Bilkent University

Department of Computer Engineering

Senior Design Project

Project short-name: Planetarium

Low Level Design Report

Boran Yıldırım, Kıvanç Gümüş, Ümitcan Hasbioğlu, Hüseyin Eren Çalık, Yiğit Bedişkan

Supervisor: Uğur Doğrusöz
Jury Members: Fazlı Can and Uğur Güdükbay

Definitions, Acronyms and Abbreviations

TCP: Transmission Control Protocol
UDP: User Datagram Protocol
UML: Unified Modelling Language

1. Introduction

In this report, the low-level architecture of Planetarium is proposed. The trade-offs of our design and engineering standards will be explained. What follows will be the in documentation guidelines. After that, information about the packages and interfaces of Planetarium's systems will be presented. Finally, the class diagrams and a detailed look into each of Planetarium's software components will conclude the report.

a. Object design trade-offs

In software development, when choosing to enhance a certain feature of a program, usually another one has to be sacrificed. Almost all decisions with respect to design, comes with certain trade-offs and implications. For this reason, we spent a lot of time identifying Planetarium's trade-offs during the design process to create the most optimized system according to the project's needs. In the following sections, trade-offs we considered will be presented.

a. Reliability vs Efficiency

Since efficiency is important in online games which need rapid updates and changes in game screen, it is prefered to give importance to efficiency instead of reliability. Because, if the user is not provided with a proper game view, the reliability constraint also won't be satisfied. A UDP server is used instead of TCP a server in order to make the client and server connection as fast as possible. Although, TCP is a more reliable protocol, it is preferred to use reliable UDP to make the game efficient.

b. Portability vs Performance

The game aims to be reachable for a larger group of people so portability is an important issue to make the game available in all platforms. Because of this reason we use Unity game engine which is cross-platform application. We prefer to give importance to portability instead of performance. The game could have been implemented for a single platform however we decide to use as many platforms as possible for wider usage. Portability is preferred instead of performance to make the game reachable from any platform.

c. Functionality vs User-Friendliness

The game is about civilizations interacting in cosmos so it is mimicking a far future real life scenario. The game's relevant to the real life and it is being science-based; provides developers the ability to extend the features of the game as much as we want. Yet, the game has to provide certain level of entertainment in which the user won't get lost so that they can enjoy the application rather than struggling to work themselves through overfilled features. To deliver this, the primary game features(e.g. gathering resources, market transactions etc.) are designed to be complex as well as being entertaining and easily playable.

b. Interface documentation guidelines

In this report, all the class names are named in the standard 'ClassName' format, where all of these names are singular. The variable and method names are camel case and follow a similar rule as in 'variableName' and 'methodName()'. In the class description hierarchy, the class name comes first, seconded by the attributes of the class, and finally concluded with the methods. The detailed outline looks similar to the one presented below:

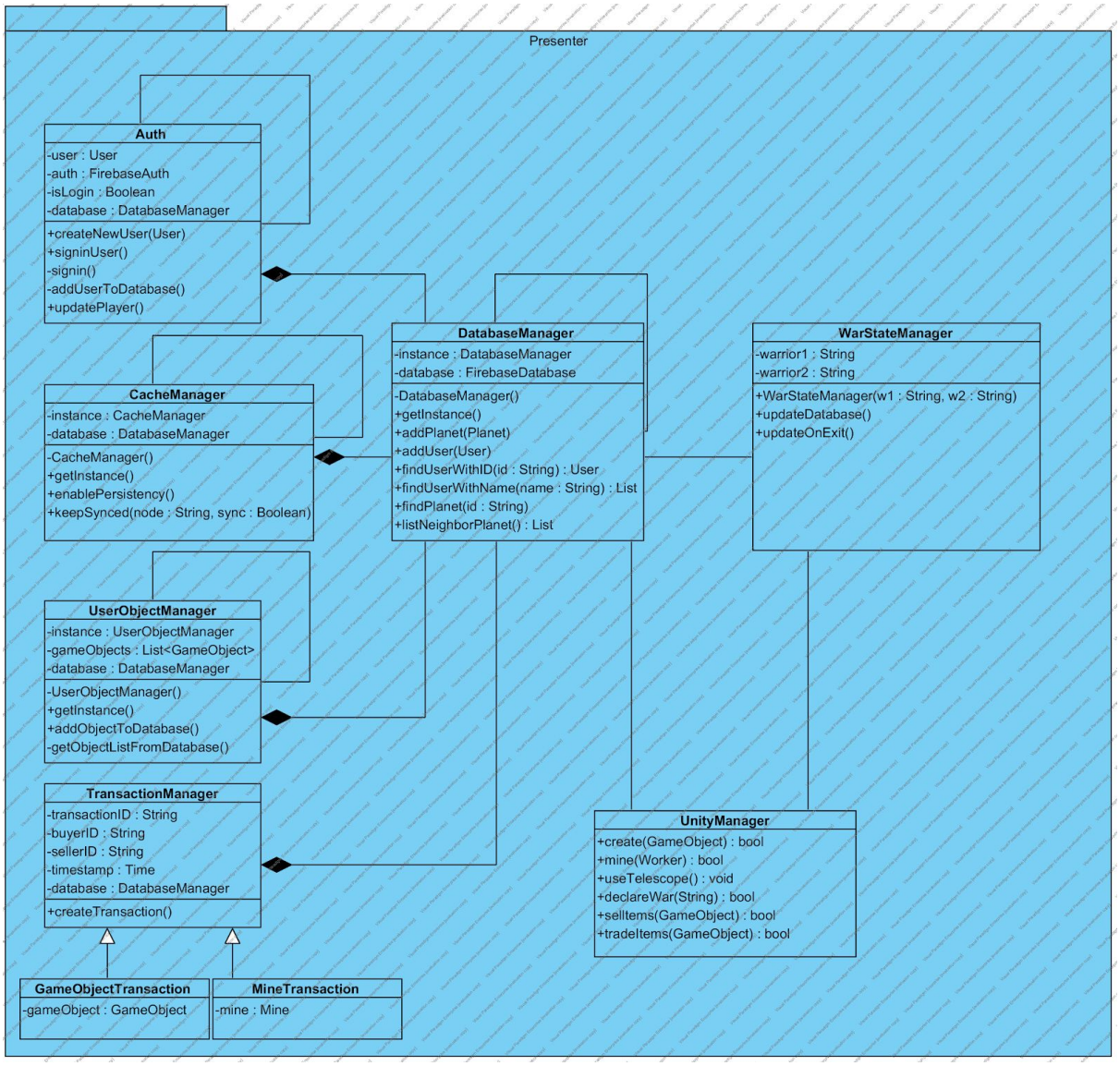
Class Name
Description of Class
Attributes
Type of Attribute : Name of Attribute
Methods
NameOfMethod(ParametersOfMethod): | Description of Method

c. Engineering standards

For the descriptions of the class interfaces, diagrams, scenarios, use cases, subsystem compositions and hardware depictions, this report follows the UML guidelines.

2. Packages

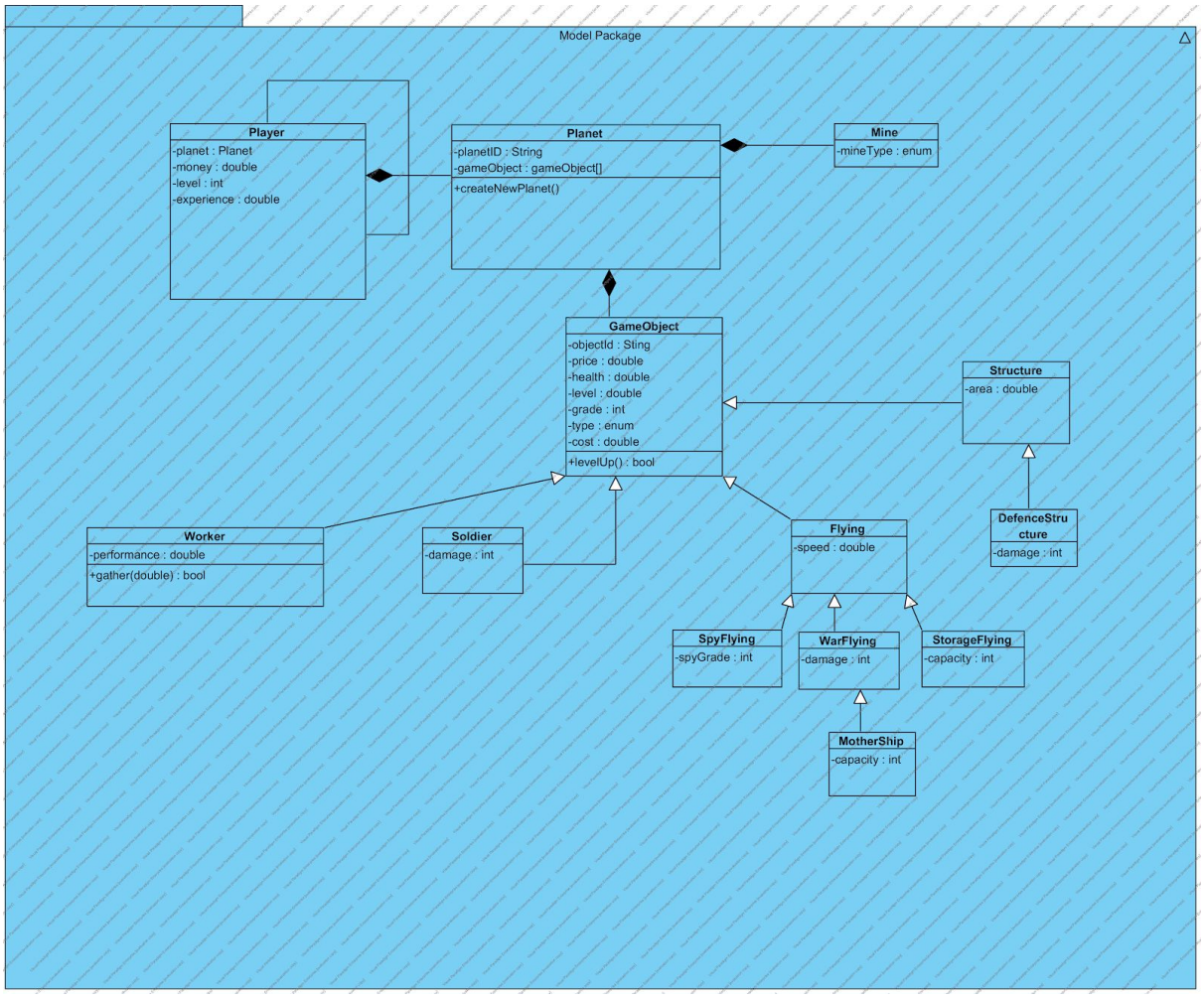
a. Presenter Package



Auth: is responsible for user related tasks in database. Login/Signup and profile updates are done from this class.
CacheManager: manages caching of database values in local. We as a developer will choose which values will be cached.
UserObjectManager: stores user's game objects in database and load them from database.
TransactionManager: works as a bank, transfers money between users and send new items to them.
DatabaseManager: is responsible for database access and querying.
WarStateManager: is responsible for managing the multiplayer game system with the Photon Unity server.

UnityGameEngine: responsible for rendering graphics and initialize objects visually.

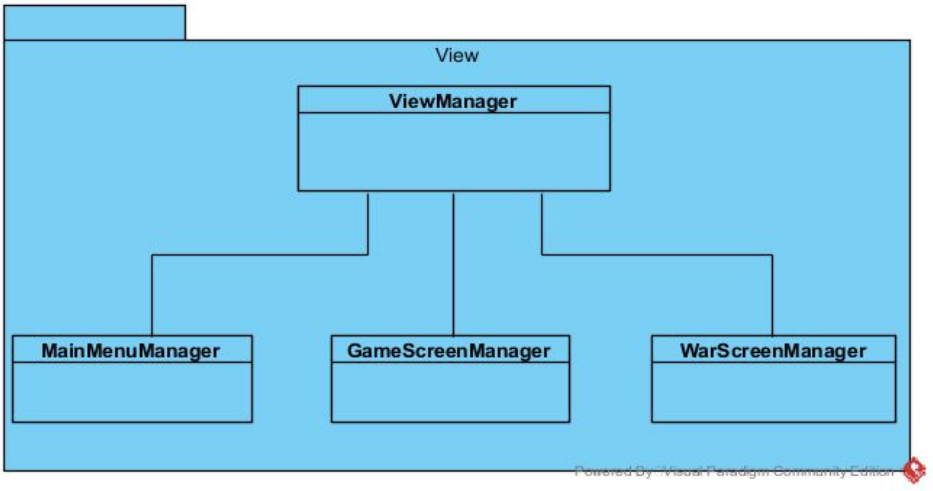
b. Model Package



Player: is responsible for modelling player object that will be used by Unity throughout the game.
Planet: is responsible for modelling planet object that will be controlled and modified by the player throughout the game.
Mine: holds the type of the mines that will be a resource to be gathered by the player
GameObject: parent of major game objects that will be generated and destroyed throughout the gameplay by the player and also the game engine.
Worker: holds the values for the worker object that will be controlled by the player to gather resources from his/her planet.
Soldier: holds the damage data for the soldier object that will be used by the player for battles.
Flying: parent of flying objects
SpyFlying: holds the data for the spyflying object which will be used by the player to spy other planets.

WarFlying: holds the data for the warflying object which will be used by the player to attack other planets.
StorageFlying: holds the data for the storage flying object which will be controlled by the user to transport mines from a planet that is defeated to player's own planet.
MotherShip: holds the data of the mothership of the player.
Structure: holds the data for structure objects which are the buildings and defence structures.
DefenceStructure: child of structure which additionally includes damage data.

c. View Package



ViewManager: parent of the main menu, game screen and war screen managers.
MainMenuManager: sets up the main menu view.
GameScreenManager: sets up the game(planet) view where the user gameplay is.
WarScreenManager: sets up the battle(war) scene where two different users battle with each other.

3. Class Interfaces

a. Presenter

Auth
Manages user registration, login and information update operations.
Attributes
instance: Auth user : User auth: FirebaseAuth isLogin: boolean database: DatabaseManager
Methods
Auth()	private constructor for singleton getInstance()	returns the instance of Auth createNewUser(User):	creates the new user in database signinUser()	signin for already created account signin()	called from createNewUser() and signinUser() to signin user addUserToDatabase()	adds the new user to database after it is created updatePlayer()	updates phonenumber and username

DatabaseManager
Manages database operations of the game.
Attributes
instance:DatabaseManager database:FirebaseDatabase
Methods
CacheManager()	private constructor for singleton getInstance()	returns the instance of DatabaseManager addPlanet(Planet)	adds the planet datas to database addUser(User)	User object findUserWithName(name:String) User	returns User object with given id findUserWithID(id:String) User	returns User object with given id findPlanet(id : String)	return Planet object with given id listNeighborPlanet() : List

WarStateManager
Manages databese operations during the war state of the game.
Attributes
warrior1:String | User1 warrior2:String |User2
Methods
WarStateManager(w1:String, w2:String) | private constructor UpdateDatabase() UpdateOnExit()

UserObjectManager
Manages database operations for User object.
Attributes
instance:UserObjectManager database:DatabaseManager gameObject:List<GameObject>
Methods
UserObjectManager() | private constructor for singleton getInstance() | returns the instance of UserObjectManager addObjectToDatabase() |add singleton object to database getObjectListFromDatabase()| returns singleton object from database

TransactionManager
Manages database operations for users transactions.
Attributes
transactionID: String buyerID:String sellerID:String timestamp:Time database:DatabaseManager
Methods
createTransaction() | creates a pipe for transaction between two users

GameObjectTransaction
Manages the game object trades between players.
Attributes
gameObject: GameObject

MineTransaction
Manages the mine trades between players.
Attributes
mine: Mine

UnityManager
Manages the Unity Game Engine
Attributes
gameObject: GameObject
Methods
create(GameObject) bool | create game object for player and returns boolean. mine(Worker) bool | mine with given worker object. useTelescope() | decleareWar(String) | starts war against given user id. sellitems(GameObject) bool | sells given gameObject. tradeItems(GameObject) bool | trade given gameObject.

b. Model

Player
Player model
Attributes
planet:Planet money:double level:int experience:double

Planet
Planet model
Attributes
planetID : String gameObject: gameObject[]
Methods
createNewPlanet() | creates new planet

Mine
Mine model
Attributes
mineType : enum

GameObject
Game object model
Attributes
objectID : String price : double health :double level : double type : enum cost : double
Methods
levelUp() bool | makes the given objects level up.

Worker
Worker model
Attributes
performance : double
Methods
gather(double) : bool

Soldier
Soldier model
Attributes
damage : double

Structure
Structure model
Attributes
area : double

DefenceStructure
Defence structures model
Attributes
damage : int

Flying
Flying model
Attributes
speed : double

SpyFlying
Spy spacecraft model
Attributes
spyGrade : int

WarFlying
War flying model
Attributes
damage : int

StorageFlying
Storage flying model
Attributes
capacity : int

MotherShip
Mother ship model
Attributes
capacity : int

c. View

ViewManager
Manages the main menu, game screen and war screen views

MainMenuManager
Sets up the main menu view

GameScreenManager
Sets up the game(planet) view where the user gameplay happens

WarScreenManager
Sets up the battle(war) screen where two different users battle with each other