OBJECT-ORIENTED SOFTWARE ENGINEERING

DEBATE IT

FINAL REPORT

GROUP 1B

21301294 **YAĞIZ GANİ**

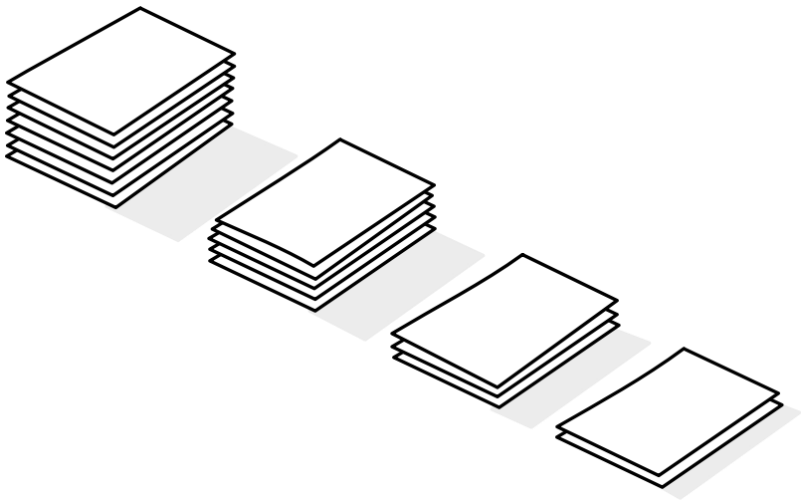
21502938 **ÇAĞATAY SEL**

21200846 **AHMET SARIGÜNEY**

21501109 **YASİN BALCANCI**

SECTION 01

17.03.2018

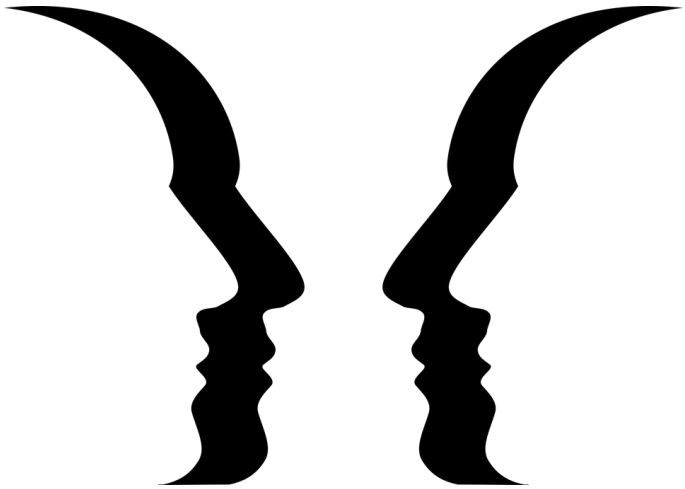


INTRODUCTION

- **In respect of game**

  The current status of our implementation, some clarifications about the design and what kind of experience we gained can be found out in this documentation. Besides, there is a user manuel part in order to introduce the limits of program.

  Nearly the absolute amount of architectural material in our data repository embodied; nevertheless we have some more work-load especially on the implementation part. We as a group still moving forward in line with our early purpose and persistence. Observing the final state of "Debate it" excites and also motivates us; therefore, we are trying our best.



IMPLEMENTATION

- **Changes in design**

In our initial design, we planned that UserHandler class will be always listening to client for requests. However, we have noticed that this is very CPU consuming and not necessary. Instead of such a design, a new UserHandler thread will be created for each request and when the request is finished, thread will stop executing.

DataReceivable interface will have an additional method to update data receivement progress to UI. This way, user will be able to see that they are waiting for server's response. In our initial design, we planned that all of the UI classes should implement this interface. However, not all of the UI classes interact with server. So, only the necessary UI classes will implement this interface.

While designing the client part, we tried to define the necessary functions of the game on the UI classes. UI class names and their functions are not changed during implementation. However, there are some little changes on the methods. Classes that will connect with the server have a ServerBridge object that will request and get data from the server. Because the classes that deal with the server has to wait for data, inner classes that extend AsyncTask are implemented in order for the UI to not process before the data comes from the server. In order to display the MainFragment in it, MainActivity class and .xml file are added.

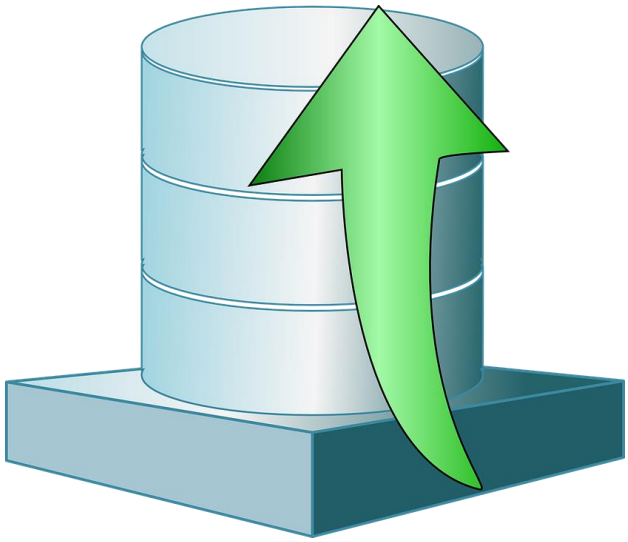
- **Implementation status**

We managed to implement UserHandler and main Server class from the Client Bridge subsystem. With these two classes we are able to establish communication between multiple clients and the server and send objects. We also implemented ServerBridge class which handles the communication in the client side. This class handles the requests coming from UI classes and forwards them to server and updates UI with respect to coming data.

We haven't started implementing BattleThread and PlayerHandler. We will implement these two classes which handles the battle logic in the next iteration.

It is also managed to implement RegisterActivity and LoginActivity classes and form related .xml files but they still are to

be completed in particular ways. BrowseBattleFragment, FinishedDebateFragment, MainActivity and MainFragment classes are estimated to be formed and completed with the missing parts of other classes until demo. We are able to take inputs in RegisterActivity and LoginActivity, send request to the server with this data and get the corresponding data from the server; set proper passes among scenes related to the corresponding button. Also LoginActivity will be considered to merge with RegisterActivity until demo presentation.

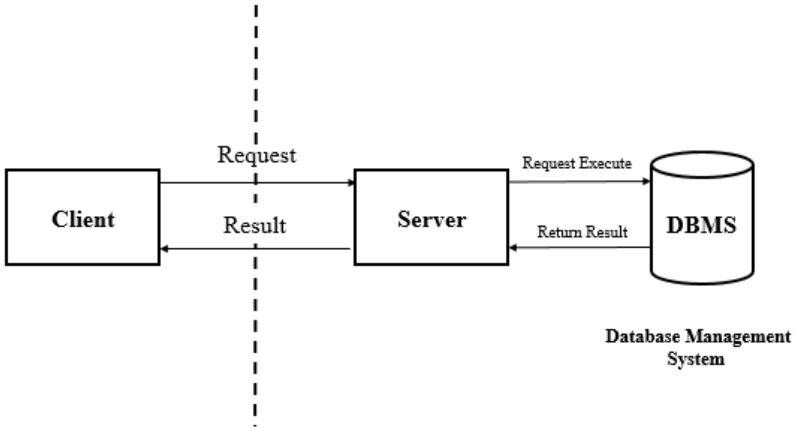


GAINED EXPERIENCE

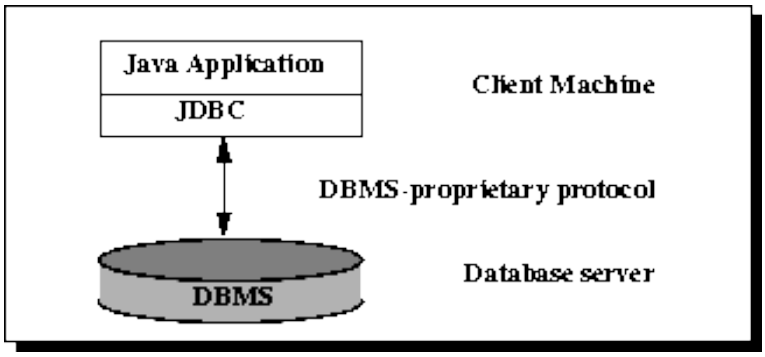
- **Database management system**

Database management issue of the program is handled by using structured query language(SQL). All kinds of datas are easily stored and manipulated in this manner.

In a nutshell, SQL is a language that provides an easy way to communicate with a database. Basically, a sturdy database can be established with the help of a few ordinary commands in SQL. Table objects are used to store all kinds of datas. The reason why we decided to use SQL is related with the complexity of our program. Using notepad or something like that could be another and the simpler way in order to process data set, however, "Debate it" requires much more efficiency in data operations like access and retrieval due to having a large amount of data. In addition to that, they have to be served users simultaneously and thus using an SQL server was considered as the right way to deal with the database management.

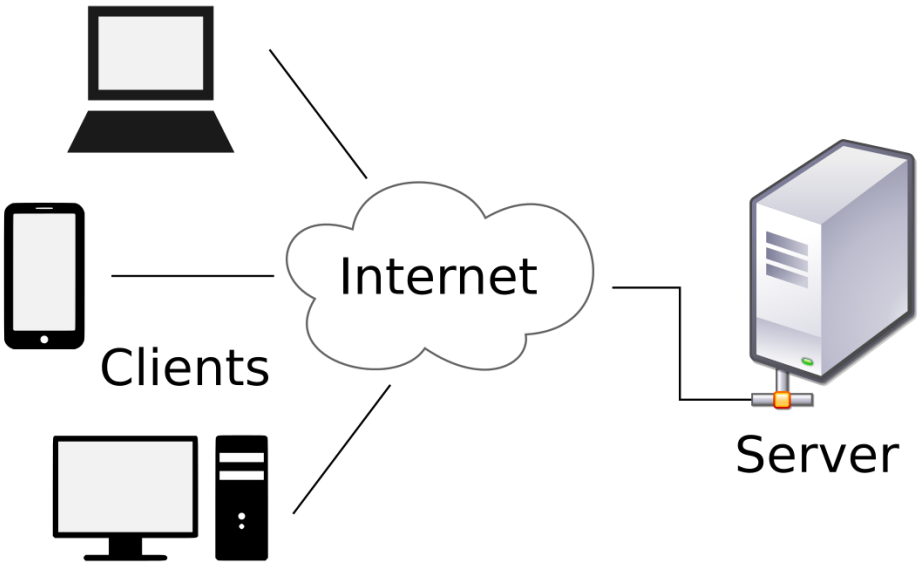


Another issue that we encountered about processing database is integrating our database system with game implementation. We are utilizing Java language to construct the game and SQL to build a stable database system for game. In this context, an application programming interface called "Java Database Connectivity" was needed to handle the interconnection between them. It basically provides our Java program to access the database management system.



- **Client-server architecture**

  To implement a server/client architecture we had to learn about multi-threading so that the server can handle multiple clients. Also, since we are implementing server in java, we learned about the Socket and ServerSocket classes in the java.net package and how to send objects through network by using these classes.



- **User interface**

  While implementing the .xml files and the related classes, relating these files is learned. Using activities, fragments, layouts, proper items and getting-displaying data via them; passing between activities with certain type of inputs (button pressed etc.), passing objects from one activity to another, stopping taking input in certain cases (if it is already taken etc.) and because we deal with a server, what to do to not perform before the data is received in order for application to not get confused are what we have learned so far.

USER'S GUIDE

- **System requirements**

  Server could be deployed to any windows machine that has jre.1.8.0_161 or higher version of JRE. Server machines firewall must allow incoming request from port 54134 (this could change) and must be in the same network as client devices.

  Client devices must have Android 4.4 (KitKat) or newer versions of android OS. They must be in the same network with server.

- **Installation**

  There is no need for installation of the server. It can be distributed as an executable file.

  Client devices can install the application from the distributed .apk file by following the standard android application installation progress.