UNIVERSITY OF STAVANGER, NORWAY

Guessing game - DAT240

(Group 14)
Asbjørn Stokka
Jone Lorentzen
Hengameh Hosseini
Vegard Matre

October 21, 2019

Github link:

https://github.com/uis-dat240-fall19/project-group-14

Abstract:

Develop a web-based game application for machine learning, creating a dataset from games played between humans. The game is about guessing the correct image by seeing as few segments of the image as possible. The game is created using a combination of HTML, CSS and JavaScript on the front end (client side) and Spring boot on Java on the back end (server side). Currently the data is stored serialized in text files, but if uploaded to a cloud service we could change it to use a database system instead.

CONTENTS

1 INTRODUCTION

Person in Charge; all

The project was to create a simple guessing game mining data for computer learning. The game has two players, one guesser and one proposer. The role of the proposer can be done by AI if there is no other human players available. The goal is for the guesser to guess the correct label for a picture chosen by the proposer, using as few as possible attempts to guess it. When the game starts, the proposer has to choose a picture and three segments from this picture. Then it is the guessers turn, where he has three attempts to guess the correct label. After that the game switch between them, the proposer chooses a segment, the guesser makes three guesses, until all segments are shown or the guesser finds the correct label. When the game is finished both players are awarded a score given by the number of segments shown. Fewer segments means a better score.

The work method was very agile.

We started by creating a UML diagram design showing the main framework of the game and what classes would be necessary to give the game its basic functionality. After doing this we discussed previous programming experience divided front end and back end tasks between the group members. From the UML diagram we also discussed interaction between the different classes and between front and back end of the game, and what APIs we would create and use for interaction between the different layers. We also made some plans for future user stories to implement in our design, and made sure our design was modularized allowed for that kind of additions.

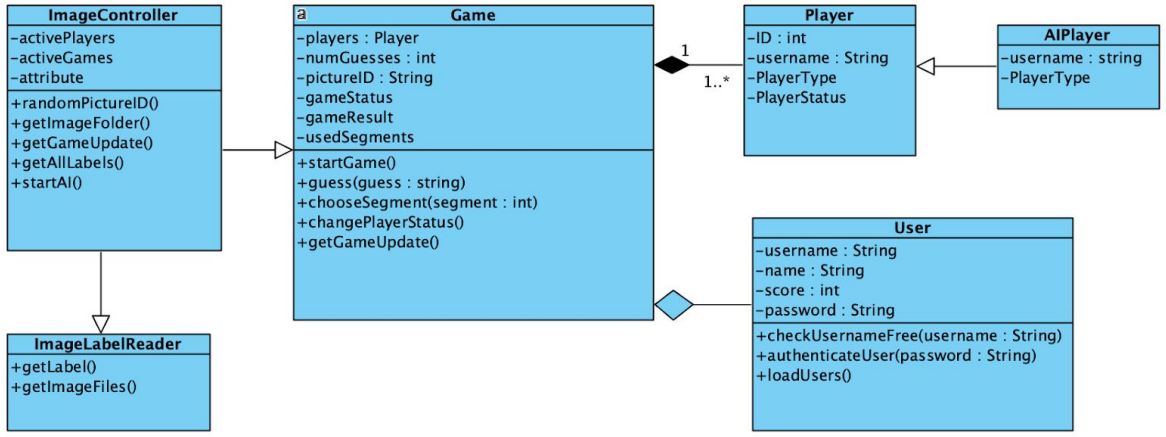
We planned for scrum meetings and what the next sprint would involved, and the more experienced coders were set to help plan how and what to work on to implement the needed code. Trying to follow clean code principles to keep the code easily readable for everyone. We used camel casing in names trying to follow the standard Java conventions. For each scrum we also reviewed code from the different classes and talked about functionality and features in it to improve the groups understanding of how things would work together.

Approaching the deadline one of the more experienced coders was planned to give some extra help where needed to push the project ahead faster.

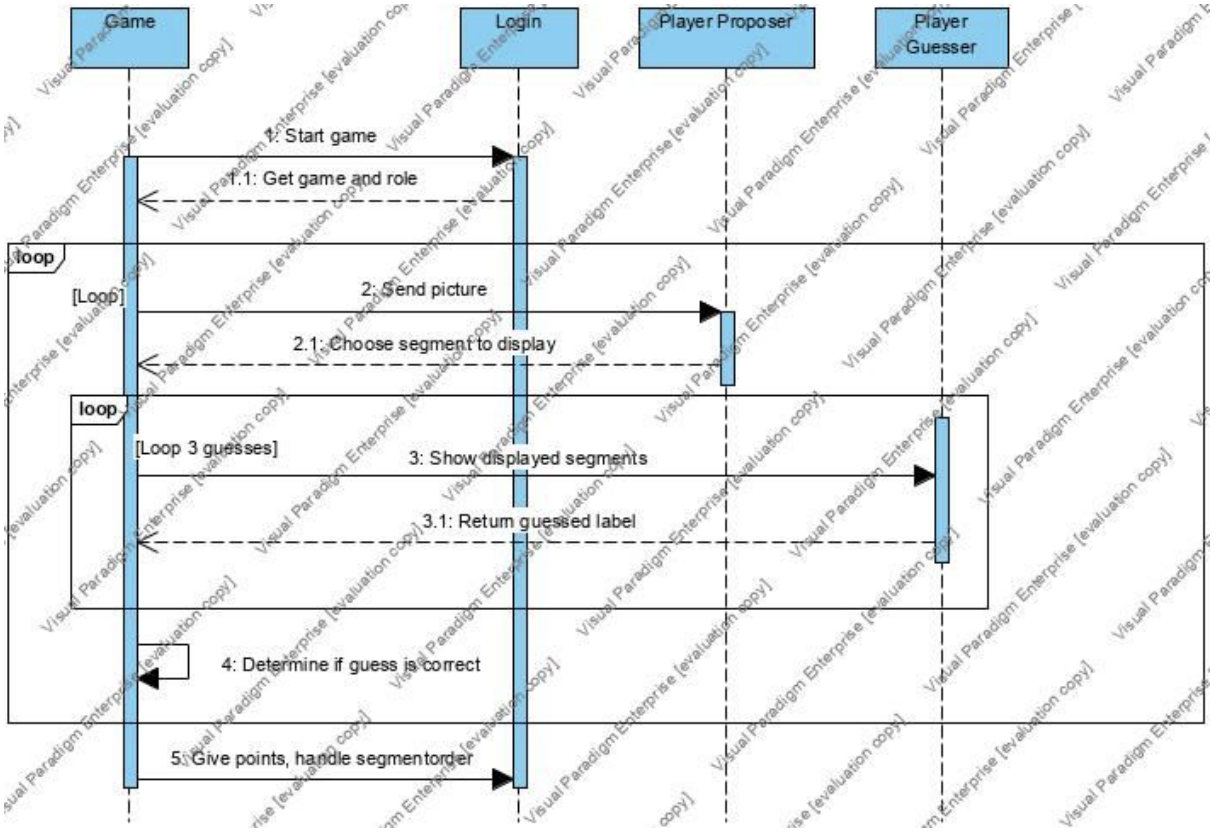
2 DESIGN

Person in Charge; all

2.1 CLASS DIAGRAM



2.2 SEQUENCE DIAGRAM



3 BACK END

Person in Charge: Jone, Vegard, Asbjørn

3.1 GAME CLASS

The Game class is the class that runs the game logic and ensures that the game works as it is supposed to. Plating the game requires 2 players, and each player needs a player type either the guesser or the proposer. Both players also needs a status that is telling the computer if the player is playing, waiting or is finished. The game object has a status so the computer can know what state the game is in. The game counts how many guesses has been done, and to change active player when three guesses has been done. It also keeps a list to keep track of which segments have been chosen.

The class has getters and setters for variables that should be available for other outside classes, protecting some content from being written to from outside sources.

guess:

The guess method is a boolean with a string input and is called when the guesser is guessing. This function adds 1 to the integer for number of guesses. Then it checks if you have made 3 guesses and if it is satisfied the players changes status. Afterwards it checks if the guess matches the picture ID that has been set in the game. If the guess is correct the game is finished and the two players will get a score. The score is given after how few segments that were chosen before the guesser guesses the correct picture ID. If the guesser guesses wrong, the method returns false.

chooseSegment:

The game also needs a method for the proposer to choose a segment. The chooseSegment method is a boolean with a segment ID as the input. When the proposer is choosing a segment there is a chance the proposer chooses a segment that already have been chosen before. Therefore every choice the proposer chooses is being checked if it is in the list of used segments. And if it is not in the list it will be added. The first time the proposer is choosing segments he should be able to pick 3 segments before it is the guessers turn. Therefore there is a check where if the size of the list is 3 or longer they switch status. This way the proposer gets to choose 3 segments the first time then 1 segment at the time after the first round.

getGameUpdate:

Because the application is dynamic it constantly needs to be updated. Therefore it needs a method that returns the state of the game and all of the information about the game. The function getGameUpdate does this where it returns a json file with the information about the game status, segments, picture ID, player role, player active and the number of guesses.

Using this method makes it possible to not refresh the whole site constantly, instead every 0,5 second a json file is being sent to update the game. Having this high refresh rate makes the game way quicker and uses less bandwidth. When a segment choice is being submitted by the proposer, the segment instantly appears in the guessers window, instead of having to wait for the site to refresh.

changePlayerStatus:

Simple method for changing the status of the players. If player 1 is the waiting part, he will be switched to the playing part or wisewersa. This method makes the code cleaner, because instead of writing the switching of the status of the players in the guess and chooseSegment methods, one can just call upon changePlayerStatus and the players would change their status accordingly.

startGame:

To start a new game, the proposer must choose a picture. The picture ID of the chosen picture must be set and the players status must also be set.

3.2 PLAYER CLASS

The player class is where the players are made, every player has attributes that must be fulfilled. Each player is assigned an auto-generated ID which is unique for each player. A player also needs a username, which is chosen by the user. There are two playertypes either you are the proposer or the guesser. The application auto-assigns the first player as the guesser where the player gets the option to be the guesser where an AI proposes segments for the player. If a second player joins the game, he has the option to choose a picture that he wants to use. When he picks a picture the game begins. The players have a status that changes accordingly after the other player has made their move. The player class needs a reference to the game, therefore a getter and a setter for the game must be implemented in the player file. This connects the player class to the game class.

3.3 USER CLASS

The user class is the class used to create the users of the game. It is in this class your username, name and password is created. The difference between this class and the player class is that the player class is used only when you play the game and don't exist outside of the game. The user class is where the user of the game are created so the user are able to play the game.

The user needs username, name and password to create a user object. The password is then hashed using the PasswordHasher class. The users are then stored in a hashmap and to different treesets. The reason behind this is that the hashmap makes it easy to get the users whenever a user is needed. The treesets are used for leaderboard and allUsers. Treesets are used because with a comparator the treesets can be sorted as needed. One treeset are sorted alphabetical and the other one are sorted by the users score. This way it is easy to collect all users sorted alphabetical when allUsers are used. The leaderboard uses the treeset sorted by score so it is easy to create a leaderboard by getting the first players in the treeset.

Methods for updating the treesets for score and name is implemented. The way the update works is by clearing the trees and then write each user from the hashmap, then the users are stored into the treeset again. This way the treesets for name and score are updated. Uses this method when there is a change made on the user. For example when someone plays and get score. Then it is necessary to update the treeset for score. Those two methods did not get used because the group decided on a different option to get a leaderboard by using datatables.

The user have methods to check if the username is taken or free and to authenticate users who try to login. In the user class there also is getters and setters. This is for score, name and username. Also there is one addScore used for adding new score to previous score.

The two comparators are in the user class. The first one sorts by checking the users name and sort them in alphabetical order The second comparator sorts by checking the users score and sort from high to low.

3.4 IMAGEGAMECONTROLLER CLASS

The ImageGameController file is the link between the backend and frontend. It is here the frontend gets what it needs to do and shows us. In this file, there are seven routes. These are the routes frontend uses. The routes are called “/game”, “/gameLogin”, “/gameUpdate”, “/sendUpdate”, “/leaderboard” and “/allUsers”. It also contains the methods getImageFolder and getAllLabels.

game:

This is the route the game is played. The program checks if the user is in the active player list. If the player is in the list, the program will also check if the player status is playing and what type the player is.. If the player is a proposer, the player will choose a picture and three segments to send to the guesser. If the player is a guesser the player will receive a picture and three segments. The guesser must then make 3 guesses on the picture label. And so on goes the game until it is finished.

gameLogin:

This route tells what's about to happen after one is logged in. One first gets a cookie unique to the username(ID). Then the program checks if there is another user that is waiting to play. If there is another player waiting, those two get matched together and start playing. If there is no other player waiting one will be added to the list for waiting players.

The way this happens is by setting a player as waiting player.. If there is someone as waiting player already the new player will match with this player. The new player will be put as a guesser and the player waiting will be put as proposer. But if waiting player equals to “None” the new player will become the waiting player waiting to get matched with a new player. When the waiting player gets matched the player gets removed from the being the waiting player. Then the function to start the game are ran. This function is in the Game-file. Both of the players are then put in the hashmap for active players.

Leaderboard:

The purpose of the leaderboard route is to show the top K players where one chooses K oneself. The default value of K are set to 3 if no value is chosen. Then the program goes through the hashmap the users are stored in. Then for each user saved in the hashmap the users name and score is added to two separate array lists for score and name. The two lists are used to make the leaderboard.

gameUpdate:

It is here all the information gets collected and updated. This is done by calling a method from the Game class. The way it works is that it goes through all players in the active player hashmap and run the update method on every player, and then all the player objects will be updated.

sendUpdate:

The routes purpose is to send updates about the game to the players. The way this happens is by going through the hashmap of active players and run the method getUpdate on every element in the hashmap. This way the game gets all necessary information updated. Afterward, the program will check whose turn it is and what will happen next.

allUsers:

This one gets all registered users. This is done is by getting a treeset that is sorted alphabetically and iterate it over to a list. This list is then put into a hashmap and returned back. This way one will get a table over all the players sorted in alphabetical order.

getImageFolder and getAllLabels :

The method getImageFolder is used to get the folder name for the picture. This method takes in a list of files and then goes through the list and if the file name is found it returns the folder name. The folder name is then returned.

The method getAllLabels is used to get all picture labels in the label reader. It uses the label reader to go through the label file and adds all labels to a list. The list is then returned.

3.5 AI CLASS

This project is a game for computer learning, and because of that it was only logical to demonstrate how it is possible to extend the Player class to create a basic AI player. The AI player overrides the setPlayerStatus method to activate and do it's turn as soon as it is signaled to be set active. As proof of concept we created a basic AI that simply creates random numbers and tries to feed them in as chosen segments whenever it is set as active player.

Building on this we created a SegmentStatistics class that saves the used list of segments when a game finishes. If it is a game between two human players it saves it to the list, if it is a game with an AI proposer it only saves it if it is better than the previously best recorded game with that pictureId. For machine learning this data could be used in statistical analysis to make a weighted decision on what segments seemed the most important to include for it to be possible to guess the image. We designed a simple ruleset where the segments would be awarded points depending on how well the users scored in the game (number of segments shown). To make a better and more complex algorithm for this it might seem useful to weight combinations of picture segments and/or when it was chosen. Is the last segments shown more important than the first ones?

4 FRONT END

Person in Charge: Asbjørn , Hengameh

The front end is the presentation layer of the application. Here we made that as a website using HTML/CSS, Javascript/jQuery and Bootstrap. It is everything the users sees on the website and using Javascript it lets the user dynamically interact with the game. On the server it consists of 3 template pages (login, gameScreen and gameHighscore) used by Spring boot to create the basic layout. Javascript and jQuery to dynamically modify the HTML depending on the current status of the game, it continuously retrieves (every 500ms) a json file from “/gameUpdate” on the server and uses “/sendUpdate” to send actions to the server. As we used jQuery, bootstrap and datatables on the front end the needed javascript and stylesheet files for that is imported as well as the stylesheet and javascript files for this application. We used a stylesheet file from another project that one of the group members is working on as the basis for style file for this application. As this is only the presentation layer everything in the game logic happens on the back end on the server.

login.html :

A fairly simple login page that does a post request to the server with username/password, and starts reading “lobby information” while also hiding/showing controls to indicate to the user that he/she is logged in and waiting for another player. It is possible to start an AI player if tired of waiting for a real user.

gameScreen.html:

This page loads the picture segments for the currently used picture (cinema is used as default) and shows it to the proposer. When the proposer has sent a pictureId to the server the game has a status update requesting the guesser to go from the lobby and load the same picture (and hiding it) and for the controls to change appropriately for both players to match the current state of the game. Here the interface in turn lets the users guess/choose segments until the game is won or lost. As all the updates are done in the background using Javascript/jQuery the page looks nice and smooth to the user who will just notice changes as they happen (up to 500ms delay), instead of having to reload the whole page from the server for every potential update. Using AJAX (Asynchronous JavaScript And XML (we used JSON)) has because of this become widely used for web applications and revolutionizing the way we use the internet.

leaderboard.html :

A small simple template using datatables to make a HTML table interactively sort data creating a leaderboard (sorted by score) but with optional possibilities to sort users by name, change the number of users shown and/or search users/scores.

imgGuess.js:

A file containing functions for the different buttons and also functions recurring every 500ms for the lobby and gameScreen to gather data from the gameUpdate and do respective interface changes. As it is primarily used on the gameScreen page, and there are only a few select functions (button clicks and the lobby update) on the remaining two pages all the jQuery/Javascript code for the game was put in this file.

5 SUMMARY AND CONTRIBUTIONS

5.1 SUMMARY OF THE PROJECT

At the beginning of the project the group had a meeting about how to work with project and how to do it. The group decided the best way was to prioritize making an mvp application. After the game was functional additional features could be implemented to the game.

The original plan was to have regular scrum meetings with the entire group. Because of a high total workload with university subjects and the project being initially divided in front end and back end development groups, the scrum meetings were often divided. Mid-project the group also lost one of the group members, that resigned from this subject. Facing this situation the group reorganized its resources. On the back end the regular scrum meetings were helpful to keep steady progress and a better understanding of the different classes part in the applications functionality.

In the front end part of the project there was a slow start and at the end of the project there was a lot of work left. So the front end group had to work hard in the end. Reasons behind this could be that this part of the group took a different approach on how to work.

When the group as a whole summarize how the workflow on the project has been it is clearly that backend felt it was much better work the way they did. While frontend had to work hard, backend could use time to implement features, clean code and other less important things. So for the group it was a good experience to see how different ways of working had an effect on the project.

5.2 CONTRIBUTIONS

Task	Who
Class Diagram	Jone
Sequence Diagram	Asbjørn
Game Class	Jone, Asbjørn
Player Class	Jone, Asbjørn
User Class	Vegard, Asbjørn
ImageGameController Class	Vegard, Asbjørn
AI Class	Asbjørn
login.html	Hengameh
gameScreen.html	Asbjørn, Hengameh
Img.js	Asbjørn

REFERENCES

https://github.com/uis-dat240-fall19/assignments -Github link to assignments and project folders

https://jquery.com/

https://getbootstrap.com/

https://datatables.net/