Requirements Engineering
SEM-Group 86

Ivar de Bruin 4944135
Tim Anema 4953940
Laura Pircalaboiu 4777778
Marc Otten 4872541
Ilya Grishkov 4770811

November 29, 2019

Contents

1 Requirement Engineering

1.1 Functional requirements

1.1.1 Must have

1. The starting player must be chosen using a random number generator.
2. There must be a GUI showing the current position of the balls on the table and the position of the cue stick.
3. When aiming your shot, there must be a line indicating where the cue ball will go.
4. There must be a menu showing scores.
5. At the end of the game the score must be uploaded to the database with a name entered at that point.
6. There must be a way to play locally against other players.
7. At the end of each game the top five scores ever should be shown.
8. The user must be able to register for an account.
9. The user must be able to authenticate.

Game Rules(8-ball)

10. When the cue ball is potted, the other player must place it back in a place of his/her choosing.
11. Each player must be assigned either stripes or solids.
12. These types must be assigned based on the first ball potted after the break shot.
13. When the first ball hit by a player is not his/her, it is counted as a foul and his/her turn ends and the other player may place the cue ball.
14. The black ball belongs to no player unless he/she has potted all of their assigned balls(striped or solid).
15. When the black ball is potted before all other balls belonging to that player are potted, said player will lose the game immediately.
16. When a ball is potted it must be removed from the table.
17. When the cue ball is potted it must instead be put back by the other player whose turn then starts.

Collisions

18. Collision between balls must be handled according to proper physics.
19. When a ball hits the wall it must bounce off correctly.

1.1.2 Should have

20. There should be a settings page for graphical and sound settings.
21. There should be statistics kept.
22. There should be animations indicating the movement of balls.
23. There should be sounds when collisions occur with balls, walls, pockets or cues.
24. There should be an option to play online against another human player.

1.1.3 Could have

25. A tutorial to explain the rules and controls to a new player could be implemented.
26. There could be spin physics.
27. There could be an option to chat with your opponent.
28. There could be additional prediction lines after hitting the first ball.
29. There could be camera controls.

Other game types

30. There could be an option to play against an AI.
31. There could be an option to play 9-ball.
32. There could be an option to play snooker.

1.1.4 Won't have

33. We will not implement jump shots.
34. We won't have micro-transactions.

1.2 Non-Functional requirements

1. We need to use Java.
2. Players must authenticate using a username and password.
3. Players should be able to delete their data in compliance with GDPR.
4. The application should run stable on the latest stable release of Windows 10, OS X and Linux (Manjaro and Arch Linux)
5. The server container could be horizontally scaleable

1.2.1 Database

6. A SQL database must be used to store scores and accounts.
7. Interaction with the database must go through JDBC.
8. JDBC's prepared statement must be used to prevent security vulnerabilities.

2 Modelling use cases

2.1 Logging in

2.1.1 Description

- Use Case: Logging in
- Author: Tim Anema
- Date: 28/11/2019
- Purpose: Authenticating with the server.
- Overview: Once the user opens the client, the client requests the user's credentials. Once the user has entered their credentials, the client will attempt to verify the user's credentials by contacting the server. If the verification is successful, the client will be able to start a game; otherwise (unsuccessful verification) a message is shown to the user.
- Cross-reference: Functional Requirement: 9
- Actors: User and server.
- Pre-conditions:
  - The user must have opened the client.
  - The user must have indicated he/she wants to login.
- Post-conditions:
  - A token was generated for use by the client
  - A refresh token was generated for use by the client
- Flow of events: See table below

User Actions	System Action	Server Action
1. User enters credentials		
	2. Client sends credentials to server	
		3. Server verifies credentials
		4. Server generates a token and a refresh token
		5. Server sends token to client
	6. Client stores token	
7. User can continue.		

2.1.2 UML Diagram

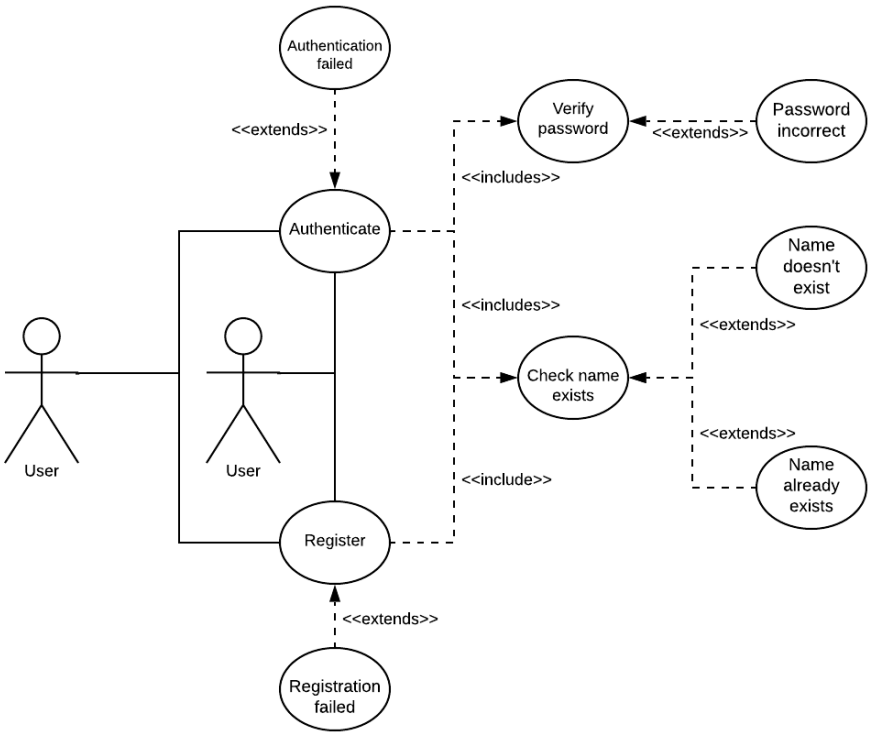

Figure 1: Logging in use case (upper use case)

2.2 Registering a new player

- Use Case: Registering
- Author: Tim Anema
- Date: 28/11/2019
- Purpose: Registering a new account.
- Overview: Once the user opens the client and indicates he/she wants to register a new account, the client asks for the required information (username and password). Once the user has entered the required information, the client will sends the information to the server. If the registration is succesful, the client will be able to login to their new account; otherwise (unsuccessful registration) a message is shown to the user.

- Cross-reference: Functional Requirement: 8
- Actors: User and server.
- Pre-conditions:
  - The user must have opened the client.
  - The user must have indicated he/she wants to register.
- Post-conditions:
  - A new user account should be saved in the database
- Flow of events: See table below

User Actions	System Action	Server Action
1. User sends information		
	2. Client sends information to server	
		3. Server makes sure the name is not taken
		4. Server saves a new account in the database
		5. Server sends a confirmation to client
6. User can continue		

2.2.1 UML Diagram

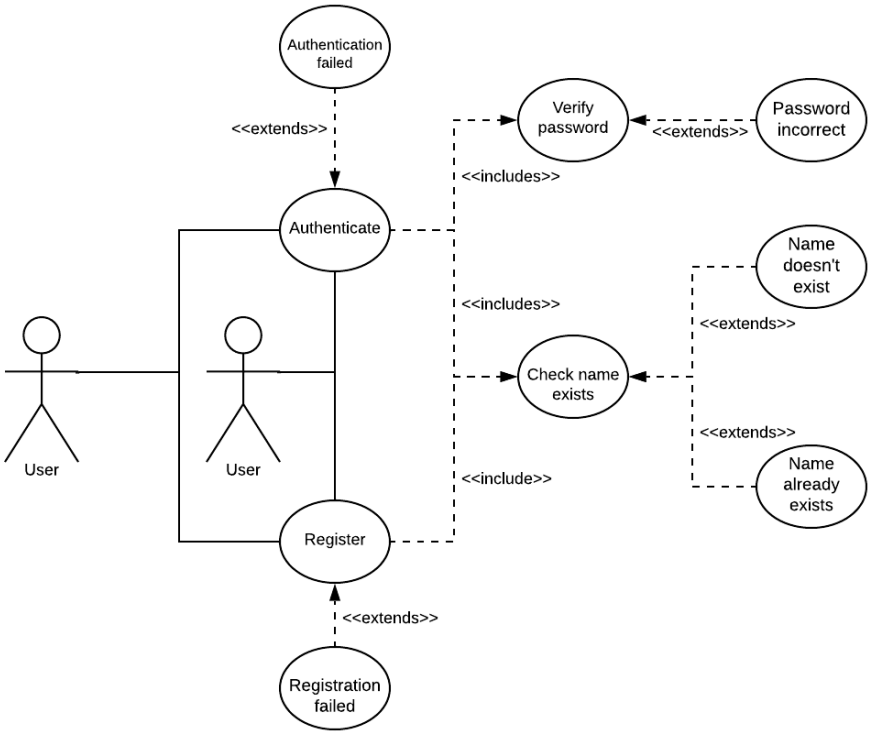

Figure 2: Registering use case (lower use case)

2.3 Starting a game

2.3.1 Description

- Use Case: Starting the game.
- Author: Marc Otten.
- Date: 28/11/2019
- Purpose: Starting and setting up the game after user decides to start playing.
- Overview: After logging in and authenticating the user can start a game. When the user presses start game the game will be set up by the application using the proper rules and objects.

- Cross-reference: Functional Requirement: 2, 6
- Actors: Players.
- Pre-conditions:
  - The user must be logged in and use a valid session.
- Post-conditions:
  - The game must be started.
  - The game must use the correct objects.
  - The game must use the correct rules.
- Flow of events: See table below

User actions	System actions
User chooses start game	
	System starts a game
	System adds table, cue and balls to the game
	System adds rules to the game
	System initializes the final game
User can now start playing	

Figure 3: Table of event flow for starting a game.

2.3.2 UML Diagram

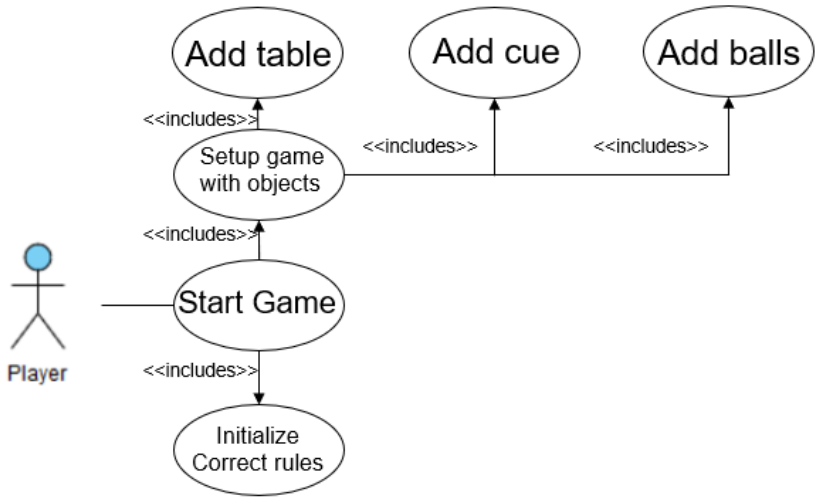

Figure 4: Starting a game use case diagram

2.4 Hitting a ball

2.4.1 Description

- Use Case: A ball gets hit by a cue or another ball.
- Author: Marc Otten.
- Date: 28/11/2019
- Purpose: Handling physics and collisions correctly.
- Overview: When a ball gets hit by either a cue or another ball the collison gets handled correctly. This mean that it applies the physics rules we implemented and handles the collisions correctly. After the ball gets hit its momentum changes. It also checks on the first with the cue ball if the ball is of the correct type, if not it is considered a foul.
- Cross-reference: Functional Requirement: 2, 13, 14, 18
- Actors: Players.
- Pre-conditions

  - The game must be started.
  - It must be the player his/her turn.
- Post-conditions:
  - The ball must have the correct momentum.
- Flow of events: See table below

User actions	System actions
User starts turn	
User hits cue ball with the cue	
	System gives cue ball momentum
	System checks collisions
	System handles collisions according to physics
	System shows trajectory of ball(s)

Figure 5: Table for event flow of hitting a ball.

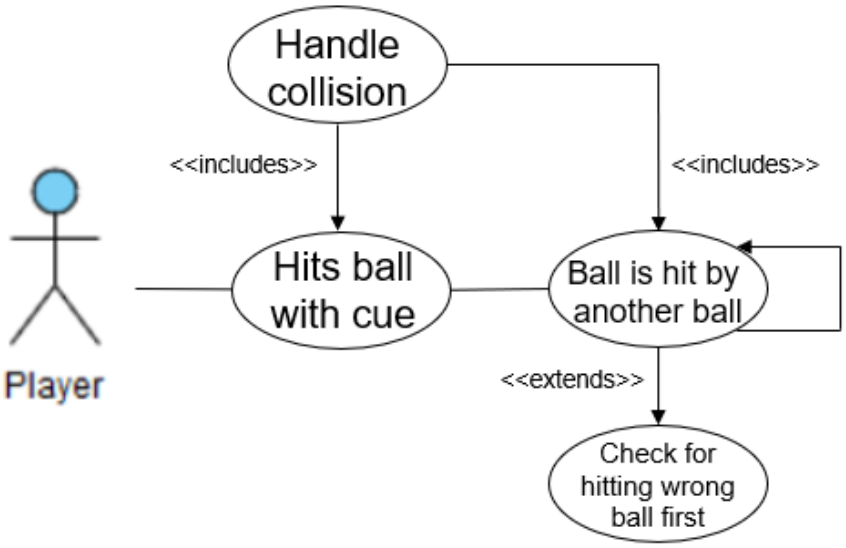

Figure 6: Hitting a ball use case diagram

2.5 Pocketing a ball

2.5.1 Description

- Use Case: Pocketing a ball.
- Author: Laura Pircalaboiu.
- Date: 28/11/2019
- Purpose: Pocketing a ball and deciding game flow afterwards.
- Overview: After a ball has been pocketed, a number of things can happen. If the ball is colored (i.e.: not the cue ball or the black ball), if it is the current player's color, then the current player will get an extra turn. If it is the opponent player's color, the opponent will get to place the cue ball wherever they want on the table.
  If the black ball has been pocketed after every other ball, the current player wins. Otherwise, the current player instantly loses. If the cue ball has been pocketed, the opposing player can get to place the cue ball wherever they want.
- Cross-reference: Functional Requirement: 13, 14, 18
- Actors: Balls.
- Pre-conditions
  - The game must have been completed.
  - The user must be in a valid session.
- Post-conditions:
  - The game must proceed based on the penalties or bonuses decided.
  - If the black ball has been pocketed, the game must end.
- Flow of events: See table 2.5.1

User Actions	System Action	Server Action
1. Ball collides with another ball	2. System updates ball locations	
3. Ball gets pocketed		
	4. System checks color of ball and decides flow of events	
	5. System sends updated game state to user.	
6. User can see the updated game state.		

Table 1: Flow of events of use case 2.6

2.5.2 UML Diagram

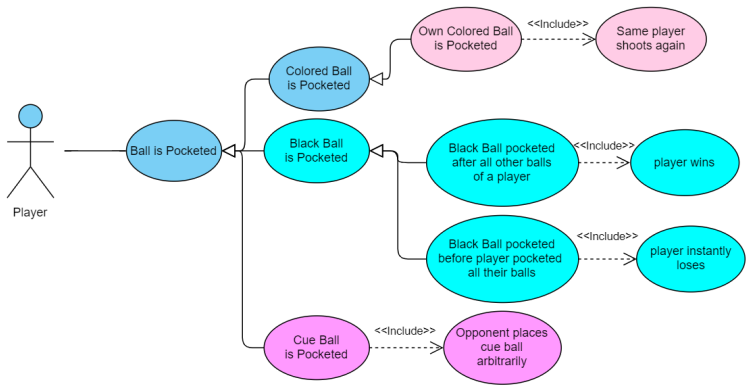

Figure 7: Pocketing a ball use case diagram

2.6 Updating the leaderboard

2.6.1 Description

- Use Case: Updating the leaderboard.
- Author: Ivar de Bruin.
- Date: 28/11/2019
- Purpose: Adding an achieved score to the leaderboard and displaying it to the user.
- Overview: After a game a player can update the name that will be shown along with his achieved score for that game on the leaderboard. This score will be send to the server. The server will add it to the database linked to the correct user. The server will then send back the updated leaderboard to the user.
- Cross-reference: Functional Requirement: 4, 5, 7
- Actors: Players and Server.
- Pre-conditions
  - The game must have been completed.
  - The user must be in a valid session.
- Post-conditions:

- The score must be stored in the database linked to the user account.
- The leaderboard must be updated to show the new top 5 scores ever.
- The name chosen by the user must be stored so entering it again is unnecessary.

- Flow of events: See table 2.6.1

User Actions	System Action	Server Action
1. User updates leaderboard name if necessary	2. System updates leaderboard name	
	3. System sends score to the server	4. Server adds score to the database
		5. Server creates new leaderboard consisting of top 5 scores ever achieved
	6. System sends updated leaderboard to user.	
7. User views the leaderboard.		

Table 2: Flow of events of use case 2.6

2.6.2 UML Diagram

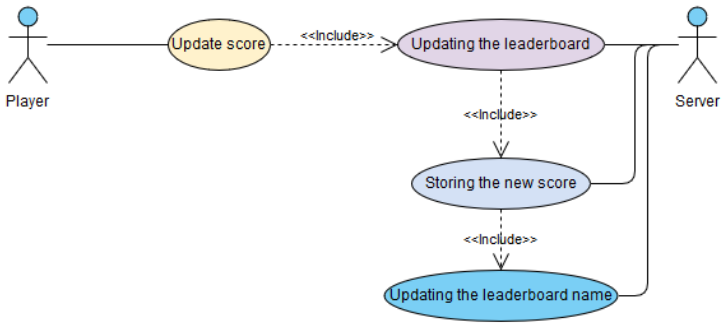

Figure 8: Adding a score to the leaderboard use case diagram