Requirements and Analysis Document for Zcrabble by Scrabblers

Niklas Axelsson, Martin Björklund, Ole Fjeldså, Gustaf Jonasson
Date: 2021-10-24
Version: Final.

1. Introduction

Zcrabble is a multi-platform game for Windows, MacOS and Linux based on the classic board game Scrabble. Zcrabble aims to make the processes of setting up and getting started with Scrabble games easier. By letting the user's computer do the boring calculations they are free to only think about what to play next. The application enables the user to play with their friends locally and to play against a computer opponent to hone their skills.

The Zcrabble application is being developed by the authors of this document with supervision from Pelle Evensen.

1.1. Definitions, acronyms and abbreviations

This section outlines a number of Zcrabble specific terms.

Play: In the context of the game, a play is the action of placing a word on the board and confirming it by ending one's turn.

Cell/Square: Cells or squares are used interchangeably and refer to the spaces on the board segmented into x*x sizes depending on the board. Specifically a square is a single one unit of space where a letter tile is placed. For example a 2 by 2 size board would fit 4 squares.

Special cell/square: Some cells/squares are worth additional points or have other modifiers when a tile is placed on them.

Board: A board is the total collection of all squares.

Letter tile: A letter tile or shortened to "tile", fits on a cell/square and has a single character belonging to whichever alfabet the dictionary is made up of. A blank tile is also a valid tile. A tile also has a score associated with each letter.

Rack: A collection of letter tiles that belong to the player and are the tiles the player can choose to play during their turn. A player may also

choose to exchange any number of tiles from the rack for an equal amount of new ones from the letter tile bag.

The bag: A container used by players to fill the rack with new letter tiles at the start of their turns(?). The game is over when the bag is empty and one player's rack is empty.

Word: Specifically in the context of the game, a game "word" is any series of letters formed by letter tiles which can be found in whichever dictionary or sets of words the game uses to reference. A word can be read vertically or horizontally but not diagonally. A word is always read from left to right or from up to down.

Score: Each player has a score associated with them which is their total accumulation of points by creating words over the course of the game. The player with the highest score wins.

Player: A human or computer (bot) who is playing the game.

2. Requirements

2.1. User stories

2.1.1. Scrabble game

Identifier: 01
Priority: 1
Status: Implemented

As a Scrabble enjoyer, I need a Scrabble video game to practice playing Scrabble in order to get better.

2.1.2. Visible letter tiles

Identifier: 02
Priority: 1
Category: View
Status: Implemented

As a player, I want to see my own letter tiles.

Confirmation

- Functional
  - Can I see my own letter tiles?
  - Can I see the letter tiles of my opponent? I should not be able to.

Tasks

- Graphical representation of a tile.
- Graphical representation of a rack.
- Draw tile and rack.
- Model tile class.
- Model rack class.

2.1.3. Place letter tiles

Identifier: 03
Priority: 1
Category: View
Status: Implemented

As a player, I need to be able to place my letter tiles.

Confirmation

- Functional
  - Can I place my letter tiles?
  - Can I only place tiles on non-occupied spots?
  - Can I only place tiles on my turn?
  - Can I only place tiles adjacent to previous tiles?
  - Can I only place the first tiles in the middle?
  - Can I only place my tiles in one direction?
  - Do the tiles I place need to form a word?

Tasks

- Enable moving tiles from rack
- Enable placing moved tile on board
- Add checks to see if valid placement

2.1.4. Take turns

Identifier: 04
Priority: 1
Category: Model
Status: Implemented

As a player, I want to take a turn, so that I can try to win the game.

Confirmation

- Functional
  - When my turn is done, is it the next player's turn?
  - I cannot take two turns in a row.
  - Can I skip my turn if I have no available words?

Tasks

- Add turn structure (what is a turn?)
- Add turn rotation logic to Game(or new class?)
- Enable passing the turn
- Add GUI to end the turn

2.1.5. Knowing whose turn it is

Identifier: 05
Priority: 1
Category: Model
Status: Implemented

As a player, I need to know whose turn it is so that I know the state of the game.

Confirmation

- Functional
    - Is it clear whose turn it is?

- Non-functional
    - Is it especially clear when it is my turn?

Tasks

- Add some GUI to show whose turn it is.
- Add some GUI to show when it is the player's turn.

2.1.6. Menus

Identifier: 06
Priority: 1
Category: View
Status: Implemented

As a player, I need an easy to use menu system/start menu so that I can easily start a new game (in my preferred game mode).

Confirmation

- Functional
    - Are there buttons in the GUI?
    - Do the buttons do what they are supposed to?
- Non-functional
    - Are the buttons distinct?
    - Are the buttons clearly marked?
    - Is the interface visually clear?

Tasks

- Make a menu layout.
- Implement menu in scenebuilder.
- Connect menu with model.

2.1.7. Marked cells

Identifier: 07
Priority: 1
Category: View
Status: Implemented

As a player, I need the cells to clearly indicate if they possess any special properties.

Confirmation

- Non-functional
  - Do the cells contain clear information about their special properties?
  - Are the special cells visually distinguishable?
  - Are the visual differences between the types of cells perceivable to people who are visually impaired?

Tasks

- Model
  - Implement the different cell properties into the model
- GUI
  - Create images of the special cells and place them on the board as imageviews.

2.1.8. Visible board

Identifier: 08
Priority: 1
Category: View
Status: Implemented

As a Player, I need to be able to see the board so that I can make decisions about how to play the game.

Confirmation

- Functional
  - Can I see the board?
  - Does the board show the state of the game?

Tasks

- Model
    - Create a model representation of the board.

- GUI
    - Create a visual representation of the board. Divide responsibilities between controller and view.

2.1.9. Dictionary

Identifier: 09
Priority: 1
Category: Model
Status: Implemented

As a player, I need the game to use a dictionary so that the game will be correct and fair.

Confirmation

- Functional
    - Is the dictionary up to date and correct?
    - Does the game find the words correctly?
    - Does the dictionary refuse incorrect words?
- Non-functional
    - Is the dictionary algorithm fast enough?

Tasks

- Find a suitable dictionary.
- Create an algorithm for finding words.
- Implement functionality that rejects words that are incorrect.

2.1.10. Remaining letter tiles

Identifier: 10
Priority: 1
Category: View
Status: Implemented

As a player I need to see the number of letter tiles left so I can make strategic plans based on my chances to get a specific letter.

Confirmation

- Functional
    - Is the number of remaining tiles clearly displayed?

Tasks

- Implement a TileBag class. The implementation can create a number of letter tiles, randomise their order and put them in a queue data structure.
- Using the observer pattern, update the remaining letter tiles in the view.

2.1.11. Score keeping

Identifier: 11
Priority: 2
Category: View
Status: Implemented

As a player, I need to see everyone's score, so that I know the state of the game.

Confirmation

- Functional
  - Can I see the opponent's score?
  - Can I see my own score?
- Non-functional
  - Are the scores presented in a visually pleasing way? (we talked about adding a bar to visualize the score, kind of like in Chess)

Tasks

- Create a class to represent the player.
- Inside the player class, use a variable to keep track of score.
- Update the score in the view using the observer pattern.

2.1.12. Scrabble bot

Identifier: 12
Priority: 2
Category: Model
Status: Implemented

As a player I want a bot to practice against on my own.

Confirmation

- Functional
  - Can the bot find words?
  - Can the bot create actual words and place tiles on the board?

- Is the bot too slow?
- Can the bot take its turn?

Tasks

- Create a bot class.
- Make the bot class utilise the word finding algorithm in the dictionary class in order to create words.

2.1.13. Shuffle

Identifier: 13
Priority: 2
Category: Model/View
Status: Implemented

As a player I want to be able to shuffle my rack around to help me see words I might play.

Confirmation

- Functional
  - Is there a way to start a shuffle of the rack?
  - Does the rack get shuffled?
- Non-functional
  - Do the visuals get updated when the rack is shuffled?

Tasks

- Write a shuffling algorithm.
- Connect a button to the shuffle method.
- Update visual when rack gets shuffled.

2.1.14. Swap tiles

Identifier: 14
Priority: 2
Category: Model/View
Status: Implemented

As a player I want to be able to exchange tiles on my rack for new ones instead of taking my turn.

Confirmation

- Functional
  - Does the model remove the tiles from the player's rack?
  - Does the model add new tiles to the rack?
  - Do the removed tiles get added to the tilebag?

        - Is the player's turn ended by swapping tiles?
    - Non-functional
        - Are the tiles visually replaced on the screen by that player's next turn?

Tasks

- Create a menu in the GUI that lets the player swap tiles.
- Write a method that adds the swapped tiles to the tile bag.

2.1.15. Return tiles to rack.

Identifier: 15
Priority: 2
Category: Model/View
Status: Implemented

As a player I want to conveniently get all my placed tiles thrown back to my rack when I change my mind about a word, or if it was incorrect.

Confirmation

- Functional
    - Do all of the tiles get returned to the rack?
    - Are they put back at free positions in the rack?
- Non-functional
    - Are the visuals updated, and the visual tiles removed from the board and shown on the rack?

Tasks

- Write a method for getting free rack positions.
- Write a method for returning tiles to the rack.
- Update the visuals depending on how the board/rack now is.

2.1.16. Pliancy

Identifier: 16
Priority: 2
Category: View
Status: Implemented

As a player I want clear feedback from the buttons and tiles when I interact with them.

Confirmation

- Functional

        - Do the buttons give feedback when clicked or hovered over?
        - Do cells and tiles give feedback when clicked or hovered over?
    - Non-functional
        -

Tasks

- Make tiles follow the cursor when placing them on the board.
- Make buttons change color when hovering over them.
- Change the opacity of tiles when they are marked.

2.1.17. Scalable scrabble game

Identifier: 17
Priority: 2
Category: Model/View
Status: Implemented

As a Scrabbler with an 8k monitor I need to be able to resize the window so that it is still visible at high resolutions.

Confirmation

- Functional
    - Can I resize the window?
- Non-functional
    - Do the elements in the window keep their shape?

Tasks

- Implement a class that checks for scaling and make sure all elements in the window are scaled correctly.
- Implement scaling presets in the menu.

2.1.18. Simple online multiplayer

Identifier: 18
Priority: 2
Category: Model
Status: Not implemented

As a player, I want to play online against my friends.

Confirmation

- Functional
    - Can I connect to my friends over the internet?
    - Can we play the game live?
- Non-functional

- Is it easy to set up online multiplayer?
- Is there any anti-cheat?

Tasks

- Research networking/tcp/sockets
- Create message system
- Message serializer/deserializer
- Implement what messages to send
- Implement what to do when receiving messages

2.1.19. Bot settings

Identifier: 19
Priority: 3
Category: Model
Status: Not implemented

As a beginner player I need different level bots so that I can incrementally train up my abilities.

Confirmation

- Functional
- Non-functional

Tasks

- Add two more bots with different functionality than the original one.

2.1.20. Ranked online multiplayer

Identifier: 20
Priority: 3
Category: Model
Status: Not implemented

As a competitive player, I need to be able to play online against strangers at my level so that I can be challenged.

Confirmation

- Functional
- Non-functional

2.1.21. Simple tutorial

Identifier: 21
Priority: 3
Category: View
Status: Implemented

As someone who has never played scrabble before i need i ‘ quick walk through of how to play the game.

Confirmation

- Functional
  - Is there a tutorial on how to play scrabble?
- Non-functional
  - Is the tutorial easy to find?

2.1.22. Multiple languages

Identifier: 22
Priority: 3
Category: Model
Status: Not implemented

As a bilingual player, I want different languages so that I can play with my friends from all over the world.

Confirmation

- Functional
  - Do the menus support multiple languages?
  - Does the game have dictionaries of multiple languages?
  - Is there a menu option for switching between languages?
  - Do all dictionaries account for every possible word (e.g. the possibility of putting words together to create new words in all scandinavian languages)
- Non-functional
  - Is it easy to switch languages?
  - Is it easy to synchronice language choice with another player?

Tasks

- Find multiple dictionaries.
- Implement a way to switch between dictionaries.
- Implement a way to synchronice dictionaries between different players online.

2.1.23. Speed mode

Identifier: 23
Priority: 3
Category: Model
Status: Sadly not implemented

As a chess player, I want a fast (chess clock) game mode to ramp up the intensity of the game.

Confirmation

- Functional
    - Is there a “chess clock” in the game that pauses when a play is made.
- Non-functional
    - Is it easy to place tiles fast? Maybe with number and arrow keys.

Tasks

- Add a clock class.
- Create timed gamemode
- Might need to change how turns work

2.1.24. Good looking GUI

Identifier: 24
Priority: 3
Category: View
Status: Beautifly implemented

As a graphic designer, I want the game to look good so that it is soothing to look at.

Confirmation

- Functional
- Non-functional
    - Is the game beautiful to look at?
    - Is the color palette harmonic?
    - Is the shape of the game nice and square?

Tasks

- Create layout in Figma.
- Export elements from figma to resources.
- Populate Board with elements.

2.1.25. In-game chat

Identifier: 25
Priority: 3
Category: View
Status: Not implemented

As a player, I want to be able to communicate with my opponent using an in game chat function.

Confirmation
- Functional
    - Is it possible to chat with opponents

2.2. Definition of Done

- The user stories satisfy their requirements, both functional and non-functional.
- Code is well documented, public methods with javadoc, private methods have a comment that explains what it does.
- Code is readable, with clear parameter names and comments where they are needed.
- Code is tested with JUnit.
- Codebase utilises a version control system, e.g. Git.

2.3. User interface

The user interface was made using JavaFX and SceneBuilder. One fxml file is used to represent the menu bar and another fxml file is used to represent the rest of the game (i.e board, rack, scores). The fxml file containing the menu bar is injected into the other fxml file in order to create the full GUI.

Figure 1 shows an early draft of the GUI made using Paint. The colored cells indicate that these cells possess special properties, such as doubling the score of a word or letter.

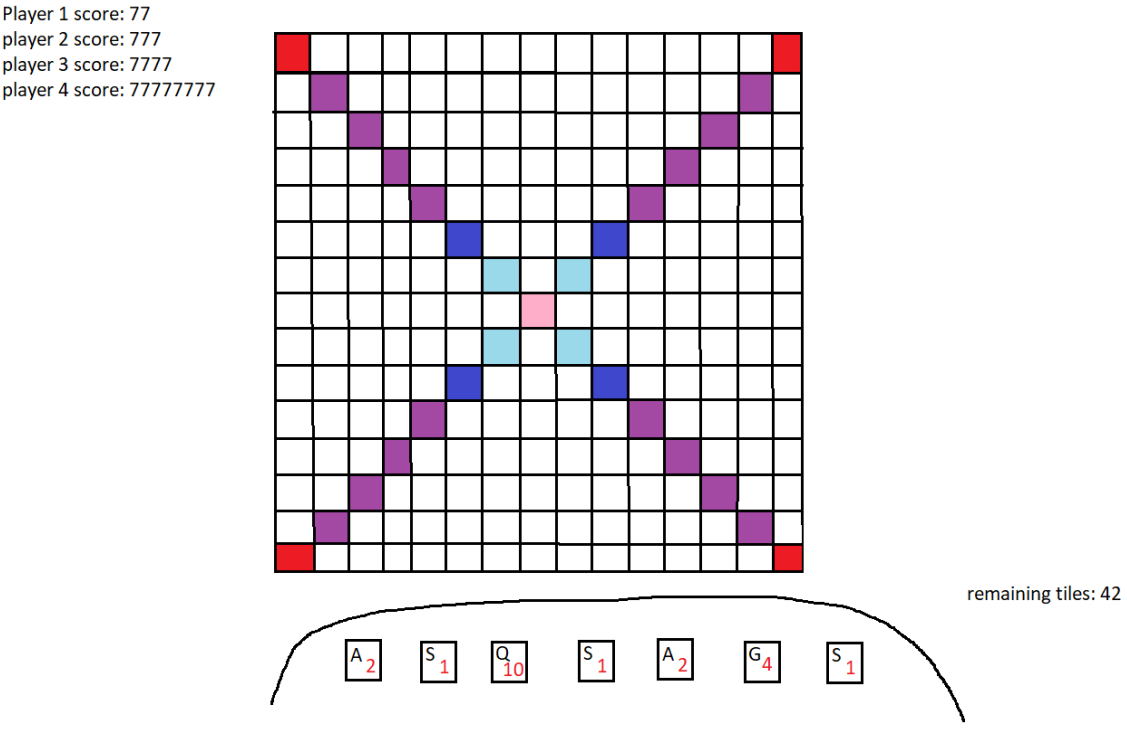

Figure 1: Early sketch of the basic GUI.

Figure 2 displays the finished start screen of Zcrabble. The background is the iconic green Zcrabble color. Pressing the start button hides the start screen and the user is met with the main view, i.e. a Zcrabble board, as seen in figure 3. By default, a game with one human player and one bot is started when the application launches.

Figure 2: The start screen.

The GUI consists of various buttons and labels that convey information to the user. A menu bar is located at the top of the application. It contains menus that let the user start a new game, change the size of the application, view the rules of Zcrabble (see figure 7), and change the theme. The GUI is dominated by the Zcrabble board, a 15 by 15 grid of cells on which tiles can be placed. Much like in the sketch in figure 1, the different colored cells indicate that these cells possess special properties. Underneath the board can the rack of the current player be found. The rack contains seven tiles that can be selected and moved onto the board.

To the right of the rack are four buttons that the player can interact with during his or her turn. The shuffle button shuffles the rack in order to help the player find words. The end turn button simply ends the player's turn. Should the player attempt to play an invalid word, the game will inform the player of this by showing a modal panel, as can be seen in figure 4.

The swap button lets the player swap their tiles should they desire. Swapping tiles counts as a turn. When the swap button is pressed, the game brings up a modal panel, seen in figure 8. In order to swap tiles, the player simply selects which tiles they want to swap and then presses the confirm button.

The return button returns all tiles that the player has placed on the board during their turn.

To the right of the board there are a number of labels that show important information about the state of the game. The P1 and P2 labels indicate how many points each player has. Further, the current player is marked in red. Under the score labels there is an image with a number inside. This number is meant to indicate how many tiles are left in the tile bag. Knowing how many tiles are left creates possibilities for interesting strategies.

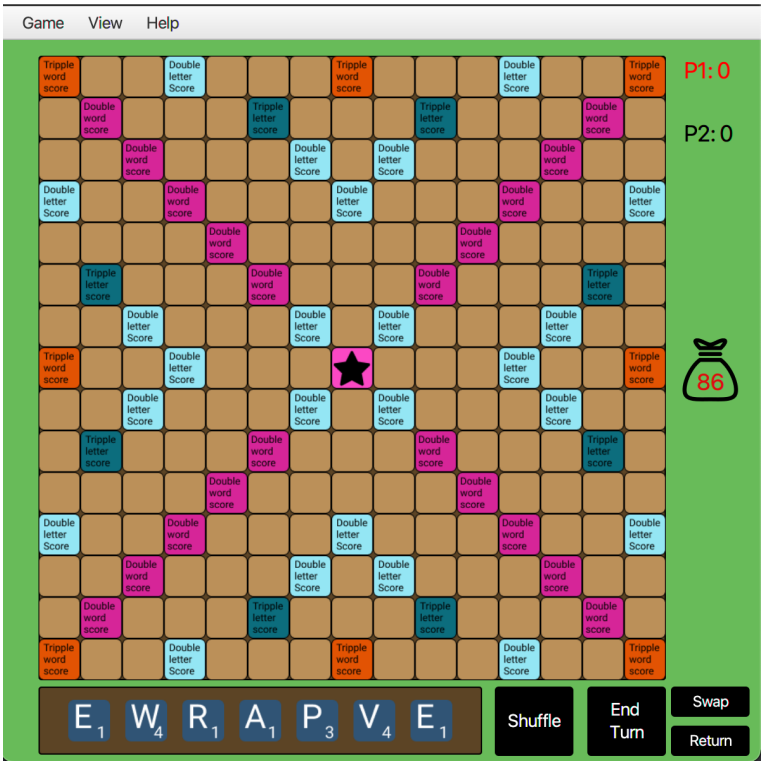

Figure 3: Finished GUI.

Figure 4: Modal panel when an invalid word is played.

The user can start playing by selecting a tile in his or her rack and moving it onto a desired cell on the board. Figure 5 shows what the game might look like after a few rounds of play. Figure 5 also showcases one of the available Zcrabble skins - Cyberpunk!

Figure 5: How the GUI looks in the middle of a game of Zcrabble.

The new game menu contains two spinners that allow the user to choose how many players and bots should participate in the next game. Pressing begin starts a new game, given that at least two players and/or bots have been selected.

Figure 6: The menu from which the player can start new games.

Figure 7: A modal panel showing the rules of Zcrabble.

Figure 8: A modal panel that lets the player swap tiles.

3. Domain model

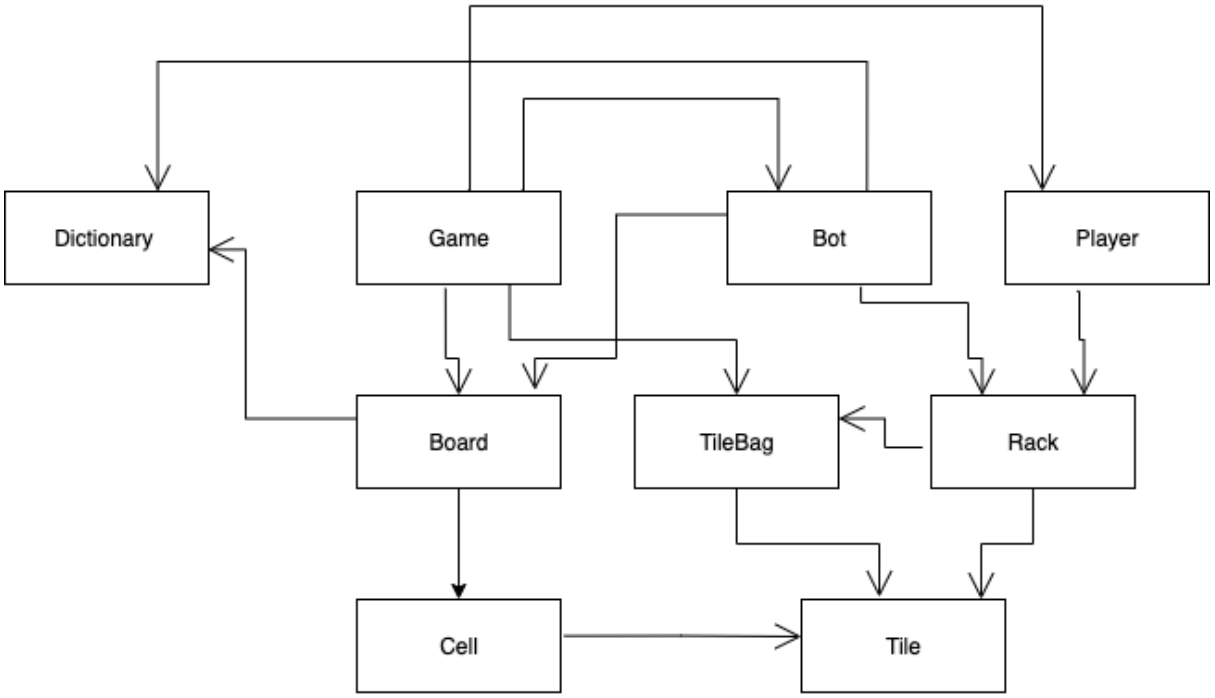

Figure 9: Domain model.

3.1. Class responsibilities

Game: Responsible for keeping track of the state of the game and has methods that change the state of the game. So it keeps track of all the players, how to end a turn and proceed to the next player's turn, it can call methods on the boards to influence them, etc. It functions as an outward face and clients call methods in it to influence the model.

Board: Has a collection of all the cells that make up a board and ways to change the board state, check if the board is valid and calculate the score.

Cell: Represents a container on the board that could hold a tile that's been placed. Also keeps track of the cell's score multipliers.

Tile: A data container that represents a tile that can be played, it needs to keep track of the letter and score value of the tile.

TileBag: Models the bag full of tiles that gets drawn from, represented by a queue in the tilebag class. Its responsibility is to hold the queue and have ways of interacting with it.

Rack: A size seven collection held by each player responsible for holding that player's usable tiles, it has methods for manipulating them.

Player: A human player, used to keep track of that player's rack and score. And has ways to modify the rack and score.

Bot: A computer player that in addition to keeping track of it's rack and score also is able to find a word that is valid and place it during its turn.

Dictionary: Loads a dictionary to use and applies it to words to check for correctness.

4. References

- Zcrabble rules can be found in the resources directory.
    - Zcrabble has some rule changes from the original Scrabble.
        - There are no wildcards in Zcrabble, instead there is an extra "Z" and "Q".
        - It is not possible to challenge other players' words in Zcrabble. Instead, the game automatically checks every play and refuses invalid words.

        - If an incorrect play is made, the player does not lose their turn but is instead allowed to make another play or skip their round.
- This codebase uses JavaFX to represent the application graphically.
- Maven is the build automation tool of choice.
- This codebase supports JDepend.
    - Run “mvn site” then “open target/site/jdepend-report/html” to read the report.

Systems Design Document for Zcrabble by Scrabblers

Niklas Axelsson, Martin Björklund, Ole Fjeldså, Gustaf Jonasson
Date: 2021-10-24
Version: Final

1 Introduction

This document aims to give an overview of how the Zcrabble game works. It contains an explanation of Zcrabble specific terms, system architecture and system design. Further, the document also contains information about dependencies and how data management is handled.

Zcrabble is a game very reminiscent of the popular game Scrabble. It is designed to be played by two to four players. Players take turns creating words based on a few letter tiles that the players have at their disposal. Points are awarded, based on a set of rules, after each turn. When the game ends, the player with the most points is victorious.

1.1 Definitions, acronyms, and abbreviations

This section outlines a number of Scrabble specific terms.

Play: In the context of the game, a play is the action of placing a word on the board and confirming it by ending one's turn.

Cell/Square: Cells or squares are used interchangeably and refer to the spaces on the board segmented into x*x sizes depending on the board. Specifically a square is a single one unit of space where a letter tile is placed. For example a 2 by 2 sized board would fit 4 squares.

Special cell/square: Some cells/squares are worth additional points or have other modifiers when a tile is placed on them.

Board: A board is the total collection of all squares.

Letter tile: A letter tile or shortened to “tile”, fits on a cell/square and has a single character belonging to whichever alphabet the dictionary is made up of. A blank tile is also a valid tile. A tile also has a score associated with each letter.

Rack: A collection of letter tiles that belong to the player and are the tiles the player can choose to play during their turn. A player may also choose to exchange any number of tiles from the rack for an equal amount of new ones from the letter tile bag.

Tile bag: A container used by players to fill the rack with new letter tiles at the start of their turns. The game is over when the bag is empty and one player's rack is empty.

Word: Specifically in the context of the game, a game "word" is any series of letters formed by letter tiles that can be found in whichever dictionary or sets of words the game uses to reference. A word can be read vertically or horizontally but not diagonally. A word is always read from left to right or from up to down.

Score: Each player has a score associated with them which is their total accumulation of points by creating words over the course of the game. The player with the highest score wins.

2 System architecture

At startup, JavaFX initialization is called and the start screen is opened. After the start button is pressed the main window is opened and a game is launched. The user can then decide to play the default game (one human player versus one bot) or create a new game from the menu at the top. Creating a new game communicates from the controller to the model to start a new game.

When a new game is started, the tiles to use and what board to use is read from text files. The "game loop" starts here and each player takes their turn placing tiles on the board and then pressing end turn to pass the turn to the next player. At the end of each turn, the current player draws up to 7 tiles on their rack from the tile bag. When tiles are placed on the board they are put in a temporary board in the model. Once the player ends their turn the model either accepts the new word and places it on the permanent board, gives out a score and passes the turn or denies it and the current player can try again. The player can also choose to exchange up to 7 of their tiles with new tiles from the tile bag, this ends that player's turn. All these user interactions (pressing buttons, moving tiles) are noticed by the controllers and communicated to the model.

Once the tile bag runs out of tiles and a player has used up their rack or all the players pass their turn without a word for 7 consecutive turns the game ends. The user can then choose to start a new game from the menu or quit the game.

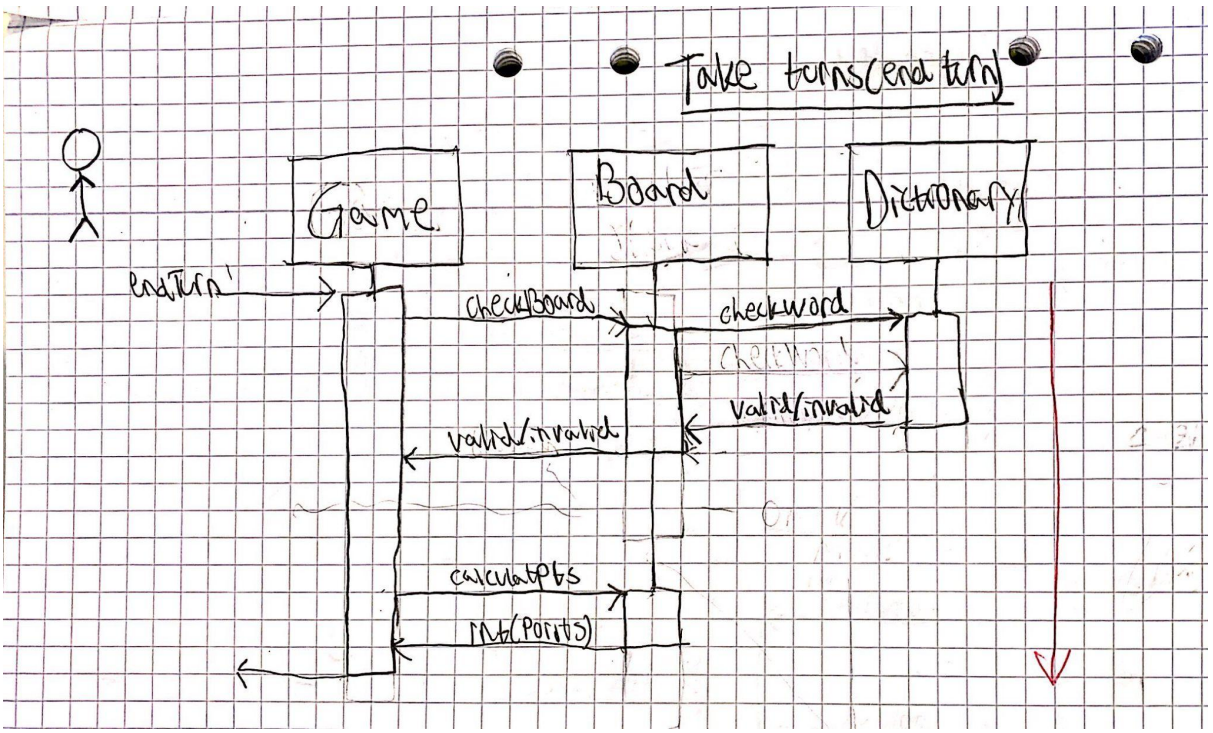

Figure 1: Sequence diagram visualizing the "Take turns" user story. This version is an early draft from september 2021.

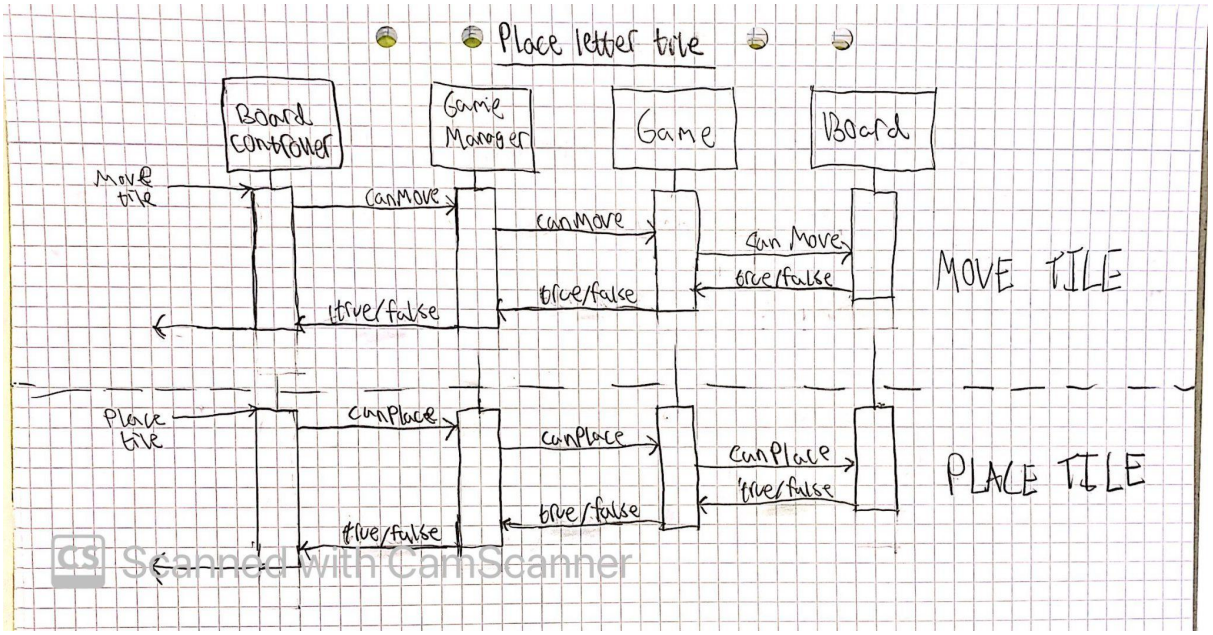

Figure 2: Sequence diagram visualizing the "Place letter tile" user story. This version is an early draft from september 2021 and is no longer representative of how the game implements this functionality.

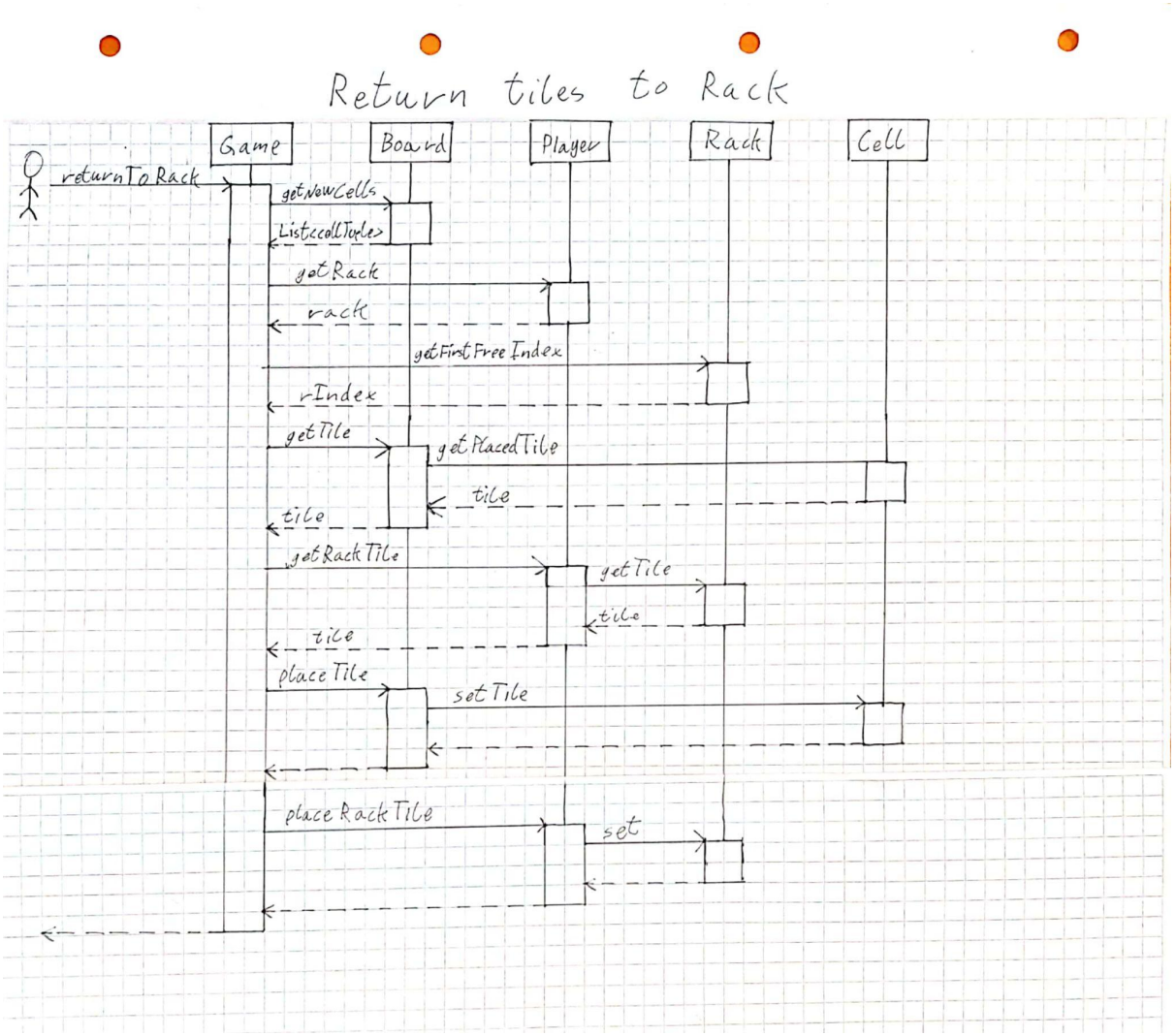

Figure 3: Sequence diagram visualizing the “Return tiles to Rack” user story. This is the implementation from 2021-10-22.

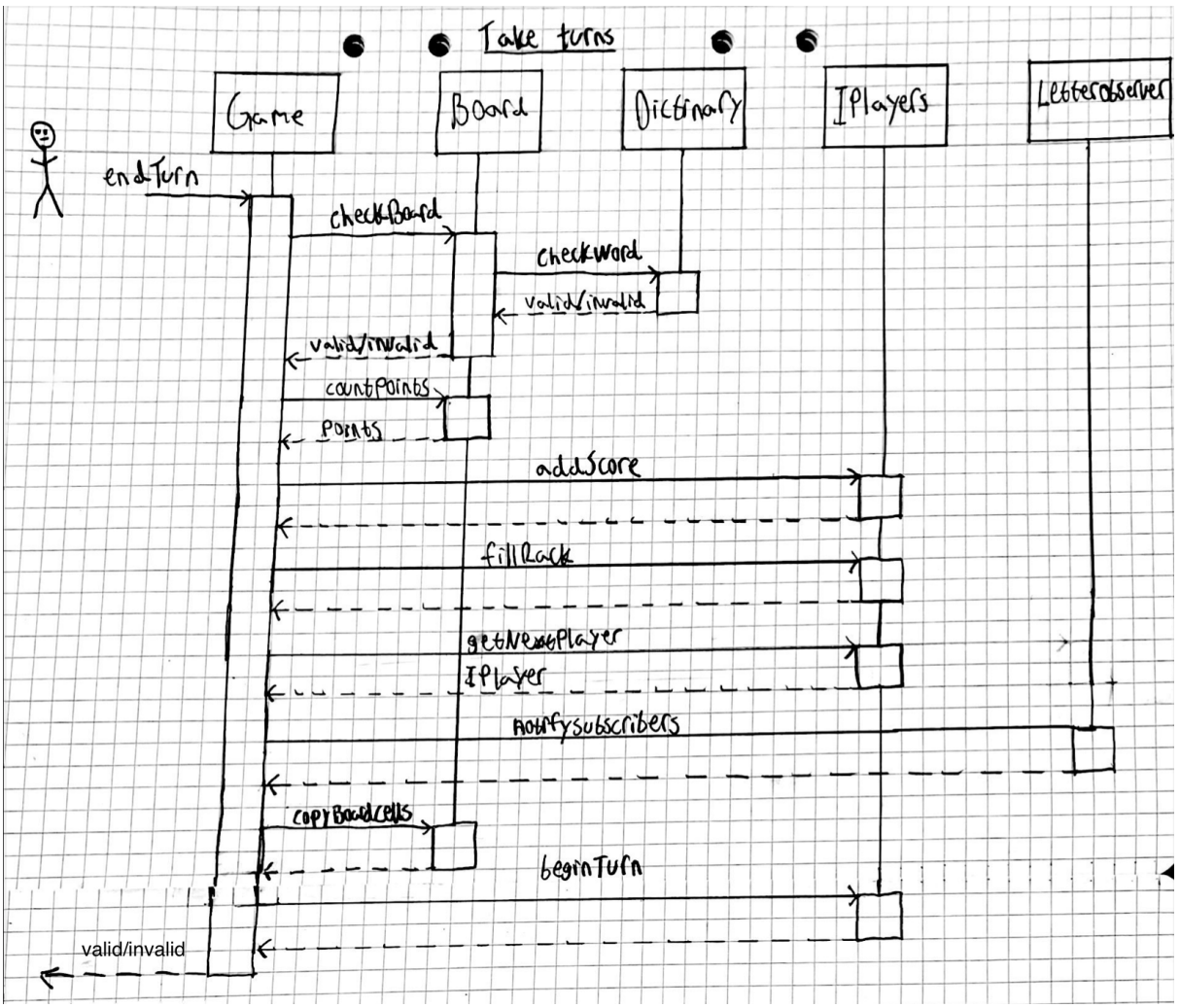

Figure 4: Sequence diagram visualizing the “Take turns” user story. This is the final version.

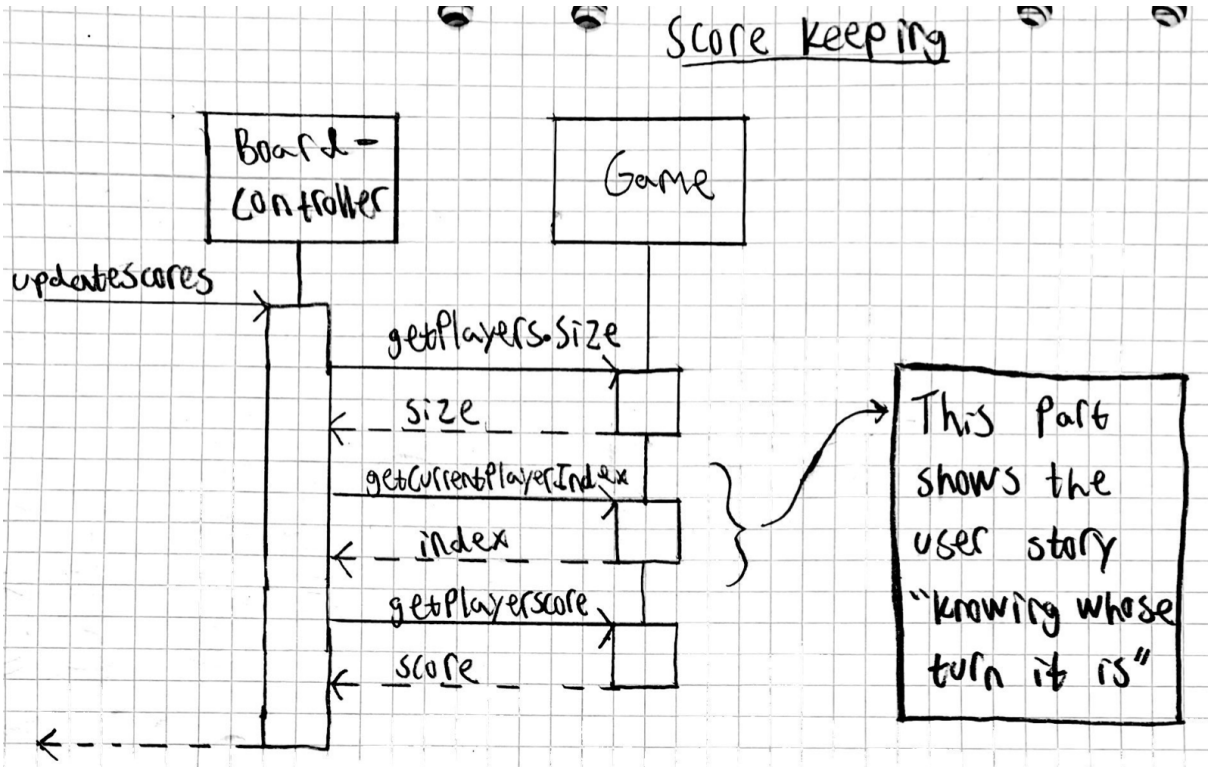

Figure 5: Sequence diagram showing the “Score keeping” user story. The middle part also visualizes the “Knowing whose turn it is” user story.

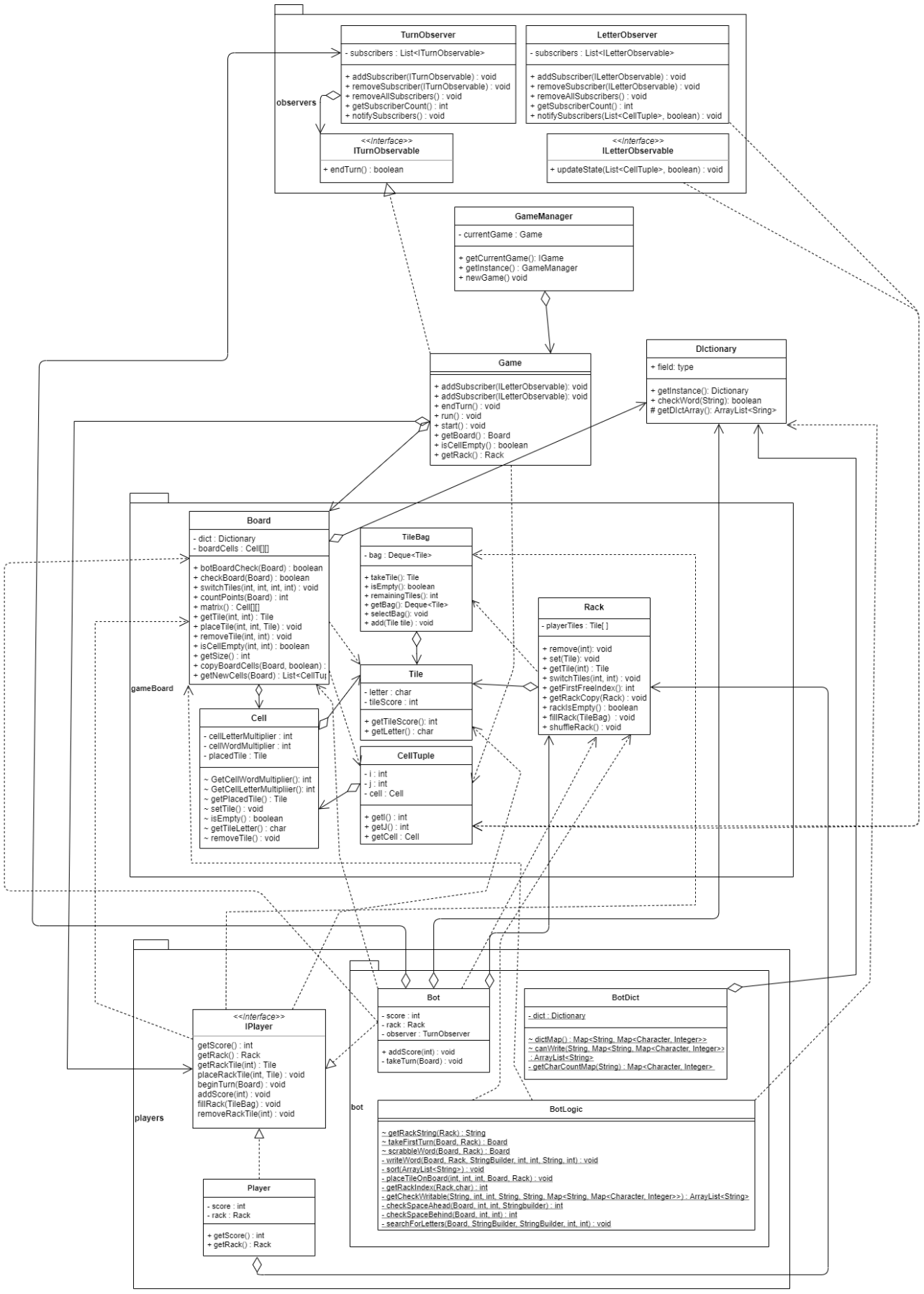

Figure 6: Design model.

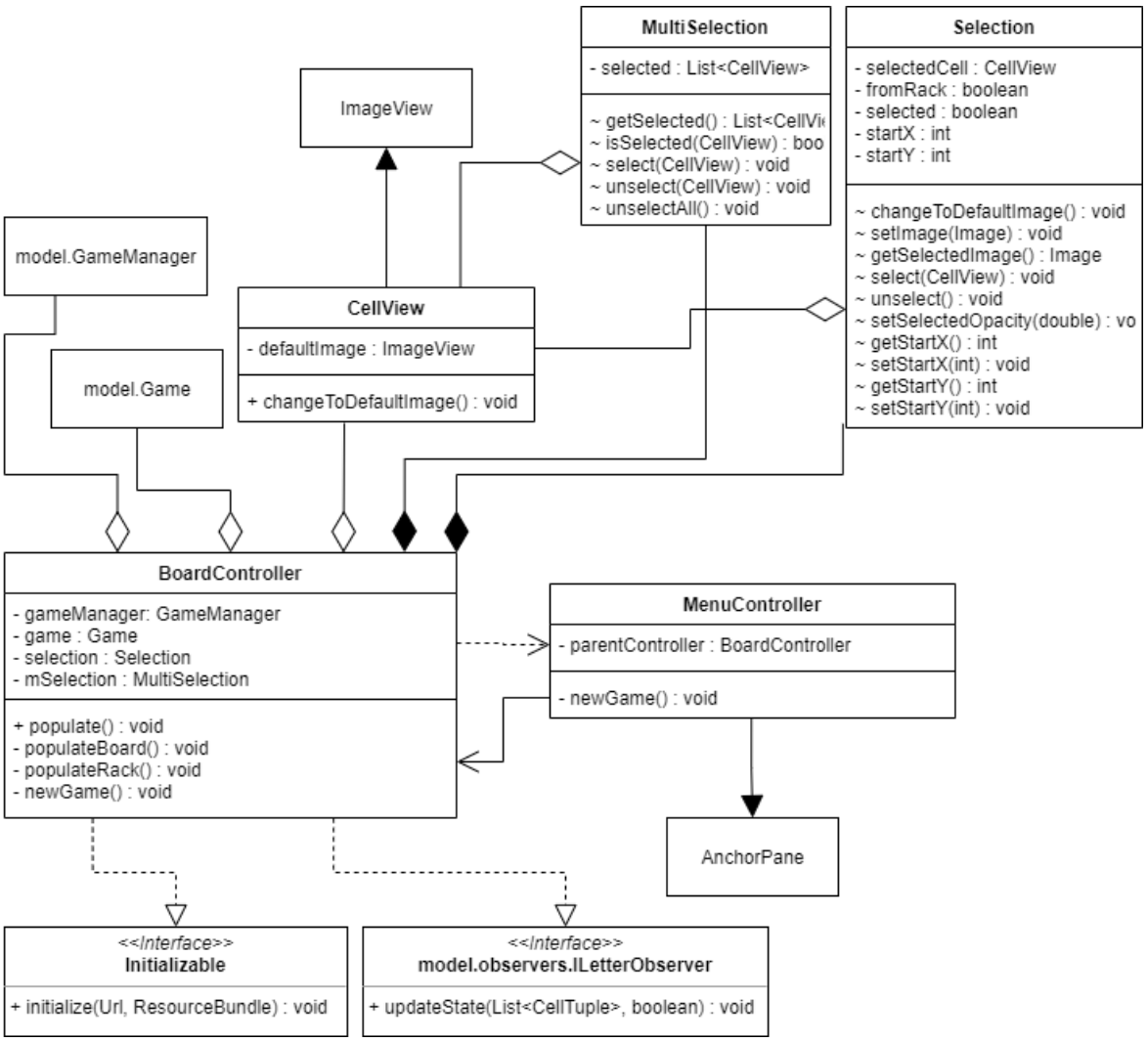

Figure 7: UML-diagram for the controller package.

3 System design

MVC

JavaFX and SceneBuilder were used to handle the graphics of the application and thus its protocols had to be followed. The application's GUI was designed in Scenebuilder and JavaFX imports a .fxml file to load the scene. Controller classes were connected to the .fxml and to GUI elements in the scene and the class is initialised when that .fxml is loaded.

It could be argued that the fxml files function as different views and since it is only possible to access the GUI elements through those classes the view and controller parts of MVC are very tightly coupled. The model however is separated as much as possible from the other modules.

The controllers that need to communicate with the model have a reference to a Game and calls the method it needs and influences the model directly. In order to avoid a circular dependency between the model and the view/controller an observer pattern was used. If the model has a change that impacts the view it publishes the changes to the views that are listening.

Relation to domain model

The design model has a very close match to the domain model. Everything in the domain is represented by a class in the design. So there is a very clear relation between each item in the domain model and its corresponding one in the design model. Since the things we included in the domain model are actual things/nouns it went well to incorporate them into the design. Some things turned out to be too big to fit in a single class so they became their own sub packages in the model, e.g. bot.

Design Patterns

The observer pattern was used twice in the code, used to update the view about changes in the model. The Game class works as a facade pattern and an entry point into the model. The singleton pattern is used to easily gain access to Game, EmptyTile and Dictionary, both of which only one instance of is needed. The codebase also uses a singleton for an enum called RandomSeed. This enum is used with a fixed seed for testing.

4 Persistent data management

- Text documents
- Icons

The game reads the data for how to populate the bag and the board from txt files. The dictionary is read from a .txt file upon starting the program.

5 Quality

Known Issues/Possible improvements:

- Players should not keep track of their own Rack and Score, there are currently problems because Rack belongs to a player but needs to be in the gameBoard package to access Tile and Cell. This creates an inconsistency in how Game communicates with the Rack. See sequence diagram “Return tiles to rack” for details.
  If the player's Rack was handled by Game there would be no need for most Rack operations to go through Player.
- Bot and Board are tightly coupled.

- When the bot is calculating, it blocks the thread and the UI stops being interactable, so running the bot/model in its own thread would be a good idea.
- Make more classes in gameBoard (e.g. Tile) be less visible, i.e make methods package private.
- Implement a singleton for an empty tile. There is currently a class called EmptyTile, but it is not used throughout the codebase.

Testing:

- Testing is done using JUnit. The tests can be found in the src/tests folder. The code coverage of the model package is at 96%.

JDepend:

- Figure 8 shows the JDepend report.

Package	TC	CC	AC	Ca	Ce	A	I	D	V
com.zcrabblers.zcrabble	1	1	0	0	11	0.0%	100.0%	0.0%	1
com.zcrabblers.zcrabble.controller	3	3	0	0	5	0.0%	100.0%	0.0%	1
com.zcrabblers.zcrabble.model	3	3	0	2	6	0.0%	75.0%	25.0%	1
com.zcrabblers.zcrabble.model.gameBoard	7	7	0	3	5	0.0%	62.0%	38.0%	1
com.zcrabblers.zcrabble.model.observers	2	0	2	2	2	100.0%	50.0%	50.0%	1
com.zcrabblers.zcrabble.model.players	2	1	1	2	2	50.0%	50.0%	0.0%	1
com.zcrabblers.zcrabble.model.players.bot	4	4	0	1	6	0.0%	86.0%	14.0%	1
com.zcrabblers.zcrabble.utils	1	1	0	1	3	0.0%	75.0%	25.0%	1

Figure 8: JDepend report.

5.1 Access control and security

- Does not apply to this project.

6 References

- Zcrabble rules can be found in project resources.
    - Zcrabble has some rule changes from the original Scrabble.
        - There are no wildcards in Zcrabble, instead there is an extra “Z” and “Q”.
        - It is not possible to challenge other players' words in Zcrabble instead the game automatically checks every play and refuses invalid words.
        - If an incorrect play is made the player does not lose their turn but is instead allowed to make another play or skip their round.
- This codebase uses JavaFX to represent the application graphically.
- Testing was done using JUnit.
- Maven is the build automation tool of choice.
- This codebase supports JDepend.
    - Run “mvn site” then “open target/site/jdepend-report/html” to read the report. Alternatively, see section 5 for a picture of the report.

Peer-review of “Rent a Car”

We were not able to build the project using gradle and there were no supplied instructions on how to build it. As such, this peer review will mainly be based on analyzing the codebase.

Comments / Javadoc

Not a single line of javadoc was found and that made it very hard to understand what the purpose of the different methods are. Since we have never seen the android api before we had no idea what anything did. So a major piece of feedback would be to add javadoc to all public methods in the code. There are some normal comments but there could be more. It would make it a lot easier for someone else to understand the code. Most variable/method/class names are verbose and class names follow a naming structure which is good.

MVC / Structure

Based on the SDD, it seems like this project aims to implement the MVC pattern. However, we can only find a package called ui. The rest of the classes do not seem to belong to any MVC package. Consequently, the overall structure of the codebase is hard to understand, especially since all the classes lack proper documentation.

Further, due to the lack of an MVC structure, the model does not seem to be as isolated as it could be. In fact, we have had a hard time identifying which classes actually belong to the model. The CarAdModel is a pure data class and is the only example of a class that definitely belongs in the model, but it should have some kind of behaviour code as well. Most other classes seem to be a mix of model, view and controller so each class doesn’t follow the single responsibility principle very well. Since many classes have several responsibilities, it seems like it could be difficult to add new functionality without modifying existing classes, which breaks the open closed principle. Because of high coupling it does seem like most of the code is not

reusable. It seems like it would be difficult to add a different view using the same model.

In SignUpPaymentFragment and AddCarAdFragment, there is quite a bit of casting (e.g. to EditText and Button). This is not very type safe. However, we have never worked with the android api before, so maybe this is necessary?

RAD/SDD

RAD:

- Definitions do not contain any definitions relevant to the domain. Why is “RAD” defined in the RAD document?
- The user stories are generally fine, although there seems to be some overlap. For instance, user stories two, three and five are pretty similar. Further, it is not clear what priority the user stories have.
- User interface looks nice. Good to get some pictures of the Figma mock up.
- Domain model looks good but is not reflected in the code. Should probably be updated. GUI can be removed from the domain model.

SDD:

- Definitions are more relevant in the SDD than the RAD.
    - Further, consider adding definitions that relate to the domain or the android api, e.g. fragment and toast.
- System architecture is not reflected in the code.
- Persistent data management section is useful for understanding some of the code base.

Tests

There are no tests. Since it is expected to cover 100% of the model writing tests should probably be a thing to focus on.

Improvements

- Higher separation between model, view and controller.
- Everything that concerns the database and the applications interaction with it we would move into the model package, these methods could then be called from the controller package.
- Add Javadoc to all public methods.
- Create tests.