CSC 431

Above The Grave

System Architecture Specification (SAS)

Team 5

Drew Barnes	Requirements Engineer
Daniel Barry	Scrum Master
Parth Sharma	System Architect

Version History

Version	Date	Author(s)	Change Comments
1.0	04/04/23	Drew Barnes Daniel Barry Parth Sharma	First Draft
2.0	04/30/23	Drew Barnes Daniel Barry Parth Sharma	Updates based on feedback

Table of Contents

Table of Figures

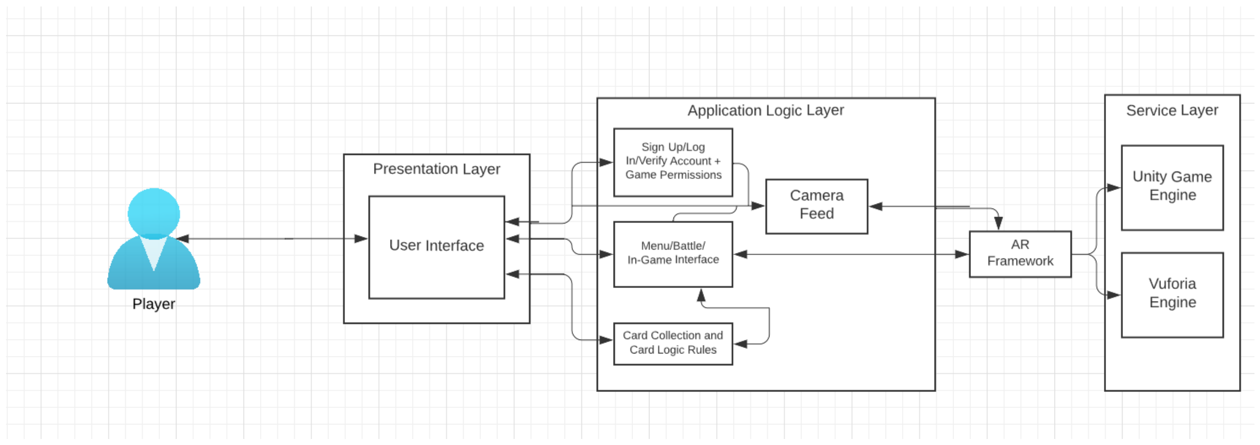
1. System Analysis

1.1 System Overview

The system of Above The Grave is an augmented reality (AR) card game that allows players to engage in battles using physical cards and their camera-equipped mobile device. The game is designed for Android and iOS devices and heavily relies on the user's phone camera to create a sense of immersion via animations/effects while playing this trading card game.

The system follows a three-tier architecture, with a presentation layer, a logic layer, and a storage/data access layer. The reasoning behind this architecture choice is its organization and reliability.

1.2 System Diagram



1.3 Actor Identification

Player

The Player is the primary actor of the system. They use the Above The Grave mobile app to capture images of physical cards with their phone camera and engage in battles with other players. The Player *can create a profile, manage their card collection, view their game progress, and change various options within the game.*

Database

The Database (aka the storage/data model) is an external system actor responsible for storing and retrieving game-related data. The Database is integrated with the game engine and handles tasks such as storing player profiles, card collections, and game progress data. The Database also manages game event logs and statistics.

1.4 Design Rationale

1.4.1 Architectural Style

We will be implementing the three-tier architecture. The motivation for this architecture is organization and reliability. This is considered the most common architectural style. Our 3-tier architecture provides independent areas for developing the application and lets each member of our team work on their own area of expertise.

This first tier is the client, also known as the presentation tier. This is the topmost level of the application and our user interface. Above The Grave's interface translates all the tasks the user needs and presents them in a way the user understands.

The next tier is application logic, also known as the logic tier. This tier is essentially the heart of the architecture. In response to a query that a user makes from the client tier, this layer processes the commands, makes logical decisions, and performs the calculations to generate an output from the storage tier. This tier will manage areas like user card permissions based on their collection and scripts to make moves/attacks in the Battle UI.

The third tier is the storage/data model. This tier contains all the user information/data and can be manipulated from the programs/scripts in the application logic layer. Above the Grave will have a filesystem representing the database. The information in our storage system can be passed to the application logic tier for processing, and then back to the user. This tier will consist of data such as a user's card collection, private information a user enters when signing up, and descriptions/moves for each card.

1.4.2 Design Pattern(s)

The design pattern we deemed applicable to this architecture is the state machine design pattern. This pattern helps us break down large tasks into independent smaller tasks. This pattern is also considered one of the most used design patterns in game development.

The idea of the state machine is hinted at in its title – you have a machine that can run different states (or programs). In Above the Grave, we may have states such as the Menu State, Game State, and GameOver State. In general, the state machine requires two basic scripts:

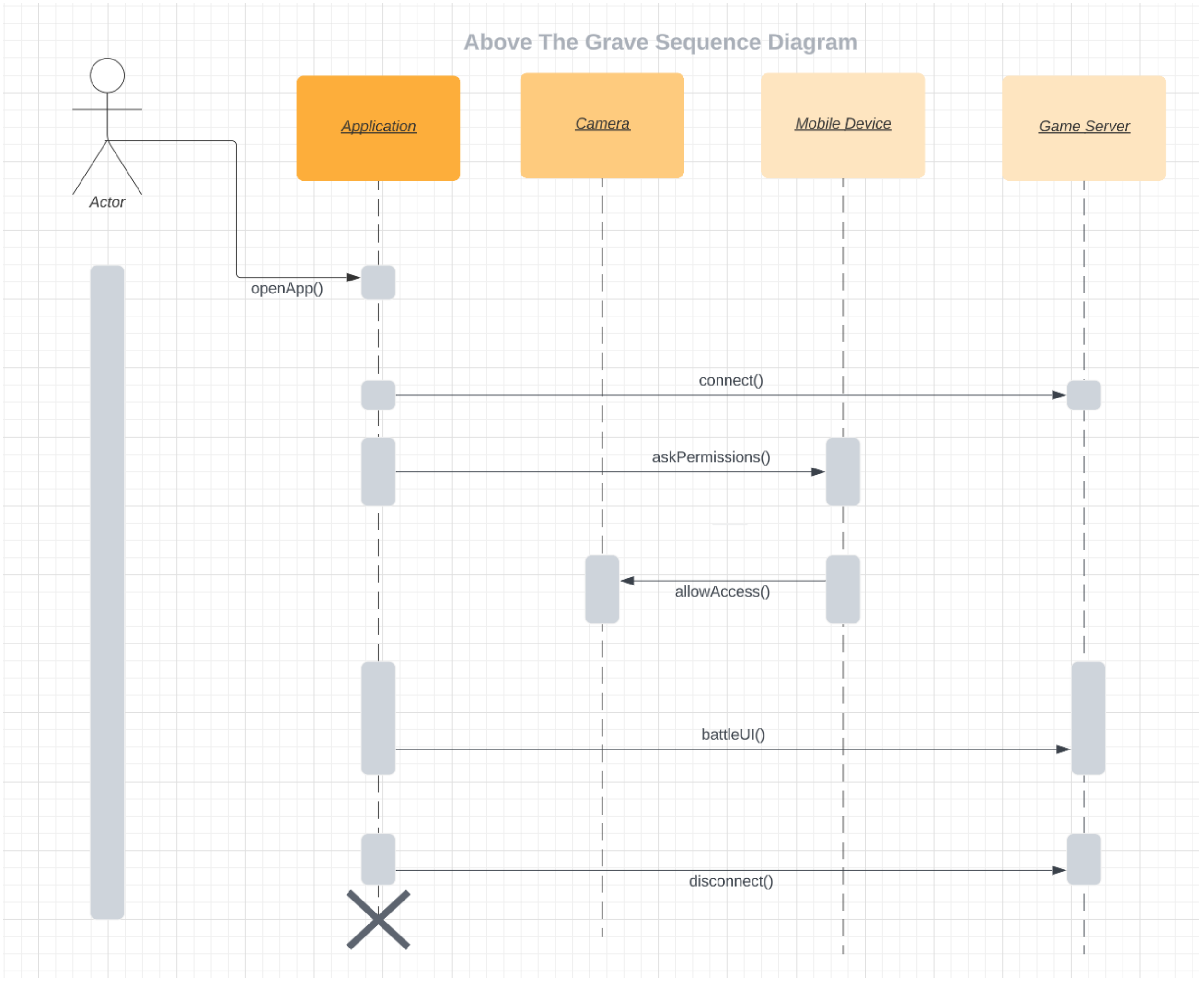
- *StateMachine: this script holds and updates our scripts*
- *BaseState: this script implements some virtual functions that will later be overwritten.*

1.4.3 Framework

The framework we will be using is Unity. Unity is the most popular cross-platform game engine. The rationale behind our choice is that Unity is supported on multiple platforms and gives developers the opportunity to create 3D games, which is the main selling point and heart of Above the Grave. It is also the most common area for AR development and is free to download. Additionally, all team members feel Unity has the friendly and intuitive user interface needed to make our developmental process smoother and more efficient.

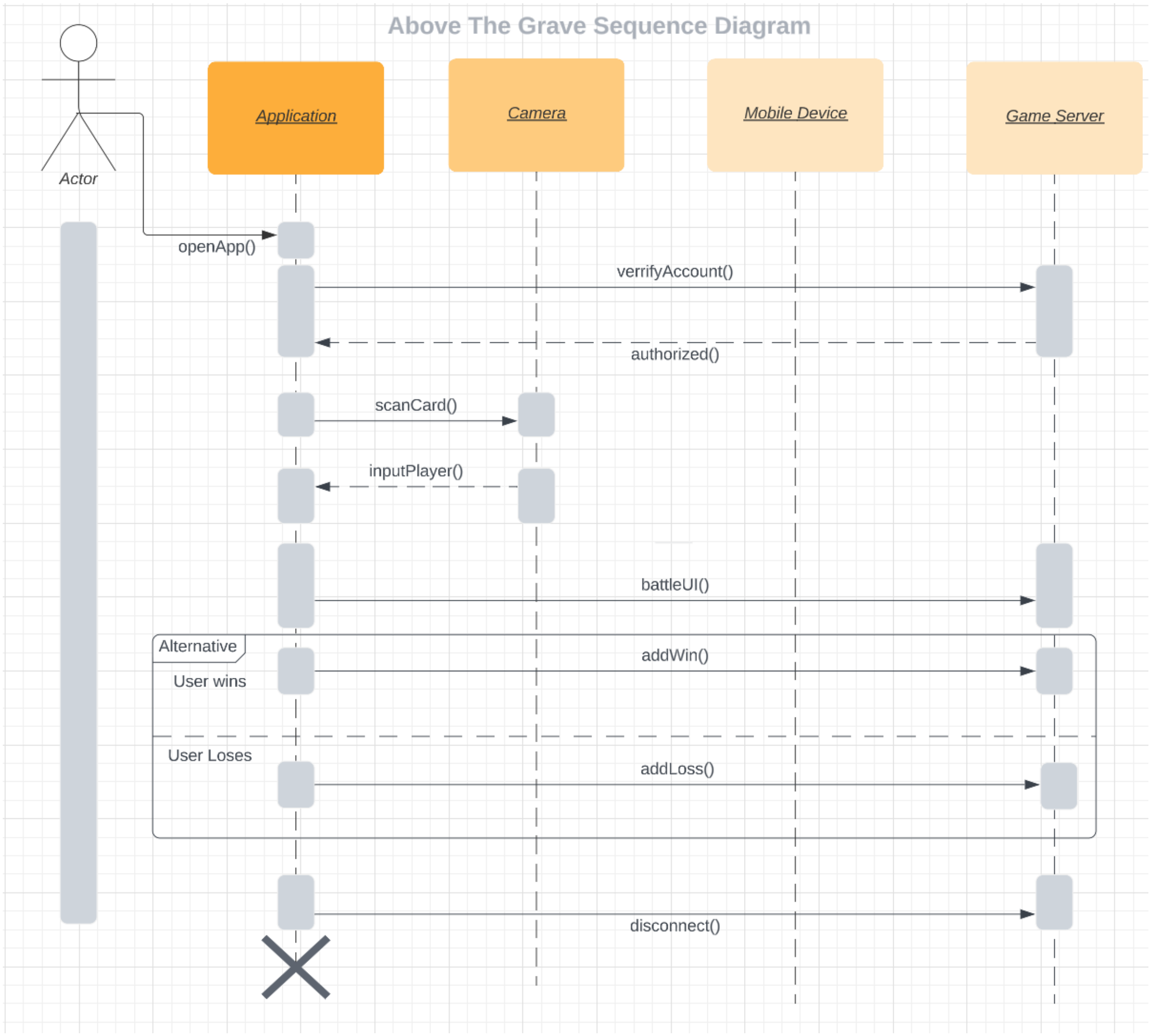
2. Functional Design

2.1 First launch of application



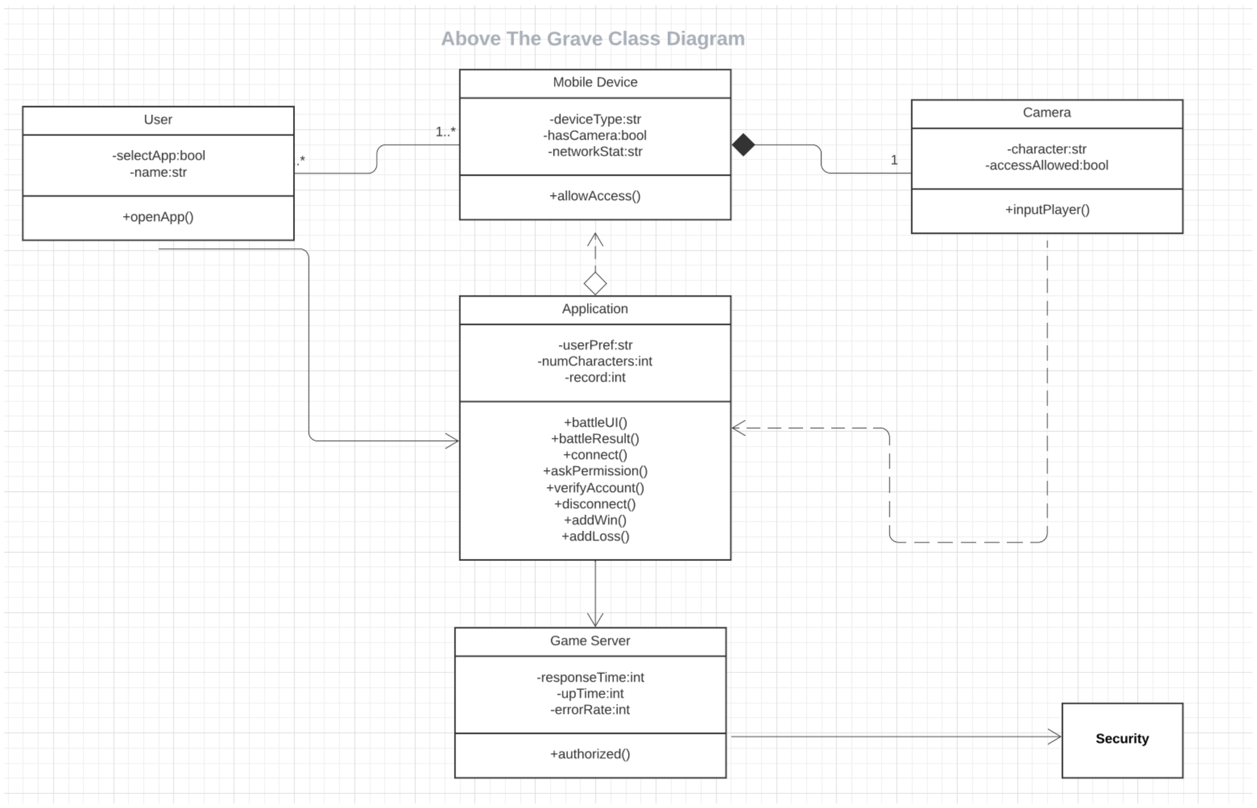
- *When the user opens Above The Grave For the first time, the app will first connect to the game server. This is denoted by connect().*
- *Above The Grave will then ask the user to grant permissions such as notifications haptics and sounds and any other preferences involved with the game experience. In the sequence chart, this is denoted by askPermissions().*
- *The mobile device class calls allowAccess(), Which allows the application to use the camera when playing the game.*
- *After this sequence of events, the application class will call battleUI(). This represents the user entering the battle interface.*
- *Lastly once the user is done playing Above The Grave the application class will call disconnect(). This represents the application disconnecting from the game server signifying the user exiting the game.*

2.2 Battling in Above the Grave



- *First, the user opens the application, which is denoted by openApp().*
- *The application class will call verifyAccount() which represents the user logging into their account. As a result, the game server will authorize the user, which is denoted by authorized().*
- *The application will then require the user to scan a card to select the character, this is denoted by scanCard(). In response, the camera will input the selected character into the application, which is represented by inputPlayer().*
- *The application class will call battleUI(), this represents the user entering the next portion of the game which is the battle user interface.*
- *Once in the battle, the user will either win or lose. If the user wins application class will call addWin() representing a win added to the user's record. If the user loses the application class will call addLosss() which represents the application adding a loss to the user record.*
- *Once the user is done battling, the application class will call disconnect(), this signifies the user is exiting the game server.*

3. Structural Design



Above The Grave Class Diagram
Mobile Device
-deviceType:str
-hasCamera:bool
-networkStat:str
+allowAccess()
User
-selectApp:bool
-name:str
+openApp()
Camera
-character:str
-accessAllowed:bool
+inputPlayer()
1..*
.*
1
Application
-userPref:str
-numCharacters:int
-record:int
+battleUI()
+battleResult()
+connect()
+askPermission()
+verifyAccount()
+disconnect()
+addWin()
+addLoss()
Game Server
-responseTime:int
-upTime:int
-errorRate:int
+authorized()
Security