MIAMI

CSC 431

Hot Air

System Architecture Specification (SAS)

Team 09

Titus Biel	Unity Developer
Tyler Tejera	Developer
John Reynolds	Developer

Version History

Version	Date	Author(s)	Change Comments
0.1	4/2/2023	Titus	Completed a rough draft of section 1.
0.2	4/2/2023	Tyler	rough draft of section 2
0.3	4/3/2023	John	Rough Draft of Section 3
0.4	4/5/2023	John	Backend Class Diagram for Section 3

Table of Contents

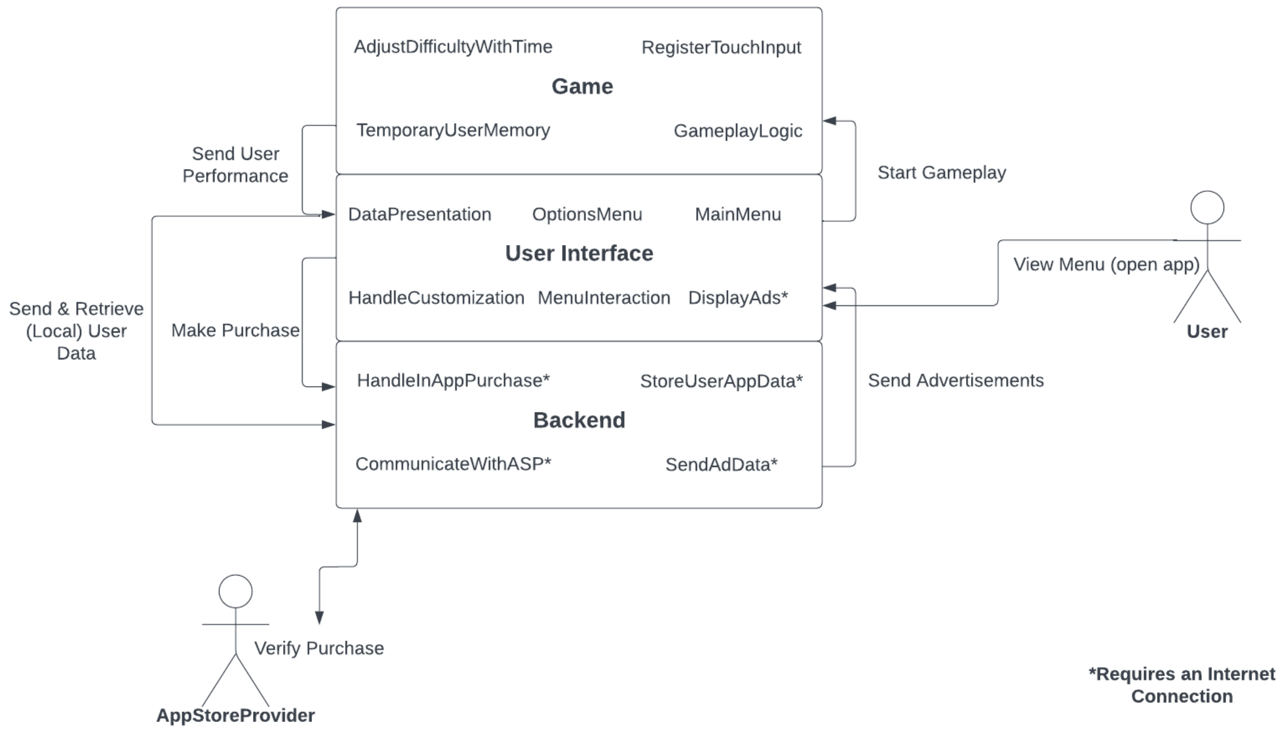
Table of Figures

41. System Analysis

41.1 System Overview

Our system consists of three separate parts, Gameplay, User Interface, and Backend, that all when working together, lead to the desired final product. Thus, the architecture that will be used is one that can adjoin separate parts together while allowing interaction between them, the Layered Architecture. On the very top layer will be Gameplay, in the middle layer will be User Interface, and on the bottom layer will be the Backend. Overall, the system will consist of these three parts and the interactions between them.

41.2 System Diagram



41.3 Actor Identification

There will be a total of two actors that will interact in the system. The first is the user who will make up the vast majority of all interactions. The second will be the App Store Server (either the Google Play Store or the Apple Store) which will aid in processing our in-app purchases in exchange for data on the transaction.

41.4 Design Rationale

1.4.1 Architectural Style

As was mentioned in 51.2, the architecture style that will be used is the Layered Architecture. The three layers will consist of a Game layer on top, a User Interface layer in the middle, and a Backend layer on the bottom. Firstly, the Game layer will cover all aspects related to gameplay and as the top layer, would be able to function even if the bottom two layers didn't exist. It will communicate only with the User Interface layer to send gameplay results. The User Interface layer will exist externally as the application's menus and internally as the application's traffic director, for instance by starting the game when the user taps play on the menu & directing data that needs to be stored to the backend. Finally, the Backend layer will consist of all data storage and foreign interactions (like with the app store) and will both receive and send data (ex. advertisements to display) when necessary to the User Interface layer.

1.4.2 Design Pattern(s)

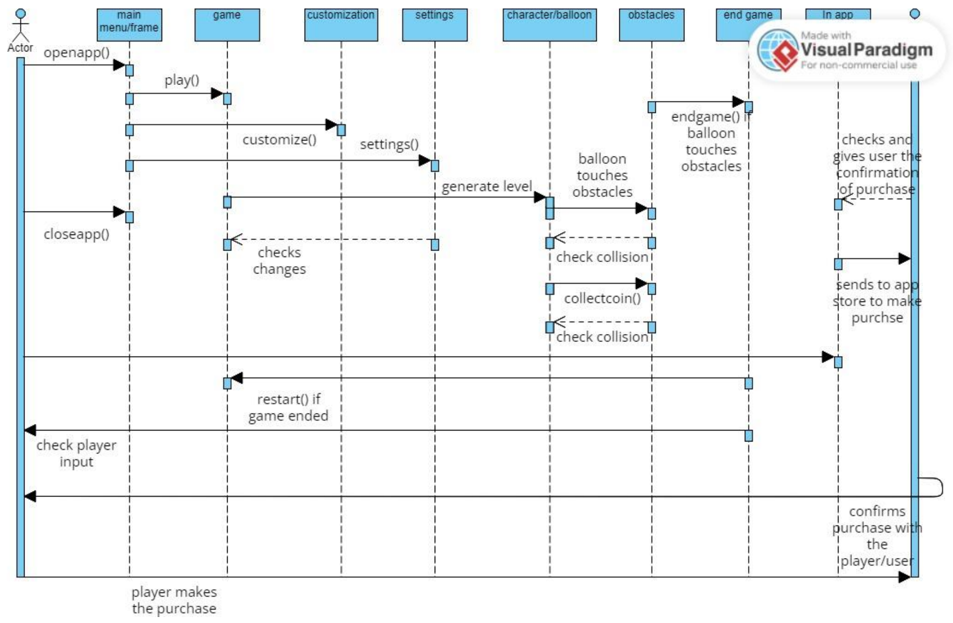
Due to the frameworks that will be used having inherent compatibility, design patterns will mostly be used to make each layer individually as efficient as possible. Firstly, the Game layer will consist of gameplay that is essentially infinite and yet should get more difficult as time goes on, so we will make use of the Factory Method in order to make game generation quick and efficient. Next, the User Interface layer will consist of visual menus and advertisements, but also of data transfer which can get rather complex. Thus, we will use the Facade design pattern to attempt to simplify everything into one versatile route, much like a highway. Finally, the Backend layer will need to utilize Adaptors in order to properly & efficiently communicate with both the User Interface and the phone's respective app store when necessary. This is because it will be written in Java, while the other layers will be written in C# and because it will be transferring data.

1.4.3 Framework

In order to put out a high quality application within a relatively quick duration, we will be utilizing frameworks to speed up the process. For the Game & User Interface layers we will be using Unity Game Engine due to its good reputation, free resources, and it being free to use commercially while we have made less than $100,000 with the game. Unity also has direct compatibility with both IOS and Android which will make deploying our application directly to the respective app stores a lot more fluent and less time consuming. For the Backend we will be utilizing Firebase for a variety of reasons. Firstly, it is hosted by a reputable company (Google) and is specifically designed to work with IOS and Android. Secondly, programming can be done in Java, a language that everyone in our team knows. And thirdly, it has compatibility with Unity through SDKs, meaning that simultaneous development with Unity as a central component is possible.

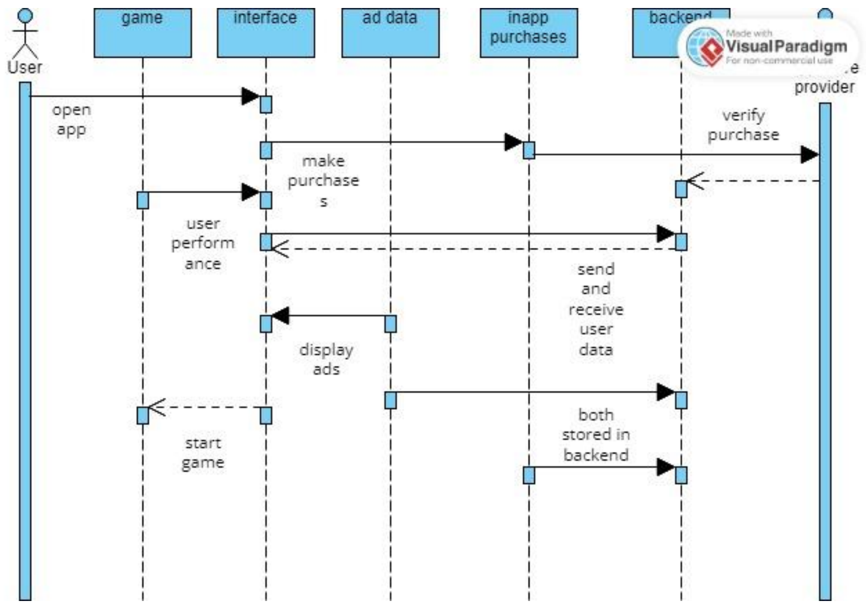
42. Functional Design

42.1 Gameplay and interface diagram



-top right is in app purchase and app store.

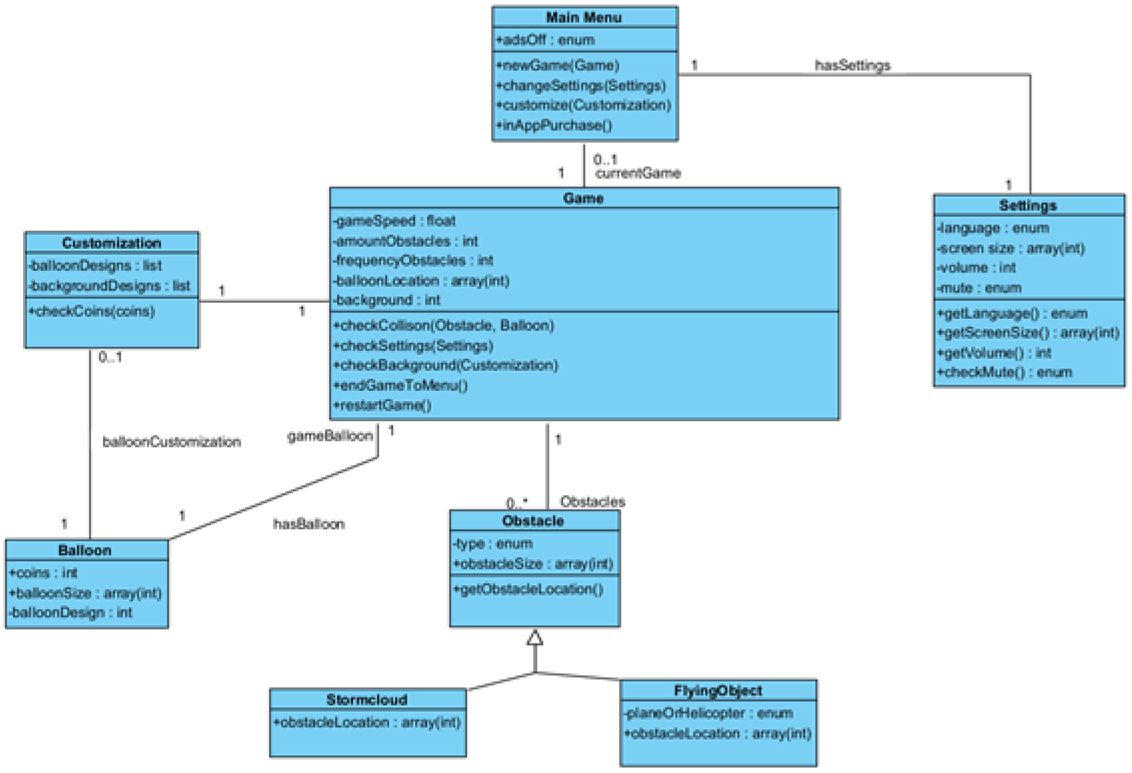
42.2 Backend and app store diagram



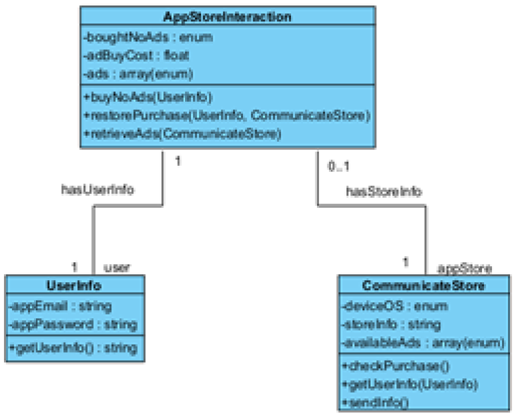
-top right is backend and app store provider

43. Structural Design

43.1 Front-end Class Diagram



43.2 Back-end Class Diagram


AppStoreInteraction
-boughtNoAds : enum
-adBuyCost : float
-ads : array(enum)
+buyNoAds(UserInfo)
+restorePurchase(UserInfo, CommunicateStore)
+retrieveAds(CommunicateStore)
1
hasUserInfo
0..1
hasStoreInfo
1
user
1
appStore
UserInfo
-appEmail : string
-appPassword : string
+getUserInfo() : string
CommunicateStore
-deviceOS : enum
-storeInfo : string
-availableAds : array(enum)
+checkPurchase()
+getUserInfo(UserInfo)
+sendInfo()