Scrabble 3

University of Mannheim
CS 306 Praktikum Software Engineering

Final Submission

Nils Becker, Luis Dreyer, Paul König, Martin Schmauch, Leon Urny

Group 3

Submitted on 31/05/2021

FSS 2021

Version 1.1

Table of Contents

1 Project Planning

1.1 Midterm report of progress

As soon as the tasks were distributed, we immediately started to set up the *GitLab* repository and made sure, everybody was able to make the required *test push*.

Everybody in the team was given the opportunity to develop in the field of their choice. We agreed on a branching model for our Git repository and formed multiple sub-branches of a *development branch*. Thereby we prevented synchronization errors and later reviewed and merged as needed. In weekly meetings we reflect on the work of every group member, agree on the next steps, and verify the scope of the project. We translated all the requirements into *GitLab*-Issues and assigned them to our project milestones. We used labels to categorize the issues as (non-) functional.

In the first and second week of the project, possible implementation strategies were discussed, and a "roadmap" was created. To get a common understanding, we started out by creating a first version of the domain-model and use-cases. First Mock-Ups were created, and our frontend development team began trying out Java FX and the possibilities of the *Scene-Builder*, aiming for a contemporary user-experience. We agreed to use *JSON* files to store data that exists independently of the game running e.g., the username or default game settings. Thus, we implemented a *JSON* Handler with the help of the Jackson library.

In the third and fourth week we derived a class structure from our domain model, implementing basic getter/setter methods and adding attributes for all domain classes. Those classes were improved throughout the development process. Concurrently, the first UI screens with basic action handlers were implemented. Furthermore, a basic game logic was implemented on the application layer and tested with JUnit tests. As we had no client-server architecture at this point of time, we created and later implemented our client-server-architecture-diagram. The earlier created use cases were reworked and transferred into our fully dressed use cases.

In the fifth and sixth week, we focussed on connecting the front- and backend. In the process we continuously checked and updated our client-server-architecture and worked in smaller teams to connect our UI-controllers with the server. We reworked our class structure and created annotated UML class diagrams to get a common understanding on what every class is doing in the context of the whole system. Furthermore, we started our AI-development, implementing and successfully testing a first AI-algorithm. We prepared the mid-term submission, created a system-sequence diagram as well as operation contracts. We reviewed our artifacts and cleaned up the repository.

Mannheim, 13/04/2021

1.2 Final report of progress

As soon as the second development phase started, we outlined all relevant functional requirements by creating Sequence Diagrams and by updating our GitLab issue board. Based on these artefacts we finished the *MVP*.

After that, we focused on the Implementation of the AI and improved the performance significantly. We decided to implement four different AI difficulties, which all relied on the same algorithm but with different parameters.

Meanwhile, we created a Statistics Screen that can be accessed in the game lobby and displays the user statistics of each and every player. Furthermore, we designed a Settings Screen where the lobby host can decide about a great variety of game settings and can customize certain parts of the game e.g., the length of the turn timer. We made this screen also available for clients, even when playing.

To meet the requirements of the *Play Tutorial* Use Case, we decided to build a fully guided tutorial mode. It allows a new player to learn the Scrabble rules as well as the functionalities of the Scrabble application.

Throughout the Project we developed meaningful Junit Tests to examine critical software parts such as *Turn*, *AiPlayer* or *GameController* before the features were implemented into the main game.

In the last two weeks we decided to set up daily meetings and regular issue board updates on GitLab to ensure fast and coordinated progress. Thereby we focused on bug fixing, styling the UI panels in a uniform design and finalizing our documents.

We reviewed the stability of our software by specifically testing edge cases. Thus, we tried to find and fix as many bugs as possible so our software ultimately offers a flawless user experience. Documentwise we applied the feedback from the first submission, designed Sequence Diagrams and improved/updated other design artefacts like the UML Class Diagram and the Architecture Diagram. Furthermore, we set up additional documents like the Developer Documentation and the User Manual.

In The End, we cleaned up the documents and styled our code according to the google style guide. For the final deployment of our Software, we cleaned up our GitLab repository, merged our development branches into the master branch and created a runnable jar as well as an executable for Windows. Further instructions on how to launch and play the game can be found in the user manual now.

Mannheim, 29/05/2021

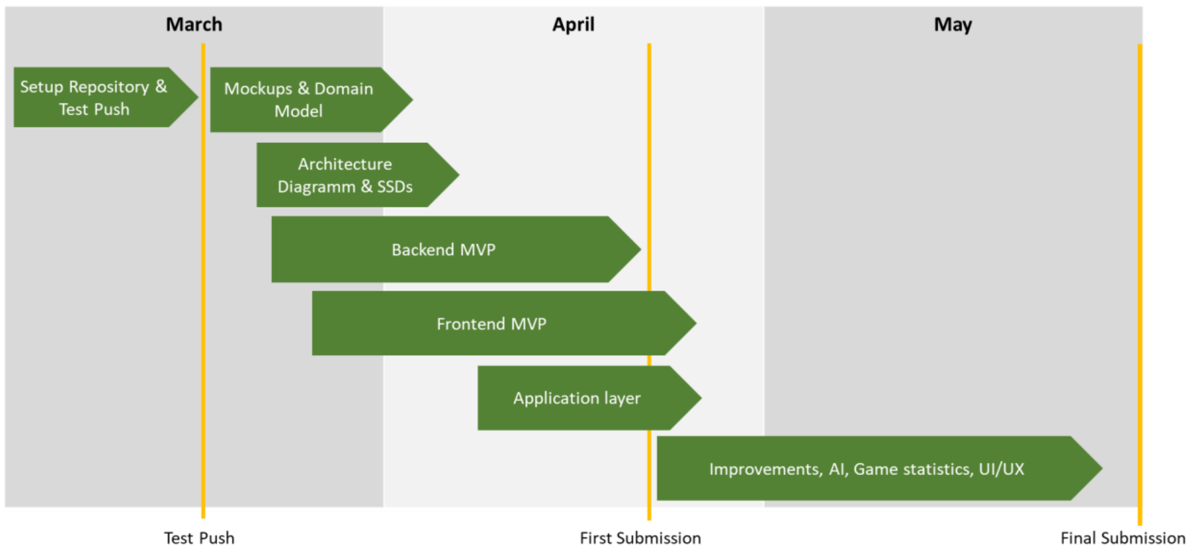
1.3 Task planning

1.3.1 Timeline

Week number	User Interface	Application	Domain	Technical Services	Architecture & Documentation
1 & 2	- create first MockUps - Explore possible implementation-strategies - Design a User Experience (Flow of User-Actions)	-	-create domain-model	-Implement JSON-Handler	-Create brief description of use-cases
3	- improve first MockUps - Implement UI-scenes as FXML-Files	- Choose UI-Controller strategy - Implement GameSettings	-Implement most important Domain-Classes (e.g., Field, Tile, Player)	-Create client-server Architecture Model	-Commit on communication interface between frontend and backend
4	- Bind FXML-scenes to Java-Classes and UI-Controllers (e.g., create unique IDs for Java FX Objects) - finish first version of LoginScreen	-Implement UI-Controllers	-Improve most important Domain-Classes (e.g., Field, Tile, Player)	-Create and implement protocol for client-server communication	-Create Fully-Dressed Use Cases
5 & 6	- finish first version of GamePanelScreen	-Implement Tile Moves and Turn Validation	- Design basic AI-Strategies - Implement basic AI-Strategies	-Link Client-Server with Lobby	- Create System Sequence Diagrams - Create Operation Contracts - Prepare all artefacts for first submission
7 & 8	- finish first version of chat - finish first version of SettingsScreen	- Implement Tile Moves and Turn Validation	-	-Client Server Validation Interaction	- Design HTML Documentation
9 & 10	-Implement more advanced designs, especially for GamePanelScreen	- Implement SettingsScreenController and make game fully customizable - bugfixes	- Evaluate different AI-Strategies and choose the best-ones - define different difficulties for AI	-Bugfixes	-
11 & 12	-Implemented Tutorial, Dark-mode, Resizable	- Fixed leave Game and Server Shutdown scenarios - various bugfixes	- finalized statistics	-	-More time for bug-fixing, add-ons and preparation of final-submission

1.3.2 Gantt Chart

The following chart visualizes our project progress and planning for the final submission.



1.3.3 Task Assignments

Layer	Task	Description	Assigned to
User Interface	LoginScreen	Implement LoginScreen in FXML and write the necessary Javacode extensions for it	*nilbecke*
	UserSettingsScreen	Implement UserSettingsScreen in FXML and write the necessary Javacode extensions for it	*nilbecke*
	LobbyScreen	Implement LobbyScreen in FXML and write the necessary Javacode extensions for it	*nilbecke*
	GamePanelScreen	Implement GamePanelScreen in FXML and write the necessary Javacode extensions for it	*mschmauc, pkoenig*
	GameSettingsScreen	Implement GameSettingsScreen in FXML and write the necessary Javacode extensions for it	*nilbecke*
	GameStatisticsScreen	Implement GameStatisticsScreen in FXML and write the necessary Javacode extensions for it	*nilbecke, lurny*
	UserStatisticsScreen	Implement UserStatisticsScreen in FXML and write the necessary Javacode extensions for it	*nilbecke*
	PlayTutorialScreen	Implement PlayTutorialScreen in FXML and write the necessary Javacode extensions for it	*nilbecke, mschmauc*
	**ScreenController*	Implement classes of Name type *ScreenController. These have to be implemented for every Screen to control and specify what the gui shows.	*mschmauc, nilbecke*

Application	GameController	Implement a GameController class, that is responsible to evaluate and verify all steps taken in the game and respond accordingly. Furthermore, write Junit-Tests for it (which have to cover 100% of written codelines)	*ldreyer, lurny*
	GameState	Implement the GameState class. This class should keep track whether the game is running or in lobby state. It refers to the GameSettings and holds the player data (including avatars) of all players in the lobby or in the game. Furthermore, write Junit-Tests for it (which have to cover 100% of written codelines)	*nilbecke, ldreyer, lurny*
	PlayerData	This class should be implemented as a help class. The PlayerData object is used for sending a minimal set of data about the player to the other players. Furthermore	*ldreyer*
	GameSettings	Implement a GameSettings class, that offers functionalities to set parameters (dictionary, multipliers, letter values etc.) before the game starts.	*ldreyer*

Domain	Field	Implement Field in Java, which represents a Gameboard Field or a Rack Field. On every Field a Tile can be placed. Furthermore, write Junit-Tests for it (which have to cover 100% of written codelines)	*lurny, ldreyer*
	Tile	Implement the class in Java and write Junit-Tests for it (which have to cover 100% of written codelines). The class is used to represent a Tile. Each must contain Tile a Letter and can be placed on a Field.	*lurny, ldreyer*
	Player	Implement Player in Java and write Junit-Tests for it (which have to cover 100% of written codelines)	*ldreyer, nilbecke*
	Turn	Implement Turn, which recognizes and validates words to calculate the turn score of a Player in Java and write Junit-Tests for it (which have to cover 100% of written codelines)	*lurny, pkoenig*
	Word	Implement the Word class in Java. The class is used to save a List of Tiles, that together form a valid word. Furthermore, write Junit-Tests for it (which have to cover 100% of written codelines)	*lurny*
	PlayerStatistics	Implement a Statistics class in Java. The class should implement methods to give easy acces to different player statistics. In addition, write Junit-Tests for it (which have to cover 100% of written codelines)	*lurny, nilbecke*

	GameBoard	Implement the GameBoard class in Java and write Junit-Tests for it (which have to cover 100% of written codelines)	*ldreyer*
	AI	Implement an AI in Java with an easy, medium, hard and unbeatable setting and write Junit-Tests for it (which have to cover 100% of written codelines)	*pkoenig*

Technical Ser-vices	JavaSocketServer	The ServerProtocol should process all messages from the client and forward them to the Server class. This class is run as a thread and is created for every connected client.	*ldreyer, lurny*
	JavaSocketClient	The ClientProtocol should process all messages from the client and forward them to the GamePanelController. This class is run as a thread and is created for every client.	*lurny, ldreyer*
	Message	Implement Message classes which are used for the client-server communica-tion	*lurny, ldreyer*

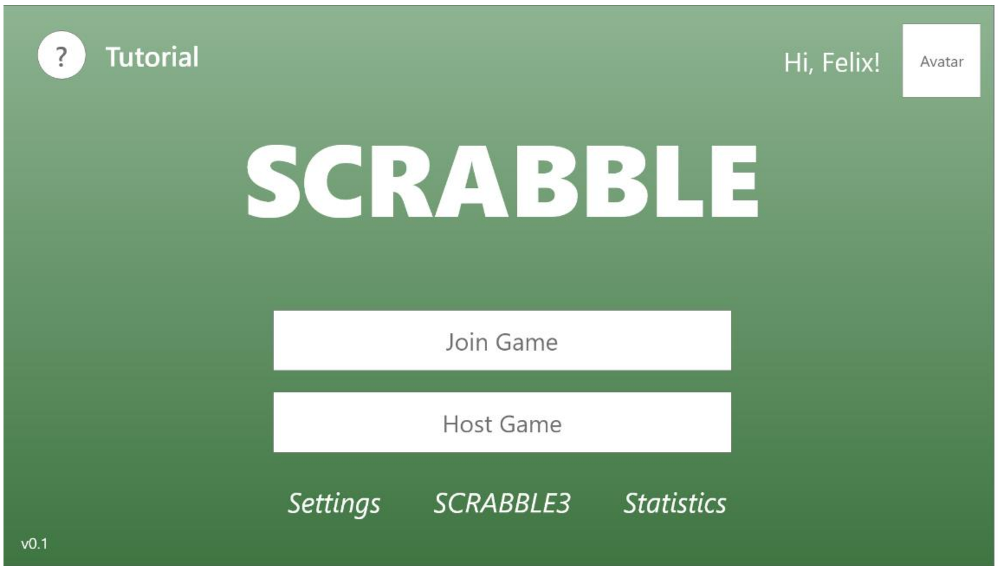
Document	Assignee
User Interface Mock-up	ldreyer
Use Cases	pkoenig
Domain Model	ldreyer, pkoenig, lurny, mschmauc, nilbecke
Short description of used (third-party) libraries	pkoenig
Architecture Diagram	ldreyer, pkoenig, lurny, mschmauc, nilbecke
Project plan:	
1. Timeline	pkoenig
2. Task Assignment	pkoenig, lurny
3. Task description	pkoenig, lurny
4a. Short written report of progress (midterm)	nilbecke
4b. Short written report of progress (final)	lurny
5. Gantt Chart	lurny
UML Diagram	lurny, ldreyer, mschmauc
Operation Contracts	mschmauc
System Sequence Diagrams	ldreyer, mschmauc
Sequence Diagrams	mschmauc, ldreyer
User Documentation	nilbecke, ldreyer
Development Documentation	pkoenig, ldreyer, lurny

2 Analysis & Design

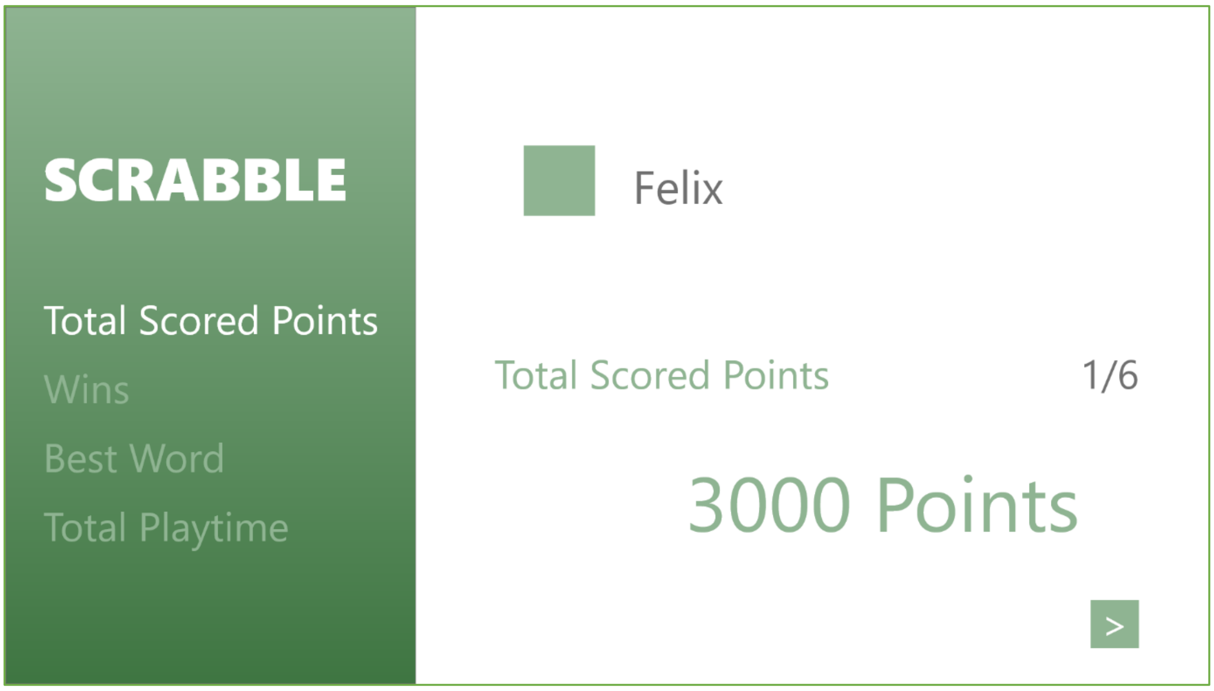
2.1 User Interface Mockups

The following Mockups were designed as early as 21st of March 2021 with Adobe XD. They were the first Artefact of our group and formed the basis for all frontend development work. They also helped to make backend design decisions.

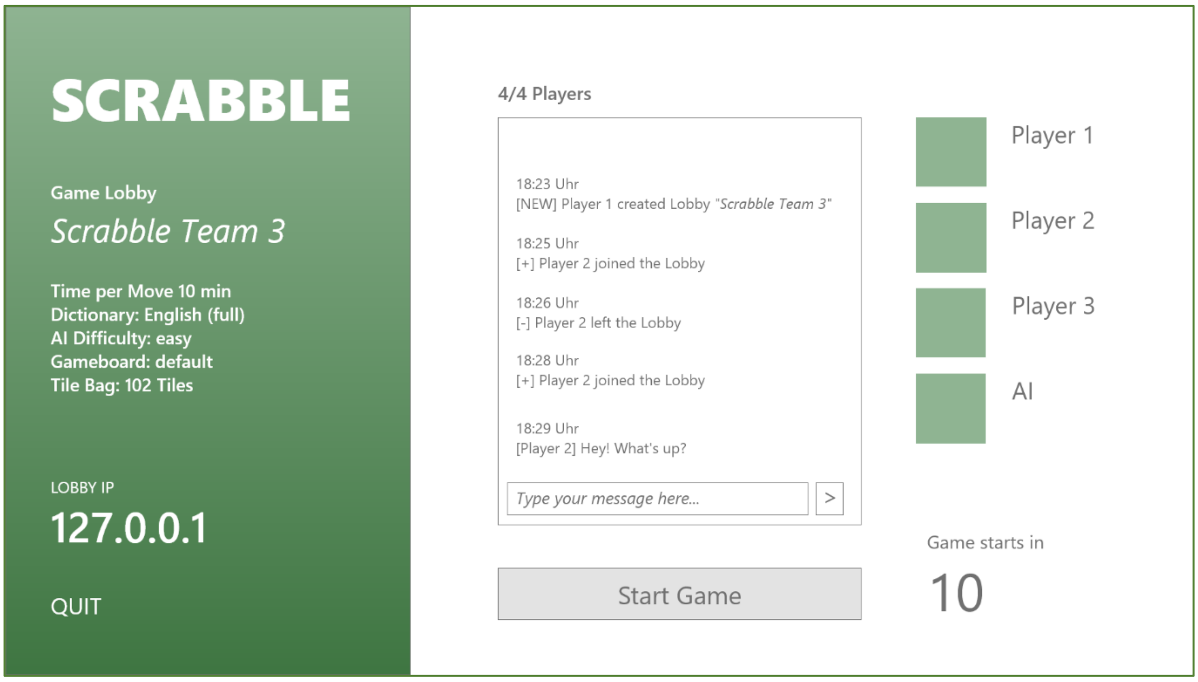
2.1.1 Main Menu Screen



2.1.2 User Statistics Screen (NEW)



2.1.3 Lobby Screen



Time per Move 10 min
Dictionary: English (full)
AI Difficulty: easy
Gameboard: default
Tile Bag: 102 Tiles

18:23 Uhr
[NEW] Player 1 created Lobby *"Scrabble Team 3"*

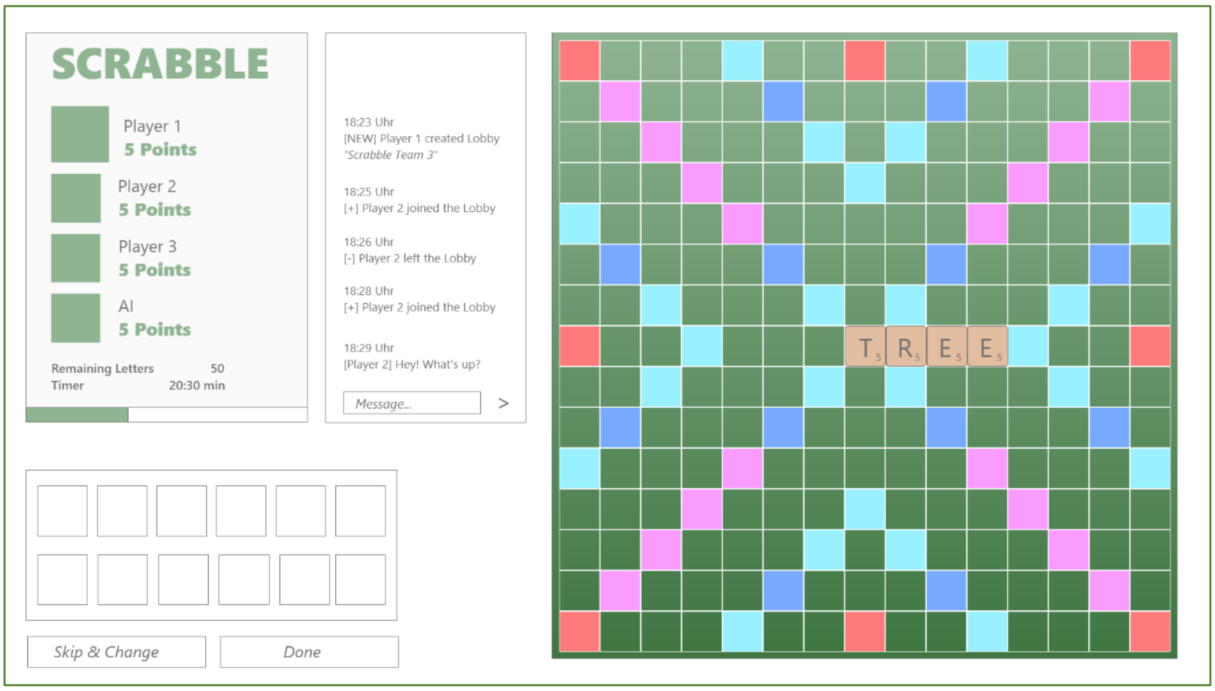
18:25 Uhr
[+] Player 2 joined the Lobby

18:26 Uhr
[-] Player 2 left the Lobby

18:28 Uhr
[+] Player 2 joined the Lobby

18:29 Uhr
[Player 2] Hey! What's up?

2.1.4 Game Screen



Remaining Letters 50
Timer 20:30 min

18:23 Uhr
[NEW] Player 1 created Lobby *"Scrabble Team 3"*

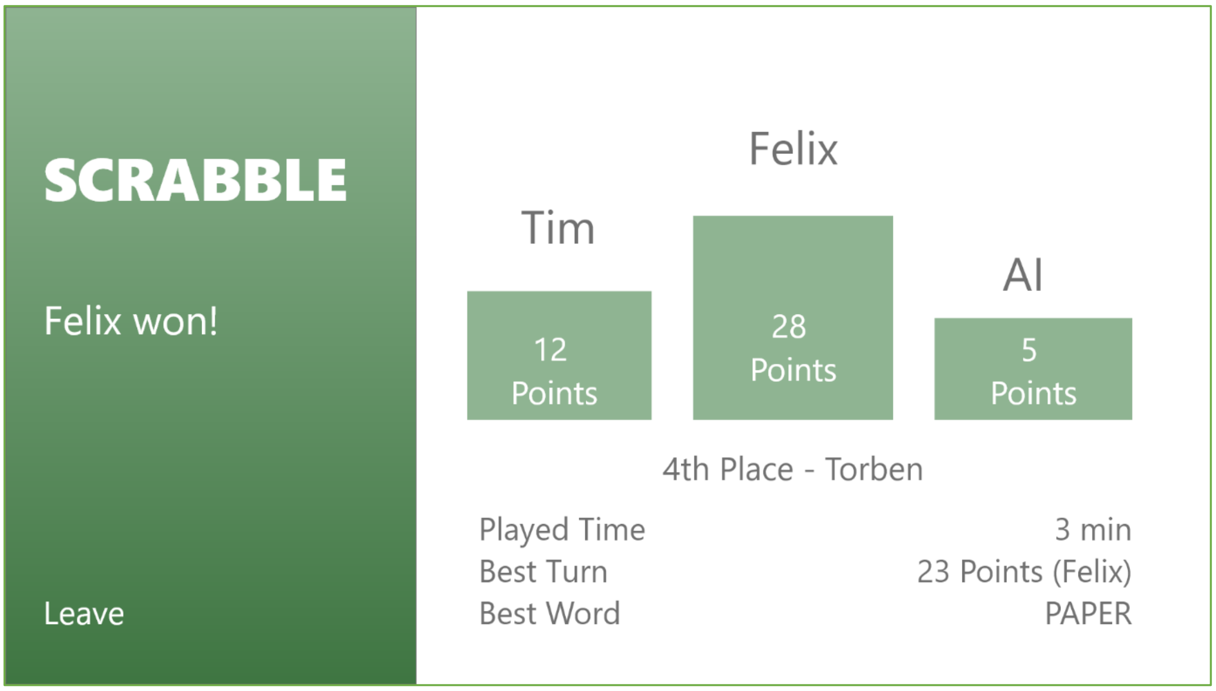
18:25 Uhr
[+] Player 2 joined the Lobby

18:26 Uhr
[-] Player 2 left the Lobby

18:28 Uhr
[+] Player 2 joined the Lobby

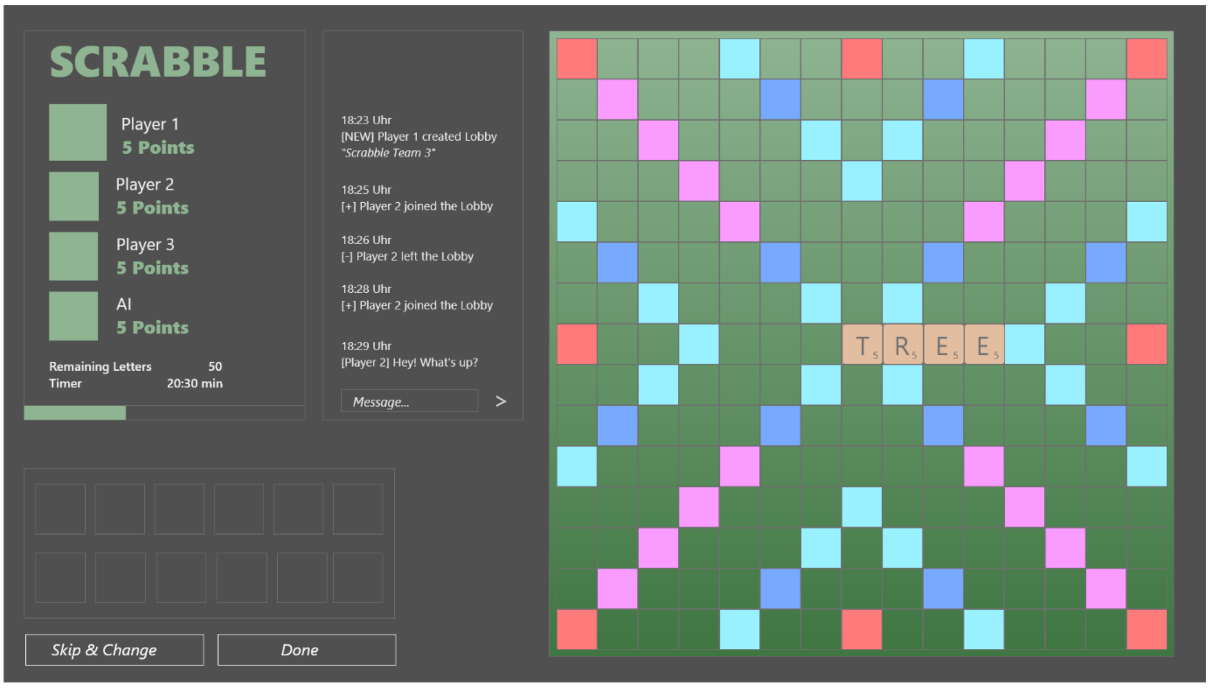
18:29 Uhr
[Player 2] Hey! What's up?

2.1.5 Leaderboard Screen (NEW)



Played Time	3 min
Best Turn	23 Points (Felix)
Best Word	PAPER

2.1.6 DarkMode (NEW)



Remaining Letters	50
Timer	20:30 min

18:23 Uhr
[NEW] Player 1 created Lobby *"Scrabble Team 3"*

18:25 Uhr
[+] Player 2 joined the Lobby

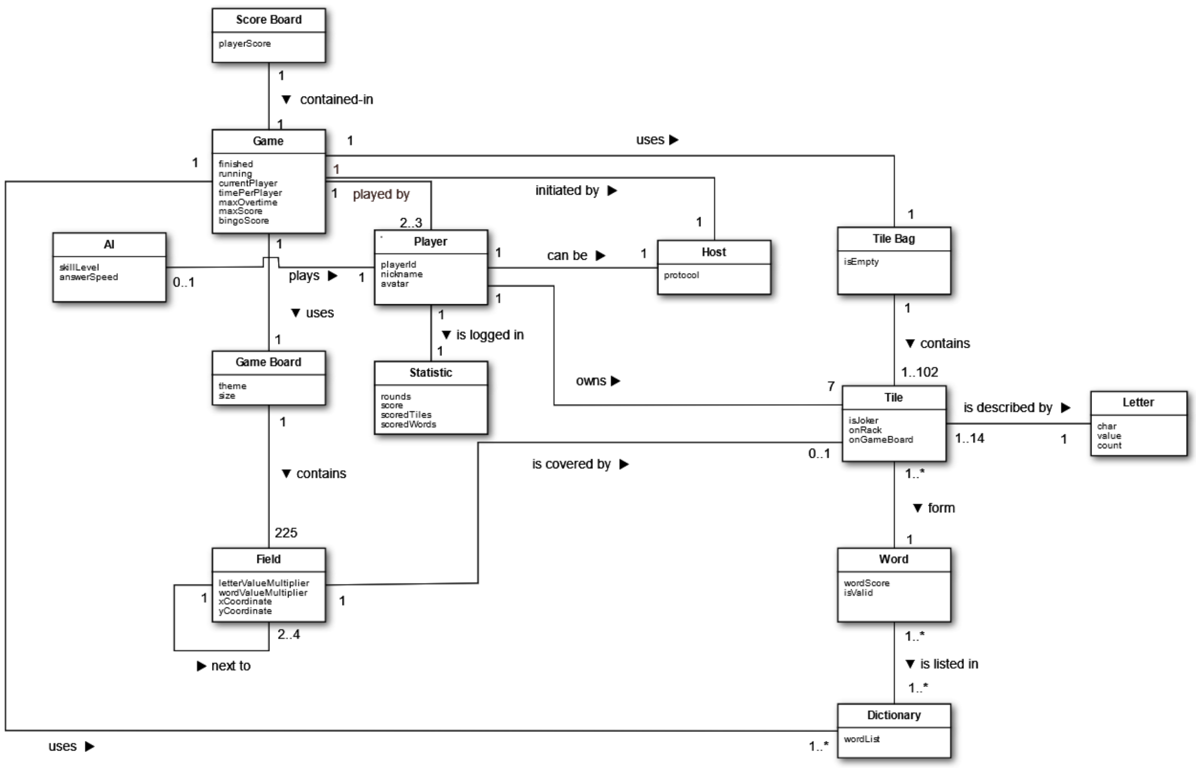
18:26 Uhr
[-] Player 2 left the Lobby

18:28 Uhr
[+] Player 2 joined the Lobby

18:29 Uhr
[Player 2] Hey! What's up?

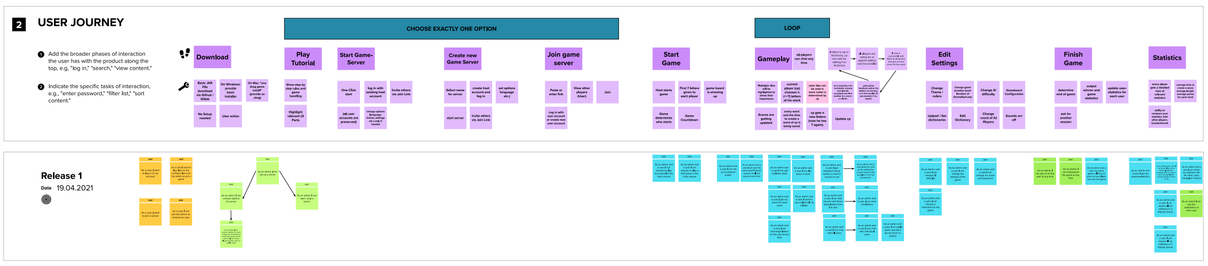
2.2 Domain Model

The second artefact we modeled all-together in real time was the domain model. We thought of the real-world Scrabble game and analyzed the Scrabble rules for fitting nouns. In a second step, we linked the found domain classes and added attributes.



2.3 Fully Dressed Use Cases

For our fully dressed use case model is used from Alistair Cockburn and added an Add-ons section. We prepared use cases on a mural board to brainstorm our ideas first, before we translated our initial approach into fully dressed use cases.


2 USER JOURNEY
Add the broader phases of interaction the user has with the product along the top, e.g. "log in," "search," "view content."
Indicate the specific tasks of interaction, e.g., "enter password," "filter list," "sort content."
CHOOSE EXACTLY ONE OPTION
LOOP
Download
Play Tutorial
Start Game-Server
Create new Game Server
Join game server
Start Game
Gameplay
Edit Settings
Finish Game
Statistics
Release 1
Date 19.04.2021

2.3.1 UC1: Manage Player Profiles

Level: Primary Tasks

Primary Actor: Game-Player

2.3.1.1 UC1-1 Create User Profile

Primary Actor: Player

Stakeholders and Interests:

- Player: Wants to create a new profile in order to profit from features like user statistics and custom avatar icon

Preconditions:

- No game is running

Trigger:

- User launches the application for the first time
- User deleted his custom player profile

Success End Condition:

- User profile is created and saved

Failed End Condition:

- none

Main Success Scenario:

1. Default user profile gets loaded
2. User gets a prompt asking for a nickname
3. User enters nickname
4. User saves the nickname to his personal profile by confirming the dialogue

Extensions:

3a. no username provided

1. Default username is used

Add-Ons:

- User can choose an avatar

2.3.1.2 UC1-2 Read User Profile

Primary Actor: Player

Stakeholders and Interests:

- Player: Wants to use his profile to log into a game lobby
- Other Players: Want to see the information about the player like player statistics

Preconditions:

- No game is running
- A custom user profile is present

Trigger:

- At launch

Success End Condition:

- User profile is loaded with username, read statistics

Failed End Condition:

- User profile could not be read properly (probably due to damaged file)

Main Success Scenario:

1. User triggers process when launching application
2. App reads profile and applies settings

Extensions:

2a Profile-file cannot be read

- Proceed with UC1-1

Add-Ons:

- read avatar and volume settings

2.3.1.3 UC1-3 Update User Profile

Primary Actor: Player

Stakeholders and Interests:

- Player: Wants to update certain aspects of his user profile

Preconditions:

- User profile has been read
- No game currently active

Trigger:

- User updates settings on user setting screen and saves changes or closes settings window

Success End Condition:

- User profile is updated with a new username and/or volume

Failed End Condition:

- User profile could not be written properly (probably due to blocked resource)

Main Success Scenario:

1. User triggers process through Join Screen
2. Application shows current user profile
3. User makes changes to profile
4. Application saves profile updates (i.e., writes it to file)

Extensions:

2a Profile-file cannot be written (first try)

- Systems informs user about error
- System asks user to close the file in any other app
- Starts another try

2b Profile-file cannot be written (second try)

- Proceed with UC1-1

Add-Ons:

- update avatar and/or volume

2.3.1.4 UC1-4 Delete User Profile

Primary Actor: Player

Stakeholders and Interests:

- Player: Wants to delete his user profile

Preconditions:

- Custom user profile has been read
- No game currently active

Trigger:

- When user indicates he wants to delete his profile

Success End Condition:

- User profile is deleted
- Application closed

Failed End Condition:

- User profile could not be deleted (probably due to blocked resource)

Main Success Scenario:

1. User triggers process through User Settings Screen
2. Application asks if user really wants to delete his profile
3. Application deletes profile from disk
4. Proceed with UC1-1

Extensions:

2a Profile-file cannot be deleted

- Systems informs user about error
- System ask user to close the file in any other app
- Starts another try

2.3.2 UC2: Play Scrabble

Level: Primary Tasks

Primary Actor: Game-Player

2.3.2.1 UC2-1 Make a Turn

Primary Actor: Player

Stakeholders and Interests:

- Player: Wants to make a valid turn, by laying down tiles, exchanging the tiles or skipping the whole turn
- Other Players: Want to see the different tile movements of the current player in real time

Preconditions:

- Game is running
- At least one tile is present on rack (usually 7)

Trigger:

- Server informs player that his turn begins

Success End Condition:

- Player got word-score
- Player received new tiles
- Next player was informed

Failed End Condition:

- Player committed invalid turn
- Timer for turn has finished and turn is invalid

Main Success Scenario:

1. Player moves a tile from his rack to the game board

Loop 1 until all tiles have been placed as desired

2. player commits turn
3. System determines whether the player formed one or more valid words (present in wordlist) in any of the possible directions
4. System presents score to all players
5. Player gets new tiles from tile bag

Extensions:

1a Player takes a tile (he placed during the current turn) from game board back to his rack

1b Player moves a tile (he placed during the current turn) to a different gameboard field

1c Player moves a tile to a different rack field

1d Player does not move any tiles, but passes turn

- Use case ends

1e Player does not move any tiles, but exchanges one or more tiles for an equal number from the bag

- Player selects tiles he wants to exchange
- Player submits tile change
- Selected tiles are added back to tile bag
- Go to step 5

1f Timer for turn has finished

- Go to step 3

2a Player placed joker tile and letter is determined by server

3a Player placed first word on game board, but it does not cover game board’s star field

- Player makes a new guess (back to 1)
-

3b No connected word layed down

- Player makes a new guess (back to 1)

3c Not all newly formed words left-to-right or top-to-bottom are on wordlist

- Player makes a new guess (back to 1)

3e Word not at least two letters long

- Player makes a new guess (back to 1)

Add-Ons:

- Player can choose to start a vote for adding word to wordlist
- Every player in game can vote for or against adding word to wordlist
- If threshold is reached, word gets added to wordlist and player receives correct scores as word stands in wordlist in the first place

2.3.2.2 UC2-2 Players receive final reporting

Primary Actor: Player

Stakeholders and Interests:

- Player: Wants to get information about the recent game like the overall leaderboard

Preconditions:

- Game is running

Trigger:

- Host chooses to end game
- Tile bag is empty and at least one player manages to play all his tiles from rack
- Six successive scoreless turns
- A player uses more than 10 minutes of play time

Success End Condition:

- Winner is determined
- Statistics are presented

Failed End Condition:

- none

Main Success Scenario:

1. Application determines winner
2. GUI is showing name of the winner and is displaying statistics

2.3.3 UC3: Play Network Game

Level: Primary Tasks

Primary Actor: Host

2.3.3.1 UC3-1 Host a game

Primary Actor: Host

Stakeholders and Interests:

- Host: Wants to create a game lobby, generate a lobby code and adjust lobby/ game settings
- Player: Want to have a lobby to join to by receiving a code from the host

Preconditions:

- Application is running in main menu
- Network connection is present

Trigger:

- Press "Host a Game"

Success End Condition:

- Server is running (for local network games only)
- Game is initialized according to GameSettings
- Game is running

Failed End Condition:

- Server is not running
- Game is not running

Main Success Scenario:

1. Server starts
2. Host waits for other players to join
3. Host can change GameSettings
4. Host decides to start the game

Extensions:

1a No Network-connection

- Host is getting a message about missing connection
- Host can try again

2a Host adds AI players

- Host chooses AI difficulty
- Go to step 3

2b Host closes lobby

- All previously connected players get shutdown message from server
- Server is stopped

3a Game started with less than two players

- Host is getting a message about missing opponents
- Go to step 3

2.3.3.2 UC3-2 Join a game

Primary Actor: Player

Stakeholders and Interests:

- Player: Wants to join a game lobby, in order to play against other players
- Host: Wants to have more players in his lobby to play against
- AI: wants to join the game in order to represent a virtual player

Preconditions:

- Server-Application is running in main menu
- Network connection is present on user- and host-side

Trigger:

- Enter the provided link (IP address)
- Press “Join Server”

Success End Condition:

- User joined a game
- Other users got a message about new game-player

Failed End Condition:

- User has not joined a game

Main Success Scenario:

1. User-Application connects with server
2. Server adds user to game-players
3. Server sends message about new gameplayer to all gameplayers
4. User-Application shows Game-lobby-screen

Extensions:

1a No Network-connection

- User is getting a message about missing connection
- User can try again

2a Username is already present in game

- No Error: Server appends 1…n as number to the username. The extended name won’t be saved, will only be used in this game

2.3.3.3 UC3-3 Chat

Primary Actor: Player

Stakeholders and Interests:

- Player: Wants to communicate with other players by text message, wants to see the scores and turn information
- Other Players: Want to see the messages from the player and might want to respond to those messages

Preconditions:

- User joined a game

Trigger:

- Message in Text-Field for chat
- Press “send” or enter key

Success End Condition:

- Message is present on the screens of all connected game-players

Failed End Condition:

- Message could not be sent
- Respective client is kicked from server

Main Success Scenario:

1. Text message is transferred to server
2. Server sends text message to each player (except origin of message)
3. Message is displayed on UI of every player

Extensions:

1a Connection error

- Client disconnects from server
- See UC3-4

2a Connection error with specific users / with all users

- Client(s) are disconnected from server
- See UC3-4

2.3.3.4 UC3-4 Leave Game

Primary Actor: Player

Stakeholders and Interests:

- Player: Wants to leave the game lobby and quit the game

Preconditions:

- User joined a game

Trigger:

- Connection Timeout
- Press on “Leave Game” or close window

Success End Condition:

- Game-Player is removed from player list
- Other users are getting a message about removed player
- If player was currently playing, tiles of turn have been cleaned
- Next player was set or if only host is left game ended

Failed End Condition:

- Other users are not getting a message about removed player

Main Success Scenario:

1. Client disconnects
2. Server removes user from game-players
3. Server sends message about removed gameplayer to all players (and if necessary what tiles to remove)
4. Client launches start screen
5. Server ends turn and informs clients whether the game ends or which player continues

2.3.4 UC4: Play Tutorial Mode

Level: Subfunction of UC2: Play Scrabble

Primary Actor: Player

Stakeholders and Interests:

- Player: Wants to learn how to play the game by playing the tutorial mode

Preconditions:

- Application is running in main menu

Trigger:

- Press “Tutorial Mode”

Success End Condition:

Tutorial Game successfully ended (as in UC2: Play Scrabble)

Failed End Condition:

- Tutorial Game left

Main Success Scenario:

1. Server is running on localhost with an AI player
2. Scrabble rules and GUI explanations are shown
3. *UC2-1*: Player makes a turn
4. *UC2-1:* AI makes a turn

Loop 2-4 until all aspects of the Game are explained

5. Finish Game as in *UC2: Play Scrabble*

Extensions:

A. At any time: Press on the designated button to get more information
B. At any time: Leave Game (as in UC3-4)

2.3.5 UC5: Display Statistics

Level: Subfunction of UC3: Play Scrabble

Primary Actor: Game-Player

2.3.5.1 UC5-1 Display Game Statistics

Primary Actor: Player

Stakeholders and Interests:

- Player: Wants to see the statistics of last game
- Other Players: Want to see the statistics of the player

Preconditions:

- Game is running

Trigger:

- Game is finished

Success End Condition:

- Game statistics are shown

Failed End Condition:

- Game statistics not added to user profile
- Game statictics not received by client

Main Success Scenario:

1. Calculate Game-Statistics
2. Save game statistics to user profile
3. Display game statistics to every user

Extensions:

2a Game statistics cannot be saved

- Step is skipped, game statistics not added to user profile

3a Game statistics cannot be displayed

- Server is closed directly, clients disconnect

2.3.5.2 UC5-2 Display User Statistics

Primary Actor: Player

Stakeholders and Interests:

- Player: Wants to see the overall game statistics of his player profile
- Other Players: Want to see the user statistics while waiting for the game to start in the LobbyScreen

Preconditions:

- Game is not running

Trigger:

- The statistics button on the login screen is clicked

Success End Condition:

- User statistics are shown

Failed End Condition:

- Error message is shown Statistic screen not open

Main Success Scenario:

1. Read user statistics
2. Display user statistics (statistics of all played games are shown)

Extensions

1a user statistics cannot be read

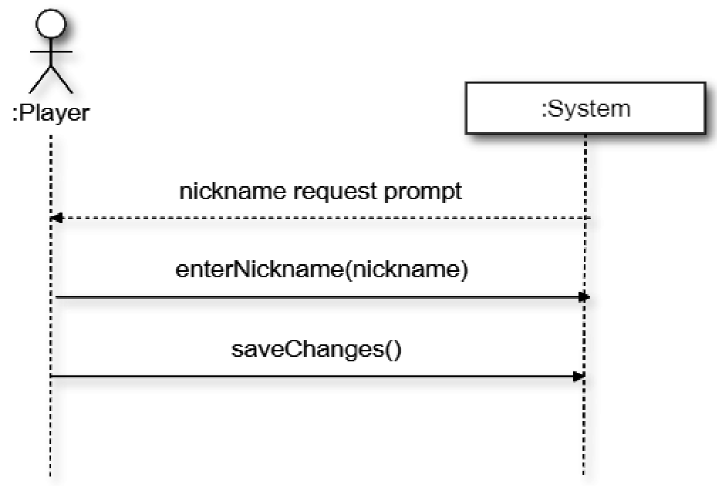
- Error message is shown, Statistic screen not open

2a user statistics cannot be displayed

- Error message is shown, Statistic screen not open

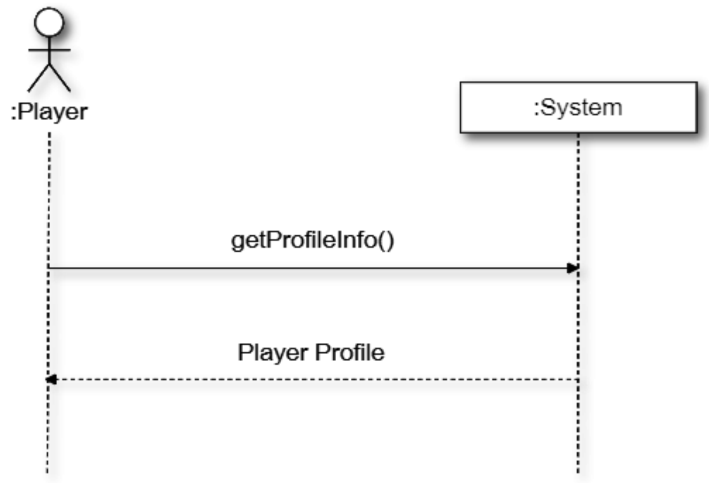
2.4 System Sequence Diagrams

2.4.1 SSD for UC 1.1: Create Player Profile



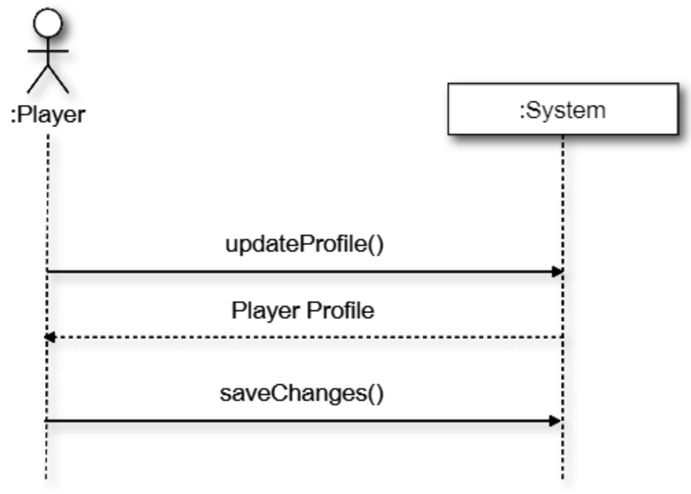
Main Success Scenario

2.4.2 SSD for UC 1.2: Read Player Profile



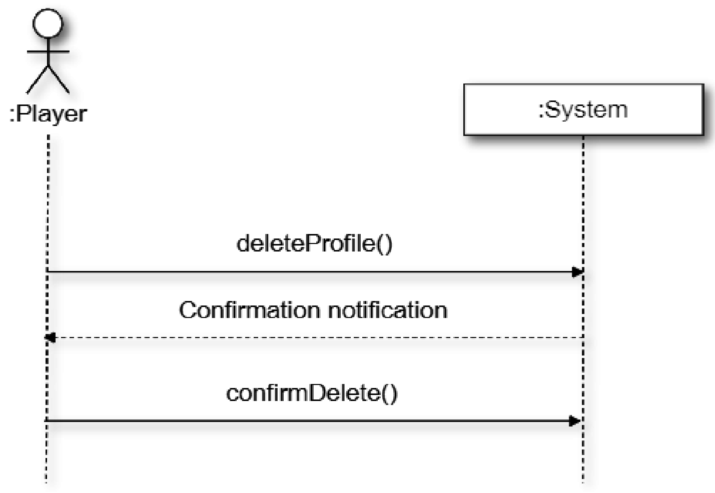
Main Success Scenario

2.4.3 SSD for UC 1.3: Update Player Profile



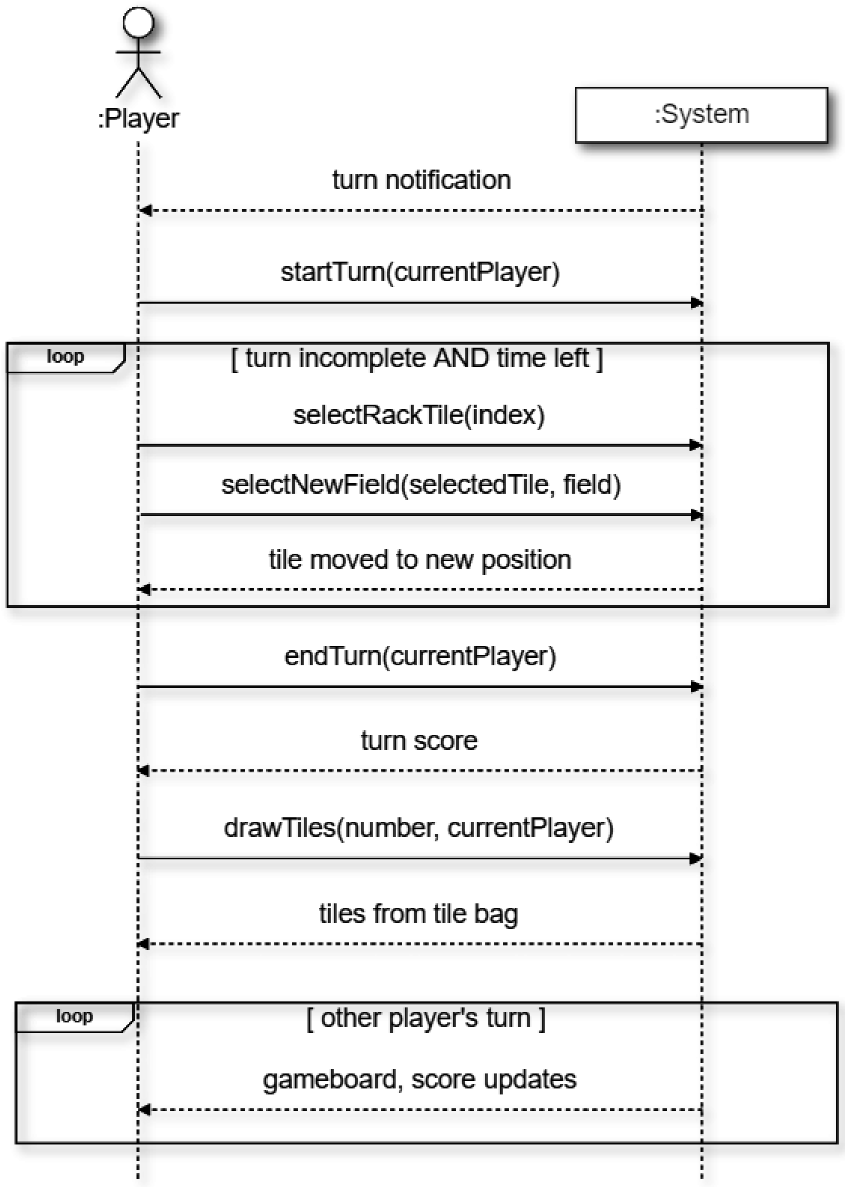
Main Success Scenario

2.4.4 SSD for UC 1.4: Delete Player Profile



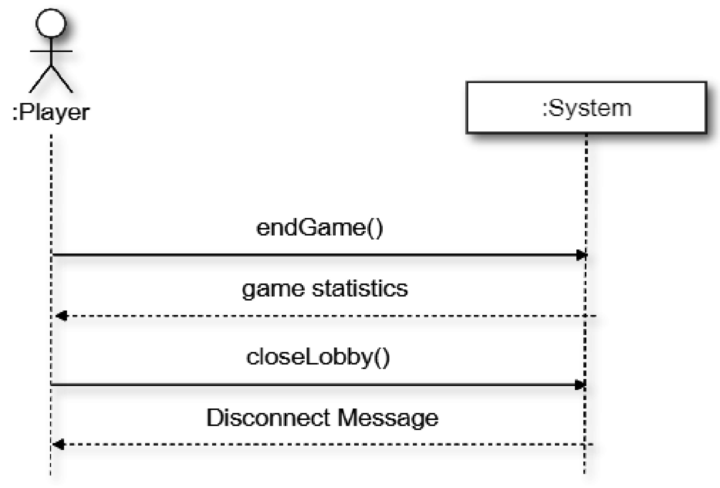
Main Success Scenario

2.4.5 SSD for UC 2.1: Make a Turn



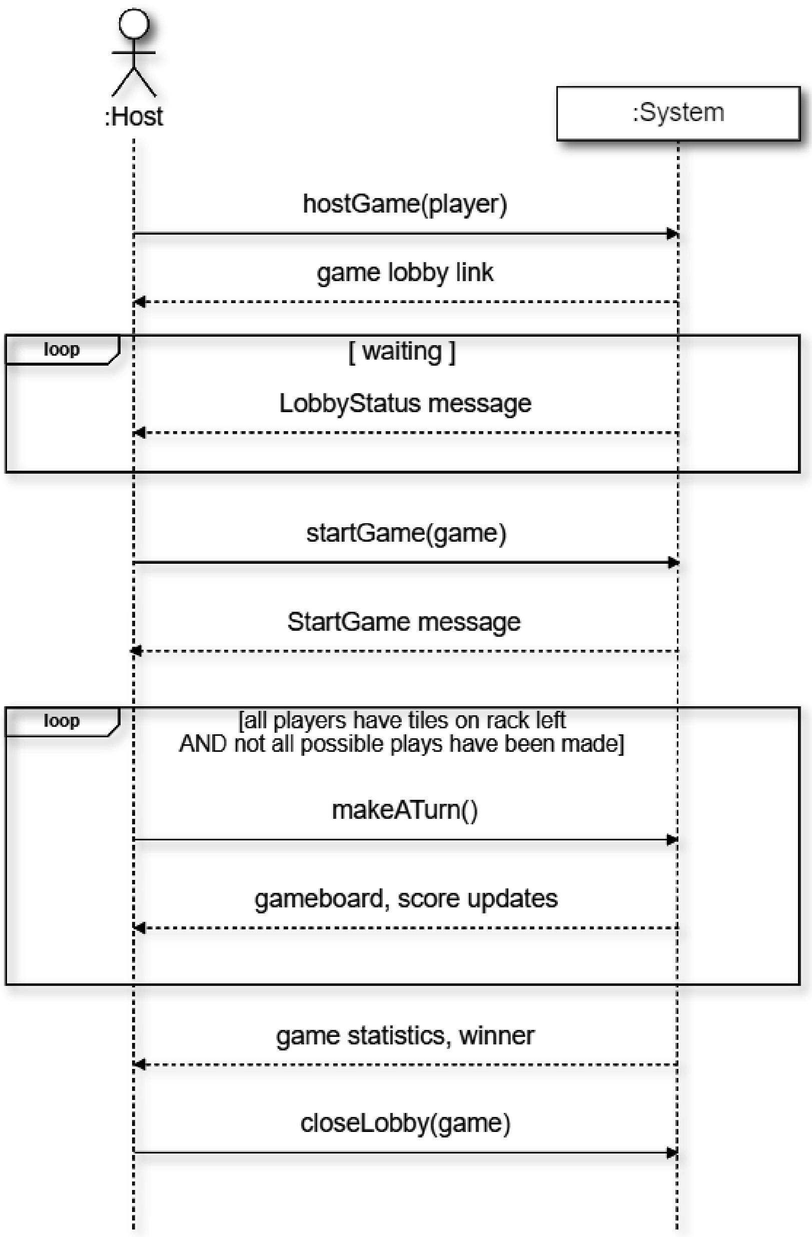
Main Success Scenario

2.4.6 SSD for UC 2.2: Players receive final reporting and can get winners name



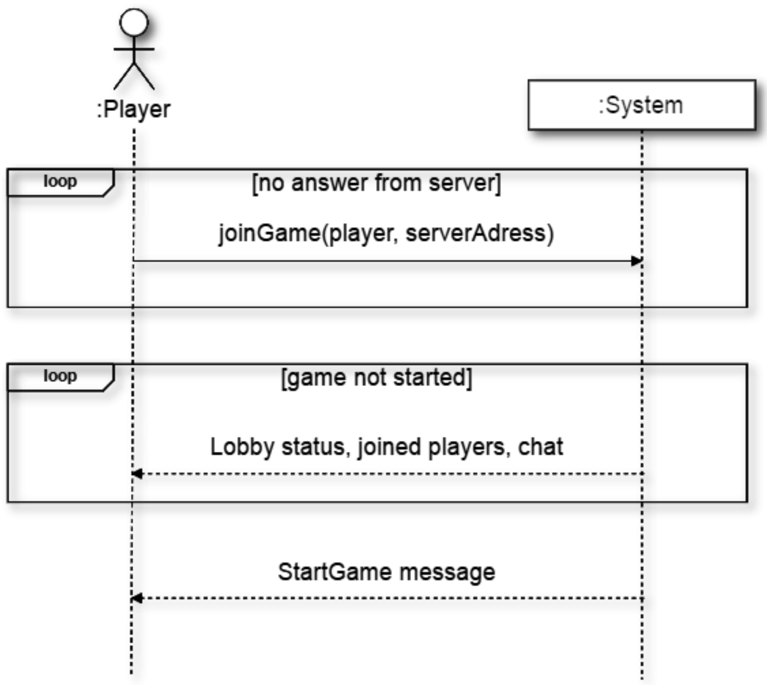
Main Success Scenario

2.4.7 SSD for UC 3.1: Host a Game



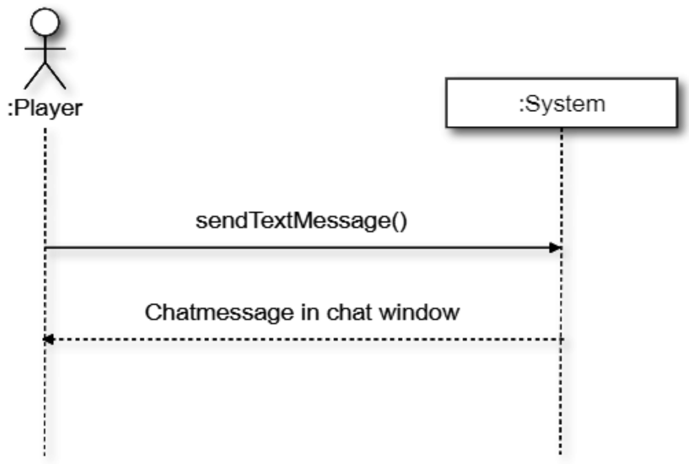
Main Success Scenario

2.4.8 SSD for UC3.2: Join a Game



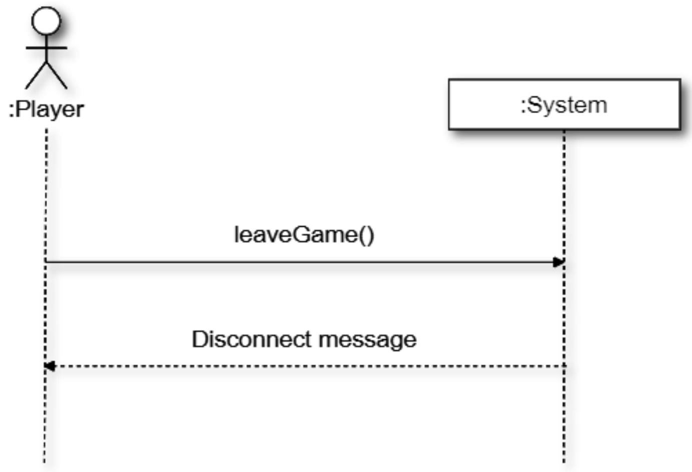
Main Success Scenario

2.4.9 SSD for UC3.3: Chat



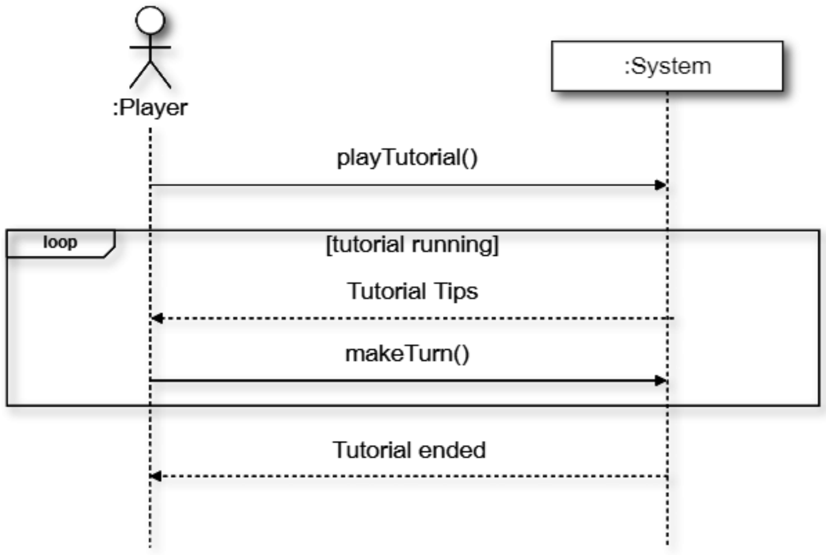
Main Success Scenario

2.4.10 SSD for UC3.4: Leave a Game



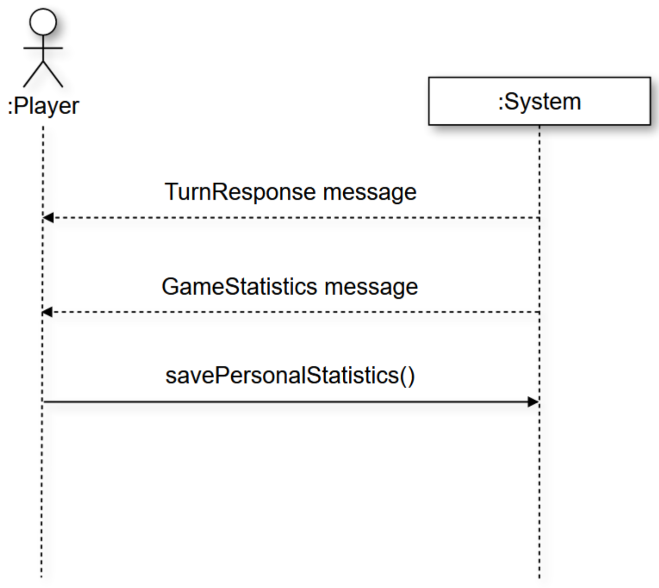
Main Success Scenario

2.4.11 SSD for UC 4.1: Play Tutorial Mode



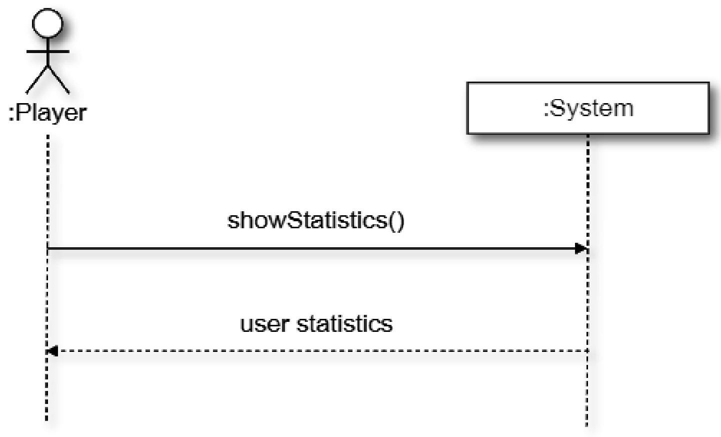
Main Success Scenario

2.4.12 SSD for UC 5.1: Display Game Statistic



Main Success Scenario

2.4.13 SSD for UC 5.2: Display User Statistic



Main Success Scenario

2.5 Operation Contracts

Operation	hostGame(player: Player)
References	UC 3.1: Host a Game
Preconditions	• Player is in main menu
Postconditions	• A server was created and runs on the local ip and game port • An AllPlayers list was created in GameState • The host was added to the player list • A lobby gui has been started for the player • The host's nickname and avatar appeared in game lobby

Operation	joinGame(player: Player, serverAdress: String)
References	UC 3.2: Join a Game
Preconditions	• Player is in main menu
Postconditions	• Player established a server connection via the client and server protocols • Player was added to AllPlayers list of the game lobby • Nickname and avatar of the player appeared in game lobby • A lobby gui has been started for the player • player´s statistic was loaded and associated with player

Operation	startGame(game: Game)
References	UC 2: Play Scrabble
Preconditions	• AllPlayers list consists of desired players • game.running is false • Game settings (timePerPlayer, maxScore, etc.) are set
Postconditions	• A game instance 'g' was initiated by the host according to game settings • A score board 'b' was created and associated with 'g' • A gameboard 'gB' was created and associated with 'g' • Dictionary 'd' has been associated with 'g' • game.running was set to true • Each player received 7 tiles on their racks

Operation	startTurn(player: Player)
References	UC 2.1: Make a turn
Preconditions	• player has same value as currentPlayer
Postconditions	• A turn 't' was created • The Player currentPlayer was associated with 't' • A Countdown timer 'c' associated with 't' was started

Operation	moveTile (oldX: int, oldY: int, newX: newY: int)
References	UC 2.1: Make a turn
Preconditions	• Countdown ‘c’ has time left • field ‘f1’ with coordinates (oldX, oldY) has been clicked • mouse has been dragged over to field ‘f2’ with coordinates (newX, newY) • mouse was released over field ‘f2’
Postconditions	• tile was removed at ‘f1’ • tile was added at ‘f2’

Operation	commitTurn(player: Player)
References	UC 2.1: Make a turn
Preconditions	• player has same value as currentPlayer • Player wants to end the turn voluntarily • Time left • Attribute isValid must be true for all Words
Postconditions	• wordScore has been calculated • Turn score was added to player's score on Scoreboard Number of tiles played in this turn by the player was checked • Operation drawTiles has been completed • currentPlayer was modified to the next player • countdown ‘c’ was reset

Operation	resetTurnForEveryPlayer(player: Player)
References	UC 2.1: Make a turn
Preconditions	• player left the Game AND is currentPlayer OR
Postconditions	• Turn has been reset and ended • Next player is allowed to perform their turn

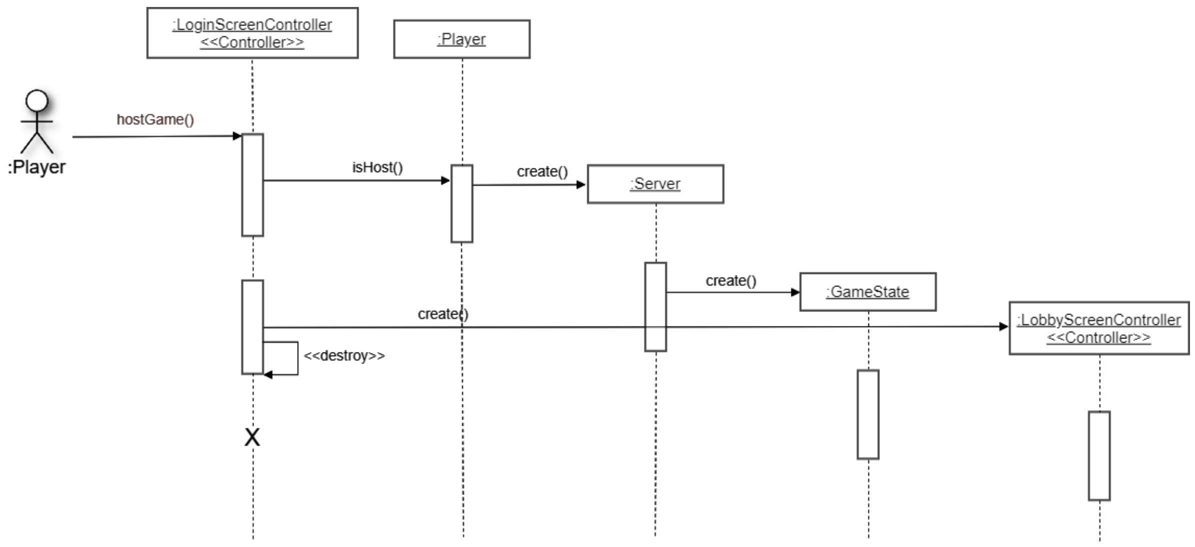
Operation	skipAndChangeTiles(tiles: List<Tile>)
References	UC 2.1: Make a turn
Preconditions	• player has same value as currentPlayer • time left
Postconditions	• Turn has been reset and ended • Selected tiles were removed from player’s rack and new ones were added • Next player is allowed to perform their turn

Operation	drawTiles(number: Int, currentPlayer: Player)
References	UC 2.1: Make a turn
Preconditions	• Turn from currentPlayer player was completed • Player owns less than 7 tiles after turn
Postconditions	• Number of tiles owned by player was updated to 7 (as long as there are enough tiles in the tile bag)

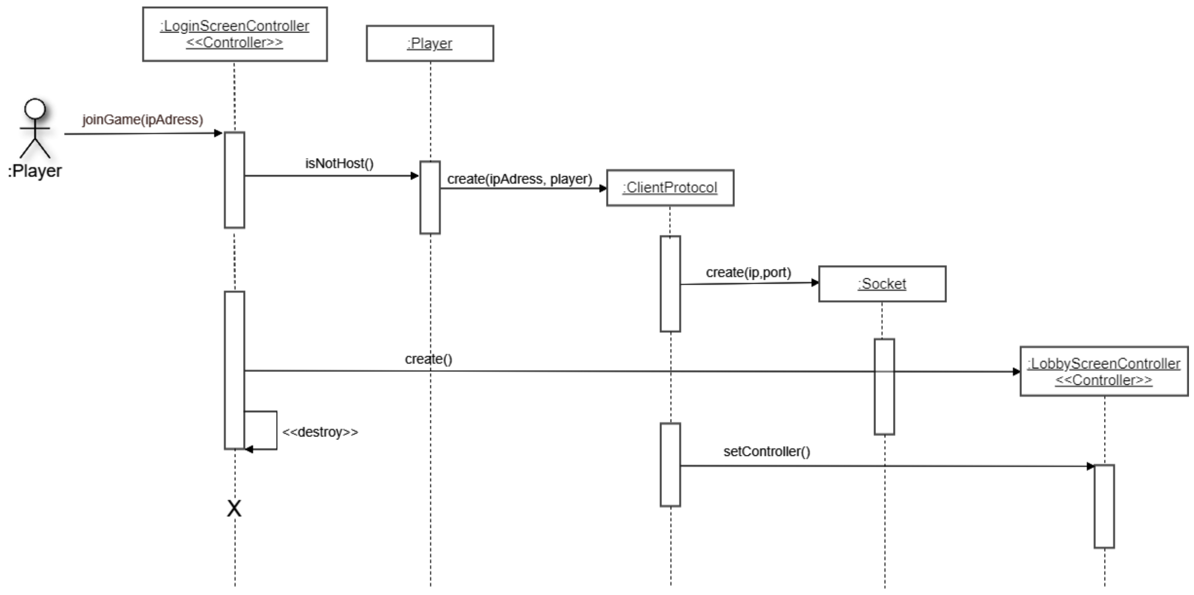
Operation	leaveGame()
References	UC 3.4: Leave Game
Preconditions	• game.running is true • timer is running
Postconditions	• game.running was set on false • player is removed from AllPlayers list • disconnect or shutdown message have been sent • timer is stopped

2.6 Sequence Diagrams

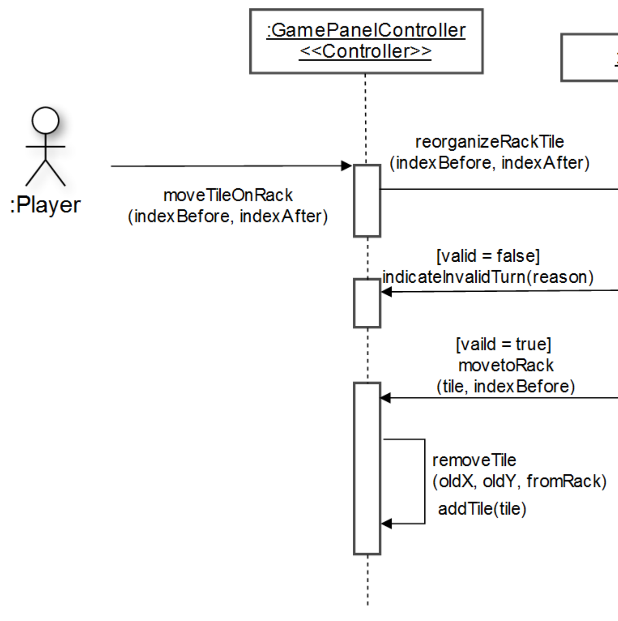
2.6.1 SD for hostGame()



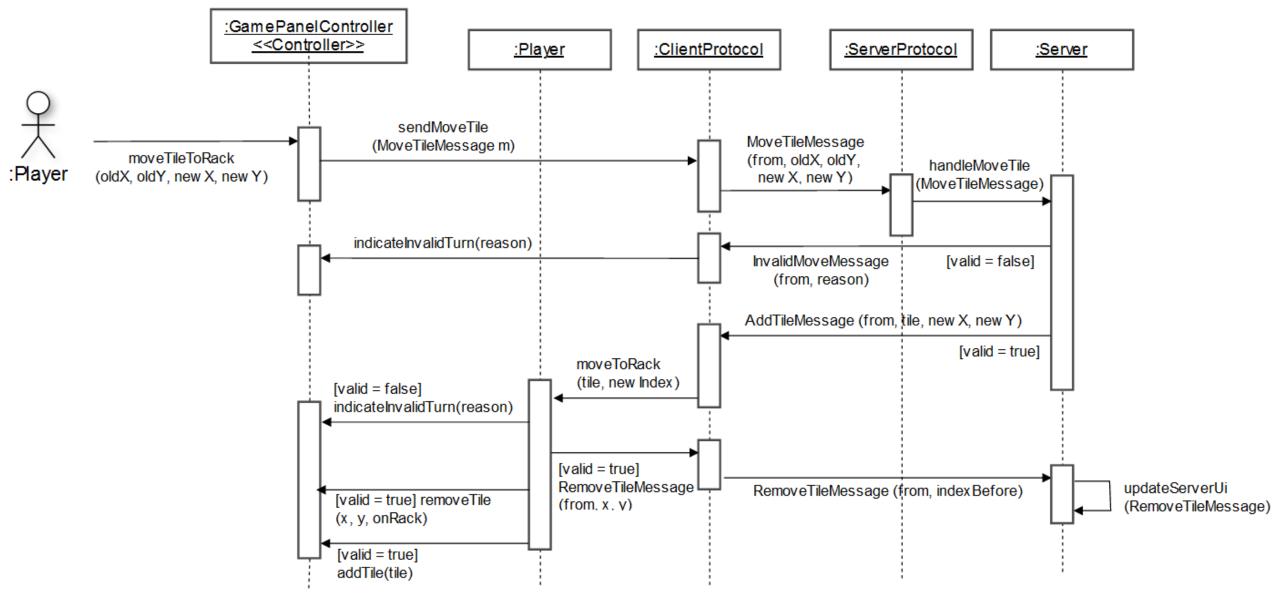
2.6.2 SD for joinGame()



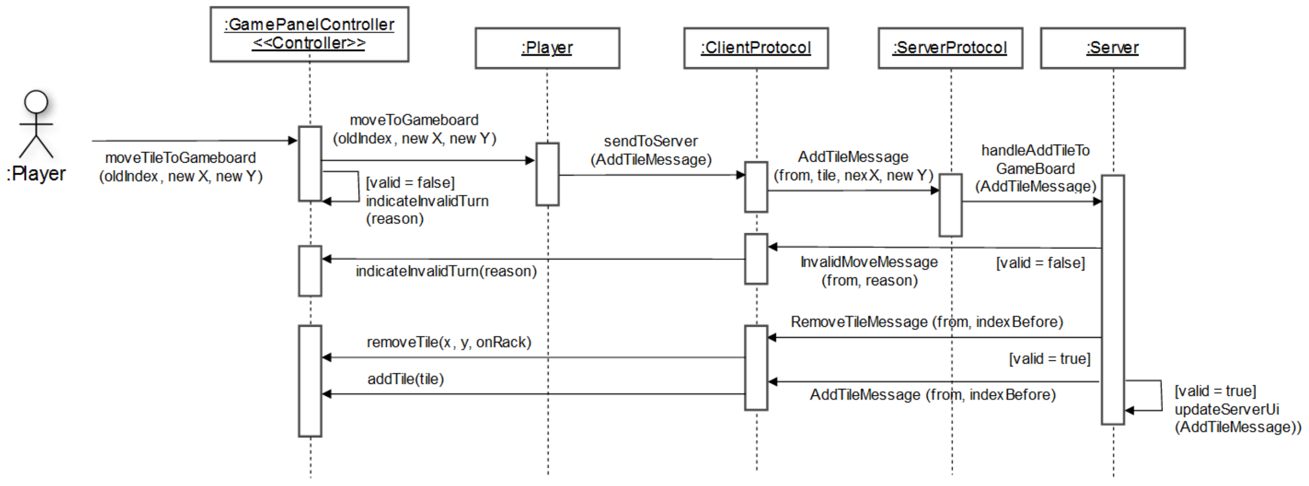
2.6.3 SD for moveTileOnRack()



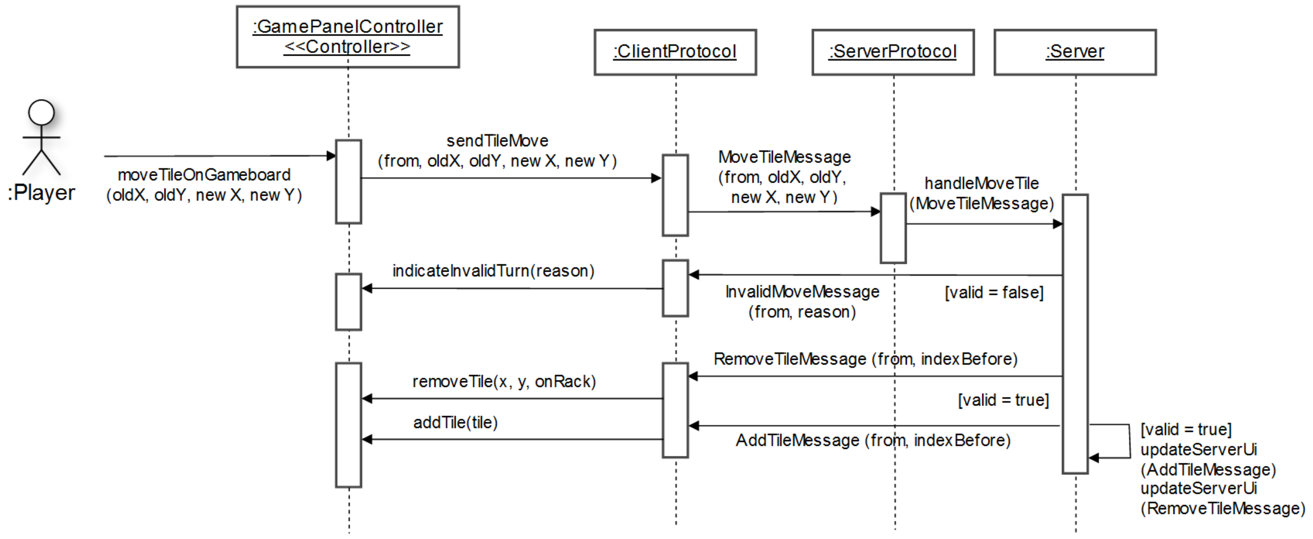
2.6.4 SD for moveTileToRack()



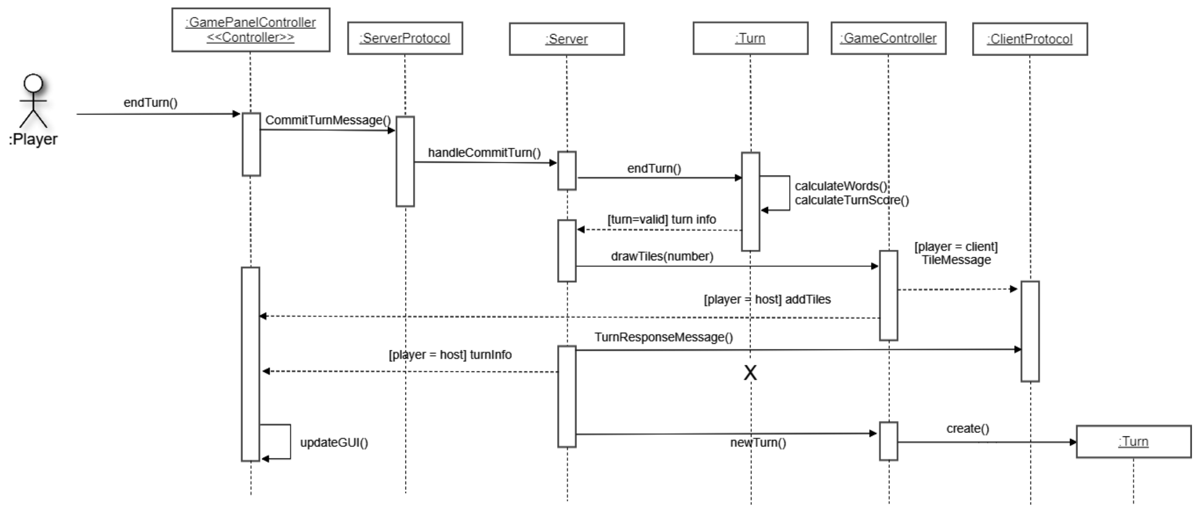
2.6.5 SD for moveTileToGameboard()


:GamePanelController <<Controller>>
:Player
:ClientProtocol
:ServerProtocol
:Server
:Player
moveTileToGameboard (oldIndex, new X, new Y)
moveToGameboard (oldIndex, new X, new Y)
[valid = false] indicateInvalidTurn (reason)
sendToServer (AddTileMessage)
AddTileMessage (from, tile, nexX, new Y)
handleAddTileTo GameBoard (AddTileMessage)
indicateInvalidTurn(reason)
InvalidMoveMessage (from, reason)
[valid = false]
RemoveTileMessage (from, indexBefore)
removeTile(x, y, onRack)
[valid = true]
addTile(tile)
AddTileMessage (from, indexBefore)
[valid = true] updateServerUi (AddTileMessage))

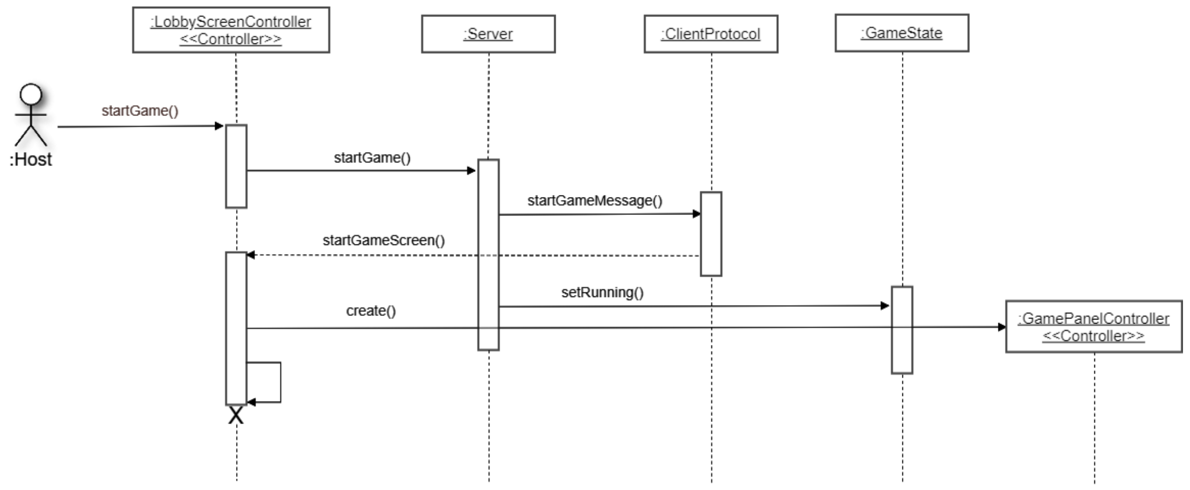
2.6.6 SD for moveTileOnGameboard()


:GamePanelController <<Controller>>
:ClientProtocol
:ServerProtocol
:Server
:Player
moveTileOnGameboard (oldX, oldY, new X, new Y)
sendTileMove (from, oldX, oldY, new X, new Y)
MoveTileMessage (from, oldX, oldY, new X, new Y)
handleMoveTile (MoveTileMessage)
indicateInvalidTurn(reason)
InvalidMoveMessage (from, reason)
[valid = false]
RemoveTileMessage (from, indexBefore)
removeTile(x, y, onRack)
[valid = true]
addTile(tile)
AddTileMessage (from, indexBefore)
[valid = true] updateServerUi (AddTileMessage) updateServerUi (RemoveTileMessage)

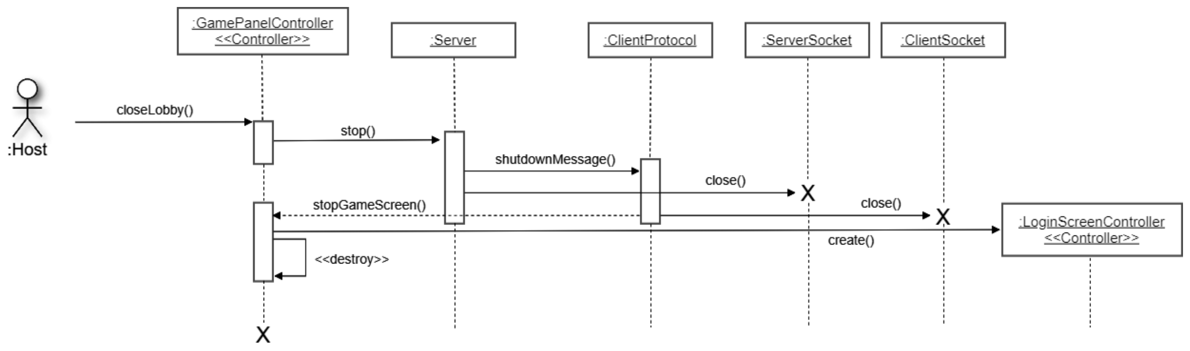
2.6.7 SD for endTurn()



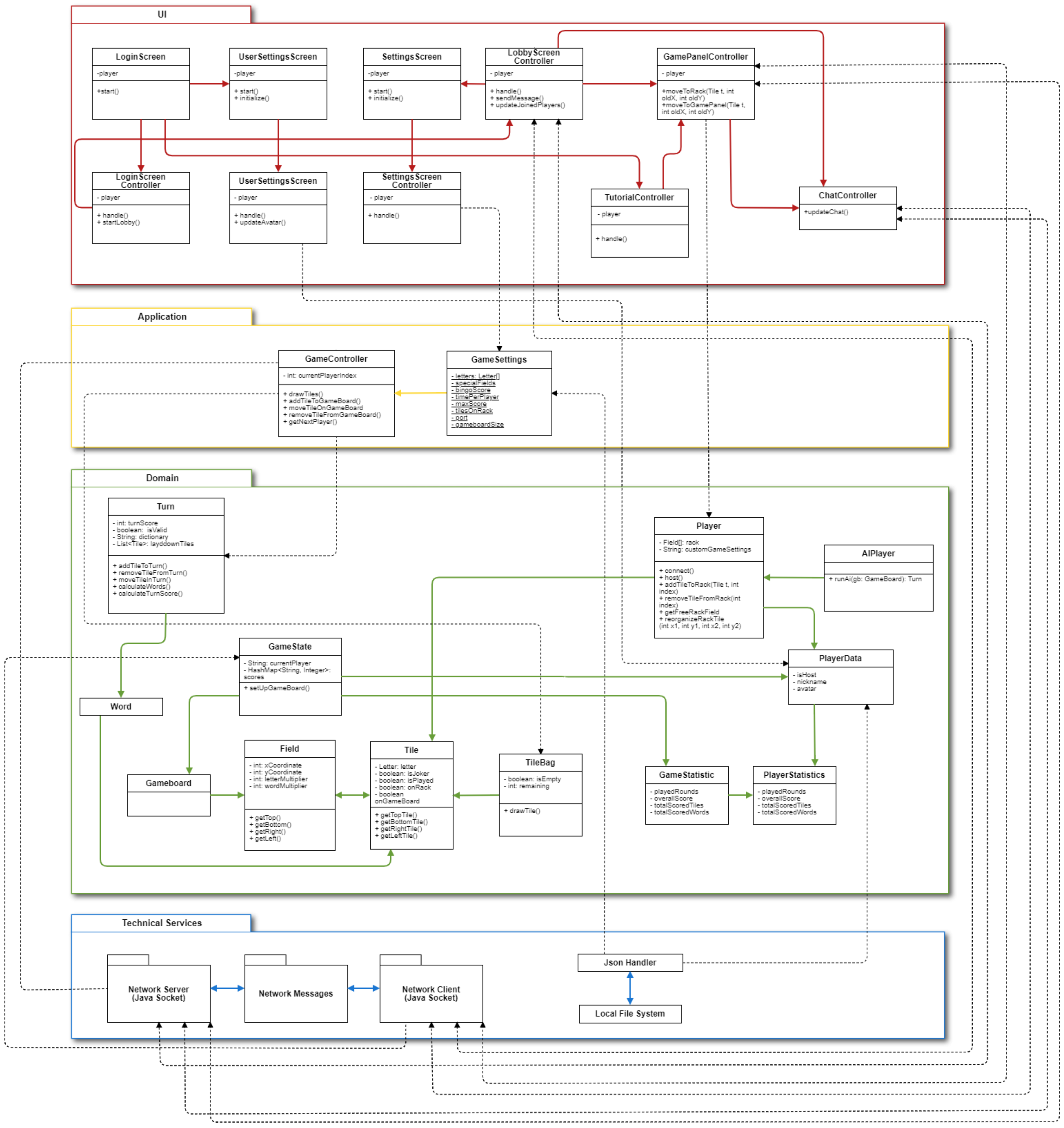
2.6.8 SD for startGame()



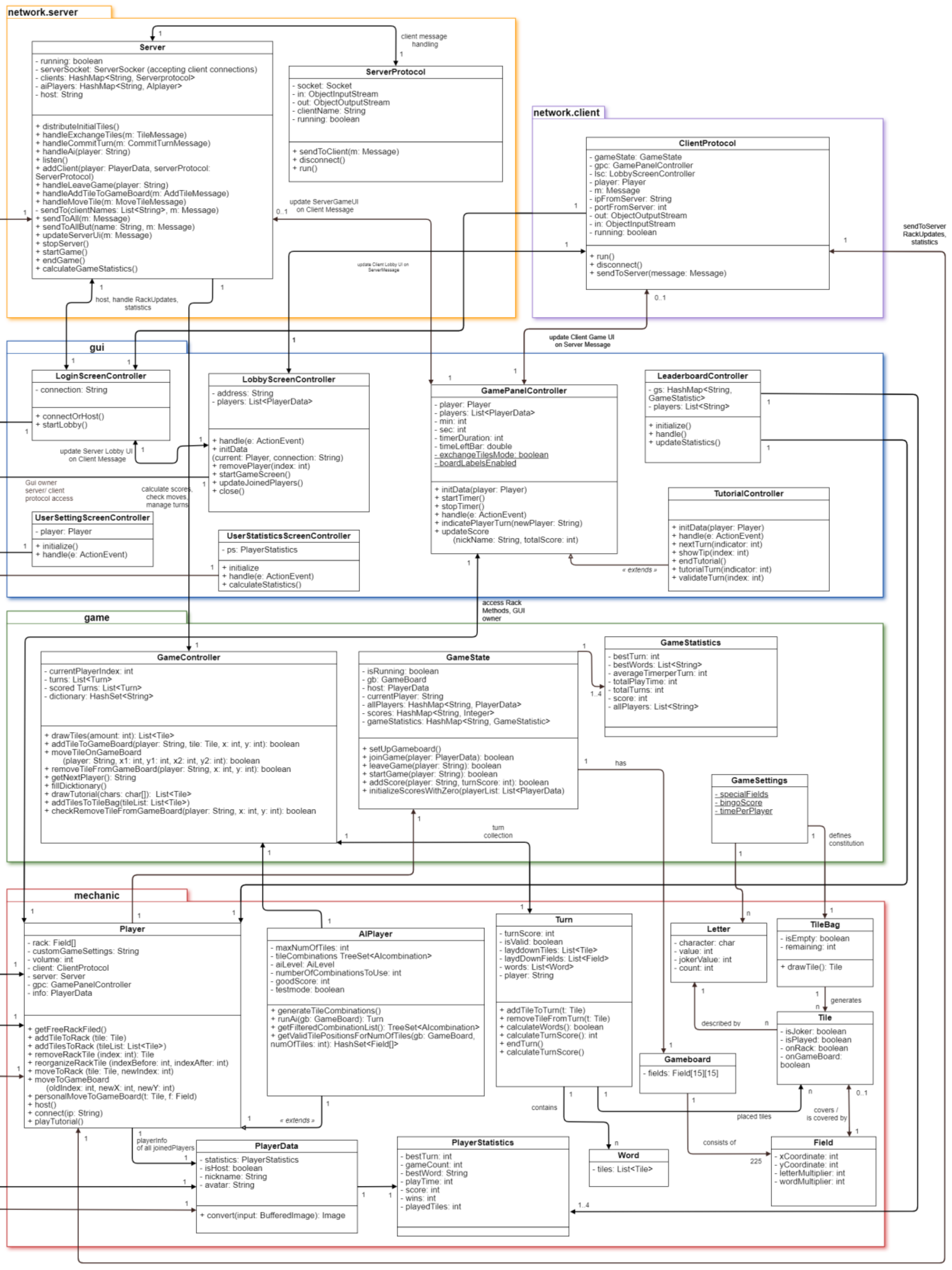
2.6.9 SD for closeLobby()



2.7 Architecture Diagram



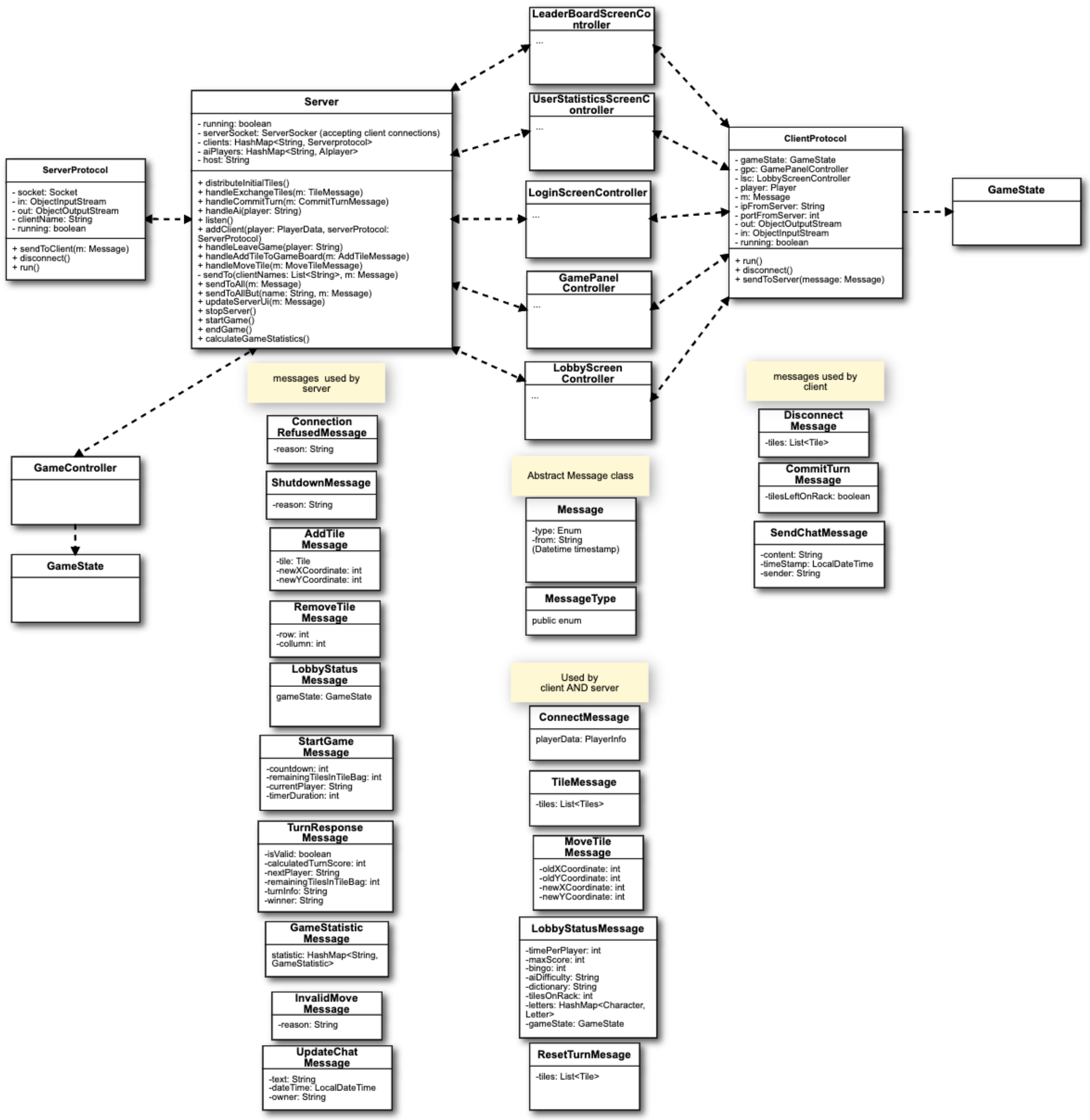
2.8 UML Diagram


network.server
Server
- running: boolean
- serverSocket: ServerSocker (accepting client connections)
- clients: HashMap<String, Serverprotocol>
- aiPlayers: HashMap<String, Alplayer>
- host: String
+ distributeInitialTiles()
+ handleExchangeTiles(m: TileMessage)
+ handleCommitTurn(m: CommitTurnMessage)
+ handleAi(player: String)
+ listen()
+ addClient(player: PlayerData, serverProtocol: ServerProtocol)
+ handleLeaveGame(player: String)
+ handleAddTileToGameBoard(m: AddTileMessage)
+ handleMoveTile(m: MoveTileMessage)
- sendTo(clientNames: List<String>, m: Message)
+ sendToAll(m: Message)
+ sendToAllBut(name: String, m: Message)
+ updateServerUi(m: Message)
+ stopServer()
+ startGame()
+ endGame()
+ calculateGameStatistics()
client message handling
ServerProtocol
- socket: Socket
- in: ObjectInputStream
- out: ObjectOutputStream
- clientName: String
- running: boolean
+ sendToClient(m: Message)
+ disconnect()
+ run()
update ServerGameUI on Client Message
update Client Lobby UI on ServerMessage
host, handle RackUpdates, statistics
network.client
ClientProtocol
- gameState: GameState
- gpc: GamePanelController
- lsc: LobbyScreenController
- player: Player
- m: Message
- ipFromServer: String
- portFromServer: int
- out: ObjectOutputStream
- in: ObjectInputStream
- running: boolean
+ run()
+ disconnect()
+ sendToServer(message: Message)
sendToServer RackUpdates, statistics
update Client Game UI on Server Message
gui
LoginScreenController
- connection: String
+ connectOrHost()
+ startLobby()
update Server Lobby UI on Client Message
Gui owner server/ client protocol access
calculate scores, check moves, manage turns
LobbyScreenController
- address: String
- players: List<PlayerData>
+ handle(e: ActionEvent)
+ initData (current: Player, connection: String)
+ removePlayer(index: int)
+ startGameScreen()
+ updateJoinedPlayers()
+ close()
GamePanelController
- player: Player
- players: List<PlayerData>
- min: int
- sec: int
- timerDuration: int
- timeLeftBar: double
- exchangeTilesMode: boolean
- boardLabelsEnabled
+ initData(player: Player)
+ startTimer()
+ stopTimer()
+ handle(e: ActionEvent)
+ indicatePlayerTurn(newPlayer: String)
+ updateScore (nickName: String, totalScore: int)
LeaderboardController
- gs: HashMap<String, GameStatistic>
- players: List<String>
+ initialize()
+ handle()
+ updateStatistics()
TutorialController
+ initData(player: Player)
+ handle(e: ActionEvent)
+ nextTurn(indicator: int)
+ showTip(index: int)
+ endTutorial()
+ tutorialTurn(indicator: int)
+ validateTurn(index: int)
« extends »
UserSettingScreenController
- player: Player
+ initialize()
+ handle(e: ActionEvent)
UserStatisticsScreenController
- ps: PlayerStatistics
+ initialize
+ handle(e: ActionEvent)
+ calculateStatistics()
access Rack Methods, GUI owner
game
GameController
- currentPlayerIndex: int
- turns: List<Turn>
- scored Turns: List<Turn>
- dictionary: HashSet<String>
+ drawTiles(amount: int): List<Tile>
+ addTileToGameBoard(player: String, tile: Tile, x: int, y: int): boolean
+ moveTileOnGameBoard (player: String, x1: int, y1: int, x2: int, y2: int): boolean
+ removeTileFromGameBoard(player: String, x: int, y: int): boolean
+ getNextPlayer(): String
+ fillDicktionary()
+ drawTutorial(chars: char[]): List<Tile>
+ addTilesToTileBag(tileList: List<Tile>)
+ checkRemoveTileFromGameBoard(player: String, x: int, y: int): boolean
GameState
- isRunning: boolean
- gb: GameBoard
- host: PlayerData
- currentPlayer: String
- allPlayers: HashMap<String, PlayerData>
- scores: HashMap<String, Integer>
- gameStatistics: HashMap<String, GameStatistic>
+ setUpGameboard()
+ joinGame(player: PlayerData): boolean
+ leaveGame(player: String): boolean
+ startGame(player: String): boolean
+ addScore(player: String, turnScore: int): boolean
+ initializeScoresWithZero(playerList: List<PlayerData>)
GameStatistics
- bestTurn: int
- bestWords: List<String>
- averageTimerperTurn: int
- totalPlayTime: int
- totalTurns: int
- score: int
- allPlayers: List<String>
has
GameSettings
- specialFields
- bingoScore
- timePerPlayer
defines constitution
turn collection
mechanic
Player
- rack: Field[]
- customGameSettings: String
- volume: int
- client: ClientProtocol
- server: Server
- gpc: GamePanelController
- info: PlayerData
+ getFreeRackFiled()
+ addTileToRack (tile: Tile)
+ addTilesToRack (tileList: List<Tile>)
+ removeRackTile (index: int): Tile
+ reorganizeRackTile (indexBefore: int, indexAfter: int)
+ moveToRack (tile: Tile, newIndex: int)
+ moveToGameBoard (oldIndex: int, newX: int, newY: int)
+ personalMoveToGameBoard(t: Tile, f: Field)
+ host()
+ connect(ip: String)
+ playTutorial()
AIPlayer
- maxNumOfTiles: int
- tileCombinations TreeSet<AIcombination>
- aiLevel: AiLevel
- numberOfCombinationsToUse: int
- goodScore: int
- testmode: boolean
+ generateTileCombinations()
+ runAi(gb: GameBoard): Turn
+ getFilteredCombinationList(): TreeSet<AIcombination>
+ getValidTilePositionsForNumOfTiles(gb: GameBoard, numOfTiles: int): HashSet<Field[]>
« extends »
playerInfo of all joinedPlayers
PlayerData
- statistics: PlayerStatistics
- isHost: boolean
- nickname: String
- avatar: String
+ convert(input: BufferedImage): Image
PlayerStatistics
- bestTurn: int
- gameCount: int
- bestWord: String
- playTime: int
- score: int
- wins: int
- playedTiles: int
Turn
- turnScore: int
- isValid: boolean
- layddownTiles: List<Tile>
- laydDownFields: List<Field>
- words: List<Word>
- player: String
+ addTileToTurn(t: Tile)
+ removeTileFromTurn(t: Tile)
+ calculateWords(): boolean
+ calculateTurnScore(): int
+ endTurn()
+ calculateTurnScore()
contains
Word
- tiles: List<Tile>
Letter
- character: char
- value: int
- jokerValue: int
- count: int
described by
TileBag
- isEmpty: boolean
- remaining: int
+ drawTile(): Tile
generates
Tile
- isJoker: boolean
- isPlayed: boolean
- onRack: boolean
- onGameBoard: boolean
Gameboard
- fields: Field[15][15]
placed tiles
consists of
covers / is covered by
Field
- xCoordinate: int
- yCoordinate: int
- letterMultiplier: int
- wordMultiplier: int

2.9 Additional Artefacts

2.9.1 Client Server UI Communication

This artefact was created during the developement process to ensure, that we as a team have a similar understanding of the client server communication. It does not claim to be correct with respect to UML conventions.



3 Implementation

3.1 IDE, Branching Model and Style Guide Conformity

As a team we decided to use Eclipse as our IDE of choice. All members were familiar with it and the given tutorials were provided for Eclipse only. We set up *eGit* and researched how to manage branches via the IDE.

To keep track of our style conformity we later installed Google style formatting and *Checkstyle Plug-in 8.39.0* to ensure our code continuously matches the style guide.

We use GitLab branches to avoid merging conflicts when pulling. We work on a develop branch only pushing working versions to the master branch for statistics. We branch personal feature branches off the development branch to make full use of GitLab's versioning capabilities. Old branches once being merged are deleted from time to time if no longer needed.

(Derived from https://nvie.com/posts/a-successful-git-branching-model/)

3.2 Short description of used libraries

By now, we used the following libraries, which are being managed by Apache Maven.

3.2.1 Apache Maven (Version 3.8.1)

A project-build automation tool. Manages build paths, installation of plug-ins and libraries and supports reproducible builds.
Ensures the same library-versions and the correct JRE 11 on all machines.

3.2.2 Jackson (Version 2.12.2)

Jackson is used as library for Json imports and exports on the technical layer. The object mapper is an easy to use Class for that purpose.

3.2.3 Java-FX (Version 11.0.1)

Framework for GUI applications in java. It is used on the UI layer.

3.2.4 JUnit (Version 4.13.2)

Framework for repeatable unit-tests. It is used for testing our classes during development and before building a runnable version of the game.

3.2.5 FontAwesomeFX9 (Version 9.2.1)

Extension to JavaFX. Provides open-source-icons for usage in GUI on UI layer.