Just One

SoPra FS2020 Milestone 2

Group 15

Charlotte Eder, charlotte.eder@uzh.ch, 17-826-090
Minh Phuong Vu, minhphuong.vu@uzh.ch, 18-730-788
Kai Mitiyamulle Arachchige, kai.mitiyamullearachchige@uzh.ch, 18-705-434
Jordan Cedeño, jordan.cedenocedeno@uzh.ch, 14-713-440
Raphael Haemmerli, raphael.haemmerli@uzh.ch, 17-714-593

Refinement of User Stories, Iteration Plans on Jira

All user stories can be found on Jira under: https://sealjira.ifi.uzh.ch/secure/RapidBoard.jspa?rapidView=46&view=planning.nodetail&issueLimit=100
The user stories can be broadly grouped into six categories:
***Game-Logic**:* Everything that has directly to do with playing the game. Examples are drawing a card, giving clues, guessing mystery words etc.
***Game-Control**: Managing different game rounds.* Creating a game, joining a game, specifying how many players and bots are allowed for one round of the game etc. These specifications will be managed through a lobby.
***Player-Control**:* Creating an account and being able to login and logout in order to save the score of each player.
***Non-Human Players**:* Bots. Should be able to give clues.
***Information**:* Information that's is important to the player. A leader board, the rules etc.
***Further Ideas**:* For example a chat function or bots that can guess words.

The User Stories have been assigned to the first five Sprints in the following way: (***Iteration Plan***)

- 2.4 – 8.4: Sprint 1: Game-Logic and Game-Control
- 9.4 – 15.4: Sprint 2: Non-Human Players and Game-Control
- 16.4 – 22.4: Sprint 3: Player-Control and Game-Control
- 23.4 – 29.4: Sprint 4: Information
- 30.4 – (6.5): Sprint 5: Further Ideas

The biggest parts of the game will be implemented during Sprint one to three since this will be the time where all the members of Group 15 have their Easter break and thus a lot of time for the Sopra. For Sprint one and two, Jordan, Râphael and Charlotte will work on the backend tasks. Kai and Minh Phuong will be occupied with the frontend. For Sprint three, four and five, someone from the backend team will most likely switch to the frontend team. Sprint five and further will be used to fix all the nitty-gritty details. If the group gets through with the work quicker than expected, further details might be implemented like a chat function or a bot that can not only give clues but also guess words.

Diagrams

Component Diagram

The main component of the component Diagram is the game "Just One". It provides the service to play the game "Just One" to external players, which can be either human or bots. Also, Just One manages its data which is essential for proving a functioning game for through a database (warehouse).
Internally, Just One is split into two main components, a client and a server. The client is occupied with presenting data to players and providing them with the possibility to give input. The server is mainly responsible for providing the game-logic but also must provide other services. One of it is player control, which lets player register and account and after that, login to it. It also provides a scoreboard which keeps track of the points the individual players score during their games. There is also a Game-Controller which manages the creation, deletion and running of different game rounds.

Class Diagram

The class diagram explains the main functions of the component "Game Logic". It consists of a game class, which can set up games, let players play a game of Just One and lets them make turns. It relies on the WordComparer class to evaluate the clues and mystery words given by the players. The game also relies on the class CardStack which maps a card stack from the real word. The card stack consists of card-objects. The game is also interacts with a ScoreBoard that keeps track of the scores of the different players. These players are represented by instances of the Player class, which has the children HumanPlayer and BotPlayer.

Activity Diagrams

The activity diagrams show the main activities of setting up a game, playing one turn of the game and playing a whole round of the game.

During the game-setup, a card-stack is created and filled with cards. After that, players are created and filled into a player queue. Then, an instance of a word comparer and a scoreboard with reference to the players are created and the game can be started.

A turn consists of the following steps: A timer is started which gives the active player (the player that has to guess the mystery word) time to choose a number between one and five. After the player has chosen a number or the timer has finished, the chosen word is shown to the passive players (players that must give clues). A new timer is started, and the passive human players give their clue through an interface. The bot players consult an API to give a clue. When all the clues are received or the timer runs out, all duplicates and other invalid clues are deleted and shown to the all the human players. A new timer starts, and the active player gets time to guess the mystery word. After the timer has finished or the active player has given his clue, points are distributed depending on the correctness and helpfulness of the guess and the clues. The scoreboard, player queue and the card stack are updated, and a new turn begins.

On a bigger scale, playing through a whole game works as the following: The main loop of turns is repeated until the card stack becomes empty (get a player out of the queue, play a turn and check after that if the card stack is empty). The global score gets updated and an end screen is shown which asks the players if they want to play another game or go back to the lobby. Depending on their choice, they are either placed into a new game queue or send back to the lobby.

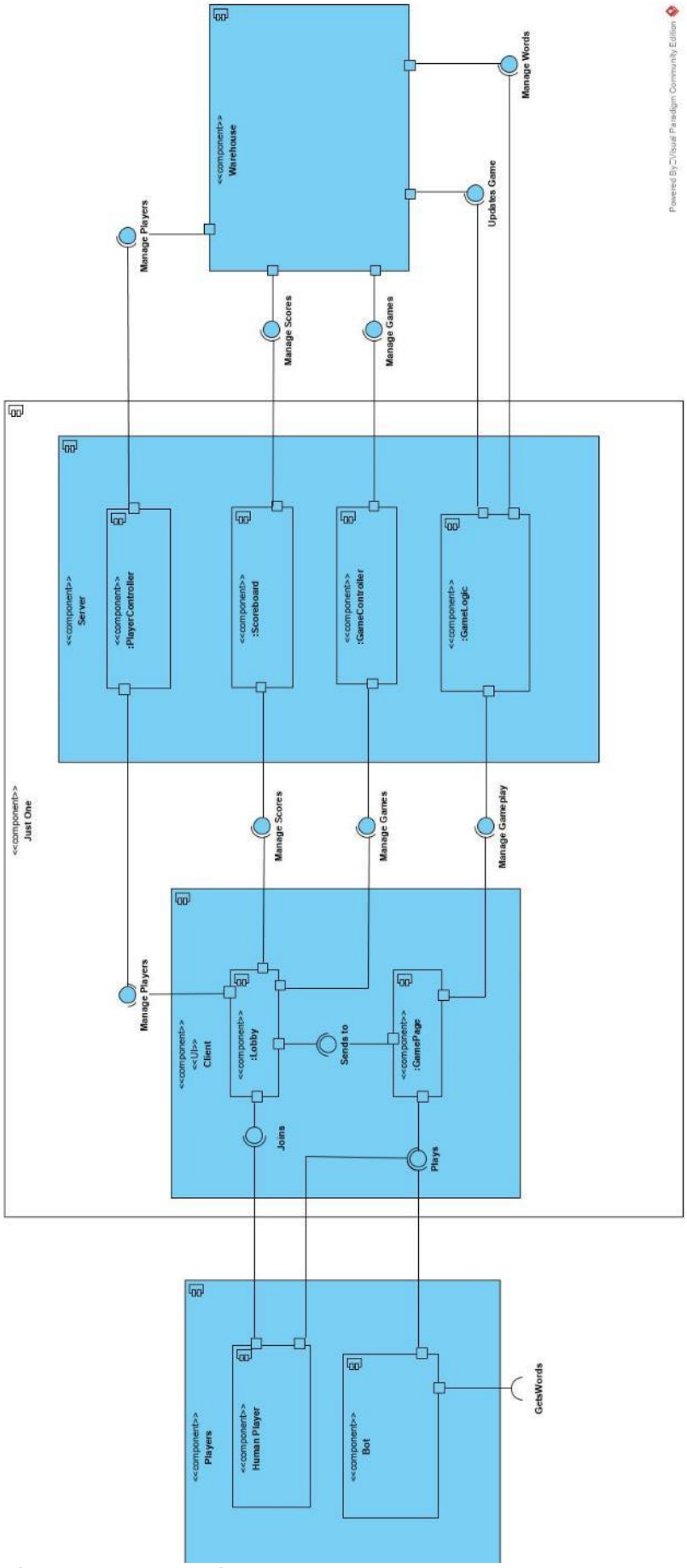

Figure 1: Component Diagram

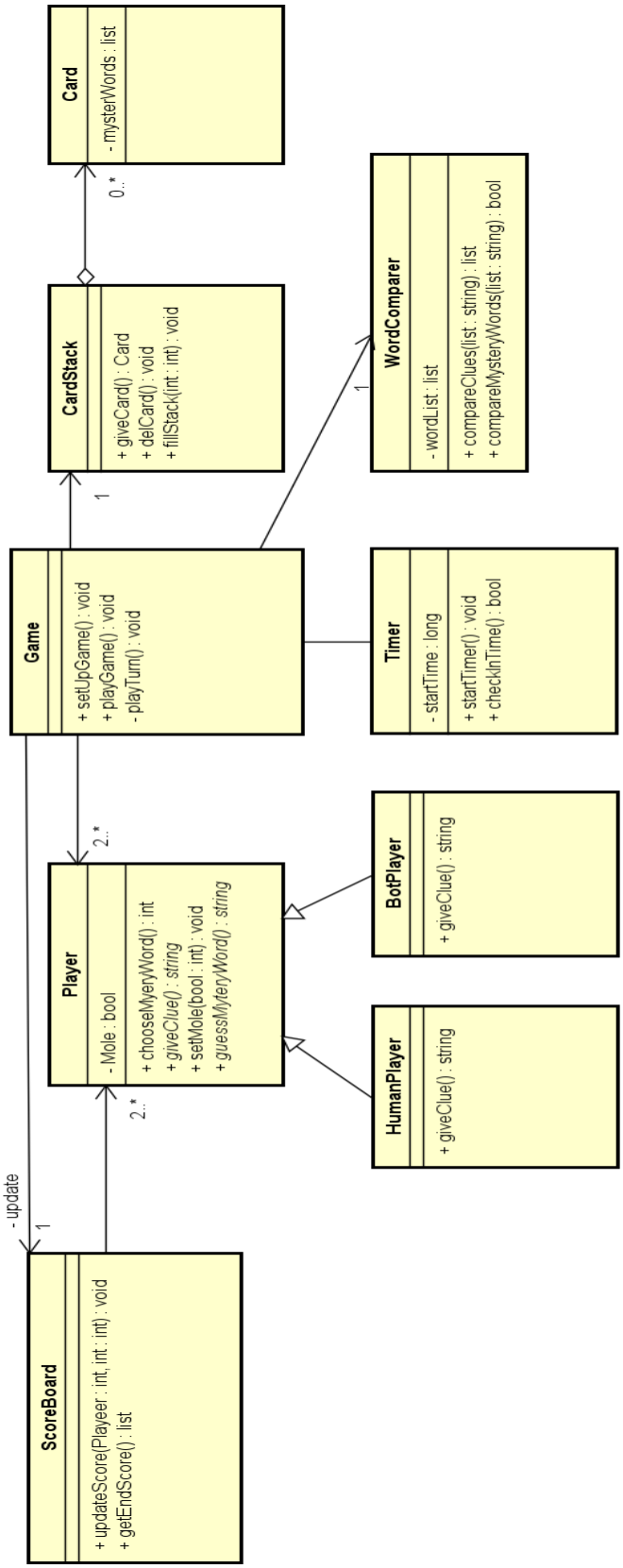

Figure 2: Class Diagram

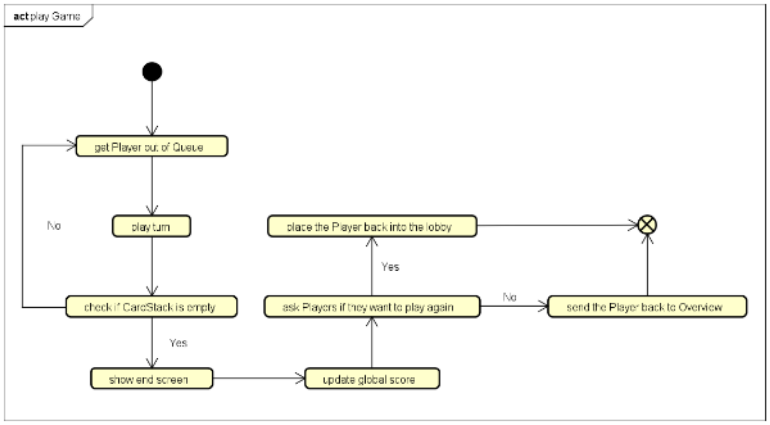

Figure 2: Activity diagram of a game

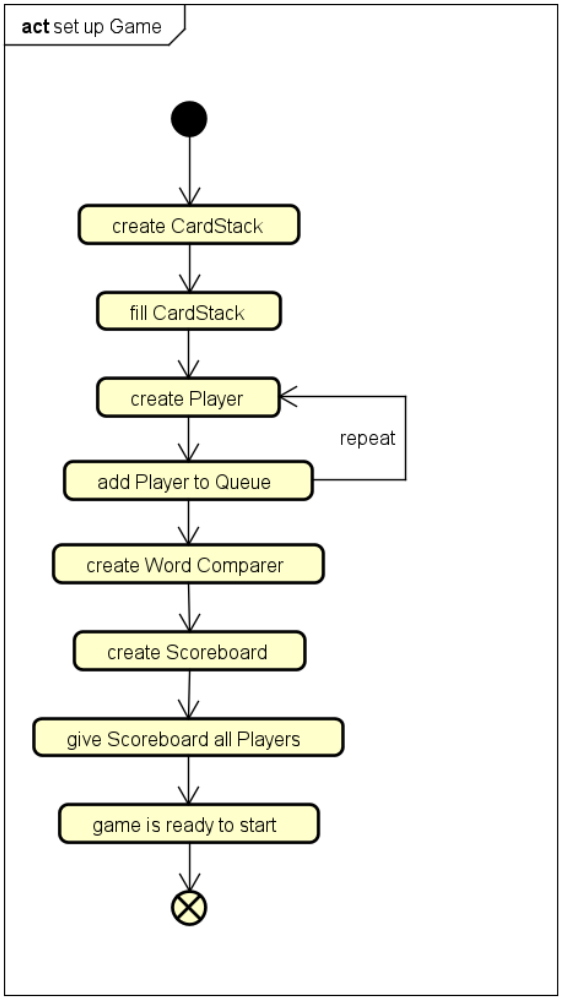

Figure 1: Activity diagram of a game creation

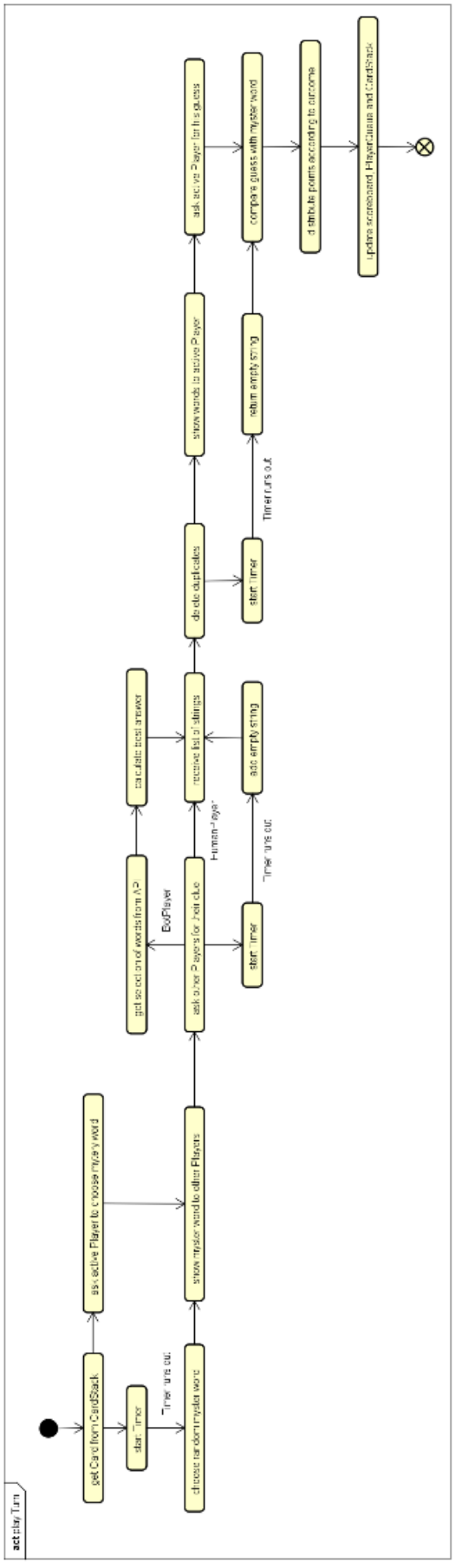

Figure 3: Activity diagram of a turn

Specification of the REST Interface and Test Cases

Login, User Registration

Mapping	Method	Parameter	Parameter Type	Status Code	Returned Value	Description
/register	POST	PlayerName <String, Password>	Body	201	PlayerId<Long>	Register a player
/register	POST	PlayerName <String, Password>	Body	409	Error: reason<String>	Player already exists
/login	PUT	PlayerName <String, Password>	Body	200	-	Login a player
/login	PUT	PlayerName <String, Password>	Body	204	-	Player is already logged in
/login	PUT	PlayerName <String, Password>	Body	401	Error:reason<String>	Login not possible because of wrong credentials
/logout	PUT	playerId<long>	Query	200	-	Logouts player
/logout	PUT	playerId<long>	Query	204		User already logged out
/logout	PUT	playerId<long>	Query	401	Error:reason<String>	A player cannot logout another player except for him or herself

Game-Related

Mapping	Method	Parameter	Parameter Type	Status Code	Returned Value	Description
/games	POST		Body	201	GameId<Long>	Create a game
/games	POST	PlayerIds List<Long> NumberOfBots <Long>	Body	409	Error: reason<String>	Zb. Because wrong Inputs like 8 Players, 20 Bots…
/games/{gameId}	PUT	playerId<long>	Query	204	-	Add user to a game
/games/{gameId}	PUT	playerId<long>	Query	409	Error:reason<String>	Zb. Game already started or full
/games/{gameId}	PUT	playerId<long>	Query	404	Error:reason<String>	Zb. The game Id or Player Id does not exist
/games/{gameId}	GET	playerId<long>	Query	200	-	Add user to a game

/games/{gameId}	GET	playerId<long>	Query	404	Error:reason<String>	Zb. The game Id or Player Id does not exist
/games/{gameId}/cards	GET	-	Query	200	Card: Words: List<String>	Get a card with words
/games/{gameId}/cards	GET	-	Query	404	Error:reason<String>	Zb. Card does not exist
/games/{gameId}/cards/words/{wordId}	POST	WordId<Long>	Query	201	Card: Words: List<String>	Set the word the active player has chosen
/games/{gameId}/cards/words/{wordId}	POST	WordId<Long>	Query	404	Error:reason<String>	Zb. Number <1 or 5<
/games/{gameId}/clues	POST	Clue <String>	Body	201	ClueList: clue, id: List<String, Long>	Save Clue, give list with valid clues back
/games/{gameId}/clues	POST	Clue <String>	Body	401	Error:reason <String>	P. ex. Player not registered to game wants to give a clue
/games/{gameId}/clues	POST	Clue <String>	Body	409	Error:reason <String>	For example active player wants to give a clue
/games/{gameId}/guesses	POST	Guess <String>	Body	201	IsGuess Valid<Boolean>	Save, guess, check if it was correct
/games/{gameId}/guesses	POST	Guess <String>	Body	401	Error:reason <String>	P. ex. Player not registered to game wants to guess
/games/{gameId}/guesses	POST	Guess <String>	Body	409	Error:reason <String>	For example passive player wants to guess
/games/{gameId}/ended	GET	gameId <Long>	Query	200	HasEnded <Boolean>, Winner: PlayerId <Long>	Check, if the game has ended and who has won
/games/{gameId}/ended	GET	gameId <Long>	Query	404	Error:reason<String>	P. ex. Game does not exist
/games/{gameId}/delete	DELETE	gameId <Long>	Query	200	-	Deleted successfully the game
/games/{gameId}/delete	DELETE	gameId <Long>	Query	409	-	Game is still running
/games/{gameId}/delete	DELETE	gameId <Long>	Query	404	-	This gameId does not exist

Player-Related

Mapping	Method	Parameter	Parameter Type	Status Code	Returned Value	Description
/players/{playerId}	PUT	PlayerId <Long>, Player	Body	204	-	Update a Player
/players/{playerId}	PUT	PlayerId <Long>, Player	Body	404	Error:reason<String>	Player not Found
/scoreboard	GET	-	Query	200	Players: List<Scores>	Get a list of a registered players and their scores
/games	GET	-	Query	200	Games: List<Game>	List of created games

Test Cases

Since the application cannot be deployed to SonarCloud and Heroku, with non-functioning tests, all the written testcases for Milestone 2 have been commented out and placed in the class UserControllerTest. The main functions that should be tested since they are crucial for a working game are from our point of view:

- POST /games: Create a game
- PUT /games/{gameId}: Put a player into a game
- POST /games/{gameId}/clues: Give a clue
- POST /games/{gameId}/guesses: Guess a mystery word
- DELETE /games/{gameId}: Delete empty games

User Interface Design – Mock-ups

When a potential Player visits Just One for the first time, he is redirected to the login page. Here he can enter his credentials if he has already created an account. Alternatively, he can click on the button “register” and register a new account. When the user has logged in, he is sent to the main lobby page where he sees an overview of all games. He can see if the game is private or public, how many players take part in a game, how many bots are added to that game and if the game has already started or is still waiting on players to join. When a player clicks on “Create Lobby” he is redirected to the game creation page. Here he can choose a name for that specific game, choose how many players will be part of the game, add two types of bots and set an optional password for a private game. Bots can either be Angels or Devils. Angels are bots that play like other players, which means that they give synonyms as clues or other helpful clues. Devils play with the goal to sabotage the game, therefore giving random words as hints instead of helpful clues. Also, bots can only give clues but not guess mystery words. Their main purpose is to make the game playable even if not enough human players are around.
When the player clicks on “create game” he gets redirected to the main lobby and should see his newly created game. By clicking on the white field under “choose game”, a player can join a game. He gets redirected to the game lobby page, where optionally a chat function might be implemented. In order to start the game, all players must check their respective “Ready?” cross. When all players are ready, a countdown is shown, and they are all send to the game-page. First, a card is drawn from the deck and is shown to the “passive players” (players that give clues during this round). The active player will only see an empty card. Then the “active player” (player that must guess the mystery word) has to choose a number between one and five. His decision is shown to the passive players. Then the passive players and bots give clues. When all clues are evaluated, they are shown to the active player. Now the active player guesses the mystery word. When the game finishes, a “game-has ended” screen is shown to all the players where they can choose to play another round or go back to the lobby.

Figure 5: Login-Page

Figure 4: Register-Page

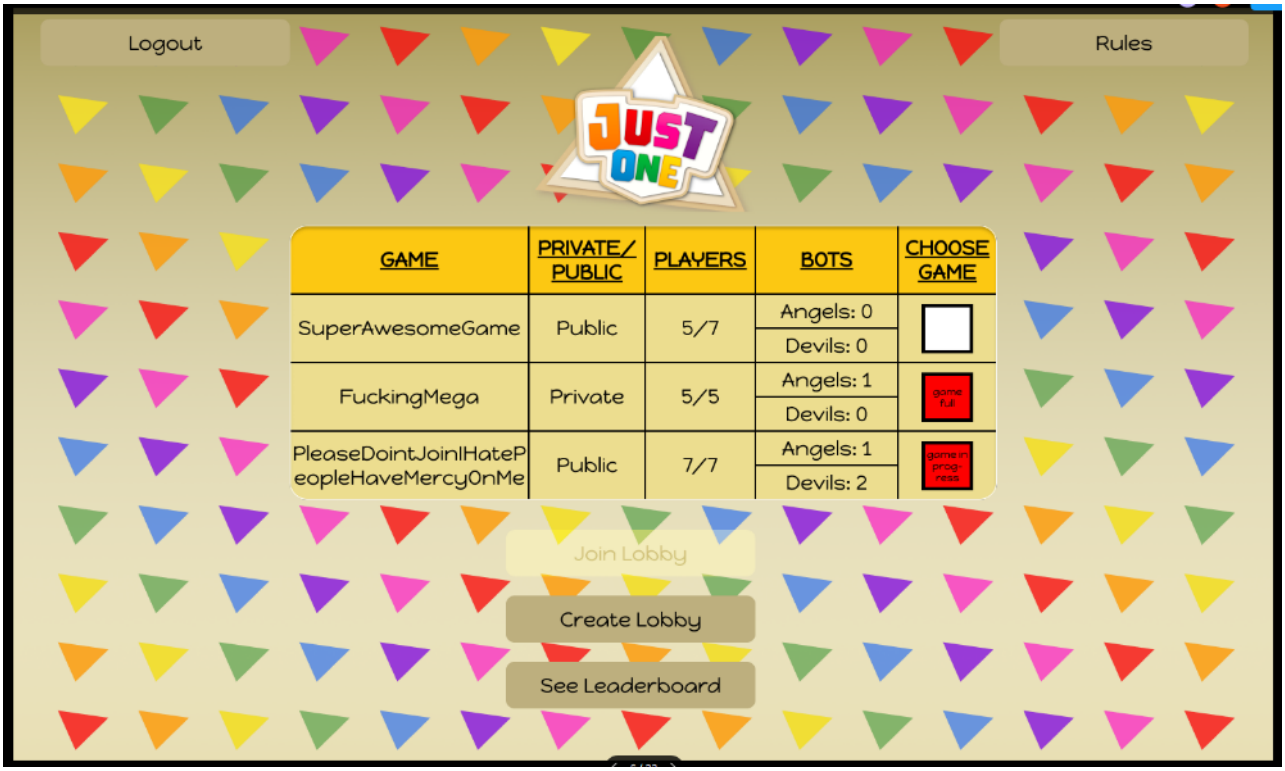

Figure 7: Lobby-Page

Figure 6: Game-Creation-Page

Figure 9: Gamelobby-Page

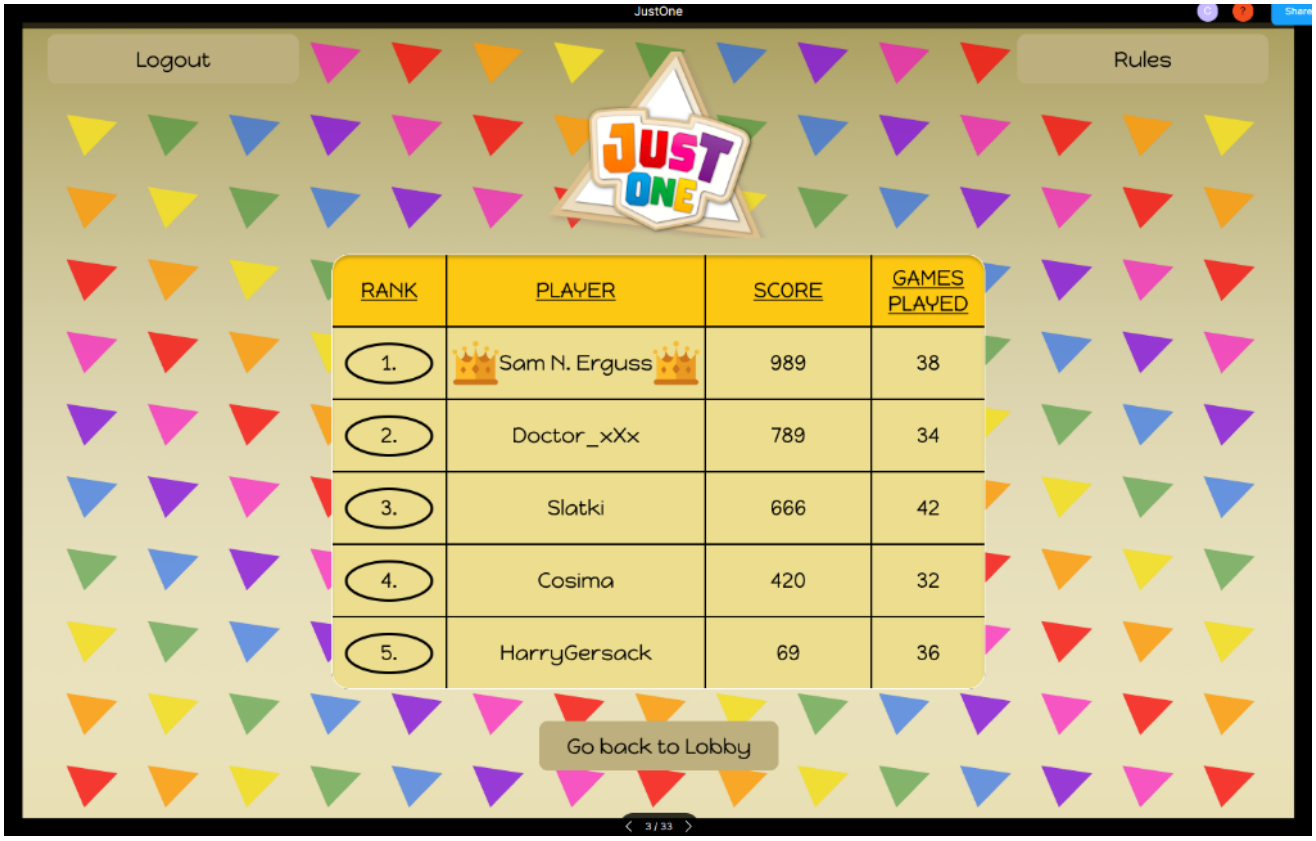

Figure 8: Leaderboard-Page

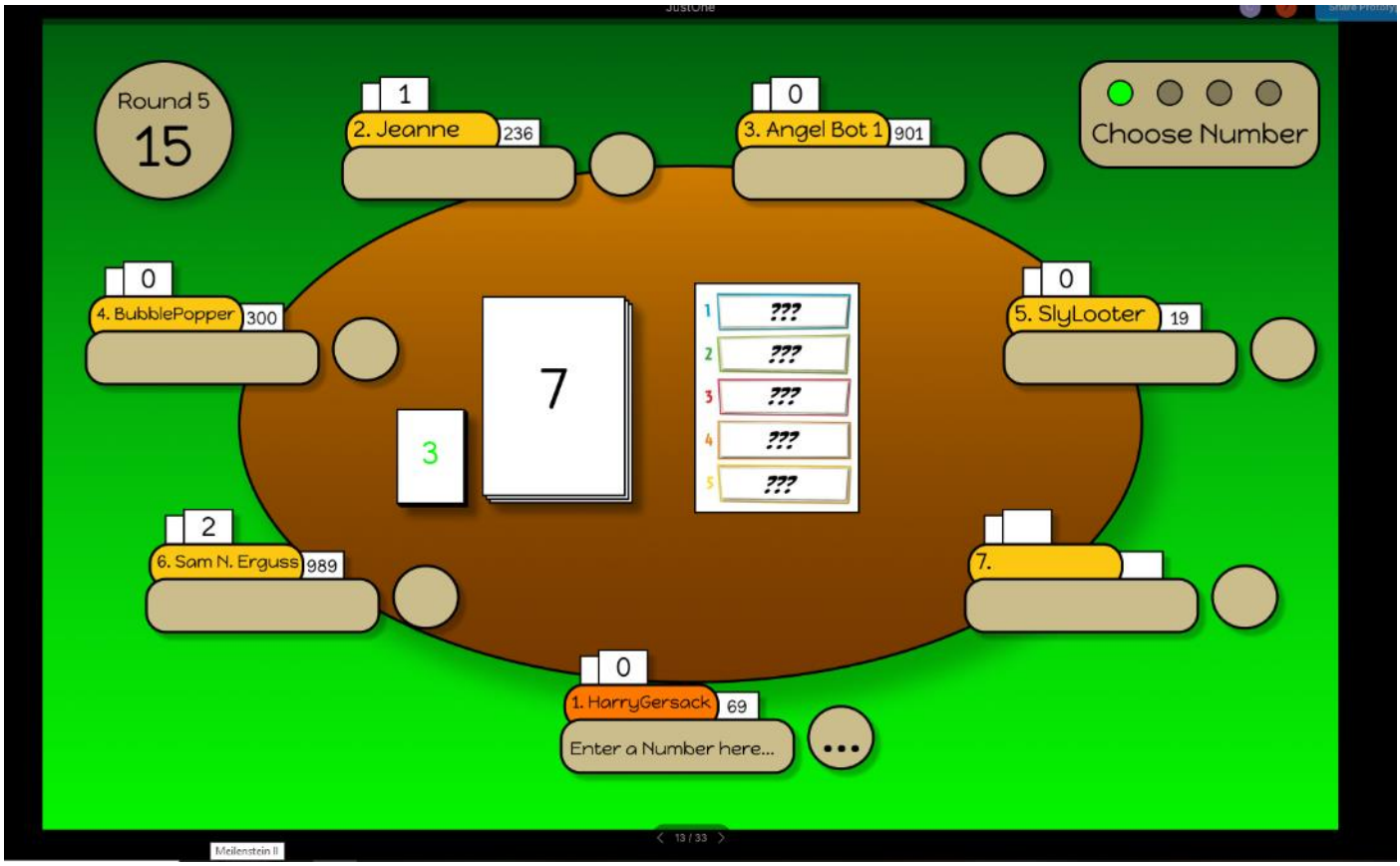

Figure 11: Game-Page: Choose a word

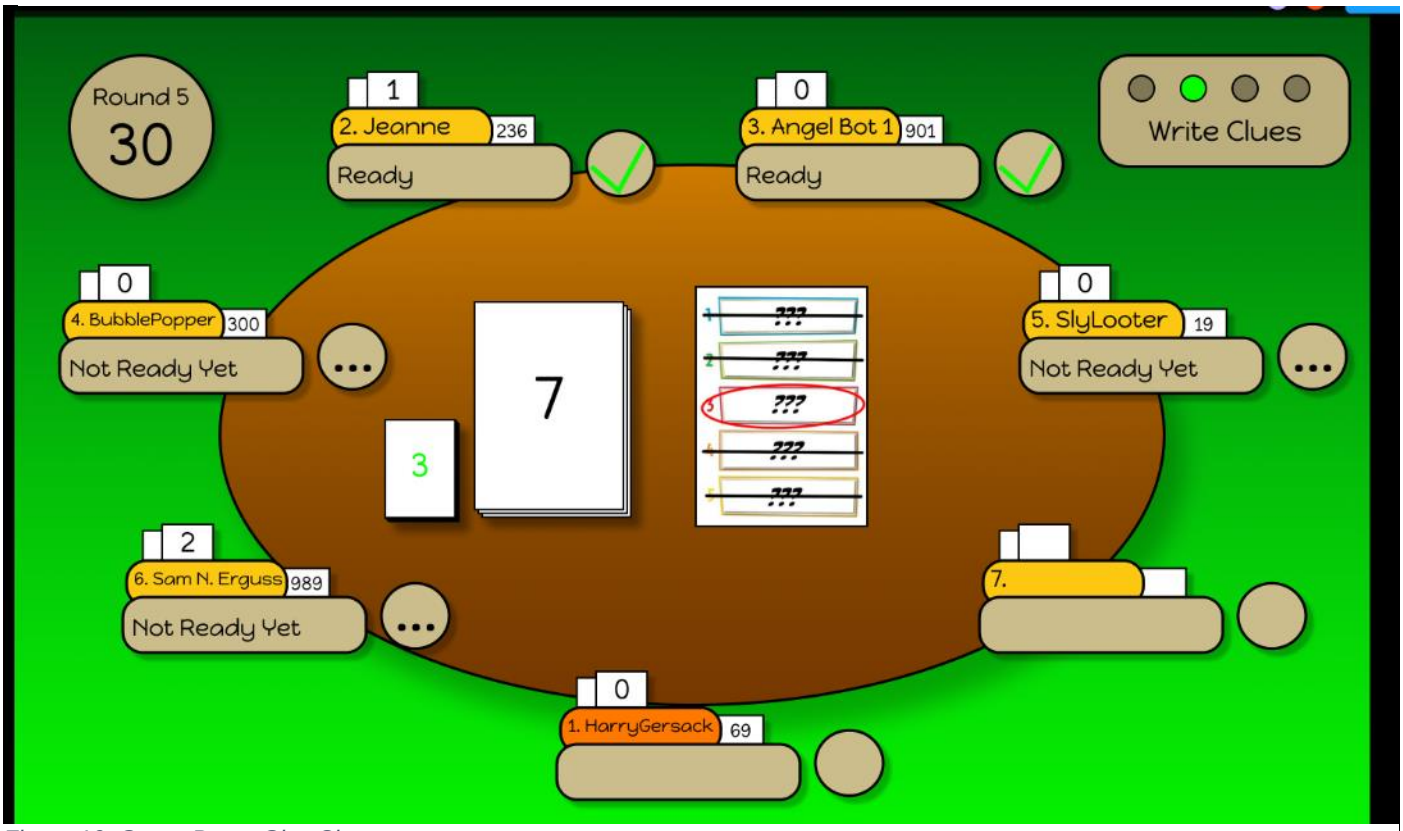

Figure 10: Game-Page: Give Clues

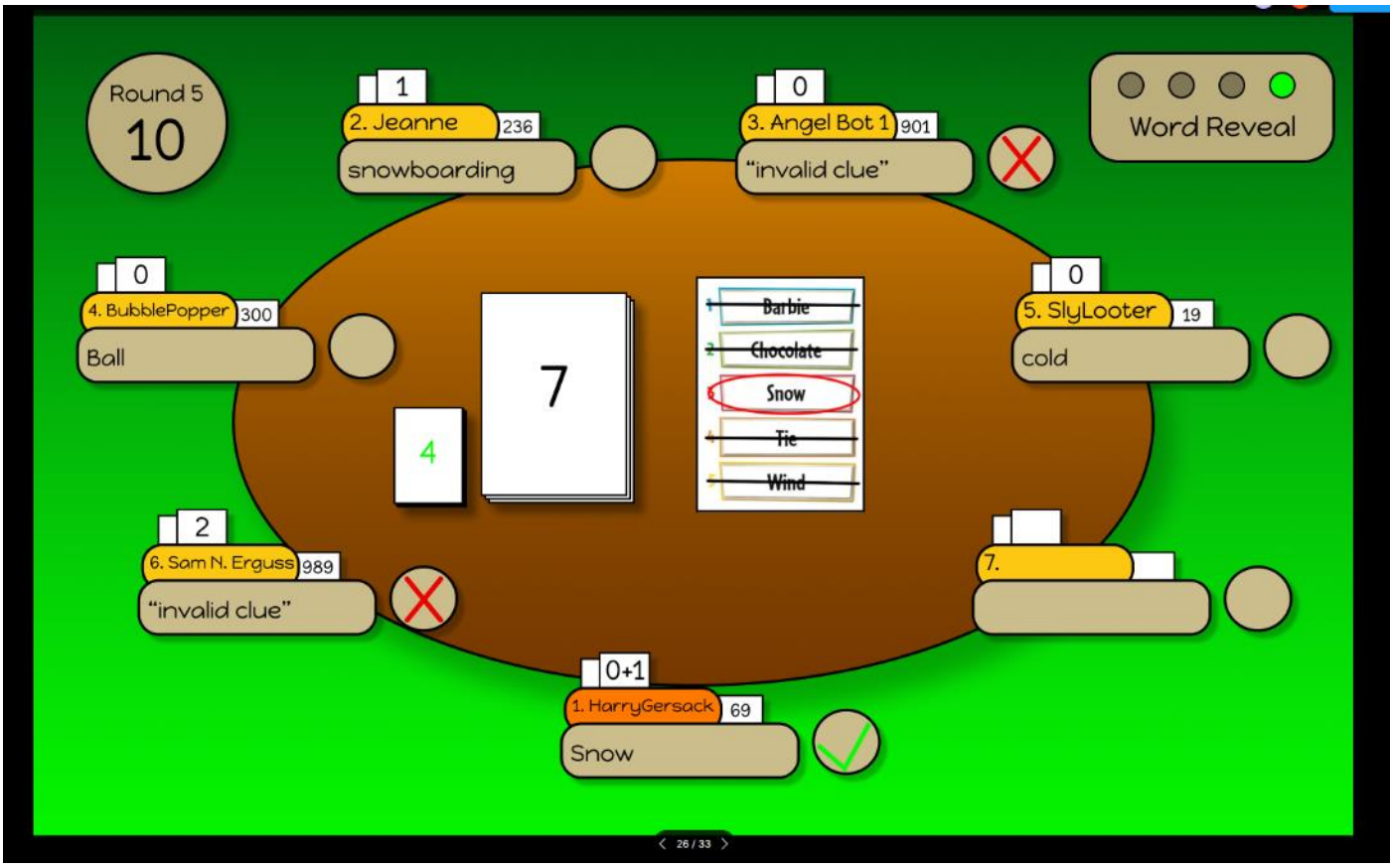

Figure 13: Game-Page: Clues are shown. Guess the mystery word

Figure 12: Game-Page: Game has ended

Links:

GitHub Group 15 folder: https://github.com/sopra-fs-20-group-15
GitHub Client: https://github.com/sopra-fs-20-group-15/client
GitHub Server: https://github.com/sopra-fs-20-group-15/server
SonarQube: https://sonarcloud.io/organizations/sopra-fs-20-group-15/projects
Heroku client: https://sopra-fs20-group-15-client.herokuapp.com/login
Heroku server: https://sopra-fs20-group-15-server.herokuapp.com/
Jira: https://sealjira.ifi.uzh.ch/secure/RapidBoard.jspa?rapidView=46&view=planning.nodetail&issueLimit=100