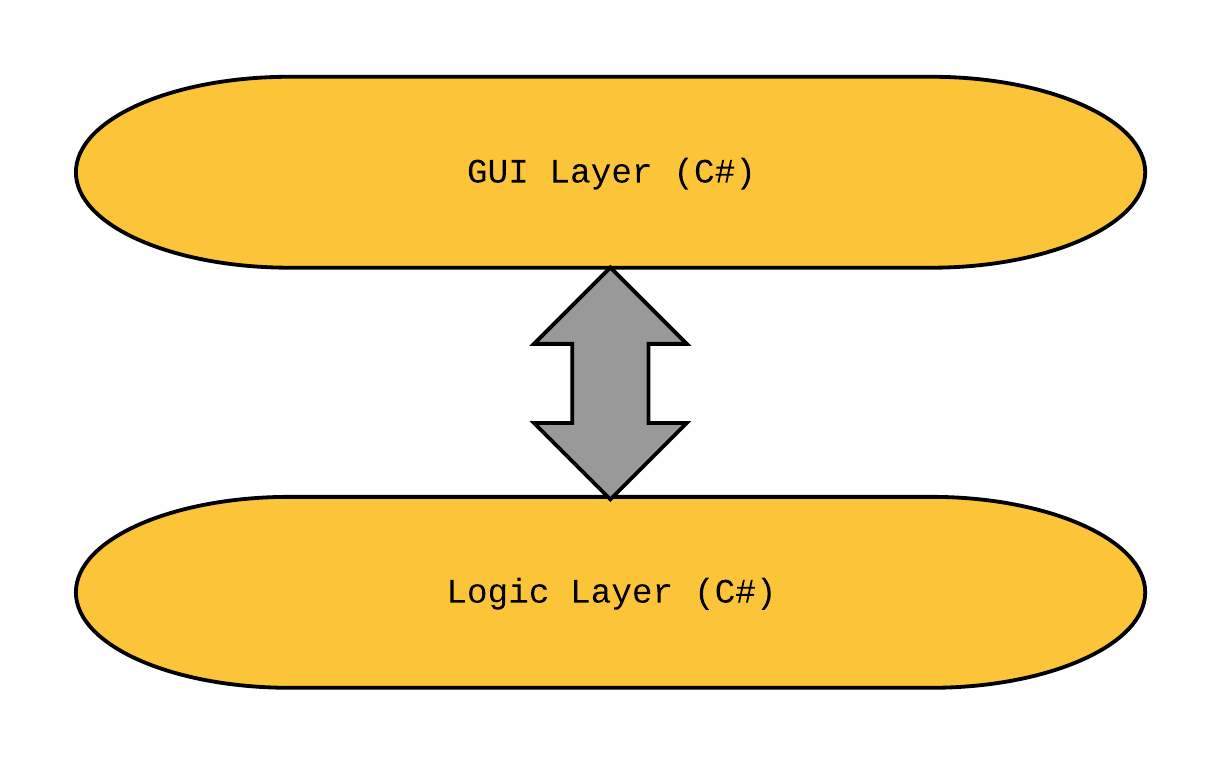
Avinash Nooka, Grant Harrison, Joseph Field III, Jared Nightingale

**1 Introduction**

This design document will provide an in-depth overview of the architecture of the Crazy Cannonz game. This document will also include diagrams to provide a visual reference for various classes that will be used in Crazy Cannonz and how they relate to one another. This document is intended for programmers and project managers for the purpose of implementing the game into actuality.

**2 Architecture**

**2.1 Introduction**

****

A high level design of Crazy Cannonz will be represented as a layer diagram consisting of the two main layers that make up the game. At the top level, a GUI layer will manage an interface that allows the user to input information. This will allow the player to select map, character, matchmaking, and settings information. Additionally, the GUI layer will consist of the sprites, physics, and graphics that the Unity engine produces. This information is then passed down to the logic layer, which is the base layer of the system. The logic layer is responsible for all computations and will consist mainly of C# scripting. The logic layer will handle events in game, such as player movements, ammunition pickups, and combat.

**2.2 Modules**

**2.2.1 Logic Layer**

The logic layer is responsible for booting, managing, and controlling the flow of the game. It will also be responsible for interpreting user input, spawning GameObjects, communicating with the GUI layer to update each frame before being displayed, as well as carrying out all other general game functions.

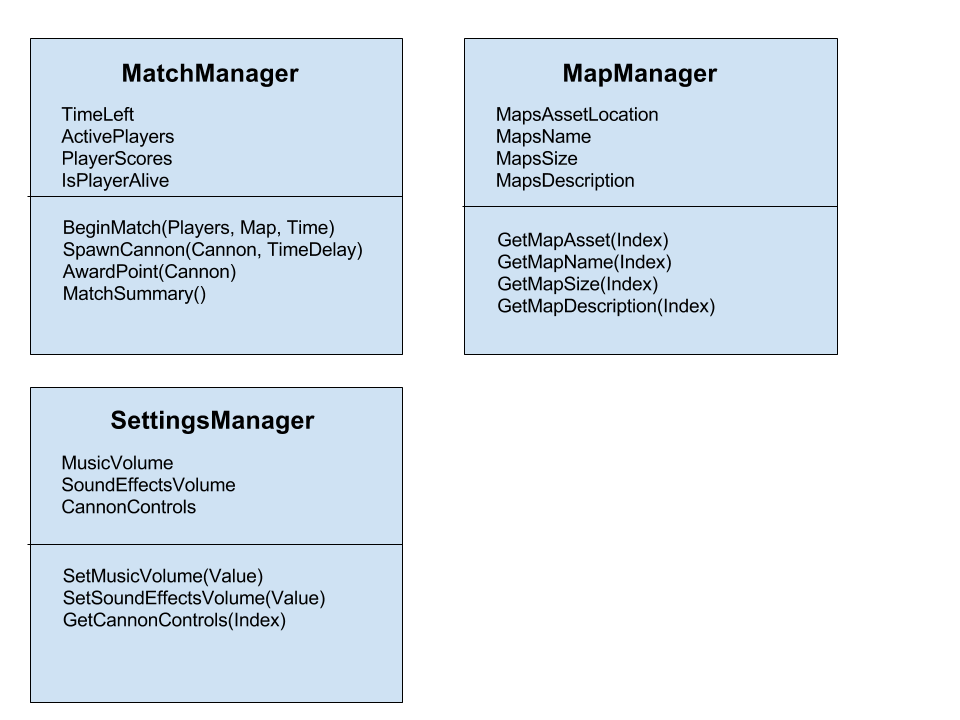
**2.2.2. GUI Layer**

The GUI layer will be responsible for creating and maintaining a user interface that allows the user to navigate a menu system. It will also allow the user to manage settings for gameplay. The entire interface will be scripted in C#, and will be connected to the logic layer to pass input information on to be processed.

**3 Class Diagram**

**3.1 Data Table Classes**

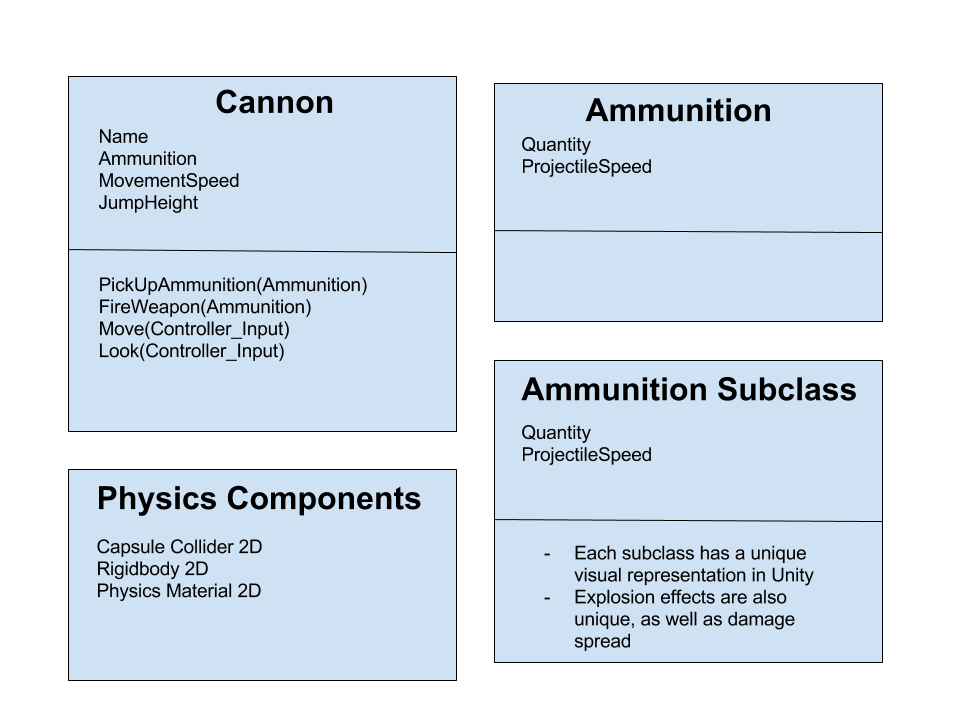
The system will be using native Unity functions to manage and manipulate data as necessary. This includes information on point totals during matches as well as user and object interactions. The figures below show various entities in the game, and their various properties and functions.



The MatchManager will be used to begin and keep track of a match. It can store information about which players are in the game, their scores, and their status of being alive or dead inside three arrays. It can award players points, and it can spawn players when they die. When time runs out, it can display a match summary and take the players back to the Matchmaking menu.

In addition, the MapManager is capable of retrieving data about the maps stored in the game. Since this data will be hard coded in, there is no need for any setters; the arrays which store the values will be prefilled.

Finally, the SettingsManager will allow for access of the controls data (information like the ‘up arrow’ will perform a jump, and so on), and will allow for function calls to change values which represent the volume of music and sound effects.

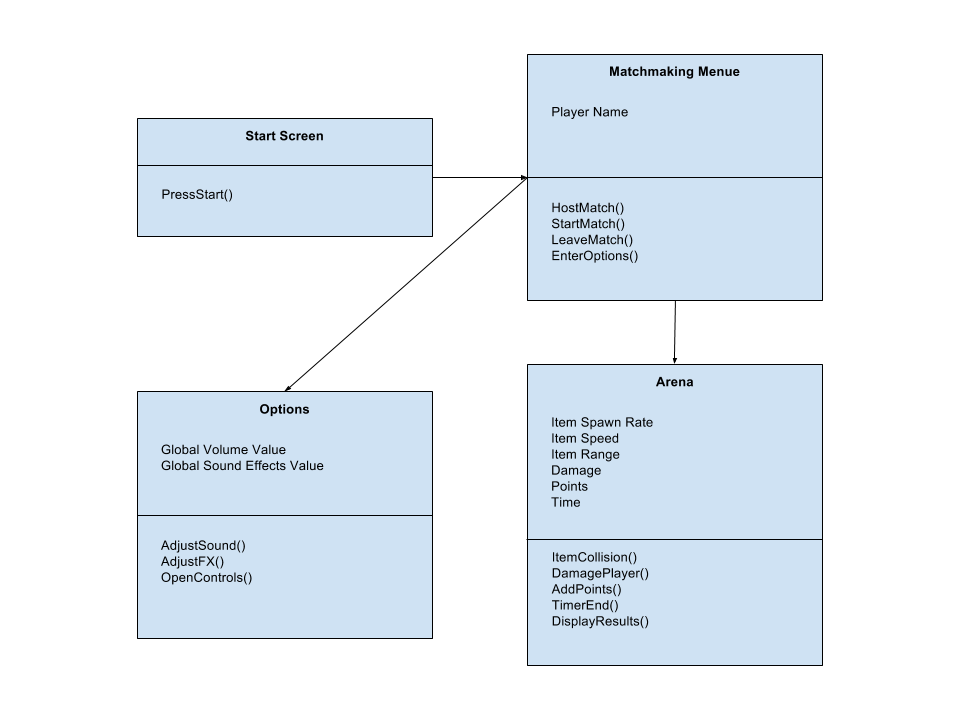


In match, each player will control a Cannon GameObject. This will have properties such as a Name, Movement speed, and Ammunition. A player shall have four basic functions when controlling a Cannon object: Pick up ammunition, fire their weapon, move, and look.

Players can pick up Ammunition GameObjects. These will have various subclasses, which will have different damage spreads, projectile speeds, and effects on player movements. This will be handled in Unity through its physics-altering components, which can be attached to the Cannon GameObject.

Each of these kinds of classes will need to have the above physics components, and this is provided by Unity as something every GameObject can have.

**3.2 GUI Layer**

****

The GUI system contains four pages: the start screen, matchmaking menu, options menu, and the arena. When the user first starts Crazy Cannonz, the start screen is presented where a user will press either a button on his/her keyboard or controller to enter the matchmaking menu. From the matchmaking menu, the user can create or join a match lobby to arena. Here, players can join the match (by pressing a button), can select a map (from a drop down list), and edit the match time length (another drop down list). The user can also access the options menu, which will allow the user to change the music and sound effects volumes via sliders, and view the controls diagram.

In the arena, the game will handle several tasks simultaneously. The game will synchronize with multiple user inputs as well as share item pickup on a first come, first serve basis. In addition, object collision to simulate damage received will be translated to points which will be added to a sum for each respective player who deals said damage. When the timer ends, the match also ends and a window will appear displaying each user’s score as well as who won the match. When the user exits the score window, they will return to the match lobby where they can decide to either continue with the same group or leave for another lobby.