Bilkent University

Department of Computer Engineering

Senior Design Project

Project short-name: Neophyte

Low Level Design Report

Ali Soyaslan, Oğuz Liv, Umut Akös, Gülce Karaçal

Supervisor: Halil Altay Güvenir

Jury Members: Uğur Güdükbay and Hamdi Dibeklioğlu

Progress Report

October 8, 2018

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Senior Design Project course CS491/2.

Contents

1 Introduction _______ *3*

1.1 Design Trade-Offs _______ 4

1.1.1 Functionality vs. Usability _______ 4

1.1.2 Security vs. Cost _______ 4

1.1.3 Performance vs. Space _______ 5

1.2 Engineering Standards _______ 5

1.3 Interface Documentation Guidelines _______ 6

1.4 Definitions, Acronyms, and Abbreviations _______ 6

2 Packages _______ *7*

2.1 Client _______ 7

2.1.1 GameGraphics _______ 7

2.1.2 ScreenController _______ 8

2.2 Server _______ 9

2.2.1 Logic _______ 9

2.2.2 Data _______ 10

3 Class Interfaces _______ *11*

3.1 Client _______ 11

3.1.1 GameGraphics _______ 11

3.1.2 ScreenController _______ 16

3.2 Server _______ 20

3.2.1 Logic _______ 20

3.2.2 Data _______ 23

4 References _______ *25*

1.0 Introduction

In a rapidly digitizing world, having technical skills is very crucial since, nowadays; almost everything requires some form of programming. As technology has been developing, we have become more dependent on it and use various technologies to accomplish specific tasks in our daily lives. Technology is being implemented in almost every section of our lives and business structures. This is the reason why, many countries such as England, Singapore, Estonia and US have started programming education in early ages, because the sooner a person learns how to create programs, the stronger their problem solving abilities get. This education also amplifies their computational and analytical thinking skills. For instance, UK made the most ambitious attempt to get kids coding, with changes to the national curriculum in 2013. ICT – Information and Communications Technology – is out and replaced by a new "computing" curriculum including coding lessons for children as young as five [1]. Such knowledge is important not only to individual students' future career prospects, but also for their countries' economic competitiveness and the technology industry's ability to find qualified workers [2].

However, it appears that Turkey is a little belated to educate children about programming compared to other countries. According to International Computer and Information Literacy Study (ICILS), who conducted among students between 6-15 years from all over the world in 2013, it has been acknowledged that only 1% of students from Turkey have advanced computer knowledge. On the other hand, 35% of students from Korea, 34% of students from Australia and 33% of students from Poland have advanced knowledge about computers and programming [3]. In order to offer an effective and simple solution for this problem, the project Neophyte will be proposed. With Neophyte, we aim to teach children how to code while making them entertained by playing different kinds of games they like. Neophyte creates a platform where children can interact with each other in an exciting way and improve their programming skills.

In this report, we aim to provide an overview of the low-level architecture and design of the system we will develop. Firstly, the design trade-off, engineering standards and documentation guidelines are described. Afterwards, the packages in our system and their functionalities are

described along with detailed class diagram views. Furthermore, interfaces of all classes in all packages are included.

1.1 Design Trade-Offs

1.1.1 Functionality vs. Usability

Functionality and usability are two main points that we focus on the design process of Neophyte. Due to the fact that our application will be used by different people from young ages, making it usable is crucial for us. The user interface should be user-friendly in this way; users can spend their time, enjoying programming rather than struggling to figure out how to play the game while writing code segments. On the other hand, Neophyte offers many functionalities such as writing codes, making contacts with other users, participating in team competitions etc. For this reason, functionality is also significant for our project. The main design goal of our system is providing the maximum possible functionality with an easy-to-use interface to users as in most of the promising software today. Thus, we will try to obtain a balance between functionality and usability.

1.1.2 Security vs. Cost

We will ask users to sign up and login to the program in order to use the application. In other words, Neophyte is a platform in which users should first log in with their credentials. Users will be allowed to change their personal information by modifying personal settings. This is the reason why, security is one of the key concerns of Neophyte. All users register to the system with a username/email and a password. The user information will only be shown to the user only. These data will be secured by using third party security system. While we are trying to achieve this, we have to consider the cost of these operations as well. Thus, we will try to get high security with a low cost as much as possible.

1.1.3 Performance vs. Space

The large scale of data also introduces a lot of possible delay which can be as a result of the requirement of saving the data to the servers. The scores should be constantly stored as well as details about personal information. Therefore, this data should be safely stored in the servers continuously and should be updated correctly. This naturally increases the processing time and slows down the system. In order to ensure fast response time, we will be saving all the data in the server time. In this way, we will ensure that the system is highly responsive to users anytime. All connections and data exchanges with the server will be handled in background threads. This way, we will be able to keep the system fast while managing lots of data at the same time.

Another aspect to this topic is that as the game will be accessed through the browser, the memory usage is a great deal for us. Many browsers these days do not support more than 2 GB of memory usage except Google Chrome. Anyhow, we still should not use that much of memory as a single application. As a result, we should keep the per-tab memory usage at 800 MB - 1200MB.

1.2 Engineering Standards

In this report, UML design principles are used in the description of class interfaces, diagrams, scenarios and use cases, subsystem compositions, and hardware-software components depictions. UML is a commonly used standard that allows simpler description of the components of a software project. The reports follow the IEEE citation guidelines for the references since they are easy to understand and very commonly used.

1.3 Interface Documentation Guidelines

This report follows the convention where all class names are singular and named with the standard ‘ClassName’ form. The variable and method names follow the same convention ‘variableName’ and ‘methodName()’. The class descriptions follow the order where the class name comes first, the attributes follow, and lastly the methods are listed. After the class names, a brief description and function of the class can be found. The detailed outline is provided below:

Class Name

➢ Description of class

Attributes

➢ Attribute name

➢ Type of attribute

Methods

➢ Method name

➢ Parameters

➢ Return value

1.4 Definitions, Acronyms, and Abbreviations

Server: The part of the system responsible from logical operations, scheduling, and data management

UI: User Interface

Client: The part of the system which users interact with

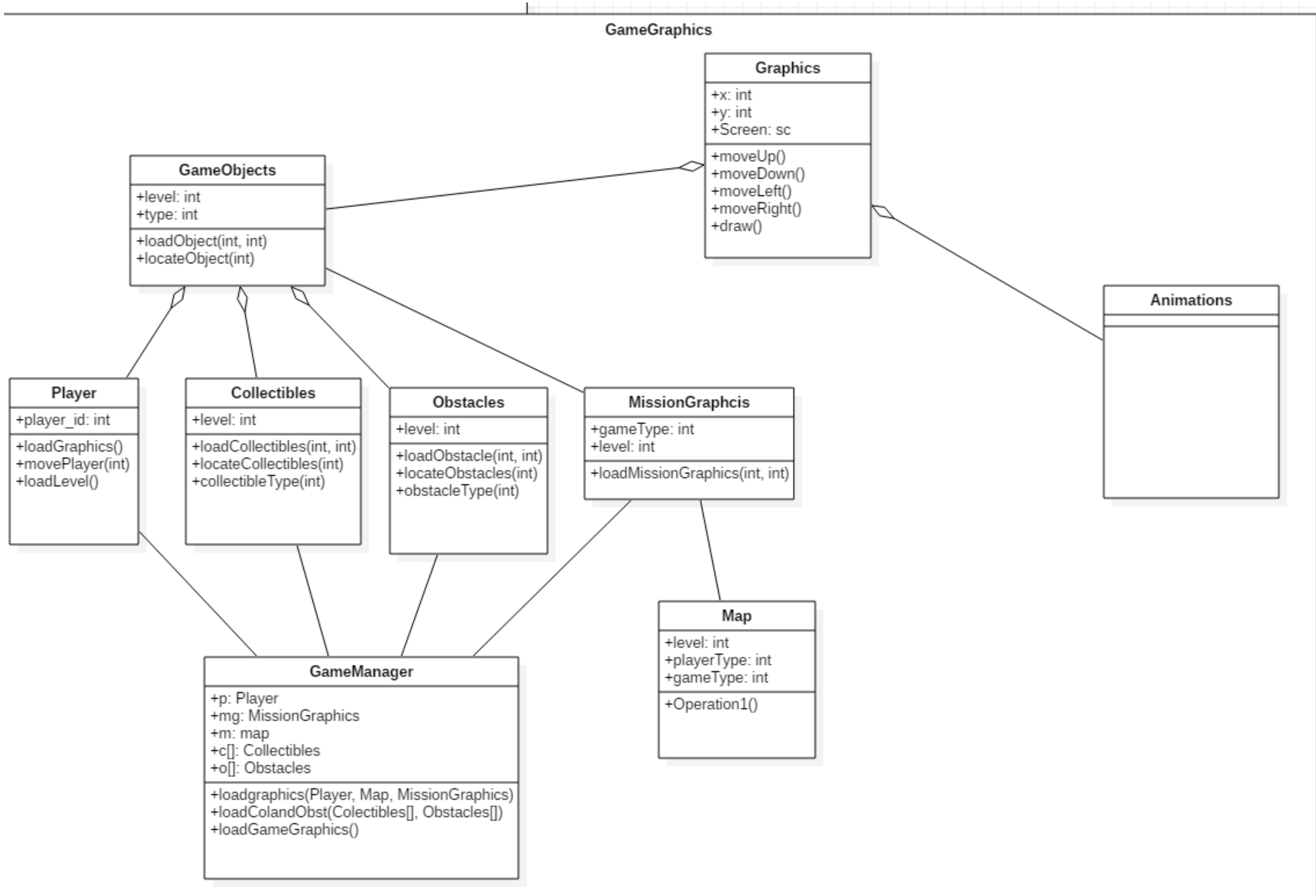
2.0 Packages

Neophyte follows a Client-Server architectural style in order to effectively respond/process concurrent user requests. The web application will constitute the client part of the system. The client requests services from the server to function and to respond the needs of the users.

The system is composed of two main packages as client and server. In the client package, there are two subpackages called GameGraphics and ScreenController. In the server package, there are two subpackages called Logic and Data.

2.1 Client

2.1.1 GameGraphics



GameGraphics package contains the classes related to Graphical User Interface.

Graphics: This class generates the visuals of the game, characters, map, static and dynamic items.

GameObjects: This class is responsible for loading and locating game objects.

Player: This class is responsible for providing dynamic information of players.

Obstacles: This class represents graphical obstacles objects that players will encounter when they reach advanced levels of games.

Collectibles: This class represents graphical collectibles objects that players will encounter during the game.

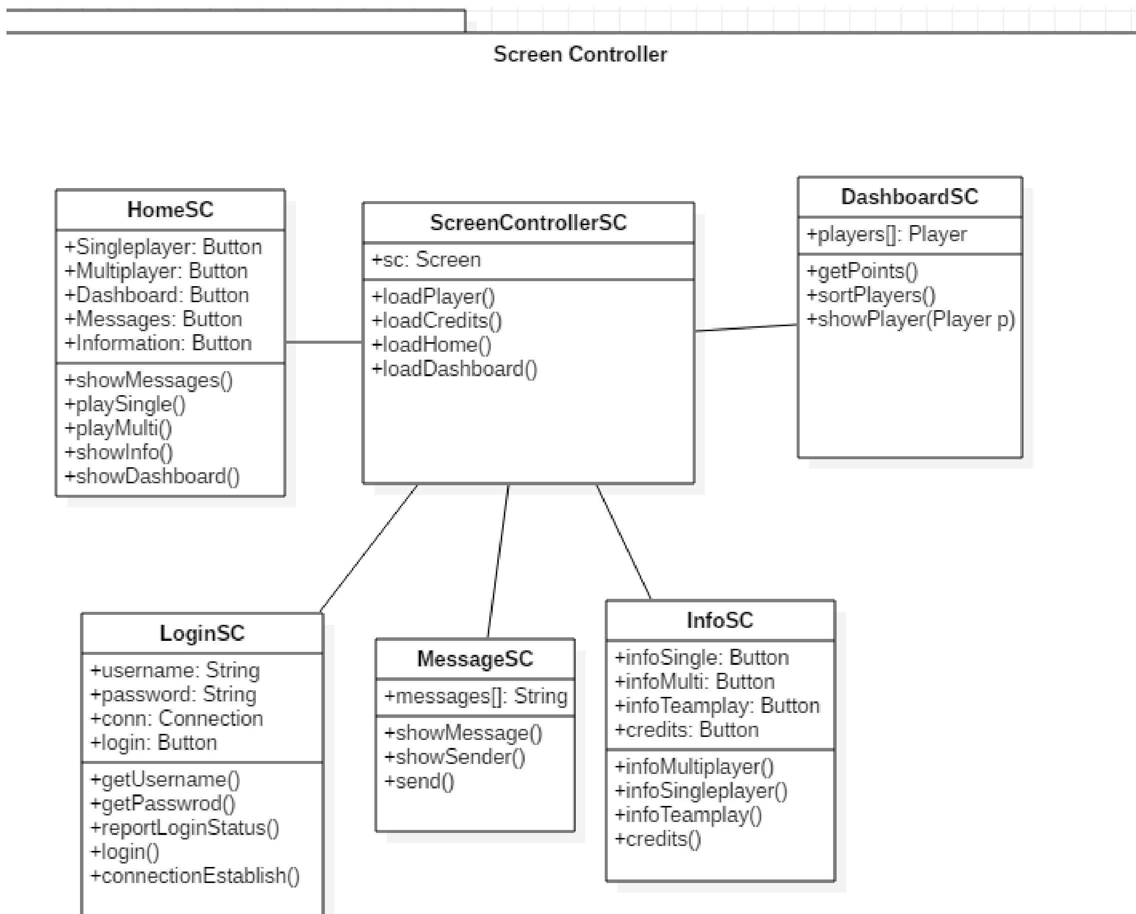
Map: This class represents the map items with graphical materials which will be generated by the game manager as the player progresses through the game.

MissionGraphics: This class loads the suitable graphics for specified level and game type.

GameManager: This class loads appropriate graphics according to player, players' levels and so on. Mainly, this class stays between game logic and game graphics. It connects suitable parts of both classes in order to make them logical and user friendly.

Animations: This class is responsible for game animation designs.

2.1.2 ScreenController

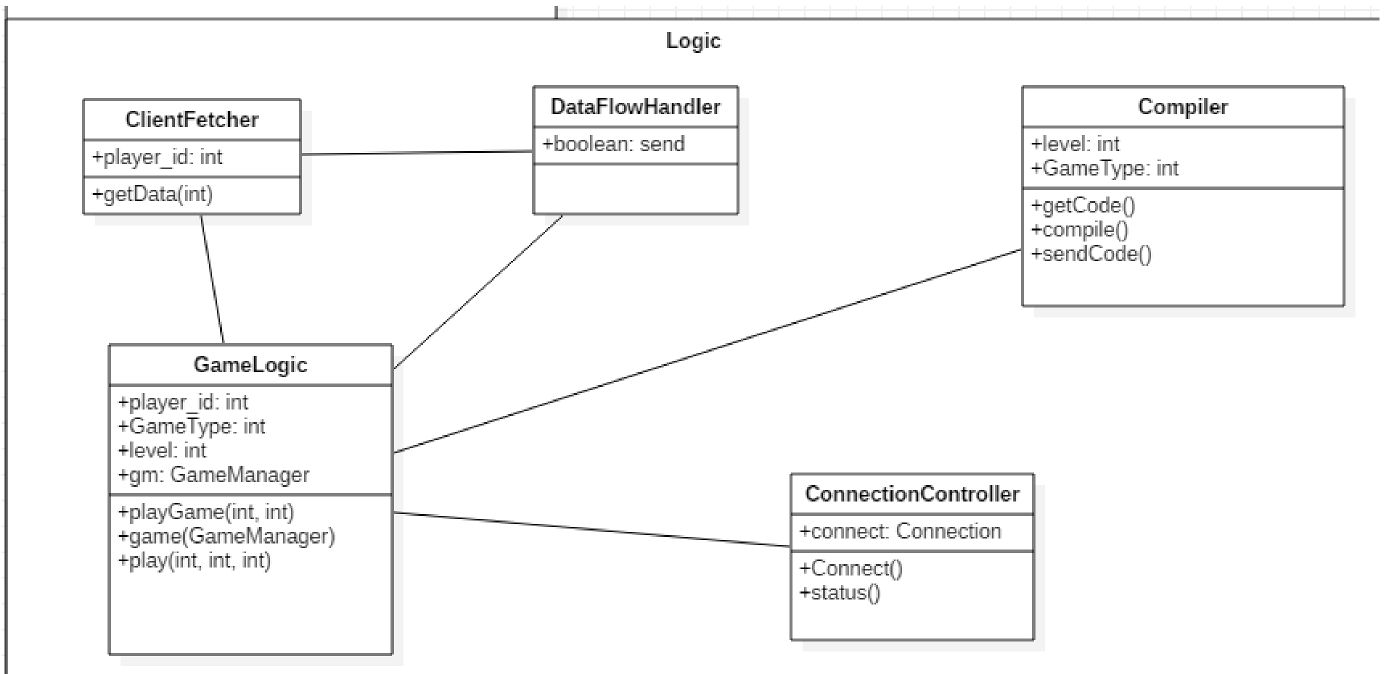


The ScreenController package handles the client functions and requests sent via the GameGraphics package.

LoginSc: This class manages the login operation of the user. At the initial startup, it is the class that user will encounter its rendered instance. It asks for basic authentication information.
HomeSc: This class manages the home screen of the program.
InfoSc: This class represents the information about a single player, multiplayer or teams.
MessageSc: This class manages direct messages sent and received among users.
ScreenControllerSc: This class manages the user interface for the main screen. All of the final outputs related to graphics are shown with this class. This class also acts as a bus to reach other classes within ScreenController.
DashboardSc: This class manages the screen that shows scores of users and their rankings among them.

2.2 Server

2.2.1 Logic



The logic package is responsible for processing requests from given clients and sending the appropriate responses.

Compiler: This class provides services to compile users' codes and reports appropriate results according to game logic.

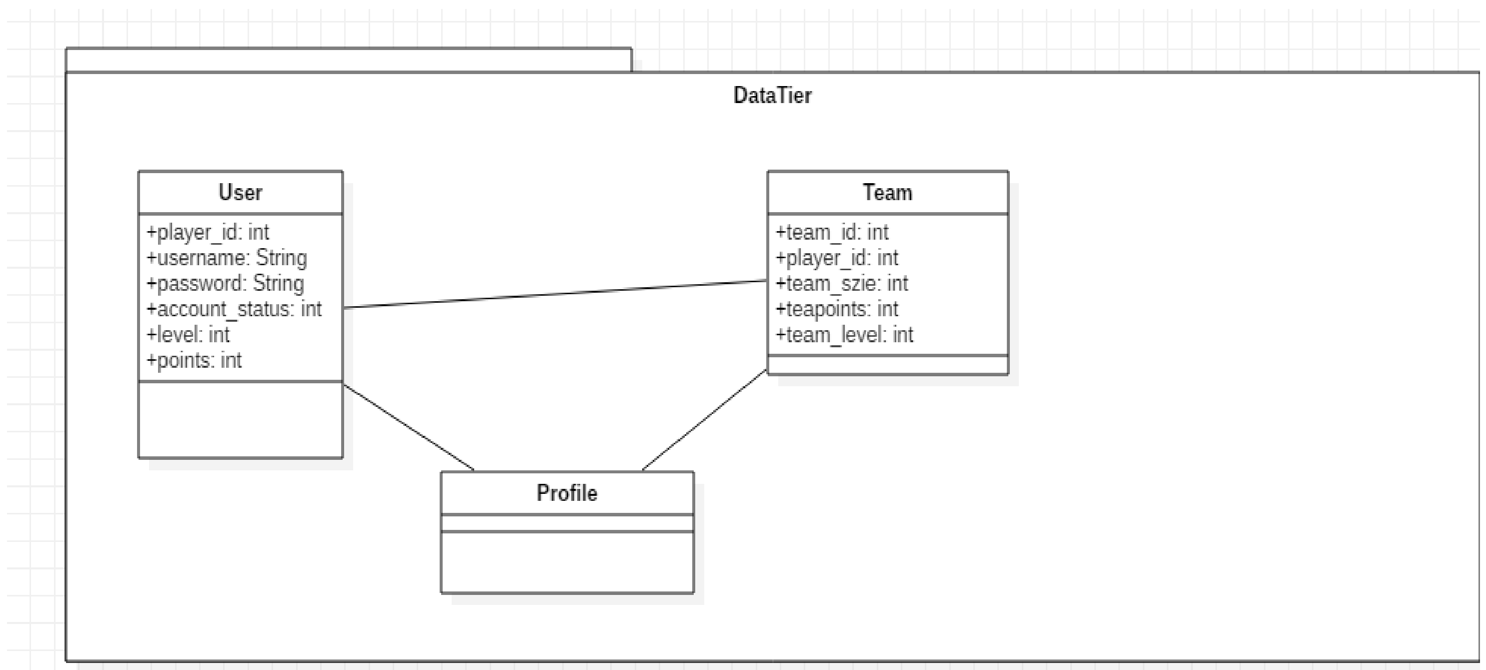
GameLogic: This class provides game logic for levels, players, as well as game graphics.

ClientFetcher: This class is for validating and for verifying the users for the game.

ConnectionController: This class represents connection logic for the game and users.

DataFlowHandler: This class represents services for messaging between users and also handles team activities such as creating a new team.

2.2.2 Data



Data package is responsible for data-related operations. Database management system (DBMS) resides in data tier and keeps the data of our application in server-side.

User: This class provides static player information such as player ID, username, level, points etc.

Profile: This class represents profile information of users.

Team: This class represents static team information such as team ID, team size, player ID etc.

3.0 Class Interfaces

3.1 Client

3.1.1 GameGraphics

Class Graphics
This class generates the visuals of the game, characters, map, static and dynamic items.
Attributes
int x
int y
Screen Sc
Methods
void moveUp(): This method calculates movements according to x,y variables and moves up.
void moveDown():This method calculates movements according to x,y variables and moves down.
void moveLeft():This method calculates movements according to x,y variables and moves left.
void moveRight(): This method calculates movements according to x,y variables and moves right.
void draw(): This method draws specific objects on screen canvas according to x,y variables.

Class GameObjects
This class is responsible for loading and locating game objects.
Attributes
int level
int type
Methods
void loadObject(int,int): This method is a helper function which is going to be overridden by its children classes. Basically, this method provides services for loading graphics of a specific object in specified game level and specified game object such as collectibles or players.
void locateObject():This method provides services for where to load related graphics of the object in the game map. Since different game objects can be on different places on the map, this is just a helper method for further objects.

Class Player
This class is responsible for providing dynamic information of players.
Attributes
int player_id
Methods

void loadGraphics(): This method loads game graphics of the player.
void movePlayer(): This method moves the player according to the user input.
void loadLevel(): This method loads the level of player.

Class Obstacles
This class represents graphical obstacles objects that players will encounter when they reach advanced levels of games.
Attributes
int level
Methods
void loadObstacles(int,int): This method loads obstacles according to the player's coordinates.
void locateObstacles(int): This method locates obstacles according to the level of the player.
int obstacleType(int): This method represents the kind of obstacles that the player will encounter.

Class Collectibles
This class represents graphical collectibles objects that players will encounter during the game.
Attributes
int level
Methods
void loadCollectibles(int,int): This method loads collectibles according to the player's coordinates.
void locateCollectibles(int): This method locates collectibles according to the level of the player.
int collectibleType(int): This method represents the kind of collectibles that the player will encounter.

Class Map
This class represents the map items with graphical materials which will be generated by the game manager as the player progresses through the game.
Attributes
int level
int playerType

int gameType
Methods
void loadMap(int,int,int): This method loads the game map according to the level, to the player type and to the game type.

Class MissionGraphics
This class loads the suitable graphics for specified level and game type.
Attributes
int gameType
int level
Methods
void loadMissionGraphics(int,int): This method loads graphics according to specified mission and to player level.

Class GameManager
This class loads appropriate graphics according to player, players' levels and so on. Mainly, this class stays between game logic and game graphics. It connects suitable parts of both classes in order to make them logical and user friendly.

Attributes
Player p
MissionGraphics mg
map m
Collectibles c[]
Obstacles[] o[]
Methods
loadGraphics(Player, Map,MissionGraphics) : This methods draws and loads layout graphics for specified mission and specified player.
loadColandObst(Collectibles[], Obstacles[]): This method draws and loads graphics for game add-ons such as collectible power ups and annoying obstacles.
loadGameGraphics(): This method loads the graphics of whole game components with the help of loadGraphics() and loadColandObst() methods.

3.1.2 ScreenController

Class LoginSc
This class manages the login operation of the user. At the initial startup, it is the class that user will encounter its rendered instance. It asks for basic authentication information.
Attributes
string username

string password
connection conn
Methods
string getUsername(): This method simply gets the username of the user.
string getPassword():This method simply gets the password of the user.
boolean reportLoginStatus(): This method represents whether the login operation is successful or not.
void login(): This method is responsible for login operation.
void connectionEstablish(): This method is responsible for establishing the connection.

Class HomeSc
This class manages the home screen of the program.
Attributes
Button message
Button dashboard
Button singleplayer
Button multiplayer

Button info
Methods
void showMessage(): This method opens messaging page for users in order to see his/her messages from different players of the game.
void playSingle(): This method gives the signal of user’s request that a single player game and its graphics.
void showDashboard(): With this method, scores and rankings of players are shown on the home screen.
void showInfo():This method loads the credit page.
void playMulti():This method gives signal for multiplayer game request.

Class InfoSc
This class represents the information about a single player, multiplayer or teams.
Methods
void infoSingle():This method loads the page that has information about how to play single player game mode.
void infoMulti():This method loads the information page that has info about how to play multiplayer mode of the game.
void credits():This method loads the credits page.

Class MessageSc
This class manages direct messages sent and received among users.
Attributes
String[] Messages
Methods
void showMessage(): This method is responsible for showing direct messages between users.
void showSender(): This method is responsible for showing senders of direct messages.

Class ScreenControllerSc
This class manages the user interface for the main screen. All of the final outputs related to graphics are shown with this class. This class also acts as a bus to reach other classes within ScreenController.
Attributes
Screen sc
Methods
void loadPlayer(): This method basically loads the player.
void loadCredits(): This method is for loading players' credits.

void loadMessages(): This method is for loading direct messages among users.
void loadHome(): This method is for loading the home screen.
void loadDashboard(): This method is for loading game rankings of players.

Class DashboardSc
This class manages the screen that shows scores of users and their rankings among them.
Attributes
Player players[]
Methods
int getPoints(): This method is responsible for getting scores of players.
void sortPlayers(): This method sorts players' scores in descending order.
void showPlayer(): This method shows players ranking.

3.2 Server

3.2.1 Logic

Class Compiler
This class compiles and reports the code according to players' actions in the game.

Attributes
int level
int gameType
Methods
string getCode(): This method gets the generated code from the user's reactions in the game.
void compile(): This method compiles the code that has been gathered from user's actions.
string sendCode(): This method sends the code to be compiled to the compile() method.

Class GameLogic
This class has the game logic according to game levels, game types and gameplay types.
Attributes
int player_id
int gameType
int level
Methods
void playGame(int,int): This method loads the appropriate game logic according to requested game type and level of the game type.

void Game(int,int,int): This method loads the suitable game logic according to specific users with the help of playeGame() method.
void play(int,int): This method starts executing game logic.

Class ClientFetcher
To make data flow suitable in the server, we need to know that which data goes to which user. This class handles this problem.
Attributes
int player_id
Methods
void getData(int): This method moves the appropriate data to specified user.

Class ConnectionController
This class manages control of connection to the server.
Attributes
Connection connect
Methods
void connect():This method establishes database connection.

int status():This method reports status of connection.

Class DataFlowHandler
This class manages the data flow throughout the server.
Methods
boolean Send():This method reports if data flow works correct or not.

3.2.2 Data

Class User
This class provides static player information such as player ID, username, level, points etc.
Attributes
int player_id
string username
string password
int account_status
int level
int points

Class Profile
This class represents profile information of users.

Class Team
This class represents static team information such as team ID, team size, player ID etc.
Attributes
int team_id
int player_id
int team_size
int team_points
int team_level

4.0 References

[1] “Coding at school: a parent's guide to England's new computing curriculum,” https://www.theguardian.com/technology/2014/sep/04/coding-school-computing-children-programming Accessed February 17, 2018.

[2] “Adding Coding to the Curriculum,” https://www.nytimes.com/2014/03/24/ world/europe/ adding-coding-to-the-curriculum.html Accessed February 17, 2018.

[3] “İlkokuldan itibaren Kodlama dersi geliyor!,” http://www.sozcu.com.tr/egitim/ ilkokuldan-itibaren-kodlama-dersi-geliyor.html Accessed February 17, 2018.