CS319

Object Oriented Software Engineering

Design Report

Space Out

Group 3-I

Doruk Çakmakçı

Arda Atacan Ersoy

Anıl Erken

Umut Berk Bilgiç

Contents

1 - Introduction

Space Out is retro style sci-fi platformer game that anybody can use to "space out" from their boring life. It is implemented using Java because, Java is one of the most common programming languages in the world. Therefore, we can reach various reliable sources to ease our job. Nearly all computers with standard hardware and software specifications can run Space Out using JVM. Space Out's graphics are 2D and its environment is designed with respect to a retro style platformer game. Not to use high graphics provides the user to get more performance and quick response time and this makes Space Out more playable. To be able to entertain the user, Space Out has different power ups with special animation effects. Local and global leaderboard systems attract more people to Space Out. From another perspective, creating a level from fundamental blocks, structures and enemies help user to unleash their creativity.

1.1 - Purpose of The System

The main goal of Space Out is to entertain the users with rich contents(character skins, retro-space themed graphics) and gameplay. Gameplay is in accordance with ease of use principle(only 3 keys on keyboard are needed). With this aspects, Space Out aims to make the users space out from their daily stress and problems.

1.2 - Design Goals

Design goals includes many aspects such as graphical advantages and game performance that makes the game more qualified and effective. These design goals come from non-functional requirements that are mentioned in the analysis report. Design goals explained elaborately in the below.

Graphical Advantages

Space Out is a 2D platformer game and that's why game does not have high-quality graphics but this case increases Space Out's game performance and in the game, user does not encounter any delays. By this performance advantage, lots of animation will be used in the game.Because of three distinct world concept, Space Out includes different materials, terrains, bricks and so on. All of these serve user to play a more attractive game.

Game Performance

Since middle quality graphics and 2D platformer game concept is used, every standard computer can run Space Out easily. Thus, in terms of performance, when user made a request in the game, response time will be very quick and user will not be exposed to delay. All of the game components have been chosen with respect to get more performance but graphics and performance relation causes a tradeoff. By considering both of their advantages and disadvantages, we try to maximize user's comfort and joy by our decision.

Interface

User interface is designed according to facilitate player's game experience and ease of learning game. In order to play Space Out, user must login or signup, but we hold these pages simple as much as we can. In the menu we only put necessary information and buttons to make the interface simpler. We only add massive features to the system such as level maker and leaderboard system, and these features are very clear and understandable in the game. Additionally, gameplay and character control buttons are arranged to make user more comfort in the game. For instance, we have used "A", "D" and "space" keys to control the character since this type of control is widely used in the gaming industry. All of design procedures are done to comfort user in the game.

Extendibility and Maintainability

One of the most important goals in the software programming is the life of program. Hence, program's subsystems, implementation and features should be appropriate for extendibility and maintainability. Java, IntelliJ, MVC pattern and other popular tools are used in Space Out. Because popularity and wide usage of these tools provide us various easiness and options while we are implementing game. Without affecting and changing many subsystems, we could add new features to the game. Wide usage of them makes Space Out more reachable and everyone can run and play it easily. Since these tools are compatible with every standard computer, mobile phone and tablet.

Modifiability, Flexibility and Adaptability

To be able to satisfy extendibility and maintainability, Space Out must satisfy modifiability, flexibility and adaptability first. As long as program is adaptable and flexible, it is also extendible and maintainable. Thus, subsystem coupling is tried to be minimized. This leads us to change unwanted features of the game easily or adding new functionalities as well. Besides using level maker, Space Out will be flexible for each user demand. On the other hand, implementing low coupling subsystems is difficult for us but it has great advantages for users.

High Availability, Reusability and Productivity

Leaderboard system and the usage of Java increase the availability of Space Out. And also by having custom level maker, our blocks, terrains, enemies and other objects are reused and they will be more effective in the game. When user creates their own level, they will be thinking like programmers and by that their productivity and reasonable thinking skill will be increased.

1.3 - Tradeoffs

Usability – Functionality:

SpaceOut's aim is to attract lots of people interest and to be able to achieve that we have to design appropriate game for each user. In this manner usability is very essential design goals for us. However, in order to keep users' attraction on Space Out, some functionalities have to be included in the space out. As long as, we add new functionalities, it makes user interface and the usability of the game more complex. Thus, we have to make smart decision to be able to balance this tradeoff.

Space Out's game play and user interface will be simple and we decide to add new features to the game such as level maker, leaderboard system. These features will not make complicate Space Out. By finishing all of these, user spends time enjoying the Space Out rather than struggling to learn it.

Space – Performance:

As we aimed at a wide range of user scale, Space Out must be as fast as possible because it should be playable for each user. To obtain the the best performance, we should keep minimum space but minimum space requires minimum features, minimum visionality and minimum functionality. In this complex relation, for global leaderboard system, we will use Google Cloud Platform's support. We will maintain a big portion of suer related data in the cloud. This way by allowing some performance downgrades we hugely improve space needs and overhead.

Efficiency – Reusability:

Efficiency is very essential in Space Out since it has some cloud-based features and lots of different characters, enemy and level-maker. If we do not have an efficient program, all of these will cause a big load on the user's computer system and this circumstance would cause some performance hiccups. On the other hand, reusability is also very important in our game. Because Space Out includes nine different level as a default and also we have a level-maker that each user can create their own level. For default levels, we will also plan to design these levels by using level-maker and also each character and other stuff must be suitable for custom levels. In these relations, reusability is very important in our case but we have to be

careful about efficiency as well. If we are not careful, we could be giving up a lot of efficiency in the name of reusability and vice versa.

Object Oriented Programming – Development Time:

To be able to develop Space Out to be more efficient, usable, reusable and functional, object oriented design is more appropriate for our task but in roder to develop object oriented software, we should analyze the game and its requirements elaborately. On the contrary, our time is limited and sometimes it is hard to finish implementation until due date. To avoid this, we started to implement Space Out as early as possible. We also try not to waste too much time on pre-planning with diagrams and other theoretical work but rather start coding and shape the future of the program partly according to the plans and the implementation.

Modifiability – Robustness:

Similar to how reusability is important for Space Out, modifiability is also because these design goals arise together, since a modular code has reusable parts. Therefore, we should implement Space Out in a modifiable way. This also facilitates to add new fetures to the game. On the other hand, robustness is another necessary design goal for Space Out because in the game lots of dynamic features will be such as level-maker, global and local leaderboard system and all of these must work properly. But if we modify Space Out frequently, this may cause problem in these features and it can be hard to solve problem in a limited time.

1.4 - Definitions, Accronyms and Abbreviations

MVC(Model View Controller): The Model-View-Controller (MVC) is an architectural pattern that separates an application into three main logical components: the model, the view, and the controller.[1]

JDK(Java Development Kit): The Java Development Kit (JDK) is an implementation of either one of the Java Platform, Standard Edition, Java Platform, Enterprise Edition, or Java Platform, Micro Edition platforms[1] released by Oracle Corporation in the form of a binary product aimed at Java developers on Solaris, Linux, macOS or Windows.[2]

JVM(Java Virtual Machine): A Java virtual machine (JVM) is an abstract computing machine that enables a computer to run a Java program.[3]

JRE(Java Runtime Environment): Java Runtime Environment (JRE) is a software package that contains what is required to run a Java program.[4]

VPC(Virtual Private Cloud): A virtual private cloud (VPC) is an on-demand configurable pool of shared computing resources allocated within a public cloud environment, providing a certain level of isolation between the different organizations (denoted as users hereafter) using the resources.[5]

GCloud(Google Cloud): Gcloud is a tool that provides the primary command-line interface to Google Cloud Platform.[6]

HDD (Hard Disk Drive): A hard disk drive (HDD), hard disk, hard drive or fixed disk is a data storage device that uses magnetic storage to store and retrieve digital information using one or more rigid rapidly rotating disks (platters) coated with magnetic material.[7]

2 - Software Architecture

2.1 - Subsystem Decomposition

We have chosen to incorporate a three-tier architecture since it is the pattern that best suits our object design. Our three-layered architecture and the subsytems can be viewed below, note that the classes within these subsystems are not included in this diagram.

The three tiers we have are ordered in this way:

1st layer is the "View" layer. It consists of LevelMaker and MenuView subsystems.

2nd layer is the "Controller" layer. It consists of AccountHandler and GameController subsystems.

3rd layer is the "Model" layer. It consists of GameModel subsystem.

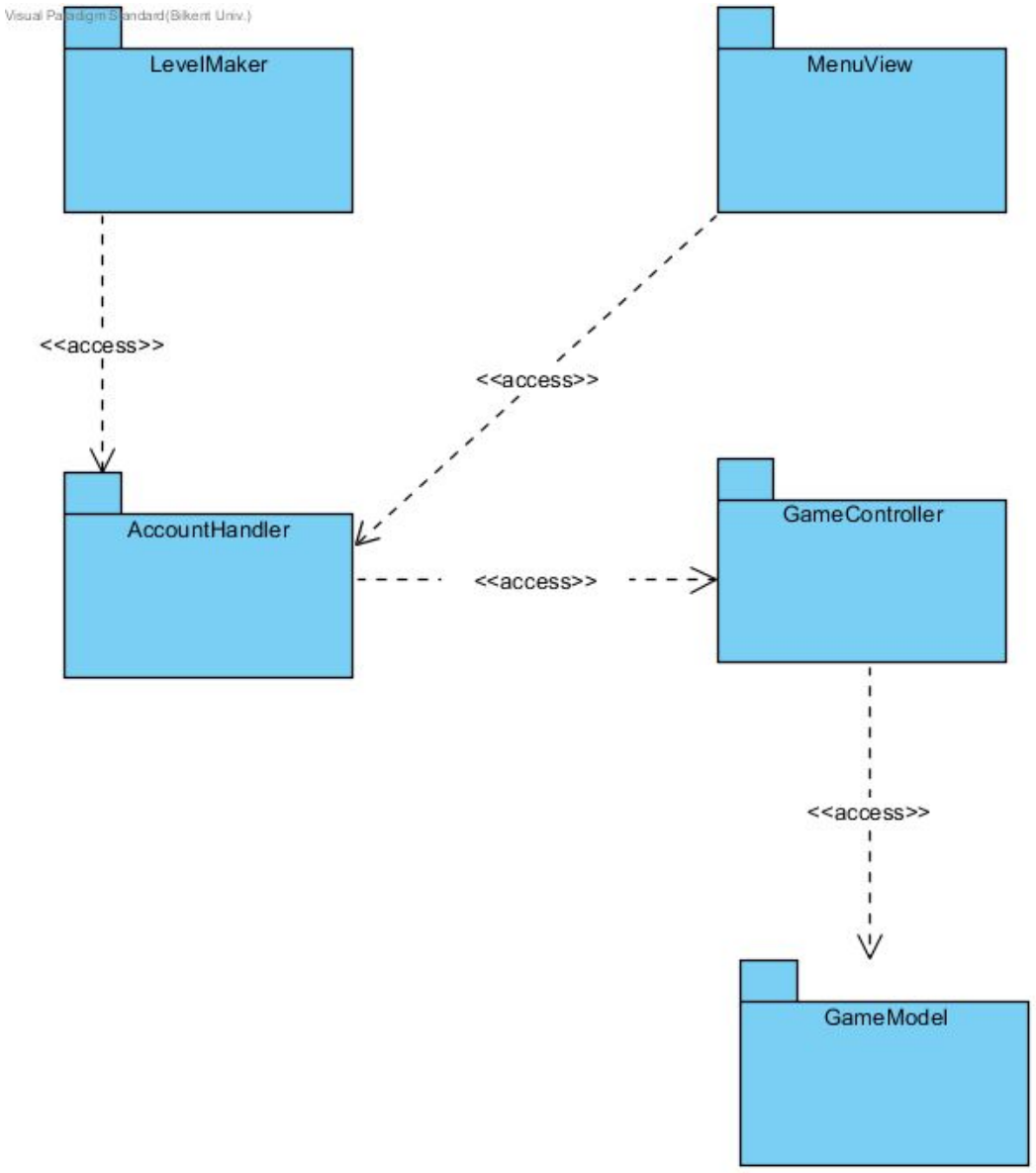
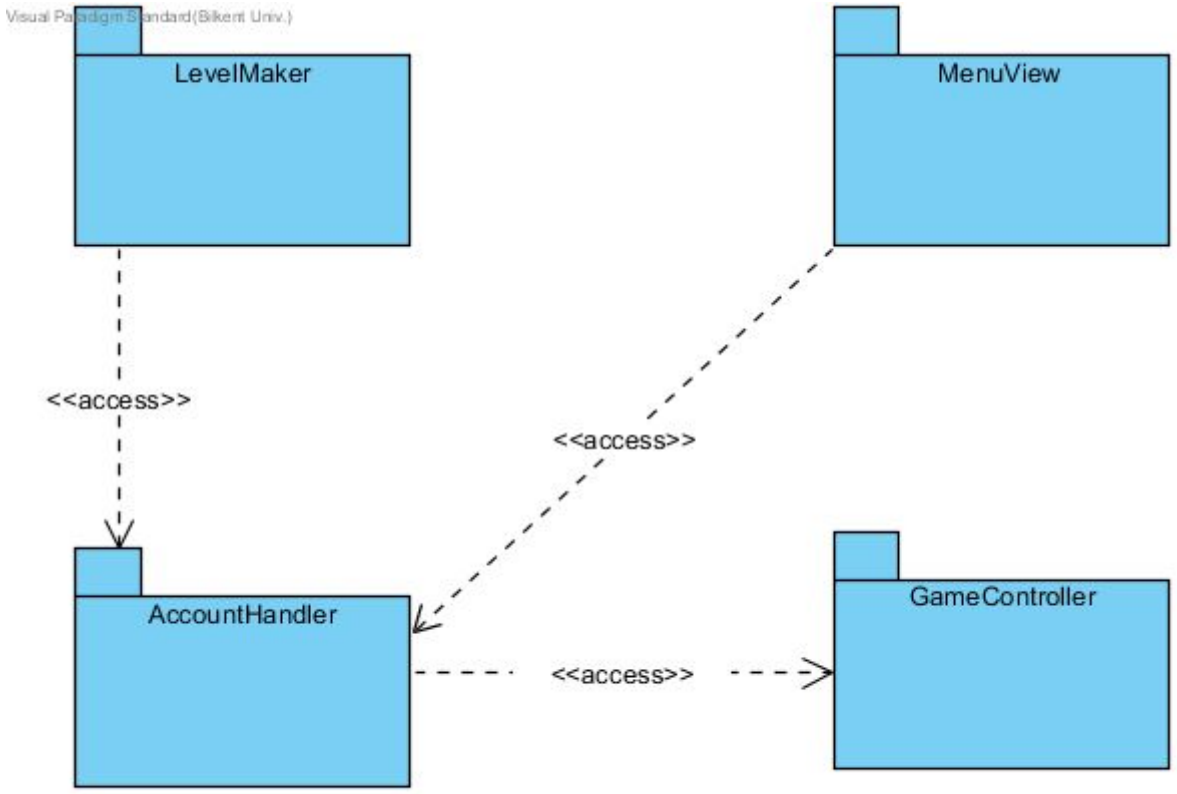


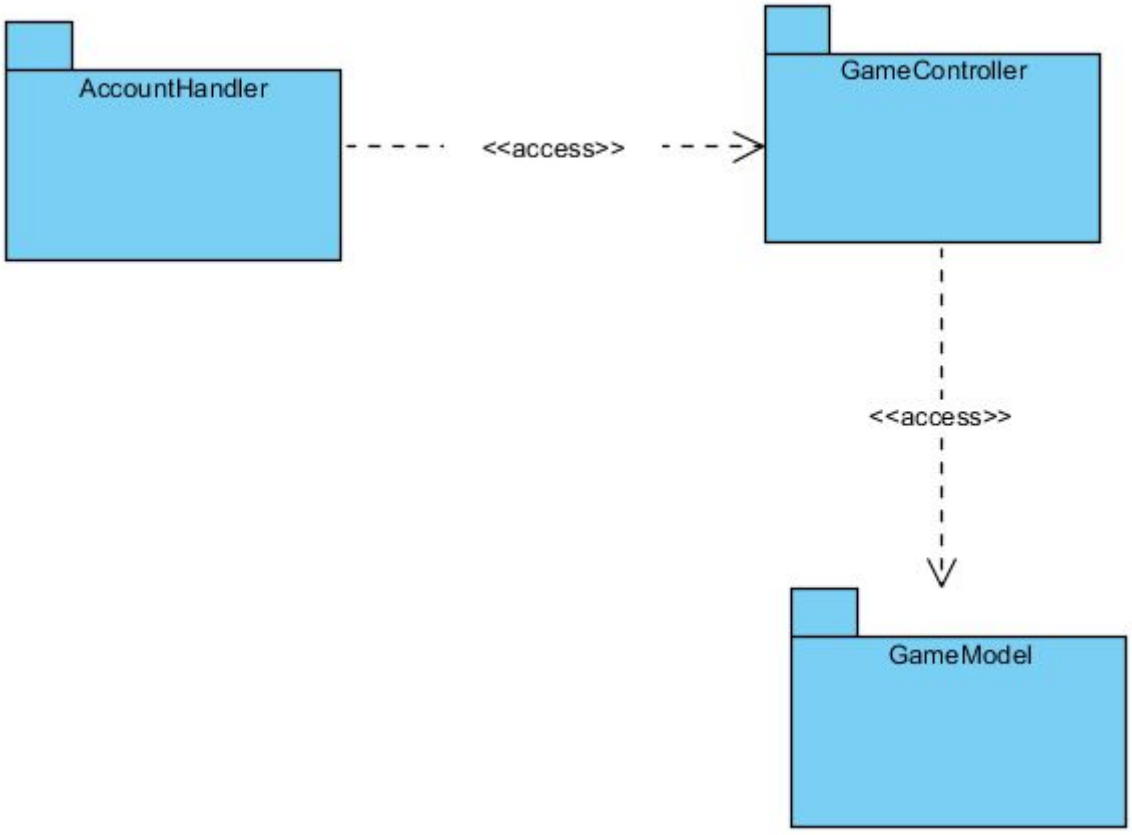
Figure 1: Subsystem Decomposition for Space Out

Interaction Between the View and Controller Layer:



In the view layer, the subsystem are responsible for things such as menus, buttons and other interact-able user interface elements. These elements use the data provided from the controller layer; namely the AccountHandler subsystem. The Level Maker subsystem within the view layer access the cloud to be able to upload and download levels from the cloud datastore. This interaction requires a connection between the level maker and the CloudQueryInterface class which is inside the AccountHandler subsystem.

Interaction Between the Model and Controller Layer:



The GameController subsystem within the controller layer utilizes the GameModel layer in order to construct and manage a playable game. The Game class within the GameController subsystem uses the data from the GameModel class within the GameModel subsystem in order to track various aspects of a game.

2.2 Hardware/Software Mapping

Space Out runs on JVM. Therefore, as a software requirement, Java must be installed on the computer where Space Out is going to be played. When java is initialized successfully, the user needs to open the .jar file to play the game. Internet connection is essential for Space Out due to cloud sync for factory levels,

achievements, updating global leaderboard and in essence, to log in or sign up to play the game.

From hardware perspective, a standard computer with average processing power, average CPU(for computations in context of the logic of the game and java requirements) and GPU(for graphics processing and display). From another point of view, keyboard and mouse is needed to play and pause the game and select menu items and create levels with the drag and drop service of the level maker respectively. Gameplay of Space Out will be better if a stronger GPU is present in the corresponding computer due to GPU acceleration present in newer and more powerful GPU's.

2.3 Persistent Data Management

Game Data will be partitioned to cloud (Google Cloud's VPC service will be used) and client HDD. Some portion of data's are dynamic such as client-created levels, leaderboard entities and rankings, character skins that are unlocked with earning the corresponding achievement and achievements itself, log in and log on username and passwords etc.. The dynamic data will be stored on cloud. On the other hand, the data for default game maps and their components, main character's default skin, sound effects, main menu and gameplay sounds, images, graphics and sprites etc. will be held locally. All levels will be files with .lvl extension but, in order not to take more disk space from the client's machine, user will need to download the client-made levels to play.

2.4 Access Control and Security

To be able to play Space Out, internet connection is needed because each user must login or sign up to the system. User's information and their datas(leaderboard scores) are kept in Google Cloud Service in a safe way. By doing that each user's scores, achievements, opened power ups, skins and custom levels will be private for him/her. Google Cloud Service has various precautions for any security issue, thus user can sign up/login and play the game in a secure way.

2.5 Boundary Condititons

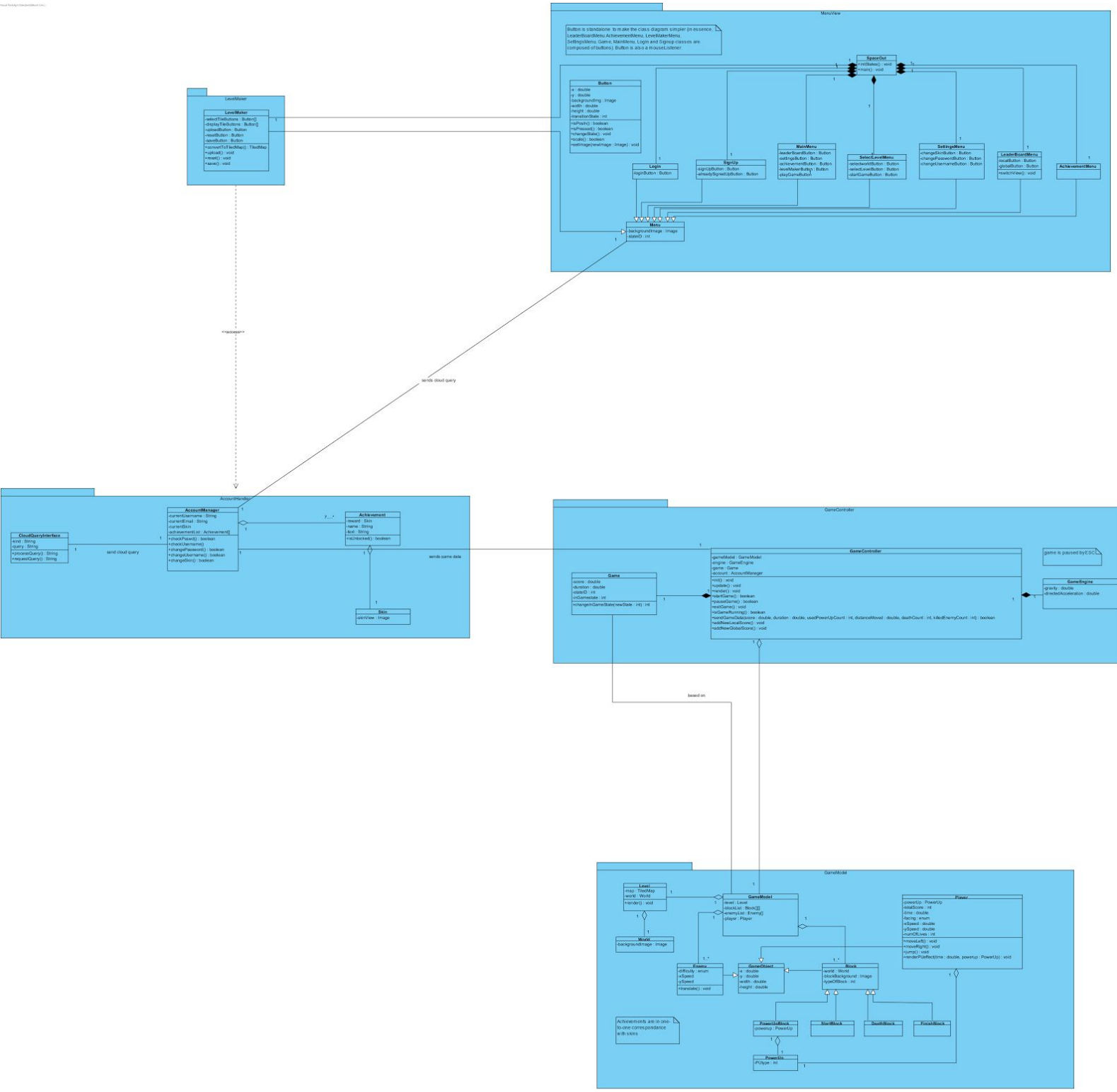
Initialization of The Game is the first boundary condition to play Space Out. Game will be extended (.jar) file. Therefore, user does not need to install the game because .jar files are executable, when user has an JVM. However, some levels and stuffs will be installed to reduce cloud database spacing.

Termination of The Game can be done 2 different ways. User can click the "Exit Game" button when he/she is in the main menu. While user is playing Space Out, user can push the "Esc" button from the keyboard after that user can click the "Exit Game" option from the game.

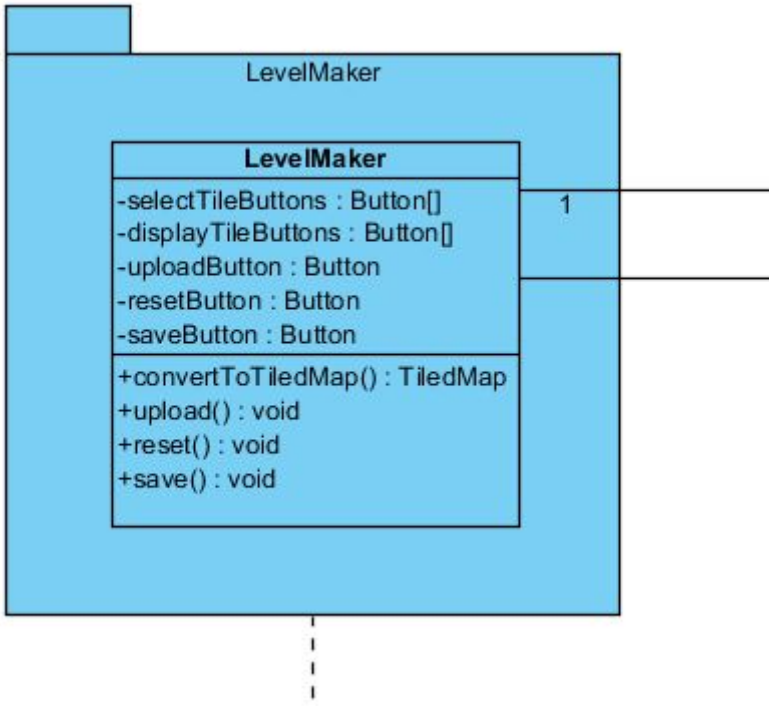
Error of The Game could occur many different ways when user play or try to open Space Out. For instance, sounds, images or Java could not be executable in that computer and user must install the required programs to run Space Out. And when user lost their internet connection his/her score will not be included in the leaderboard system and his/her game will be functionless.

3 - Subsystem Services and Low-Level Design

The following diagram shows all the classes divided into the subsystems:

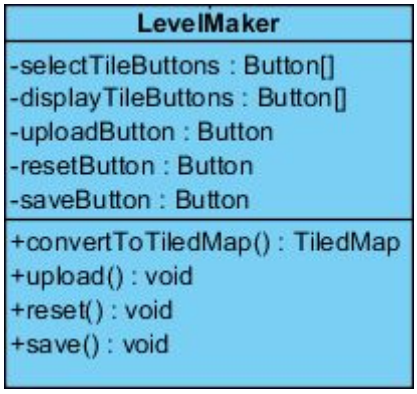


3.1 LevelMaker Subsystem



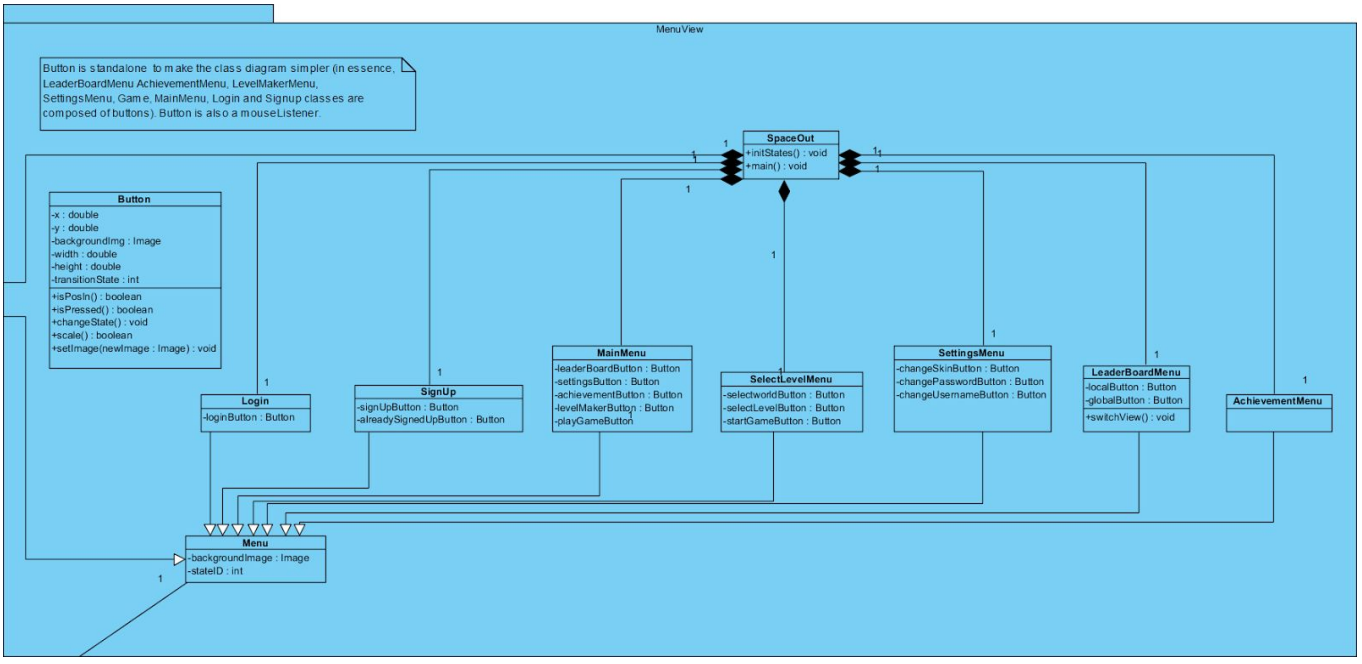
LevelMaker subsystem holds a single class: LevelMaker. This subsystem is responsible for containing the LevelMaker feature. This, althogh similar to other menu states in the application, is made a separate subsystem since its' behaviour, functionality and purpose do not resemble a simple game menu.

LevelMaker:



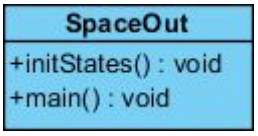
This class as a single unit represents the Level Maker functionality of our program. It is its' own state. It utilizes our own Button class to divide the screen into tiles that can be filled up with Tile images. selectTileButtons is an array of Buttons that holds the buttons for the selection pane and the displayTileButtons is an array of Buttons that holds the buttons for the canvas pane. The other three buttons are upload, reset and save buttons. Upon pressing the uploadButton, the created and saved level will be uploaded to the cloud datastore with the help of CloudQueryInterface class from the AccountHandler subsystem. convertToTiledMap() method converts the selected elements in the canvas to a .tmx file that is than usable by our application. It returns this .tmx file as a TiledMap object.

3.2 MenuView Subsystem



MenuView subsystem is responsible for holding all the menus together as states. It is able to switch between these states on demand. This decides what will be showed on the screen. The menus display the data provided by the AccountHandler subsystem.

SpaceOut:



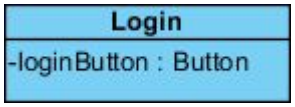
We are building a state based game and we think that all screens of our games are states, so all of them need to be classes. Then we also need a class that wraps all the state classes and initialize them, run the main method. This will be the SpaceOut class. It has all state classes (Login, SignUp, MainMenu, SelectLevelMenu, SettingsMenu, LeaderBoardMenu, AchievementMenu, LevelMakerMenu) in its constructor, initialize them on initStatesList method (that is a special function for Slick), and runs the whole game by its main method.

SignUp:

This class is for responsible for the signup menu that first appears when Spaceout is started. It will provide the user interface with text fields, and buttons to sign-up, or to navigate to login menu. Like all state classes it has a stateID property, and like all menus it has a background image. Since our sign-up system is cloud

based, it sends cloud queries using AccountManager class, to the CloudQueryInterface class that basically parses, and requests cloud queries.

Login:



This class is responsible for user login. As name suggests, it is very similar to SignUp class. It has user interface elements for e-mail/username and password. (Since classes like TextField are so trivial, they are not mentioned in class diagram.) This class also sends cloud queries via AccountManager and CloudQueryInterface classes, to check the login credentials are valid or not.

MainMenu:

MainMenu class is responsible for providing necessary operations for main menu of SpaceOut. It has buttons to navigate the user to gameplay, levelmaker, achievements menu, settings menu and leaderboard menu.

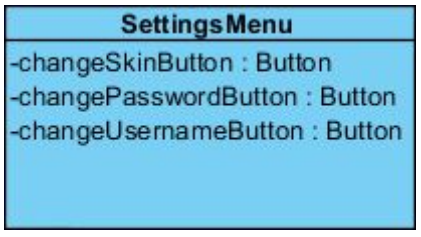
SelectLevelMenu:

This class provides a menu for user to select the world and level to play. It also holds a button to start the game. This state appears when user selects the “Play” option in the main menu. Since we provide a level maker, the levels are made by players will be accessible via cloud. That’s why SelectLevelMenu sends cloud queries in order to choose and play custom levels.

LeaderBoardMenu:

This menu provides global and local leaderboards. It has buttons to choose between them. Since global leaderboard is cloud based, it sends cloud queries and receives data via CloudQueryInterface. On the other hand, local leaderboard receives its data from AccountManager class.

SettingsMenu:



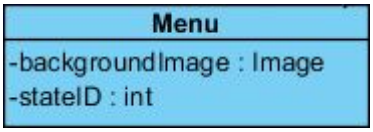
This menu manages various account settings. By the use of changeSkinButton the current skin of the user is changed. It modifies the selected skin data through AccountManager class' changeSkin() method. The other two buttons; changePasswordButton and changeUsernameButton are used to change

the password and username of the account, again, through AccountManager's methods.

AchievementMenu:

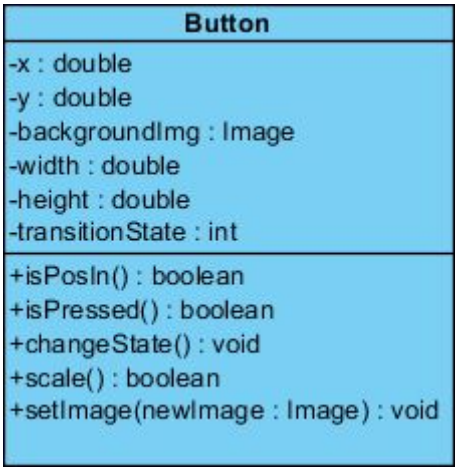
This class provides a menu that user sees the unlocked achievements. It takes the data from AccountManager class that has an attribute named achievementList –which is an array of Achievement class objects. More detail will be given about Achievement class.

Menu:



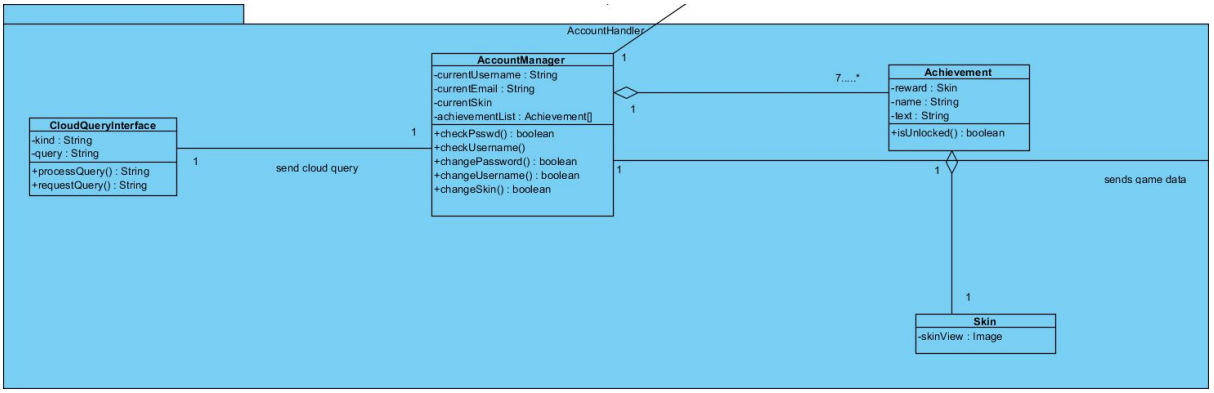
We have many different menus in our game, but all of them have some common properties. That's why we implement a parent class named "Menu", that has background image, state id and relationship with AccountManager class. Our other menu classes extend this class ("is a" relationship), so that they also have these attributes and relationships. We think that this is a nice way to implement object oriented design.

Button:



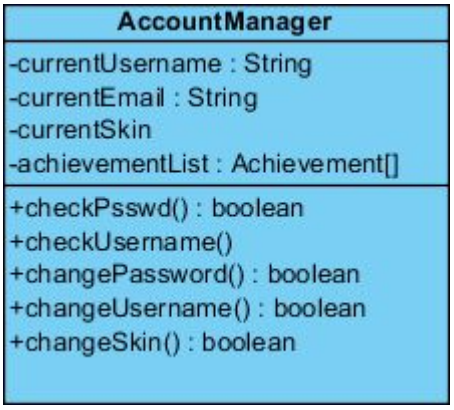
We implement our own button class, in order to do the state transitions easier and to provide a unique user interface. Button has background image, x-y coordinates, width, height attributes, and isPressed() method to detect button press. Also it has a method to change the game state that will be used after button presses. Instances of Button class are used in all of our menu classes.

3.3 AccountHandler Subsystem



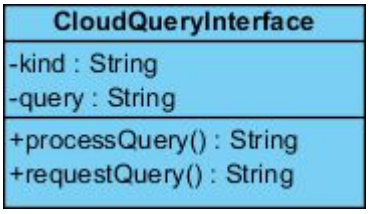
AccountHandler subsystem manages all the attributes tied to a single account. It heavily communicates with the cloud datastore in order to gather user data.

AccountManager:



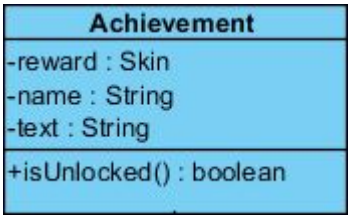
This class holds the necessary user information. Since menu classes are used to display and update user data, and we hold some of the data in AccountManager and some of them in the cloud, AccountManager, all menu classes and CloudQueryInterface have a strong relationship. AccountManager sends and receives cloud queries from CloudQueryInterface. Also it is related with GameController class, since GameController needs user data like chosen player skin, and needs to send game data to cloud for example in order to have a rank in global leaderboard. The class has attributes to hold user data like username, e-mail, achievement list and current player skin, and methods related to login and user preferences, like to check password, check username, change e-mail, change password and change player skin etc.

CloudQueryInterface:



We will send data to cloud and receive data from cloud. In order to do that, we need to build request queries and refine the received queries from cloud. Since we have to do this same processes many times, we decided that creating a cloud interface and handling this processes by its methods can help us, so that we don't need to write the code over and over again. That's why this class has String attributes for query and its kind, and methods to process or request them. It has a strong relationship with AccountManager and Menu, as mentioned in AccountManager class' explanation.

Achievement:



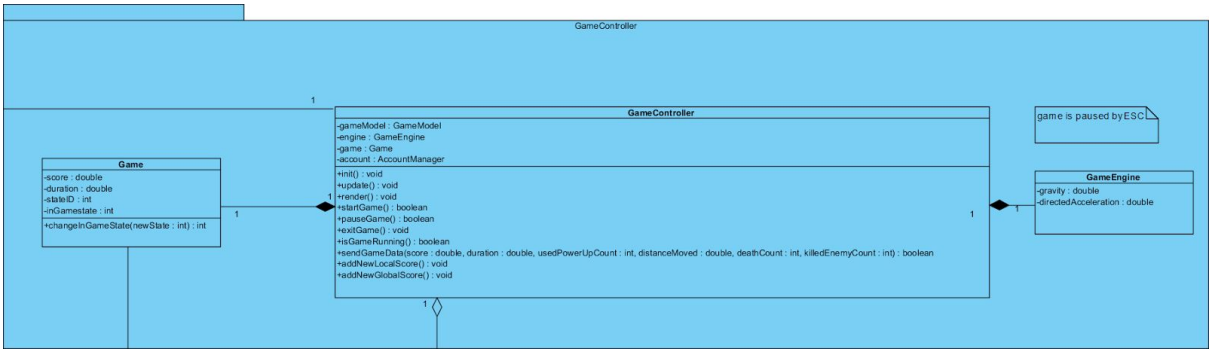
An achievement instance has a name, a text to describe the unlock criteria and a Skin object as reward. An array of achievements is held in AccountManager in order to regulate the unlockable character skins. Also, achievement objects have

isUnlocked() method and it uses data that comes from GameController class, to check if an achievement is unlocked during the last gameplay. This data is received using AccountManager class.

Skin:

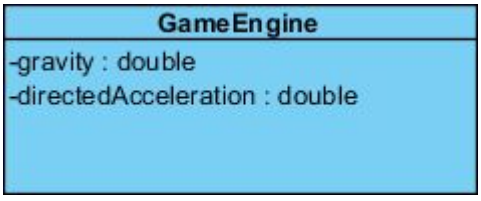
Character skins have an image that regulates the character view. Achievement objects and skin objects have one to one correspondence, i.e. each achievement has a skin. That's why a skin object has the relationships that are described in Achievement class, and AccountManager class has current skin attribute and change skin method.

3.4 GameController Subsystem



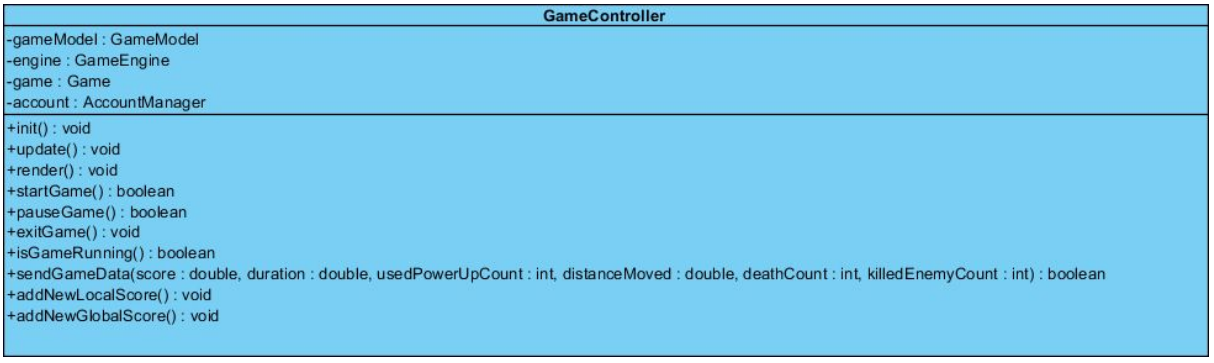
GameController subsystem is responsible for all the logic regarding the game. It manipultes game model based on the state of the game. It also communicates with the cloud in order to upload user/game data through the AccountHandler subsystem.

GameEngine



This class decides what happens the game objects based on their current state. It takes gravity and other acceleration into account while doing so. It calculates which gameObjects are currently colliding with each other to prevent things passing through each other.

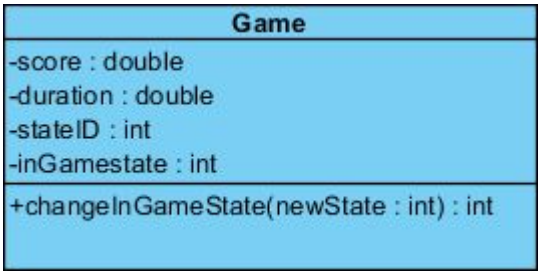
GameController



The game controller class is the "brain" of any game session in our application. With the update() method, the physical aspects of the game (game models) are animated/moved by using the gameEngine. It utlizes the Game class by getting several attributes bound to a level and passing them to the account (AccountManager object) by it's sendGameData method.

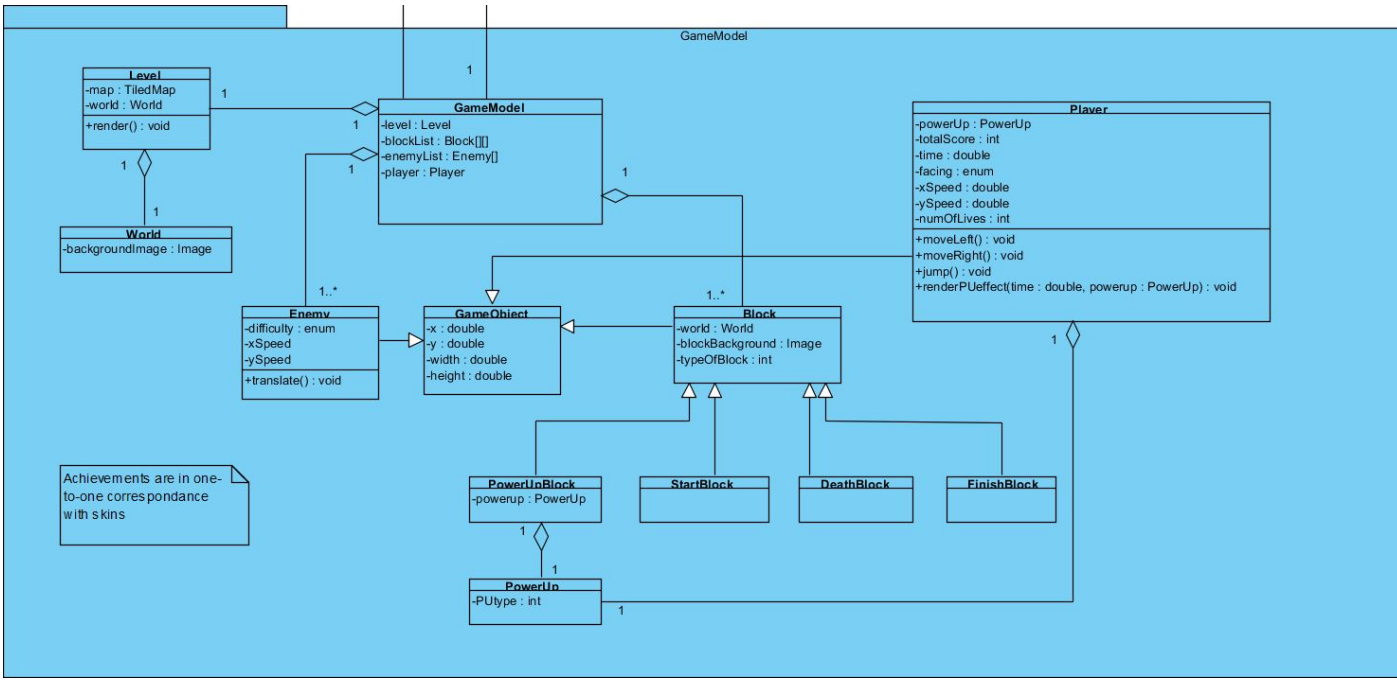
There are also some game state related methods that handle the state of which the game belongs to at any given moment. Start, pause and exit game methods are used to switch from these inner states.

Game



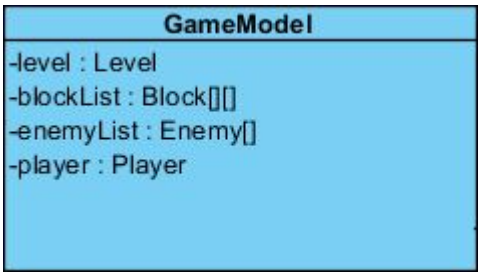
The game class holds essential data about the game such as the number of kills as "score", the time spent in the game so far within "duration" and the current in-game state within "inGameState". It is responsible for providing current data to the GameController. By its changeInGameState method it is able to alter the in-game state.

3.5 GameModel Subsystem



The GameModel subsystem is the only subsystem in the “Model” layer. It contains all the game objects and things that the game controller can manipulate.It also holds contains the Player class which is what the user controls.

GameModel:



GameModel class regulates the in-game interface. It has an object of Level class which holds the map and the world of the game and draws it. GameModel also holds the list of enemies and blocks that has an effect on regulating death of the player, powerups etc.

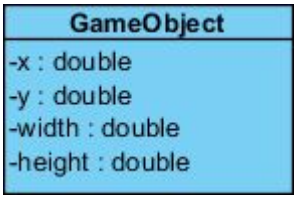
Level:

A Level object has the map of the game and the world which is Mars, Moon or Earth. Level class draw the level using these objects.

World:

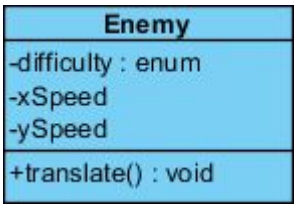
A World object has a background image that will be used to draw the game levels.

GameObject:



This is a parent class that has the common properties of game objects like enemy, block and the player. Common properties include x-y coordinates, width and height. These are implemented as attributes of GameObject.

Enemy:



This class is one of the children of GameObject class. Obviously, it is used to implement the enemy characters in the game. Each enemy has a difficulty level, speed and a method to regulate its movement, along with the attributes of its parent class. It is also one of the classes that form the GameModel.

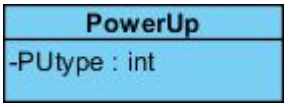
Block:

Block is another child of GameObject class. Besides GameObject's attributes, it has world and blockBackground attributes to determine the view of a block, and an integer attribute to indicate the type of the block. It is another class that forms the GameModel, and it has different types as child classes.

PowerUpBlock:

It is a block type that gains player a power up when contact is occurred. Besides attributes of its parent classes, it has a powerup type attribute, which is an instance of Powerup class and used to determine the type and effect of powerup.

PowerUp:



PowerUp class has an integer attribute to decide the type of powerup. This can mean extra player speed, extra live etc. It is used in PowerUpBlock and Player classes. Player class renders the effect of the powerup using its type.

StartBlock:

The block that indicates the start of the level. It is child of Block class, and grandchild of GameObject class.

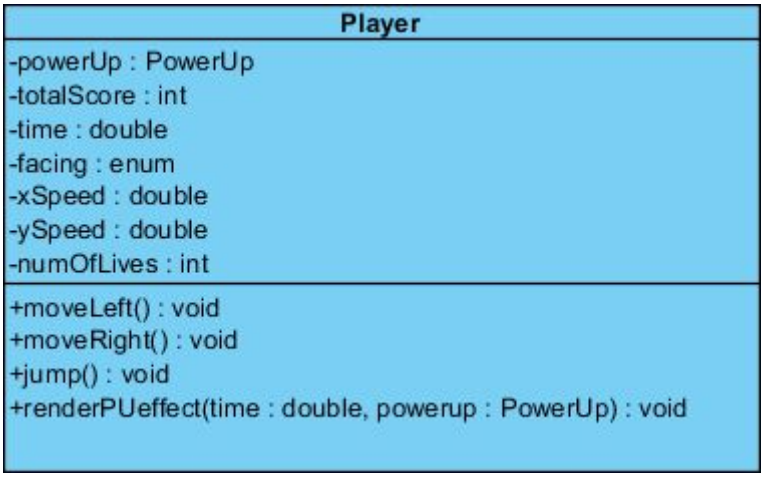
DeathBlock:

The block that causes to death of the player in case of a contact.

FinishBlock:

The block that indicates the end of the level.

Player:



As name indicates, this is what user controls during the gameplay. It is the character of the game. It is also a child of GameObject class, so it has its attributes too. It has a Powerup object, so that it will modify the attributes like speed, number of lives and total score. To do this, Player class has a method named renderPUeffect. Player has attributes about total score and total game duration, since they can unlock achievements. It also has methods to regulate the character movement (move right, move left, jump), and attributes to regulate the character view with respect to these movements.

4 Improvement Summary

With this iteration of the design report a lot of changes has been made and a lot of new sections has been added.

In section 1, a “Trade-off” section has been added which indicates several trade offs that were/will be kept in mind for the implementation.

Regarding section 2 and 3, a whole revamp of the subsystems and class were made. We switched from a 4-tier architecture to a 3-tier architecture since we felt it fit better to our implementation. A inter-layer interaction section was also added to section 2 to further clarify how these new layers will interact with each other. We have also expanded the subsystem and class descriptions on the 3rd section of the report.

Overall, the design of the application has changed in a way that makes more sense and covers a lot more ground and features compared to before.

5 - Glossary and References

[1] https://www.tutorialspoint.com/mvc_framework/mvc_framework_introduction.htm

[2] https://en.wikipedia.org/wiki/Java_Development_Kit

[3] https://en.wikipedia.org/wiki/Java_virtual_machine

[4]https://en.wikipedia.org/wiki/Java_virtual_machine#Java_Runtime_Environment_from_Oracle

[5] https://en.wikipedia.org/wiki/Virtual_private_cloud

[6] https://cloud.google.com/sdk/gcloud/

[7] https://en.wikipedia.org/wiki/Hard_disk_drive