

RaceTrack

a product by PathFinder

Authors

Marco Forster
forstma1@students.zhaw.ch

Manuel Berweger
berweman@students.zhaw.ch

Marvin Tseng
tsengmar@students.zhaw.ch

Dan Hochstrasser
hochsdan@students.zhaw.ch

21. May 2020

Contents

1	Project Idea	3
1.1	Baseline	3
1.2	Idea	4
1.3	Client Benefits	5
1.4	Competition Analysis	6
1.5	Main Use Case	7
1.6	Resources	8
1.7	Risks	9
1.8	Profitability	10
2	Analysis	12
2.1	Use Case Model	12
2.1.1	Overview	13
2.1.2	Main Use Cases	14
2.1.2.1	Use Case UC7: Play one Game Session	14
2.1.2.2	Use Case UC13: Load Track	17
2.1.3	Secondary Use Cases	20
2.1.3.1	Use Case UC1: View High Score	20
2.1.3.2	Use Case UC4: Set Game Options	21
2.1.3.3	Use Case UC10: Load Game Session	22
2.1.3.4	Use Case UC11: Save Game Session	23
2.1.4	Minor Use Cases	25
2.1.4.1	Use Case UC2: View Rules	25
2.1.4.2	Use Case UC3: View Credits	25
2.1.4.3	Use Case UC6: Reset Game Session	25
2.2	Additional Requirements	25
2.2.1	Functionality	26
2.2.1.1	General	26
2.2.1.2	Game Rules	26
2.2.1.3	Special Requirements	27
2.2.2	Usability	28
2.2.3	Reliability	28
2.2.4	Performance	28
2.2.5	Supportability	28
2.2.6	Scalability	28

2.3	Domain Model	28
3	Design	30
3.1	Software Architecture	30
3.1.1	Distribution Model	32
3.2	Design Artifacts	32
3.2.1	Design Class Diagram (DCD)	33
3.2.2	Selected Interaction Diagrams	33
3.2.2.1	Starting a new game from the main menu	33
3.2.2.2	Executing a turn in the game	34
3.2.2.3	Last player crosses the finish line	34
4	Implementation	35
4.1	Code Implementation	35
4.1.1	Packages	37
4.1.1.1	com.pathfinder.racetrack.model	37
4.1.1.2	com.pathfinder.racetrack.view	38
4.2	Code Quality	38
4.3	Code Documentation	39
4.4	Test Concept	40
4.5	Installation Guide	41
4.5.1	macOS	42
4.5.2	Windows	43
4.6	Source Code	43
5	Results	45
5.1	Achieved Goals	45
5.2	Open Issues	46
5.3	Future Developments	47
6	Appendix	49
6.1	Project Management	49
6.1.1	Construction Phase: 20.04.2020 - 24.05.2020	51
6.1.1.1	Milestone M3: 22.05.2020	51
6.1.1.2	Iteration 6: 04.05.2020 - 24.05.2020	51
6.1.1.3	Iteration 5: 20.04.2020 - 03.05.2020	51
6.1.2	Elaboration Phase: 09.03.2020 - 19.04.2020	52
6.1.2.1	Milestone M2: 17.04.2020	52
6.1.2.2	Iteration 4: 06.04.2020 - 19.04.2020	52
6.1.2.3	Iteration 3: 23.03.2020 - 05.04.2020	53
6.1.2.4	Iteration 2: 09.03.2020 - 22.03.2020	53
6.1.3	Inception Phase: 24.02.2020 - 08.03.2020	54
6.1.3.1	Milestone M1: 06.03.2020	54
6.1.3.2	Iteration 1: 24.02.2020 - 08.03.2020	54
6.2	References	54

Wherever we have used masculine pronouns only, this has been done solely for reasons of readability, however the female form is also always intended.

Chapter 1 Project Idea

All important points (except for 2.2.6 - 2.2.9) of the updated Project Outline have been summarized in this chapter.

1.1. Baseline

In today's world, almost anything is being digitalised. **PathFinder** wants to bring childhood memories and the nostalgic game experience of the popular paper and pencil game mostly known as *Racetrack* to the digital age. Not only can PathFinder and its partners turn a profit from this venture, but it can also reduce the general paper waste by providing a digital alternative to its original counterpart on paper.

1.2. Idea

The underlying idea of **RaceTrack** is a round-based racing game where players have to strategically choose their next move out of multiple reachable positions. These positions are calculated based on their previous turns and the position of other players. As the game calculates the selectable moves for each player based on their previous choices, the game gets more interesting and challenging with each passing turn. The players will be needed to demonstrate all their strategical and thoughtful decision-making skills to be able to beat their opponents. As the game is based on the original pencil and paper game *Vector Race* (also known as 'Racetrack' or 'PolyRace'), the game can also help to improve the foresight thinking and basic understanding of vector mathematics of each player.



Figure 1.1: Turn in the game (Mock-up)

1.3. Client Benefits

RaceTrack brings multiple benefits to its users respectively players:

- Players can play the game either alone or together with friends.
- No paper is needed and wasted. Instead, larger computer screens ensure better visibility than on a sheet of paper.
- Custom tracks can be created by players, and once loaded into the game, can be played on over and over again.
- Game sessions can be saved and reloaded, later on rounds don't have to be played in one session.
- Different choosable game modes make for a diverse gaming experience.
- A game can instantly be restarted with the same settings (e.g. same number of players, same track, same cars, etc.).
- Players can compare their scores on the high scoreboard.
- There's also an educational aspect where players can learn about the concept of vectors and vector addition:
 - Players can train their reasoning abilities by playing the game.
 - It can even help to improve decision-making skills.

1.4. Competition Analysis

Research has shown that the game *Racetrack*, also known as *Vector Race*, has never been realized as a computer desktop game [1] [2]. Currently, only two mobile applications on Android with the same concept as RaceTrack are being distributed on the Google Play Store [3]. However, these games are not suitable for local multiplayer or educational purposes, as the screen size is limited by the smartphone.

Thanks to the implementation with the programming language Java, RaceTrack is platform-independent. This gives the user the advantage of running RaceTrack on all different kinds of operating systems like Windows, macOS, or Linux. Besides, a larger audience is reached, and therewith a larger potential of paying customers.

RaceTrack distinguishes itself by adding the ability to load in custom tracks into the game and the option to play the game with distinct game modes, with each of them handling situations (e.g. when crashing in the game) differently. It is also planned to be able to add special items onto the track, e.g. boosts or obstacles. This feature can be activated during the creation of the game round.

1.5. Main Use Case

The main use case of RaceTrack is a player or a group of players playing a round of RaceTrack:

1. On his computer, the player starts the application, which he already downloaded and installed.
2. After the main menu finishes loading, the player starts a new game by selecting the corresponding entry in the menu.
3. The player configures the coming game session with the options to his liking, e.g. which track to be played on, the number of players, the colors of the cars, which game mode to be played with and more. In the end, he lets the configured session to be created by the game.
4. After the game has started, the player can make his first turn.
5. The first player selects his move for his current turn, based on the given possible moves the player can choose from. The available options are being shown on the track.
 - The player's possible moves are calculated at the beginning of his turn, based on his current velocity and previously selected moves.
6. The player's car moves to the selected position on the track.

As a turn-based game, every other player gets to do his turn next, before the same player gets the ability to do another turn. Each player repeats steps 5-6 until it's the first player's turn again.

7. The game continues, turn after turn, until every player crosses the finish line.
8. While playing the game, the following rules are being enforced:
 - It is not possible to drive backward.
 - It is not possible to drive to a position already occupied by another player.
 - It is not possible drive past the track limits, doing so will result in different consequences, depending on the game mode selected before the start of the session.
9. After the session finishes, the players' results are displayed in comparison with the track's high score.
10. The player can choose to return to the main menu, restart the session with the same settings, or to end the game (application) completely.

1.6. Resources

The development of RaceTrack will be split between 4 programmers. Every member of the development team needs to have advanced knowledge in programming with Java. With the use cases, it is possible to split the tasks and resources between the developers. Also, knowledge in vector mathematics is needed to know possible directions for the car in the next turn.

Every developer needs a working computer to build and test the game. Git is used as a version control system and GitHub (hosted by ZHAW) as a remote repository, with this configuration, changes can be easily tracked, retraced and reverted if necessary. The project status is being tracked by using the *Projects* function in GitHub using Kanban Board boards. *Azure Pipelines* and *SonarCloud* are used for *Continuous Integration and Deployment* and ensuring code quality and security respectively. 400 to 500 man-hours are projected to be needed for developing the game and managing the project.

1.7. Risks

Following risks have been identified:

- Since PathFinder is an emerging startup, it does not offer the same expertise as established game development companies.
- Also, with a game like RaceTrack, the rules of the game must be well known to the player to ensure the *fun* factor.
- With a small team consisting of only four developers, the temporary loss of a teammate due to illness or other diseases can heavily impact the progress of the project.
- Implementation of graphical user interfaces unfamiliar.
- The used programming language Java is not optimized for game development.

1.8. Profitability

With the required man-hours multiplied by the salary, the development costs of the application RaceTrack is estimated at 60,000 Swiss Francs. Additionally, there will be expenses of 1'000 Swiss Francs for licenses and infrastructure. Since our game can be downloaded for free, our income is generated by Microtransactions. In future releases, users will be able to purchase Add-ons such as new race tracks or additional cars with striking looks. The prices for these Add-ons range from 2 to 15 Swiss Francs. With a predicted user base of 30'000 users and average revenue of 2 francs per installation, we expect the game to become profitable within a year.

Chapter 2 Analysis

The most important results of the problem analysis are documented in this chapter.

2.1. Use Case Model

All identified use cases of RaceTrack are described in this section. Each use case is identified by an id in the form of **UCn** (e.g. UC1, UC2, ...). Use cases are added and removed during development but don't change their ids during the entire development cycle (Reason why some use cases, e.g. UC5, don't exist anymore).

2.1.1. Overview

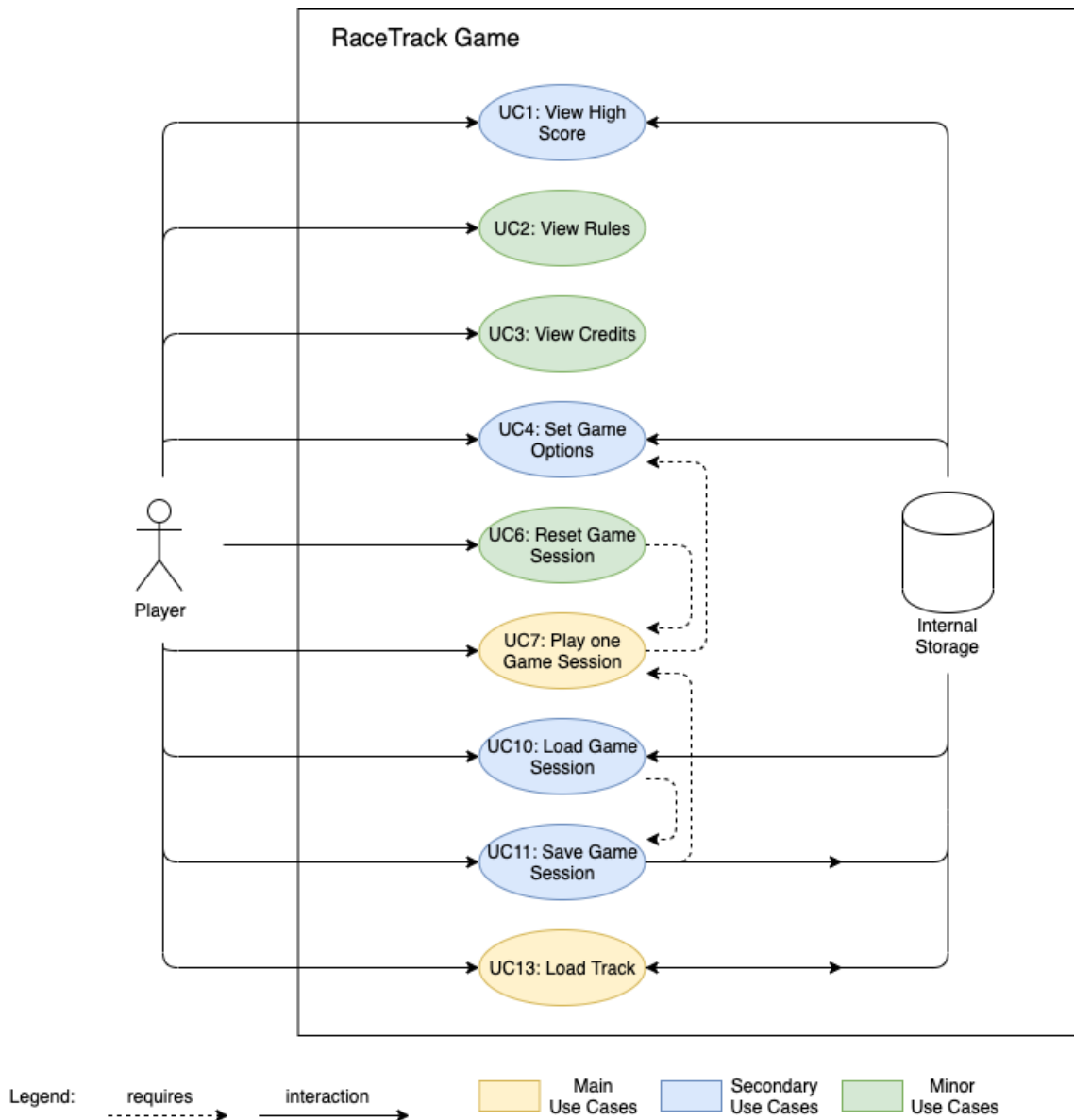


Figure 2.1: Overview of all defined use cases of RaceTrack

- UC6 requires UC7, as there is no session to reset if no session is currently being played.
- UC7 requires UC4, as the options for the game session need to be defined by the player before the session can be started.
- UC10 requires UC11, as a game session first needs to be saved before it can be loaded again.
- UC11 requires UC7, as there is no session to be saved if no session is currently being played.

2.1.2. Main Use Cases

2.1.2.1. Use Case UC7: Play one Game Session

Scope: RaceTrack Game

Level: User Goal

Primary Actor: Player

Stakeholders and Interests:

Player: Wants to play one game session of RaceTrack, either alone or with up to three friends.

Preconditions: A track has been loaded (UC10) and the game session has been successfully started with the player's settings (UC4).

Success Guarantee: The game session has been successfully played. Every player has reached the finish line and the high score board is being displayed (UC1).

Frequency of Occurrence: Once for every game session played.

Main Success Scenario

1. The session starts and the system places each player's car on the starting line.
 2. The order of the placement is from the top left to the bottom right.
 3. The first player to start is determined randomly.
 4. The player is given a total of maximum 9 possible moves he can choose for his car during his turn.
 - (a) The possible moves for the player's car are calculated before each turn, based on the player's velocity, previous turns, and potential obstacles on the track (e.g. another player's car).
 - (b) With each move the player can accelerate, decelerate, or maintain his current velocity.
 - (c) All available moves the player can take are shown on the track.
 5. The player chooses his move out of the possible moves.
 6. The player's car drives to the selected position on the track.
 - (a) Once a player reaches the finish line, his placement is saved and can no longer participate in the current race.
- Each player repeats steps 4. - 6. for his turn until every player reaches the finish line.*
7. The session ends and the scores of the just-finished race are displayed.
 8. The players can restart another session with the same settings or return to the main menu.

Alternative Flows

1. At any time, a player drives past the track limits:
 - (a) When the **Go Kart** game mode (*Continue with minimum velocity and no acceleration until the track is reached again*) is selected during game creation:
 - i. The player's velocity resets.
 - ii. The player does not accelerate until the track is reached again.
 - (b) When the **Formula One** game mode *Retire from the race* is selected during game creation:
 - i. The player *crashes* and receives a corresponding notification about it.
 - ii. The player retires from the race and will not be able to do any more moves during this game session.
 - (c) When the **Bobby Car** game mode *Reset position to last valid point and reset velocity* is selected during game creation:
 - i. The player *crashes* and receives a corresponding notification about it.
 - ii. The player's velocity resets.
 - iii. The player's car resets to his last valid position on the track.

2. At any time, a player can drive to a position already occupied by another player:
 - (a) The game calculates the player's possible moves and realizes that some possible moves are already occupied by another player.
 - (b) The game will not show the invalid move to the player as one of the next possible moves anymore.
3. When *special items* have been selected as an option while starting the game:
 - (a) Special items e.g. boosts or obstacles are being randomly placed across the track.
 - (b) When driving on to a special item, its special ability will be activated.

Special Requirements

- Recognisable assets. Cars, track limits, special items and the starting/finish line must be easily recognizable.
- The car's movement has to be smooth.
- It has to be clear which player's move it currently is.

Technology and Data Variations List

- Player input is entered by a mouse click or keyboard (0-9).

UI Sketch

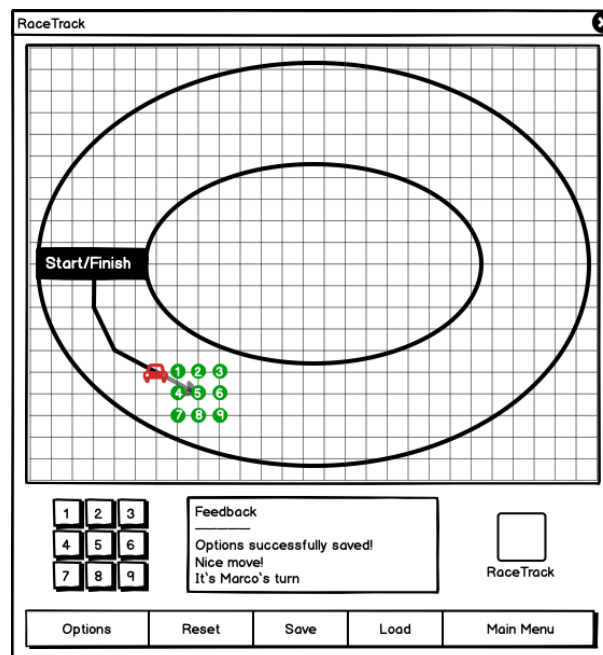


Figure 2.2: UI Sketch for Use Case UC7

System Sequence Diagram

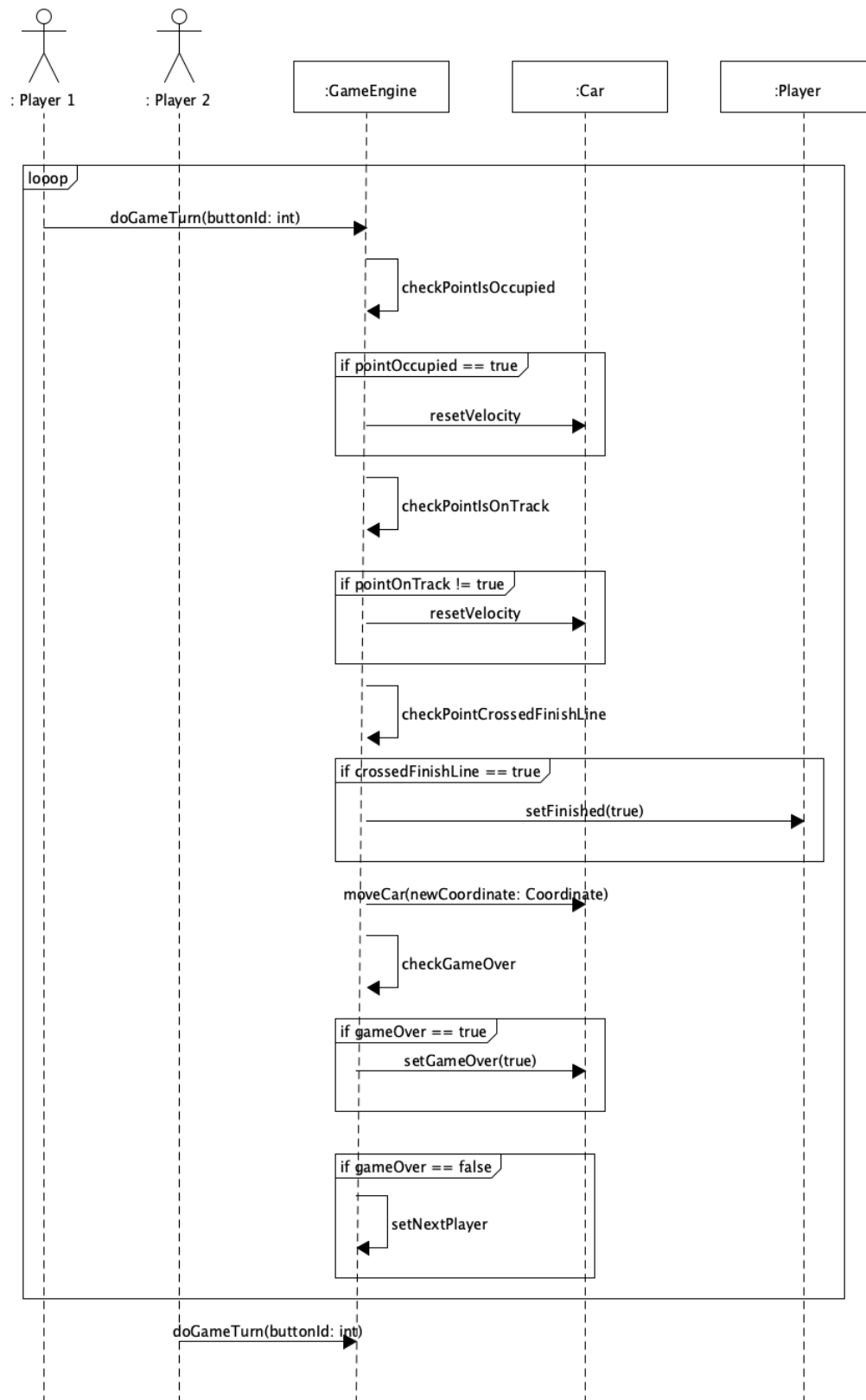


Figure 2.3: System Sequence Diagram of Use Case UC7

2.1.2.2. Use Case UC13: Load Track

Scope: RaceTrack Game

Level: User Goal

Primary Actor: Player

Stakeholders and Interests:

Player: Wants to load a track from a bitmap image.

Preconditions: A bitmap image that complies with all specifications for a track file (see *Special Requirements*).

Success Guarantee: The track has been successfully loaded into the game and no errors have occurred during the upload process.

Frequency of Occurrence: Once for each new track uploaded.

Main Success Scenario

1. The player selects the *Load Track* entry in the menu.
2. The player selects a valid bitmap image, which complies with all specifications for a track file (see *Special Requirements*).
3. The system uploads the selected image into the game and converts it into a track.
4. The player can upload another track or return to the main menu.

Alternative Flows

1. When the player wants to delete an existing track:
 - (a) The player selects an existing track from the list in the tracks menu.
 - (b) The player confirms the deletion by clicking on the *Delete Track* button.
 - (c) The system removes the track from the game.
 - Default tracks can not be deleted and a comprehensible error message will be displayed when trying so.
2. When the verification of the image as a track fails:
 - (a) The game will respond with a comprehensible error message that the image can not be verified as a track.
 - (b) The player will be allowed to retry the upload process with another image or to return to the main menu.

Special Requirements

- After the file has been uploaded, the track needs to be choosable during the set-up of a new game session (UC4).
- The bitmap image needs to comply with these specifications or else the game will not be able to verify it as a track:
 - The image must only consist of the following colors (*can still be changed during development*):
 - * #C5C5C5 : represents the drivable track.
 - * #000000 / #FFFFFF : represents the starting/finish line.
 - * Every other color will be interpreted as not to be part of the drivable track (out of track boundaries).
 - The starting/finish line must be placed either vertically or horizontally, it cannot be placed diagonally.

- Below is an example of a correct bitmap image:

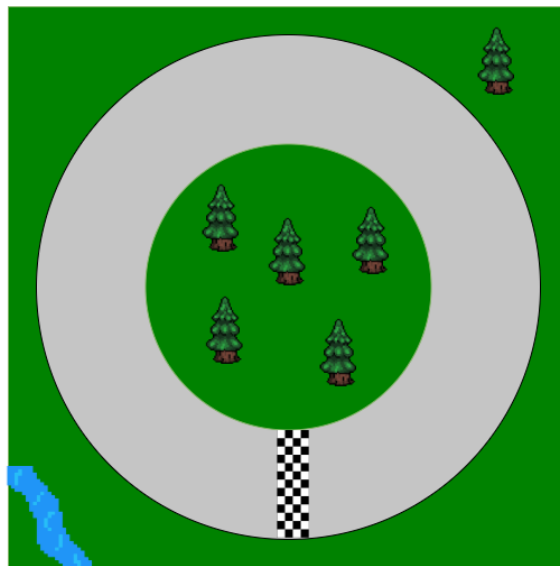


Figure 2.4: Example of a valid track to be used in Use Case UC13

Technology and Data Variations List

- The bitmap file needs to be of the file format *PNG*.

UI Sketch

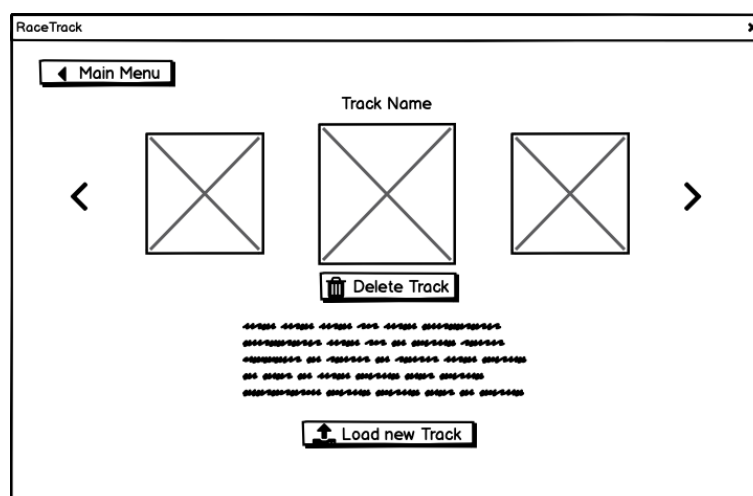


Figure 2.5: UI Sketch for Use Case UC13

System Sequence Diagram

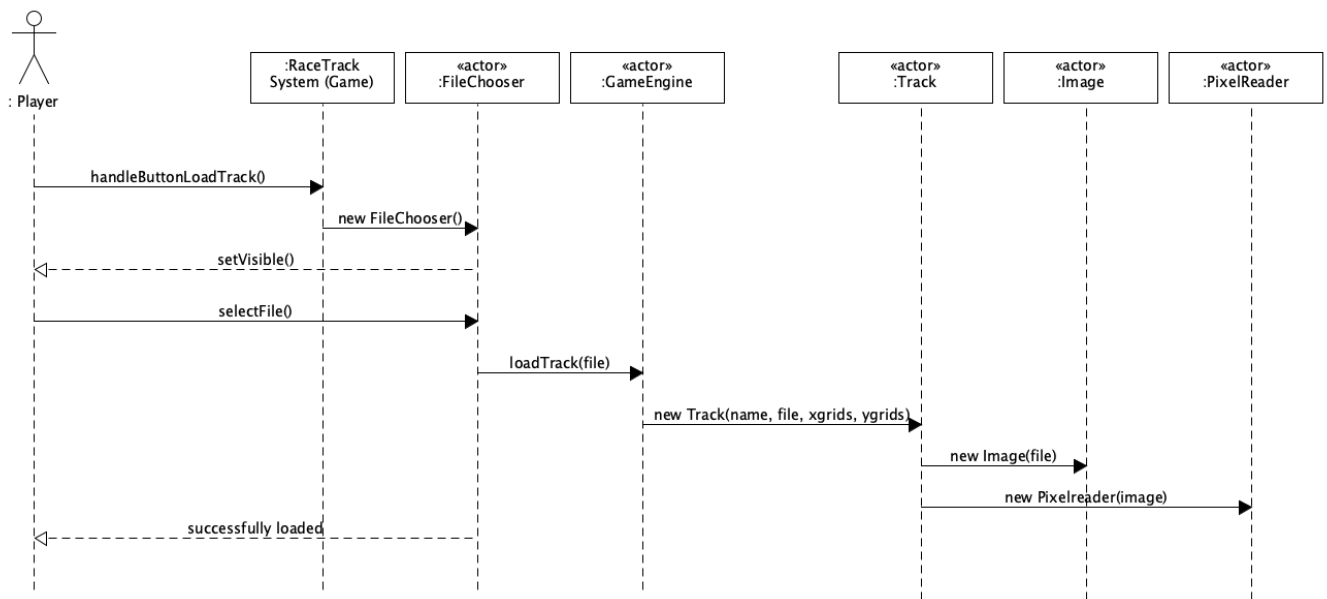


Figure 2.6: System Sequence Diagram of Use Case UC13

2.1.3. Secondary Use Cases

2.1.3.1. Use Case UC1: View High Score

Scope: RaceTrack Game

Level: User Goal

Primary Actor: Player

Stakeholders and Interests:

Player: Wants to look at the high score of a particular track.

Preconditions: At least one game session has already been played on the track (UC7), else there are no scores to be displayed for the track.

Success Guarantee: The high scores of the track are successfully shown.

Frequency of Occurrence: Every time the player selects the *View High Score* button in the menu.

Main Success Scenario

1. The player selects the *View High Score* entry in the menu.
2. The player chooses for which track the high score should be displayed.
3. The player chooses which game mode the high score should be displayed.
 - (a) Three available game modes can be chosen during the creation of a new game session (UC4).
4. The high score board for the particular track and game mode will be shown.

Alternative Flows

1. When a game session finishes (UC7):
 - (a) The scoreboard for the just-finished game session (track and game mode) will be displayed.
 - (b) In the scoreboard, the scores of the participated players will be displayed with the current high score of the track and game mode, that just has been played on.

Special Requirements

- At least one round must have been completed on the track before the high score can be displayed.
- The game session must be completed, before its scores get written to the high score list.

Technology and Data Variations List

- The high scores are saved in a *.rtsave* file.
- The *.rtsave* file follows the JSON syntax.

UI Sketch

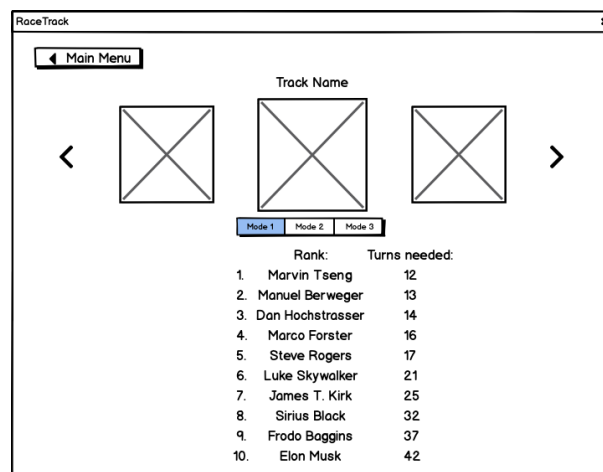


Figure 2.7: UI Sketch of Use Case UC1

2.1.3.2. Use Case UC4: Set Game Options

Scope: RaceTrack Game

Level: User Goal

Primary Actor: Player

Stakeholders and Interests:

Player: Wants to configure the options for the coming game session (UC7).

Preconditions: No running game session at the moment.

Success Guarantee: The preferences entered by the user are successfully applied to the created game session.

Frequency of Occurrence: Once before each new game session.

Main Success Scenario

1. The player selects the *New Game* entry in the menu.
2. The player configures the settings for the new game session to his liking.
 - (a) The player defines for each participating player the name and car color.
 - (b) The player defines how the game should handle crashes by selecting one out of three available game modes:
 - i. **Go Kart** : *Continue with minimum velocity and no acceleration until the track is reached again.*
 - ii. **Formula One** : *Retire from the race.*
 - iii. **Bobby Car** : *Reset position to last valid point and reset velocity.*
 - (c) The player selects the track to be played on.
3. The player finishes the setup for a new game session by clicking on the *Start Game* button.

Alternative Flows

1. When not all four player fields are filled out during game setup:
 - (a) The game starts the session with only filled out players, players with no name are ignored.

Special Requirements

- The settings must be easily understandable.
- Language internationalization on the text displayed.
- Only previously loaded tracks (UC13) can be chosen during the game setup.

Technology and Data Variations List

- The settings for the game session must be saved in a *.rtsave* file.
- The *.rtsave* file follows the JSON syntax.

UI Sketch

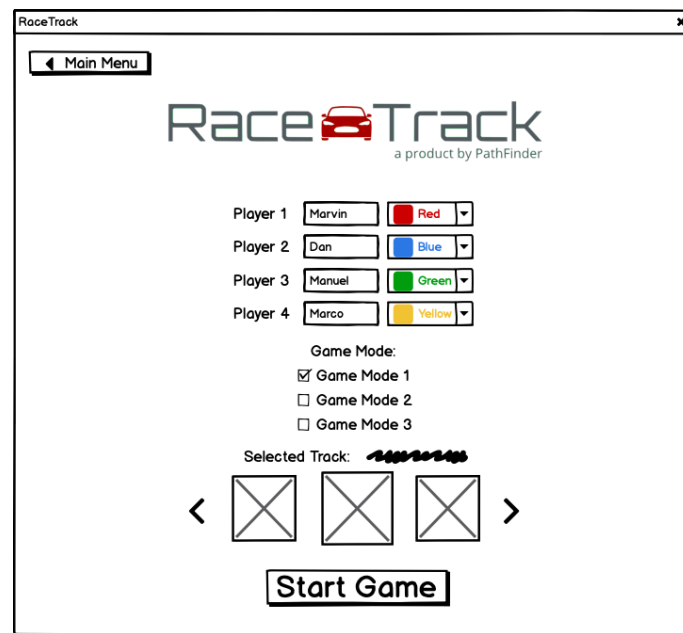


Figure 2.8: UI Sketch for Use Case UC4

2.1.3.3. Use Case UC10: Load Game Session

Scope: RaceTrack Game

Level: User Goal

Primary Actor: Player

Stakeholders and Interests:

Player: Wants to load a previously saved game session (UC11).

Preconditions: A valid *.rtsave* file containing all the necessary pieces of information to restore a past game session.

Success Guarantee: A saved game session has been successfully loaded from a *.rtsave* file.

Frequency of Occurrence: Once per saved game session to load.

Main Success Scenario

1. The player is currently in the main menu of the game.
2. The player selects *Load Game* in the main menu.
3. The player chooses one of the previously saved game sessions in the list to be loaded.
4. The system loads the previously saved game session with the same state as saved.

Alternative Flows

No alternative flows present.

Special Requirements

- When the save file is loaded, the game state should be the same as before.

Technology and Data Variations List

- The resulting file that gets loaded must be a *.rtsave* file.
- The *.rtsave* file follows the JSON syntax.

UI Sketch

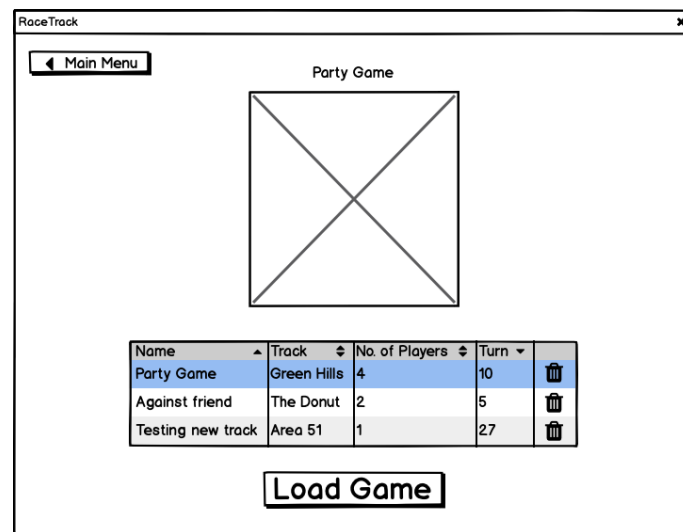


Figure 2.9: UI Sketch for Use Case UC15

2.1.3.4. Use Case UC11: Save Game Session

Scope: RaceTrack Game

Level: User Goal

Primary Actor: Player

Stakeholders and Interests:

Player: Wants to save a running game session (UC7) that hasn't been saved yet.

Preconditions: A currently running game session (UC7).

Success Guarantee: A running game session has been successfully saved in a *.rtsave* file.

Frequency of Occurrence: Usually once per game session.

Main Success Scenario

1. A game session is currently being played (UC7).
2. The player selects the *Save Game* entry in the menu of the game window.
3. The game shows a success message to the player, that the session has been successfully saved.

Alternative Flows

1. Any time the player wants to delete a previously saved game session:
 - (a) The player is currently in the main menu of the game.
 - (b) The player selects *Load Game* in the main menu.
 - (c) Instead, to *load* the selected game session, the player chooses to *delete* the saved game session.

Special Requirements

- All information needed to save the current game state must be included in the save file.

Technology and Data Variations List

- The resulting file that gets created with a save must have the extension *.rtsave*.
- The *.rtsave* file follows the JSON syntax.

- Below is an example of such a `.rtsave` file:

```

1 {
2   "currentPlayerId": 0,
3   "gameMode": "com.pathfinder.racetrack.model.GoKartEngine",
4   "trackName": "The Misty Forest",
5   "players": [
6     {
7       "id": 0,
8       "car": {
9         "velocityHistory": [
10          {
11            "xy": [
12              3,
13              -1
14            ]
15          }
16        ],
17        "route": [
18          {
19            "xy": [
20              3,
21              17
22            ]
23          },
24          {
25            "xy": [
26              3,
27              16
28            ]
29          }
30        ],
31        "color": {
32          "red": 1.0,
33          "green": 0.0,
34          "blue": 0.0,
35          "opacity": 1.0,
36          "platformPaint": {
37            "argb": -65536,
38            "r": 1.0,
39            "g": 0.0,
40            "b": 0.0,
41            "a": 1.0,
42            "type": "COLOR",
43            "proportional": false,
44            "isMutable": false
45          }
46        },
47        "crashed": false,
48        "offTrack": false
49      },
50      "name": "Player 1",
51      "finished": false,
52      "retired": false,
53      "rank": 0,
54      "nbrOfMoves": 0
55    }
56  ]
57 }

```

Listing 2.1: Example of a Save File

UI Sketch

Already visible on the UI-Sketch for Use Case UC1: Play one Game Session.

2.1.4. Minor Use Cases

2.1.4.1. Use Case UC2: View Rules

Before the player starts playing the game, he wants to look at the rules of the game, so he knows how to play. The player opens the rule set in the main menu by clicking on the corresponding entry. The player should be able to easily understand the aim of the game by reading through the rules.

2.1.4.2. Use Case UC3: View Credits

The player is interested in the current game version and/or the developers of the game. In the main menu, he opens the credits dialog by clicking on the *View Credits* button. In the credits dialog, he can easily determine the current game build and/or the names and contact information of the developers.

2.1.4.3. Use Case UC6: Reset Game Session

The player is not satisfied with the current state of the game, based on the moves the player has previously made. To reset his moves and start the game from the beginning, the player clicks on the *Reset* button in the game window. Then the game will ask the player, if he is sure about the action. If he says yes, all elements in the game will be set to the initial state, otherwise, no changes will be made.

2.2. Additional Requirements

2.2.1. Functionality

2.2.1.1. General

- The game will be played on the same device and GUI.
- The game will be implemented to allow distributed sessions in the future.

2.2.1.2. Game Rules

Goal

The objective of the game is to be the first one to reach the finish line. If multiple players finish the race with the same amount of turns, it will result in a draw.

Environment

The game board is a checkered piece of paper with a track, starting line and finish line. All the grid points on the track are accessible by the players with their respective moves.

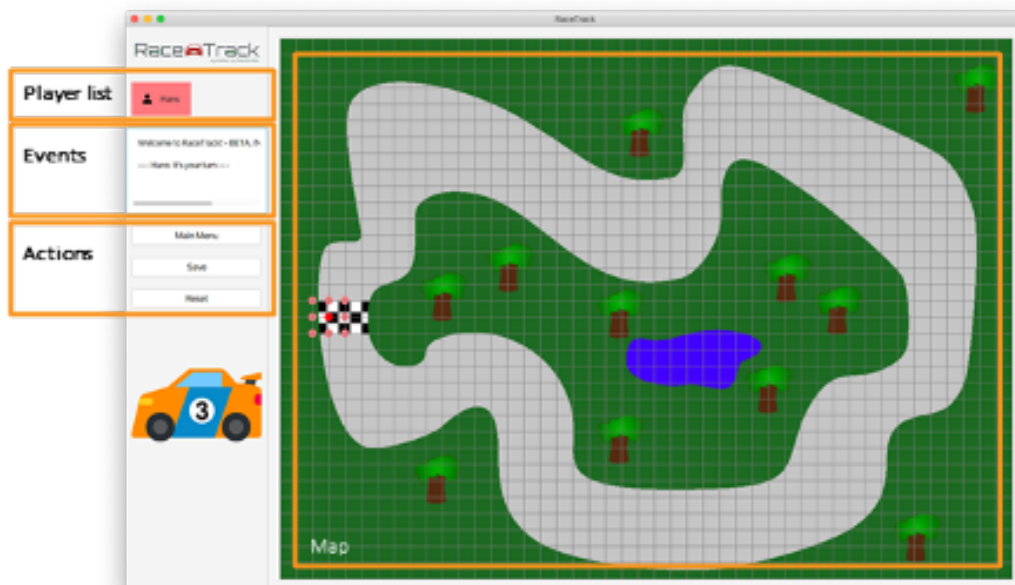


Figure 2.10: Example of the environment in the game

The course of the game

All players start from the starting line. The first player is determined randomly. After each move of a participant, the next competitor takes his turn until every player reaches the finish line. A finished player will not be able to move again.

Moving around

Each player's first move must lead to one of the five neighbors of the starting point moving forward. On the first move, it is not possible to go backward. In each following move, the so-called base for this move is determined. The base is dependent on repeating the previous move, both horizontally and vertically. This calculation in RaceTrack is also known as velocity in the game.

Example

“If the player last moved two boxes to the right and four boxes up, the main point is now two boxes to the right and four above the current starting point. The player now can move directly to the main point or one of his eight neighbors.”

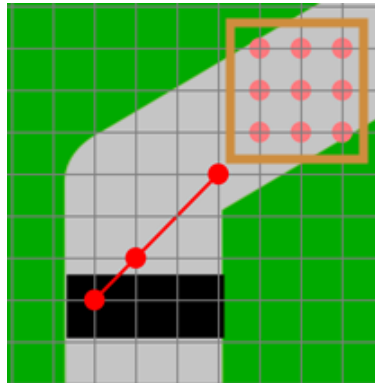


Figure 2.11: Example of how to move around the game

Conditions

The cars must stay within the track boundaries. This applies to every move. Leaving the road will lead to a crash. Depending on the selected game mode, the game will react differently to a crash. Points that are occupied by another car cannot be approached.

Game Modes and Crash Handling

There are currently **three** unique game modes available:

- **Go Kart** : *Continue with minimum velocity and no acceleration until the track is reached again:*
 - The player's velocity resets.
 - The player does not accelerate until the track is reached again.
- **Formula One** : *Retire from the race:*
 - The player crashes and receives a corresponding notification about it.
 - The player retires from the race and will not be able to do any more moves during this game session.
- **Bobby Car** : *Reset position to last valid point and reset velocity:*
 - The player crashes and receives a corresponding notification about it.
 - The player's velocity resets.
 - The player's car resets to his last valid position on the track.

2.2.1.3. Special Requirements

Custom tracks to be loaded into the game (UC13) need to comply with following requirements:

- The image must only consist of following colors (*can still be changed during development*):
 - #C5C5C5 : represents the drivable track.
 - #000000 / #FFFFFF : represents the starting/finish line.
 - Every other color will be interpreted as not to be part of the drivable track (out of track boundaries).
- The starting/finish line must be placed either vertically or horizontally, it cannot be placed diagonally.

2.2.2. Usability

RaceTrack's GUI must implement the fundamental concepts of usability to ensure great user experience. The following requirements are defined:

- Minimalistic GUI:
 - No buttons to steer the car, only mouse clicks needed.
 - A map to show:
 - * The grid paper.
 - * The Track.
 - * Starting/Finish line.
 - A list with all current players.
 - An event log where all past actions can be retracted.
- Target interaction components should be large enough for users to both discern what it is and to accurately select them.
 - minimum of 24 Pixels in width and height for buttons.

2.2.3. Reliability

The following reliability requirements are defined for RaceTrack:

- Reliability: Game breaking errors should occur at maximum once every 20 game sessions (during the Beta phase).
- Failure definition: A game-breaking error is a session of RaceTrack which was interrupted by a crash of the application.

2.2.4. Performance

Due to the simple architecture of the application, no special performance requirements are defined. Notice the usability requirement number four, which requires the response time for user feedback to be at *normal* levels (i 1s).

2.2.5. Supportability

Serviceability requirements:

1. The Java code must be documented.
2. Clean code concepts will be integrated when developing RaceTrack (See PDF *PROG1 Clean-Code-Regeln*).
3. RaceTrack is developed using the Semantic Versioning specification.

2.2.6. Scalability

There are no special requirements for scalability.

2.3. Domain Model

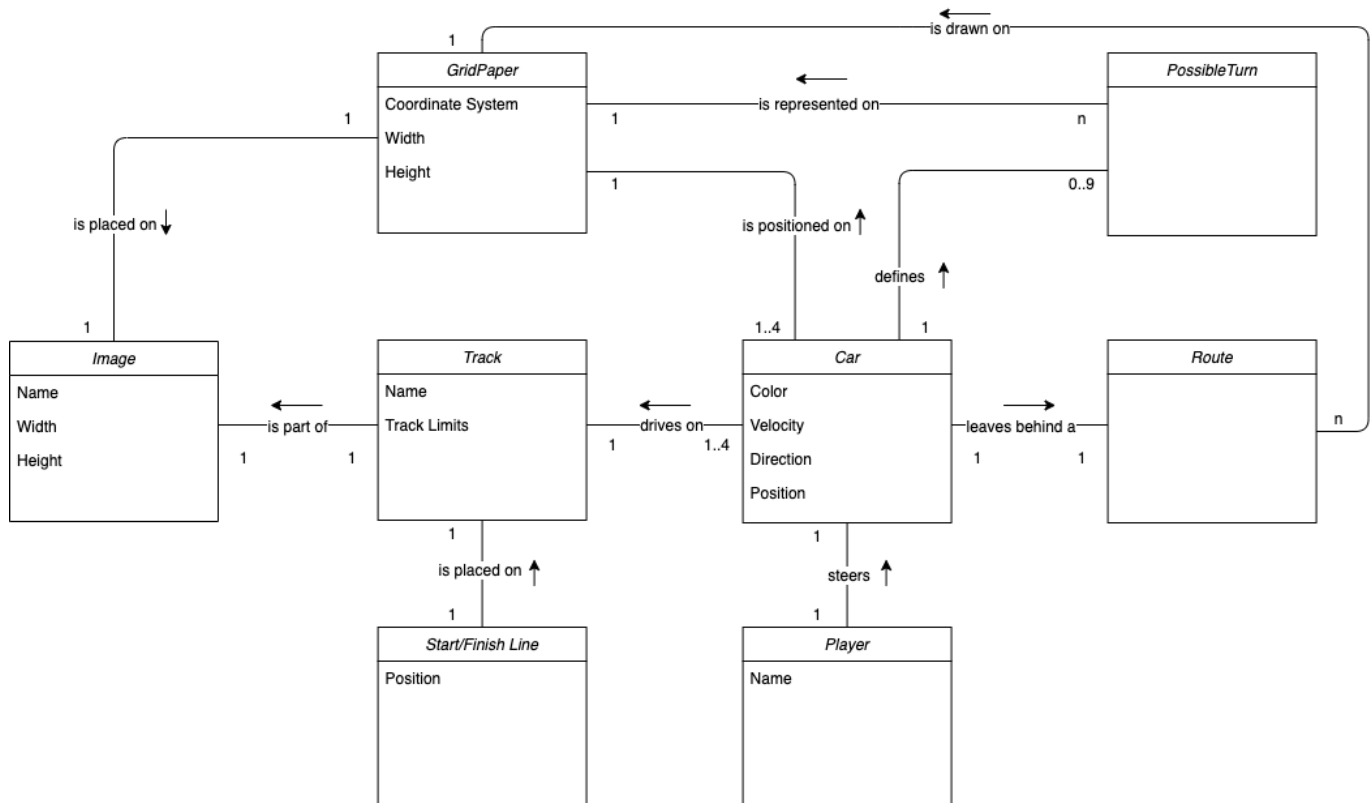


Figure 2.12: Domain Model of RaceTrack

The *Track*, with a *Start/Finish Line* placed on it, is part of an *Image*. In the end, a *GridPaper* is placed on the image, which defines the points players can drive to. There are between one and four cars that drive on the *Track*, they are placed on the *Start/Finish Line* before the game starts. Each *Car* is being steered by a *Player*. A *Car* is always being positioned on the *GridPaper*. The *Car* defines its *PossibleTurn* through his velocity and direction. Each *PossibleTurn* is represented on the *GridPaper* itself. While driving, the *Car* leaves behind a *Route*, which is also drawn on the *GridPaper*.

Chapter 3 Design

The most important aspects of the selected solution are documented in this chapter.

3.1. Software Architecture

As there weren't any special architectural requirements needed for RaceTrack, it was decided to opt for a **layered architecture**. For the development of RaceTrack, the UI will be separated from the model and application logic. This layered architecture will be used to separate the application's concerns into two stages:

- **User Interface** represents the visualization of the data that model contains (*package com.pathfinder.racetrack.view*).
- **Domain- / Application Logic** acts on both model and view. It controls the data flow into model objects and updates the view whenever data changes. It keeps view and model separate (*package com.pathfinder.racetrack.model*).

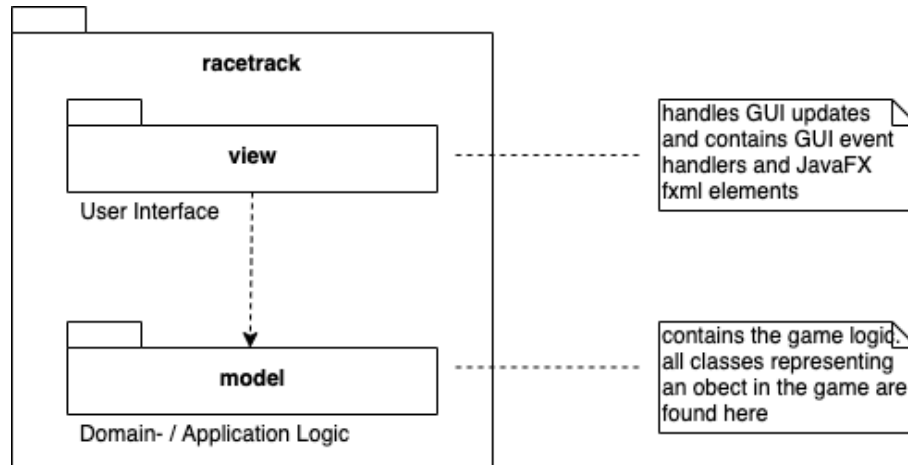


Figure 3.1: Layered implementation in RaceTrack

Not only does this abstraction separate the different aspects of the application, but it also provides loose coupling between these elements. The UI logic belongs in the view and business logic belongs in the model. Because of the separation of responsibilities, future development or modification is also easier. This layered architecture will also be easier to implement in a game, as other patterns like a pure implementation of the MVC Pattern are really difficult to implement using Java and especially JavaFX.

Package Diagram:

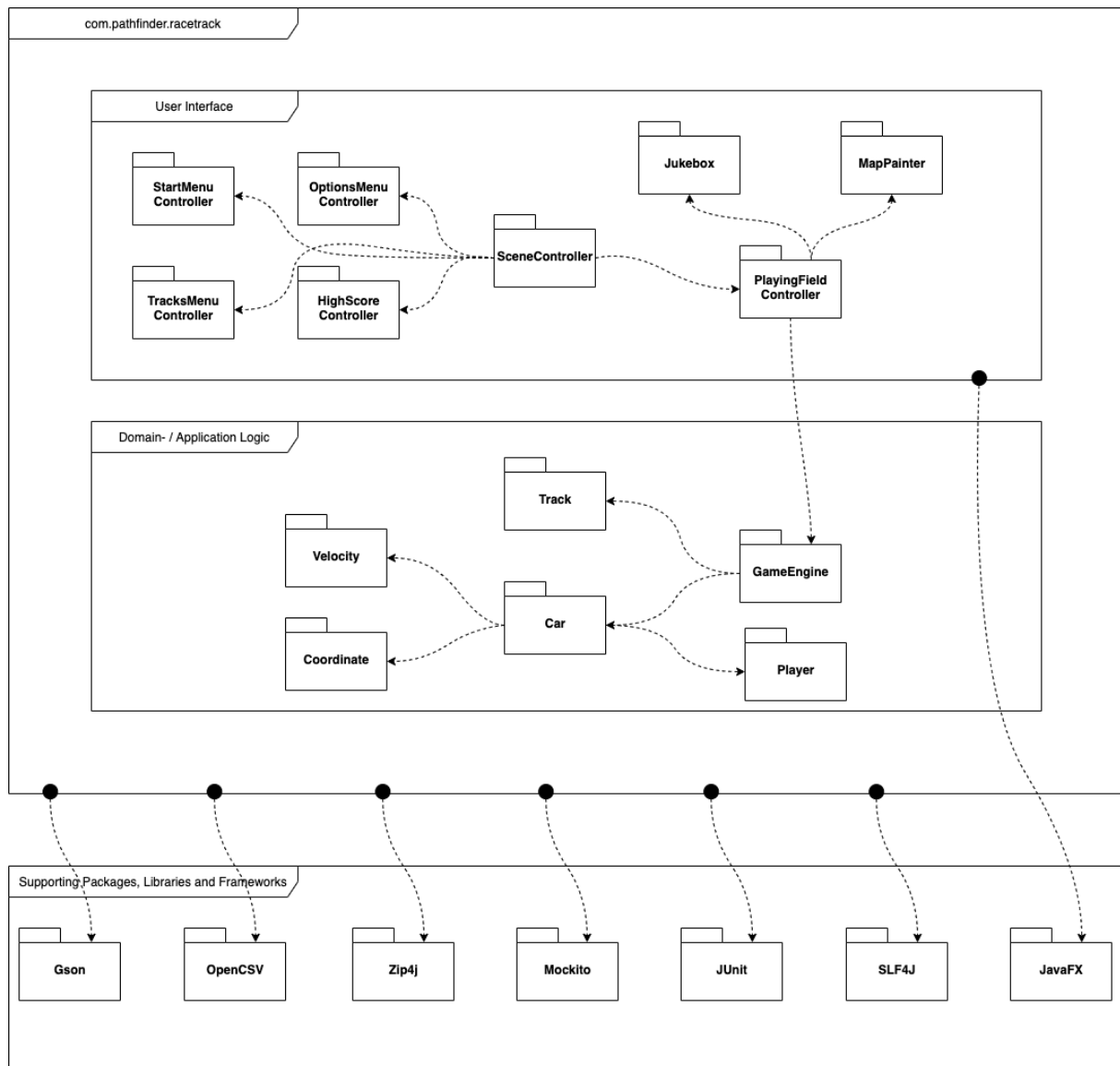


Figure 3.2: Package Diagram of RaceTrack

3.1.1. Distribution Model

As RaceTrack will be played on a single device, there isn't any need for a distribution model. But the game will be implemented to allow distributed sessions (Peer-to-Peer) in the future.

3.2. Design Artifacts

3.2.1. Design Class Diagram (DCD)

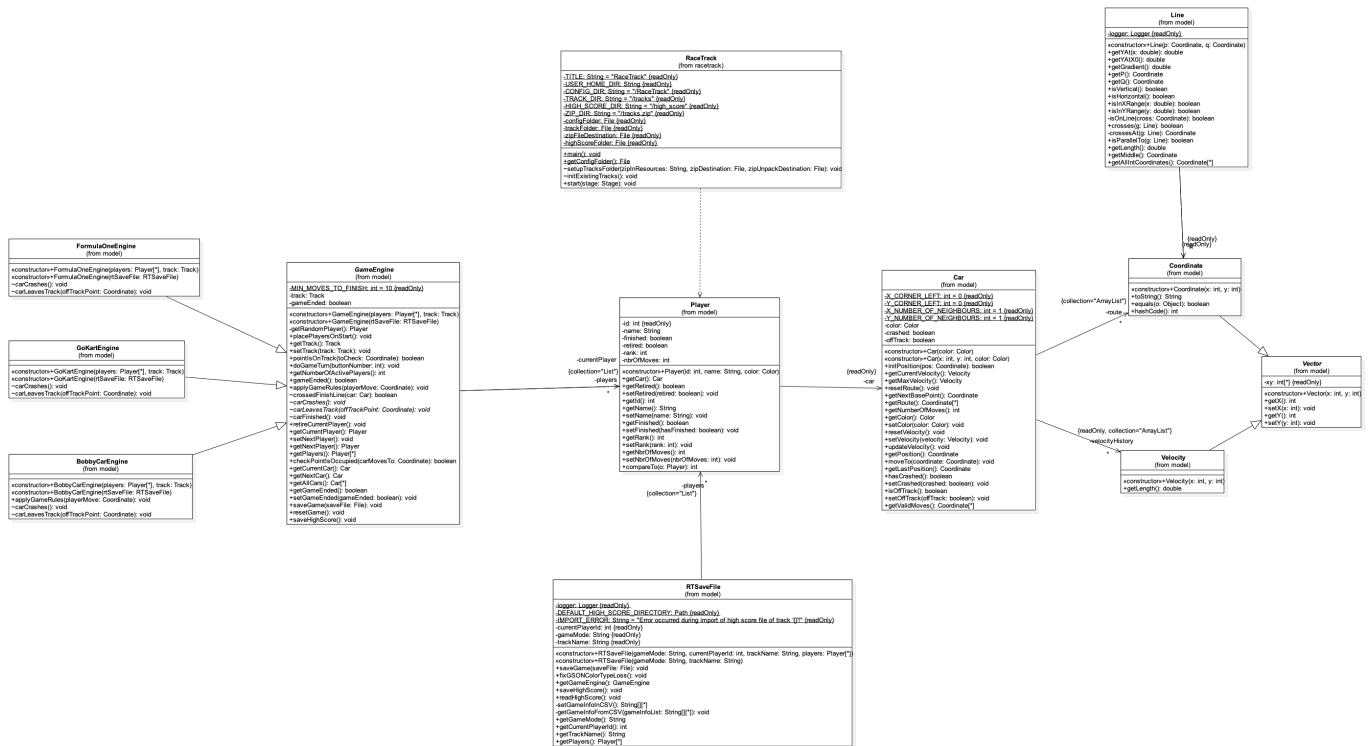


Figure 3.3: Design Class Diagram of RaceTrack

3.2.2. Selected Interaction Diagrams

3.2.2.1. Starting a new game from the main menu

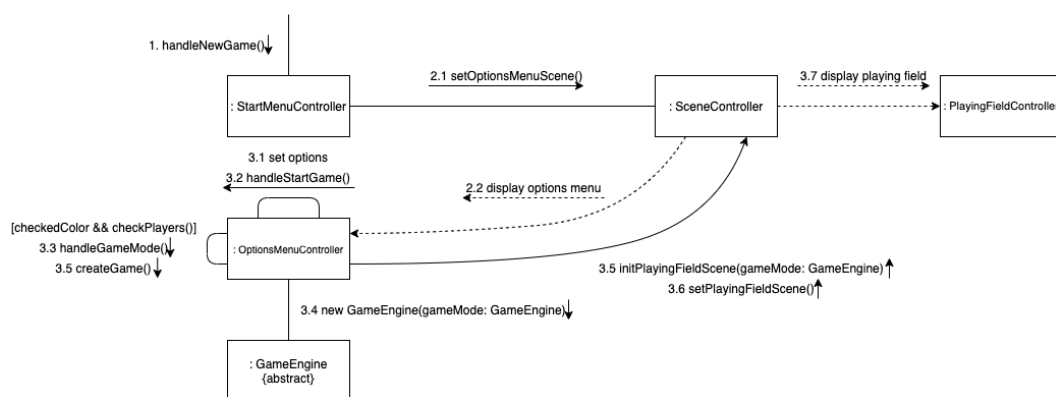


Figure 3.4: Communication Diagram for action "Starting a new game from the main menu"

3.2.2.2. Executing a turn in the game

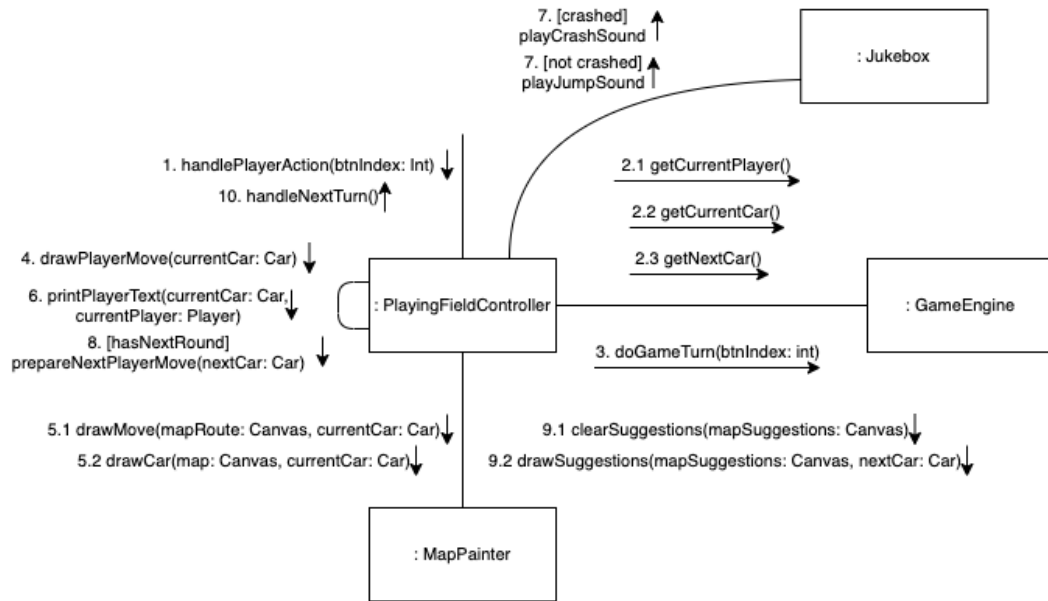


Figure 3.5: Communication Diagram for action "Executing a turn in the game"

3.2.2.3. Last player crosses the finish line

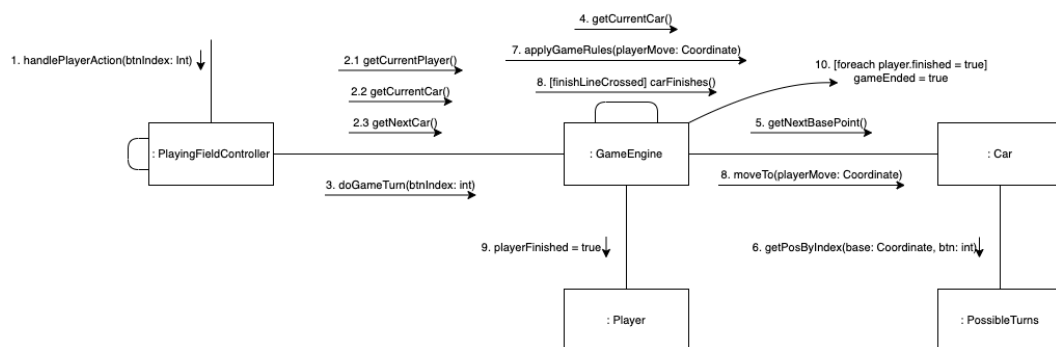


Figure 3.6: Communication Diagram for action "Last player crosses the finish line"

Chapter 4 Implementation

The most important details of the implementation of the solution as well as the performed tests are documented in this chapter.

4.1. Code Implementation

The work between the developers was distributed by using Kanban Board Boards within the *Projects* function in GitHub, it also gave the project manager the ability to track the current status of the project. GitHub (hosted by ZHAW) was also used as the remote repository for the team. *Azure Pipelines* and *SonarCloud* were used for *Continuous Integration and Deployment* and ensuring code quality and security respectively. The most recent build of RaceTrack gets automatically built through the use of *Azure Pipelines*.

As described in section *Software Architecture*, RaceTrack implements a layered architecture, which can also be taken from the project structure in the source code itself:



Figure 4.1: Java Project Structure of RaceTrack

4.1.1. Packages

4.1.1.1. com.pathfinder.racetrack.model

A *GameEngine* represents the brain of a game session, each game mode (currently three in total) extends the abstract class *GameEngine* with its extensions. There are currently three game modes available.

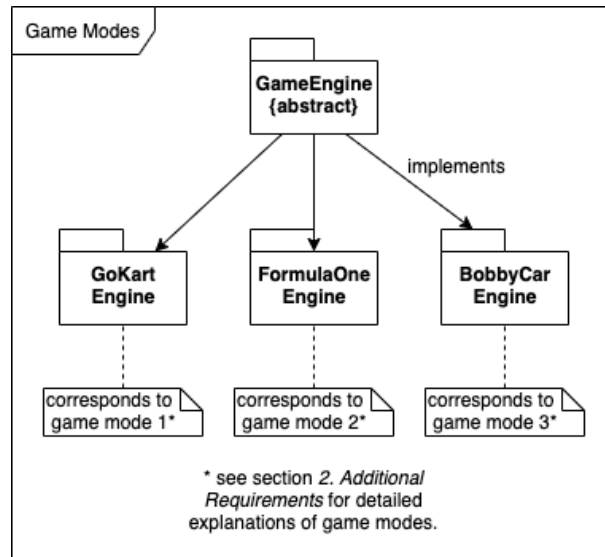


Figure 4.2: Implementation Overview of the different Game Modes

As per layered architecture, the models are being updated in the domain logic, a *GameEngine* in RaceTrack to be exact. But there are still some connections and dependencies between some models themselves: e.g. *Velocity* and *Coordinate* both extend the abstract class *Vector*, which is used by *Car*, which is also owned by a *Player*.

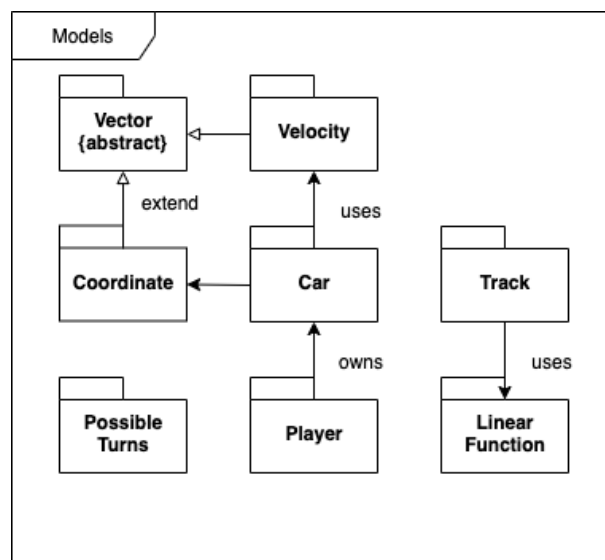


Figure 4.3: Implementation Overview of the different Game Models

4.1.1.2. com.pathfinder.racetrack.view

Player input through the GUI is handled by the *controllers*. Each controller is responsible for a different section of the GUI (e.g. *StartMenuController* for input in the main menu), except for the *SceneController*, which manages what menu is currently viewed by the player.

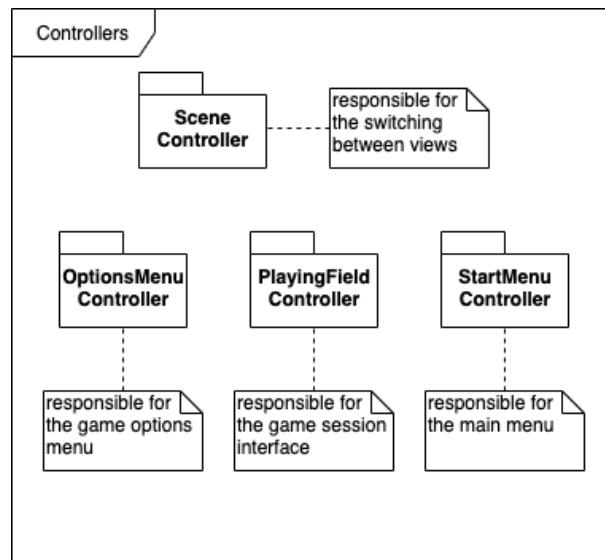


Figure 4.4: Implementation Overview of the different Game Controllers

The *View* package also holds the classes *Jukebox* and *MapPainter*, which are both only utility classes for the actual user interfaces (*Jukebox* for adding sounds effects and *MapPainter* for drawing the playing field).

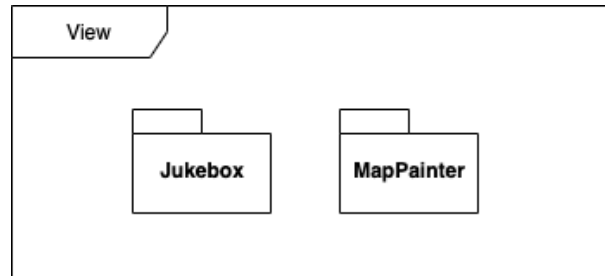


Figure 4.5: Implementation Overview of the View Components

The actual interfaces are being built by **JavaFX** through *.fxml* files, which are located in the *resources* directory in the source code.

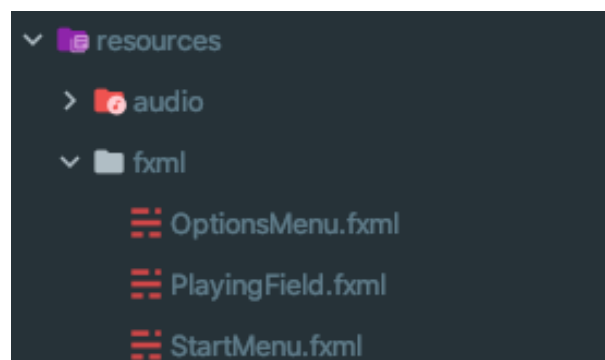


Figure 4.6: Implementation Overview of the View Components

4.2. Code Quality

RaceTrack is being developed using the leading cloud-based solution for continuous analysis of code quality, reliability, and security **SonarCloud**. Not only does **SonarCloud** automatically detect bugs, vulnerabilities, code smells and, other issues, it also elevates the team's coding quality to a new standard.



Figure 4.7: SonarCloud Summary for RaceTrack

RaceTrack currently maintains the highest Reliability Rating **A**, the highest Security Rating **A** and an outstanding Test Coverage of **81.5%**, and all of that still in the Beta Phase, with only room for further improvement.

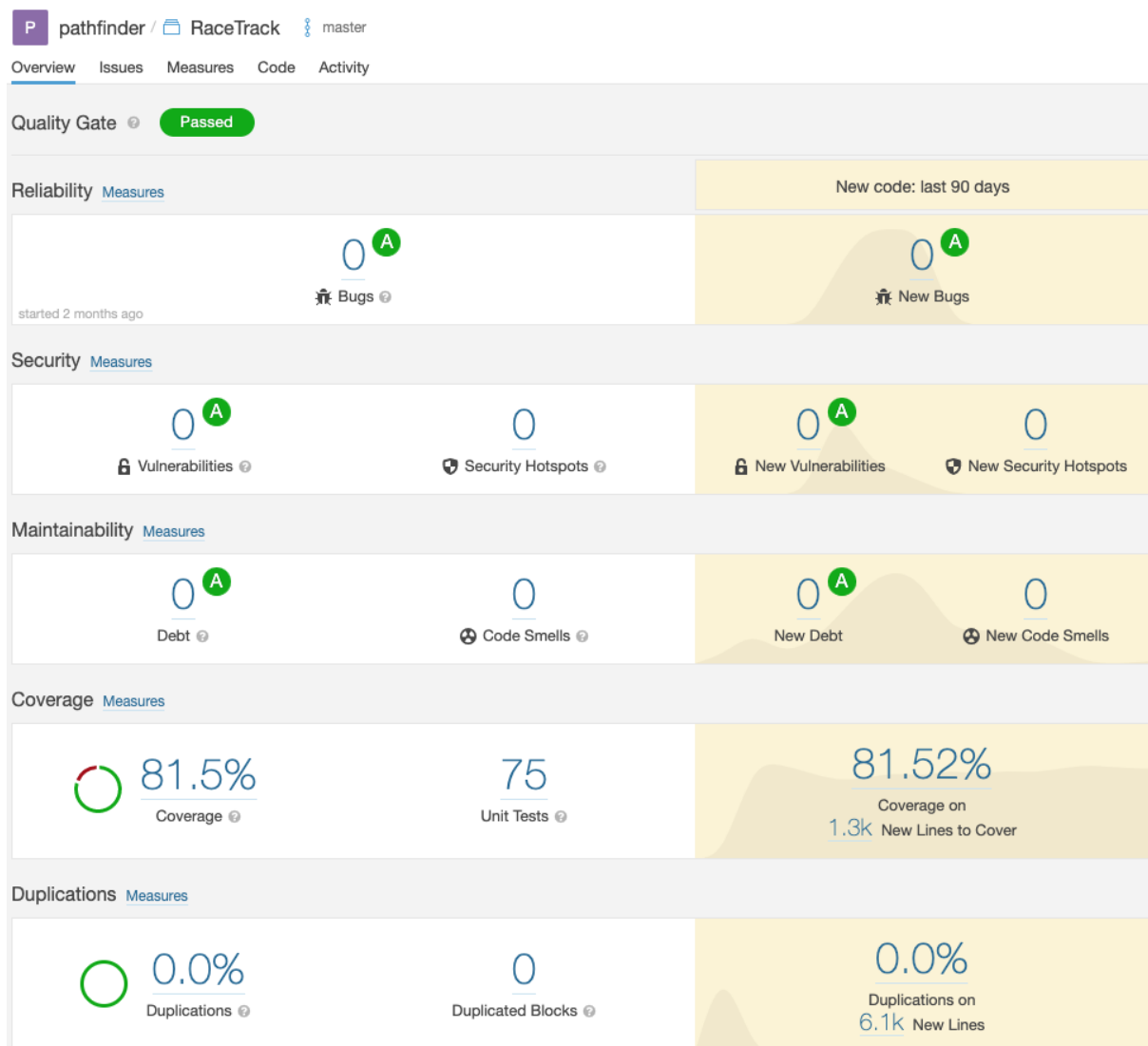


Figure 4.8: SonarCloud Dashboard for RaceTrack

In the current build version of RaceTrack, there are also no *code smells* (code that is confusing or difficult to maintain) present.

4.3. Code Documentation

During the development of RaceTrack it was seen to pay close attention to the concepts of Clean Code (See PDF *PROG1 Clean-Code-Regeln*). All public methods and attributes have also been noted with JavaDoc comments, while private methods and attributes have been commented if deemed necessary. The most recent JavaDoc of RaceTrack can be found at the GitHub repository (https://github.zhaw.ch/PathFinder/PSIT3-FS20-IT18ta_WIN-Team5) under the section *Documentation*.



Figure 4.9: Documentation section in the GitHub repository

Package com.pathfinder.racetrack.controller

Class	Description
BobbyCarEngine	Variation of the game RaceTrack, after a crash the position of the car resets to the last valid one.
CreditsController	Controller of the Credits fxml-view
FormulaOneEngine	Variation of the game RaceTrack, after a car crashed the player retires from the game
GameEngine	Contains the logic and rules of the game
GoKartEngine	Variation of the game RaceTrack, after a crash the car continues with the minimum velocity and no acceleration until the track is reached again
HighScoreController	Controller of the high score fxml-view
OptionsMenuController	This class represents the logic of the options menu
PlayingFieldController	Controller of the actual game
RulesController	Controller of the rules fxml-view
SceneController	This class controls the different scenes of the UI (User Interface)
StartMenuController	This class represents the logic of the start menu
TracksMenuController	Class representing the logic for the Tracks Menu

Copyright © 2020. All rights reserved.

Figure 4.10: JavaDoc extract of RaceTrack

4.4. Test Concept

Tests for all two main packages *model* and *view* have been created and executed, the created test cases can be found in the directory `src/test/java/com.pathfinder.racetrack`.

In the *view* package, the different JavaFX GUI controllers (holding event handlers and more) are tested (e.g. *OptionsMenuTest* or *TracksMenuTest*) amongst all the helper classes for the view (e.g. *ExceptionDialogTest* or *JukeboxTest*). In the *model* package, all the different used models are tested (e.g. *GameEngineTest*, *CarTest*, or *VelocityTest*).

A total of **75** unique test cases are tested and verified with a total test coverage of **90%** on Class level, **87%** on Method level and **83%** on Line level, with all scopes exceeding the **80%** test coverage threshold:

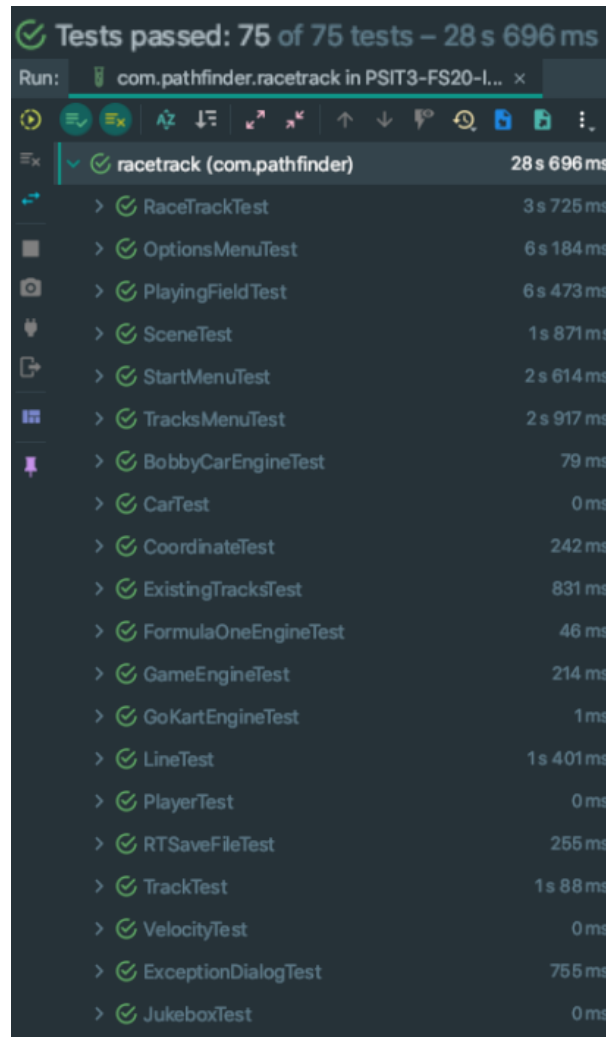


Figure 4.11: Overview of test cases in RaceTrack

Coverage: com.pathfinder.racetrack in PSIT3-FS20-IT18ta_... x

90% classes, 83% lines covered in 'all classes in scope'

Element	Class, %	Method, %	Line, %
com.pathfinder.racetrack	90% (28/31)	87% (267/306)	83% (1153/1374)

Figure 4.12: Test Coverage in RaceTrack

4.5. Installation Guide

The most recent builds of RaceTrack for **macOS** and **Windows** (x64) can always be found on the GitHub repository (https://github.zhaw.ch/PathFinder/PSIT3-FS20-IT18ta_WIN-Team5) under the section *Nightly Build*:

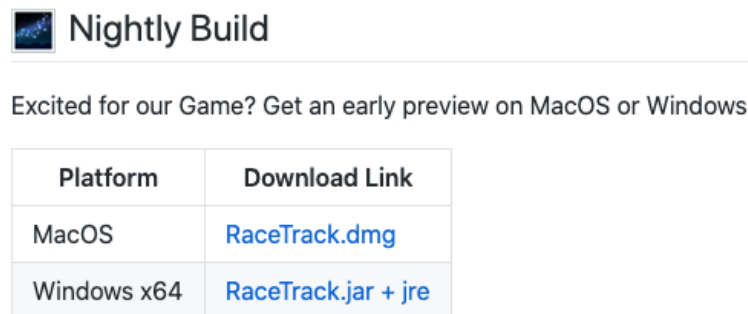


Figure 4.13: Documentation section in the GitHub repository

4.5.1. macOS

1. Execute the *RaceTrack 1.0.dmg* file and copy *RaceTrack.app* to your *Applications* folder.

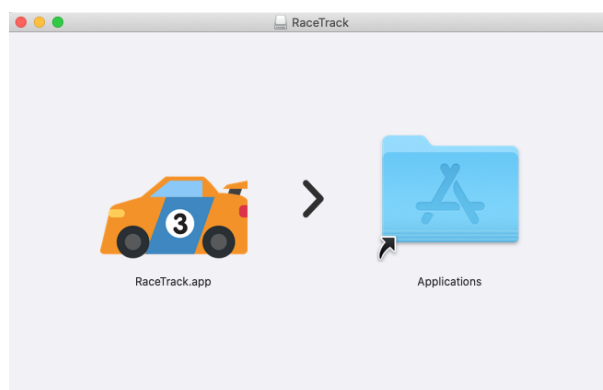


Figure 4.14: Installation Guide for macOS: Copy to Applications folder

2. On the first time RaceTrack needs to be open via right-click, or else you'd not be able to skip the verification error message.

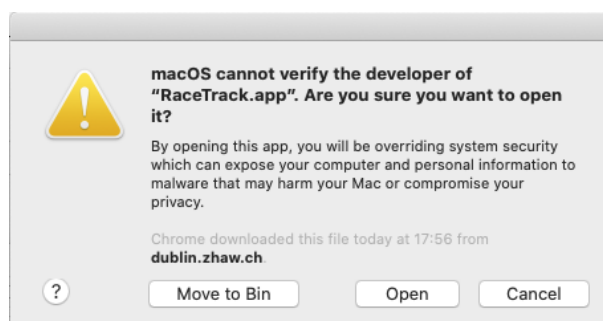


Figure 4.15: Installation Guide for macOS: Skip verification error

3. After the first time, RaceTrack can be opened like any other application on your Mac.

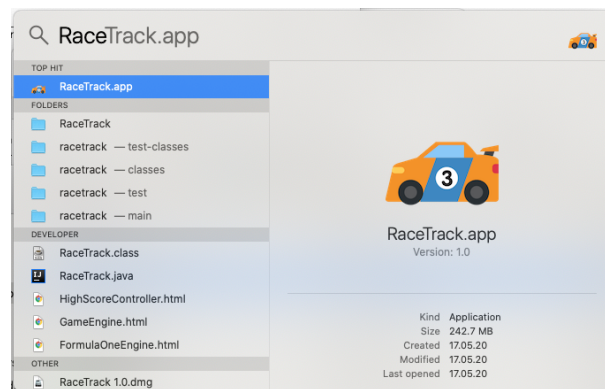


Figure 4.16: Installation Guide for macOS: Start RaceTrack via Spotlight

4.5.2. Windows

1. Unzip the *RaceTrack_Windows64.zip* archive and execute *RaceTrack.exe* in the just unzipped folder.

Name	Date modified	Type	Size
jre	5/17/2020 6:10 PM	File folder	
config.json	5/17/2020 3:42 PM	JSON File	1 KB
RaceTrack.exe	5/17/2020 3:42 PM	Application	342 KB
RaceTrack-0.0.0-jar-with-dependencies.jar	5/17/2020 3:16 PM	JAR File	15,424 KB

Figure 4.17: Installation Guide for Windows: Unzip folder and execute .exe

4.6. Source Code

The complete source code (including Unit Tests) can be found attached to this documentation *Implementation_PSIT3-FS20-IT18ta-WIN_Team5.zip* or on GitHub (<https://github.zhaw.ch/PathFinder/PSIT3-FS20-IT18ta-WIN-Team5>).

Chapter 5 Results

The project results are summarized in this chapter concerning the original project idea.

5.1. Achieved Goals

All of the teams' original goals for this project have been achieved, better put all planned use cases for this project have been implemented:

- Main Use Cases:
 - ☒ Use Case UC7: Play one Game Session
A round of RaceTrack can successfully be played with up to four people simultaneously. The track is being recognized correctly and the velocities gained are also calculated accurately. The crossing of the finish line is recognized error-free, too.
 - ☒ Use Case UC13: Load Track
Additional custom tracks, which comply with all the rules for a custom track, can be imported without a hassle. Invalid tracks are correctly refused with an appropriate error message. Tracks can be deleted too, except for the default tracks in the game.
- Secondary Use Cases:
 - ☒ Use Case UC1: View High Score
The high score of each track and game mode can successfully be displayed, this is also true for additionally added custom tracks. The correct track and game mode is also displayed correctly after the end of a game session.
 - ☒ Use Case UC4: Set Game Options
Game options before starting a new game can accurately be set and the new game starts with the correct configuration. The game starts correctly with the precise number of players entered.
 - ☒ Use Case UC10: Load Game Session
A previously saved game session (*.rtsave* file) can be imported into the game error-free with the exact game state as before the save. The continued game session behaves without any errors or anomalies.
 - ☒ Use Case UC11: Save Game Session
A currently running game session can correctly be saved as a *.rtsave* file.
- Minor Use Cases:
 - ☒ Use Case UC2: View Rules
Rules are being displayed correctly on the rules page.
 - ☒ Use Case UC3: View Credits
Credits are being displayed correctly in the credits page.
 - ☒ Use Case UC6: Reset Game Session
A running game session can successfully be reset. The reset game session behaves without any errors or anomalies.

5.2. Open Issues

Currently, there are only two unresolved issues still open for RaceTrack:

- A confirmation should be needed before the system resets a running game session.
- The game should be restartable after the High Score window is being displayed.

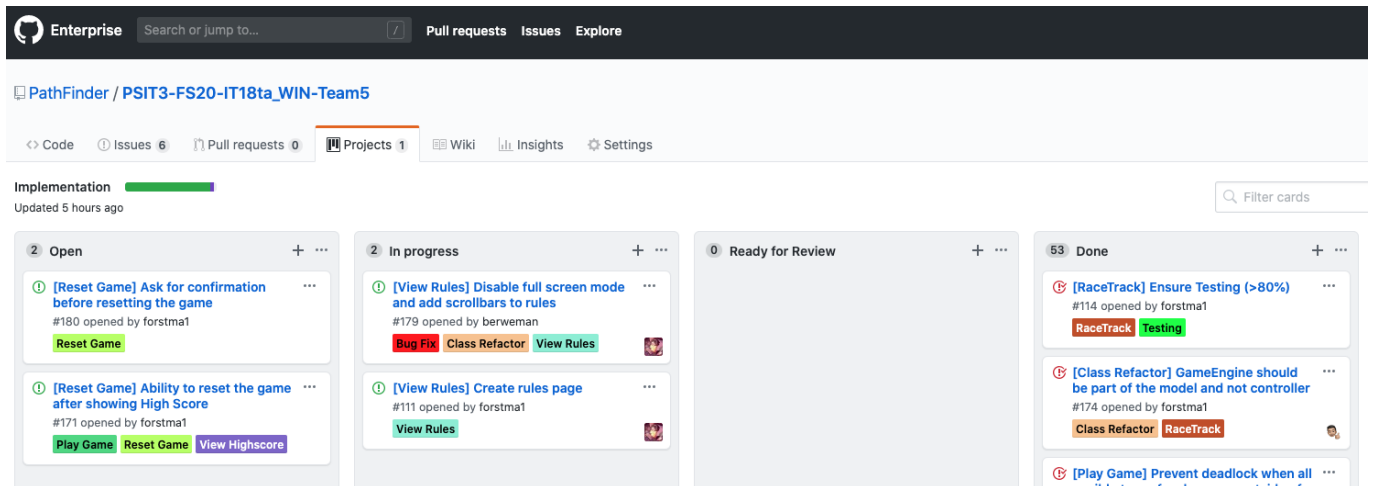


Figure 5.1: Current Open Issues for RaceTrack on GitHub

5.3. Future Developments

The future possibilities for RaceTrack are endless, there are a lot of additional features that can be implemented:

- Special items that can be placed on the track (boosts, obstacles, etc.)
- Touch support
- Ability to zoom into the game
- Manual modification of the grid density for a track
- Additional game modes and tracks
- A new color scheme
- Built-In editor for custom tracks
- and many more...

Chapter 6 Appendix

- Project Management
- Registers
- Attachments

6.1. Project Management

Listing of the rough timeline of the iterations. The planned expenditure and goals are listed together with the actual expenditure and achieved goals per iteration. After each iteration an iteration-assessment is being created, where reached goals are analyzed and additional measures may be defined. The detailed planning for the next iterations and risks have been constantly updated.

Phase	Iteration	Start/Duration [Week]	Goals	Planned [h]	Work	Actual Work [h]
Inception	1	1 / 2	Project draft created, use cases defined, domain model drafted	60 h		50 h
Milestone	M1	End of week 2	Vision, goals and requirements defined	Phase 1 Total: 60 h		50 h
Elaboration	2	3 / 2	Use case model finished, UI prototype present	60 h		70 h
	3	5 / 2	Domain model finalized, main use cases first iteration implemented	60 h		80 h
	4	7 / 2	Prototype finished, architecture stable, main use cases finished	60 h		80 h
Milestone	M2	End of week 8	Architecture as PoC verified	Phase 2 Total: 180 h		230 h
Construction	5	9 / 2	Secondary use cases finalized	80 h		80 h
	6	11 / 3	Minor use cases finalized, user manual finished	100 h		120 h
Milestone	M3	End of week 13	Beta Release of RaceTrack	Phase 3 Total: 180 h		200 h
				Project Total: 420 h + 80 h reserve		480 h

Table 6.1: Project Timeline

6.1.1. Construction Phase: 20.04.2020 - 24.05.2020


6.1.1.1. Milestone M3: 22.05.2020

Start/Duration [Week]: End of week 13



Planned Work [h]: 180 h

Actual Work [h]: 200 h

(Achieved) Goals:

-  Beta Release of RaceTrack

Additional Measures:

-  Milestone has been completed successfully.
-  End of project.





6.1.1.2. Iteration 6: 04.05.2020 - 24.05.2020

Start/Duration [Week]: 11 / 3



Planned Work [h]: 100 h

Actual Work [h]: 120 h

(Achieved) Goals:

-  (Secondary) Use Case UC1 (View High Score) finalized
-  (Minor) Use Cases finalized
-  RaceTrack manual finished
-  Final report finished

Additional Measures:

-  All for this iteration planned goals have been achieved.
-  All implementation work on RaceTrack finished.





6.1.1.3. Iteration 5: 20.04.2020 - 03.05.2020

Start/Duration [Week]: 9 / 2




Planned Work [h]: 80 h

Actual Work [h]: 80 h

(Achieved) Goals:

-  (Secondary) Use Case UC4 (Set Game Options) finalized
-  (Secondary) Use Case UC1 (View High Score) finalized
-  (Secondary) Use Case UC14 (Save Game Session) finalized
-  (Secondary) Use Case UC15 (Load Game Session) finalized

Additional Measures:

-  Almost all for this iteration planned goals have been achieved.
-  Implementation of UC1 not finished, still a bug with the view of the high score after the end of the game session.
-  No additional measures needed, UC1 almost done.

6.1.2. Elaboration Phase: 09.03.2020 - 19.04.2020

Elaboration phase has been extended by an additional iteration because of the ongoing COVID-19 pandemic and Easter holidays.

6.1.2.1. Milestone M2: 17.04.2020

Start/Duration [Week]: End of week 8



Planned Work [h]: 180 h

Actual Work [h]: 230h

(Achieved) Goals:

-  Architecture as PoC verified

Additional Measures:

-  Goal for this milestone has been achieved.
-  50 h more used during the additional iteration, but main use cases finished.







6.1.2.2. Iteration 4: 06.04.2020 - 19.04.2020

Start/Duration [Week]: 7 / 2





Planned Work [h]: 60 h

Actual Work [h]: 80 h

(Achieved) Goals:

-  Finished prototype
-  Finished *Solution Architecture* documentation
-  Development pipeline set upped
-  Architecture stable
-  (Main) Use Case UC7 (Play one Game Session) finalised
-  (Main) Use Case UC13 (Load Track) finalised

Additional Measures:

-  All for this iteration planned goals have been achieved.
-  Iteration not part of the original timeline added additionally during the COVID-19 crisis.
-  20 h more needed as planned, as finishing documentation and main use cases needed more time than planned.
-  No additional measures needed, as the finished prototype already has a lot of features implemented.





6.1.2.3. Iteration 3: 23.03.2020 - 05.04.2020

Start/Duration [Week]: 5 / 2





Planned Work [h]: 60 h

Actual Work [h]: 80 h

(Achieved) Goals:

-  Use Case UC7 (Play one Game Session) first implementation realized
-  Use Case UC13 (Load Track) first implementation realized
-  Domain model finalized
-  System Sequence Diagrams for main use cases finished

Additional Measures:

-  All for this iteration planned goals have been achieved.
-  Domain Model has already been finalized in the last iteration.
-  20 h more needed as planned, as the main use case UC7 is by far the biggest and most complex use case to implement.
-  No additional measures needed, as other use cases are less complex to implement.





6.1.2.4. Iteration 2: 09.03.2020 - 22.03.2020

Start/Duration [Week]: 3 / 2





Planned Work [h]: 60 h

Actual Work [h]: 70 h

(Achieved) Goals:

-  Use cases specified in detail
-  UI prototype present
-  Use Case Model finished
-  First iteration of the domain model drafted

Additional Measures:

-  All for this iteration planned goals have been achieved.
-  Domain Model has already been finalized.
-  10 h more needed as a lot of design- and architectural questions needed to be answered.
-  No additional measures needed.

6.1.3. Inception Phase: 24.02.2020 - 08.03.2020




6.1.3.1. Milestone M1: 06.03.2020

Start/Duration [Week]: End of week 2



Planned Work [h]: 60 h

Actual Work [h]: 50 h

(Achieved) Goals:

-  Vision defined
-  Goals defined
-  Requirements defined

Additional Measures:

-  All goals for this milestone have been achieved.
-  No additional measures needed.







6.1.3.2. Iteration 1: 24.02.2020 - 08.03.2020

Start/Duration [Week]: 1 / 2





Planned Work [h]: 60 h

Actual Work [h]: 50 h

(Achieved) Goals:

-  *Project Outline* finished
-  Project draft created
-  Use cases defined
-  Domain model drafted
-  Architecture drafted
-  Development pipeline set up finished

Additional Measures:

-  Almost all for this iteration planned goals have been achieved.
-  Set-up of development pipeline not possible yet, as additional planning is still needed.
-  Goals achieved with 10 hours less than planned.
-  Development pipeline will be configured during the next iteration and shouldn't be a problem, as an additional 10 h development time will be taken as a reserve from the last iteration.

6.2. References

Number	Reference
[1]	Microsoft. (2020). Microsoft Store - Search Results [Online]. URL: https://www.microsoft.com/en-us/search?q=vector+race [Accessed 28. Feb. 2020].
[2]	Apple. (2020). Games - Mac App Store Downloads on iTunes [Online]. URL: https://apps.apple.com/us/genre/mac-games/id12006?mt=12 [Accessed: 28. Feb. 2020].
[3]	Google. (2020). Google Play - Search Results [Online]. URL: https://play.google.com/store/search?q=vector%20race [Accessed 28. Feb. 2020].

Table 6.2: References

Chapter 6 Listings

2.1 Example of a Save File 24

Chapter 6 List of Figures

1.1	Turn in the game (Mock-up)	5
2.1	Overview of all defined use cases of RaceTrack	13
2.2	UI Sketch for Use Case UC7	15
2.3	System Sequence Diagram of Use Case UC7	16
2.4	Example of a valid track to be used in Use Case UC13	18
2.5	UI Sketch for Use Case UC13	18
2.6	System Sequence Diagram of Use Case UC13	19
2.7	UI Sketch of Use Case UC1	20
2.8	UI Sketch for Use Case UC4	22
2.9	UI Sketch for Use Case UC15	23
2.10	Example of the environment in the game	26
2.11	Example of how to move around the game	27
2.12	Domain Model of RaceTrack	29
3.1	Layered implementation in RaceTrack	31
3.2	Package Diagram of RaceTrack	32
3.3	Design Class Diagram of RaceTrack	33
3.4	Communication Diagram for action "Starting a new game from the main menu"	33
3.5	Communication Diagram for action "Executing a turn in the game"	34
3.6	Communication Diagram for action "Last player crosses the finish line"	34
4.1	Java Project Structure of RaceTrack	36
4.2	Implementation Overview of the different Game Modes	37
4.3	Implementation Overview of the different Game Models	37
4.4	Implementation Overview of the different Game Controllers	38
4.5	Implementation Overview of the View Components	38
4.6	Implementation Overview of the View Components	38
4.7	SonarCloud Summary for RaceTrack	39
4.8	SonarCloud Dashboard for RaceTrack	39
4.9	Documentation section in the GitHub repository	40
4.10	JavaDoc extract of RaceTrack	40
4.11	Overview of test cases in RaceTrack	41
4.12	Test Coverage in RaceTrack	41
4.13	Documentation section in the GitHub repository	42
4.14	Installation Guide for macOS: Copy to Applications folder	42

4.15	Installation Guide for macOS: Skip verification error	42
4.16	Installation Guide for macOS: Start RaceTrack via Spotlight	43
4.17	Installation Guide for Windows: Unzip folder and execute .exe	43
5.1	Current Open Issues for RaceTrack on GitHub	47

Chapter 6 List of Tables

6.1	Project Timeline	50
6.2	References	55

Chapter 6 Glossary

Add-ons A usually commercially distributed add-on for a video game (usually extended story line, new game areas or objects). 11

Android Android is a mobile operating system based on a modified version of the Linux kernel and other open source software, designed primarily for touchscreen mobile devices such as smartphones and tablets. 7

Azure Pipelines Azure Pipelines is a cloud service that people can use to automatically build and test code project and make it available to other users. Azure Pipelines combines continuous integration (CI) and continuous delivery (CD) to constantly and consistently test and build code and ship it to any target. 9, 36

Car A players character, which will be moved around during a race. Are differentiated by different colors. 29

Continous Integration and Deployment Continuous Integration (CI) is the practice of merging all developers' working copies to a shared mainline several times a day. Continuous deployment (CD) is a software engineering approach in which software functionalities are delivered frequently through automated deployments.. 9, 36

Google Play Store Google Play, formerly Android Market, is an American digital distribution service operated and developed by Google. It serves as the official app store for devices running on "Google certified" Android operating system.. 7

GridPaper The overlaying grid put on top of the track during a round of RaceTrack. Players will only be able to move around the grid points provided by the GridPaper. 29

GUI Short for Graphical User Interface. A GUI displays objects that convey information, and represent actions that can be taken by the user. 26

Image The actual image which holds the track and starting/finish line. The image defines the track and environment by colors. 29

Java Java is a general-purpose programming language that is class-based, object-oriented, and designed to have as few implementation dependencies as possible. It is intended to let application developers write once, run anywhere (WORA), meaning that compiled Java code can run on all platforms that support Java without the need for recompilation.. 7

Kanban Board A Kanban board is one of the tools that can be used to implement Kanban to manage work at a personal or organizational level. Kanban boards visually depict work at various stages of a process using cards to represent work items and columns to represent each stage of the process. Cards are moved from left to right to show progress and to help coordinate teams performing the work. A Kanban board may be divided

into horizontal "swimlanes" representing different kinds of work or different teams performing the work. 9, 36

Microtransactions Microtransactions are a business model where users can purchase virtual goods with micropayments. Microtransactions are often used in free-to-play games to provide a revenue source for the developers.. 11

Player A player in the game (no NPCs). Each player controls a Car during a race. 29

PossibleTurn A possible turn (also called possible move) is an action a player can take during his turn of a round. Generally a player will be able to choose one out of nine possible actions to take, where his car should be moving to. 29

Route The past moves of each player will represent its car's route. The route will be represented as a trailing line behind each car. 29

SonarCloud SonarCloud is the leading product for Continuous Code Quality online, totally free for open-source projects. It supports all major programming languages, including Java, JavaScript, TypeScript, C#, C++ and many more. 9, 36

Start/Finish Line Defines the starting and end points of the track. 29

Track The actual track where the cars will drive on. Can be imported by the player from a valid bitmap image. The track defines drivable points, track boundaries and starting/finish line. 29