Exercise 1 – Requirements of the Snake

Functional Requirements

For the game Snake, the requirements regarding functionality and service are grouped under the functional requirements. Within these functional requirements, four categories can be identified using the MoSCoW model for prioritization.

Must Haves

1. The user must be able to start a new game
2. When a new game is started a game board with a 20 x 20 grid must be initialized
3. When a new board is initialized the game must create a snake in the middle of the grid
4. The initial size of the snake is one square
5. The initial direction of the snake is *North*
6. Snake's direction can have four values: North, East, South, West
7. When the keyboard is not used, the snake must move in a certain direction all the time
8. The initial speed of the snake is one movement every 0.7 seconds
9. The user must be able to control the direction of the snake by pressing one of the arrow keys on his/her keyboard
10. Snake can rotate only by 90 degrees
11. The snake must grow, when a food is eaten, by one square (this through extending onto the square the food was on)
12. The game must display the score while the user is playing
13. The game must save the scores in a database
14. The game must stop when the snake hits itself or a wall
15. The wall is represented by the edges of the playing screen
16. The game must be won when the snake occupies all but one square on the grid
17. The users must be able to authenticate themselves with the game by entering their username and password
18. The user must be able to name their games to be saved together with the obtained score when the game finishes
19. The game must display a leaderboard of the top 5 scores after the current score has been recorded
20. The game must provide new food at a random position (on which the snake is not located) after the currently available food has been eaten.
21. The game must increment the score when a food gets eaten by the food's value
22. The initial food's value is 1 point
23. The game must place a food at some random place somewhere on the grid where the snake is not located when the game is started

24. The random position is determined with the *Math* class
25. The game must keep a maximum of 1 food on the grid at a time
26. The game must remove food from its position once it gets eaten by the snake

Should Haves

27. The game must speed up by 0.1 seconds every time the snake's length becomes a multiple of 7
28. After the snake's length is 28, the game difficulty increases even more with the food starting to randomly change its place on the grid every 10 seconds if it is not eaten
29. The player can pause the game by pressing the spacebar, then the snake stops moving and keyboard events are not triggered anymore
30. The player can resume the game by pressing the spacebar, then all functionality continues to work

Could Haves

31. The game is multilingual
32. The player can choose the style of the snake from provided ones
33. The player can set the grid size
34. After pressing the start button there is a 3 second countdown displayed before the game starts
35. The snake should flash 3 times when the game is lost or won
36. There is music playing in the background when the game is being played
37. There is a sound effect when the snake eats the food
38. The game could be controllable using a cursor

Won't Haves

39. The game won't be able to be played by multiple players at the same time (either locally or over the internet)
40. The game won't allow the snake to turn in any other angular direction apart from 90 degree angles
41. The player shall be able to change the board's background by choosing an image from his/her computer files

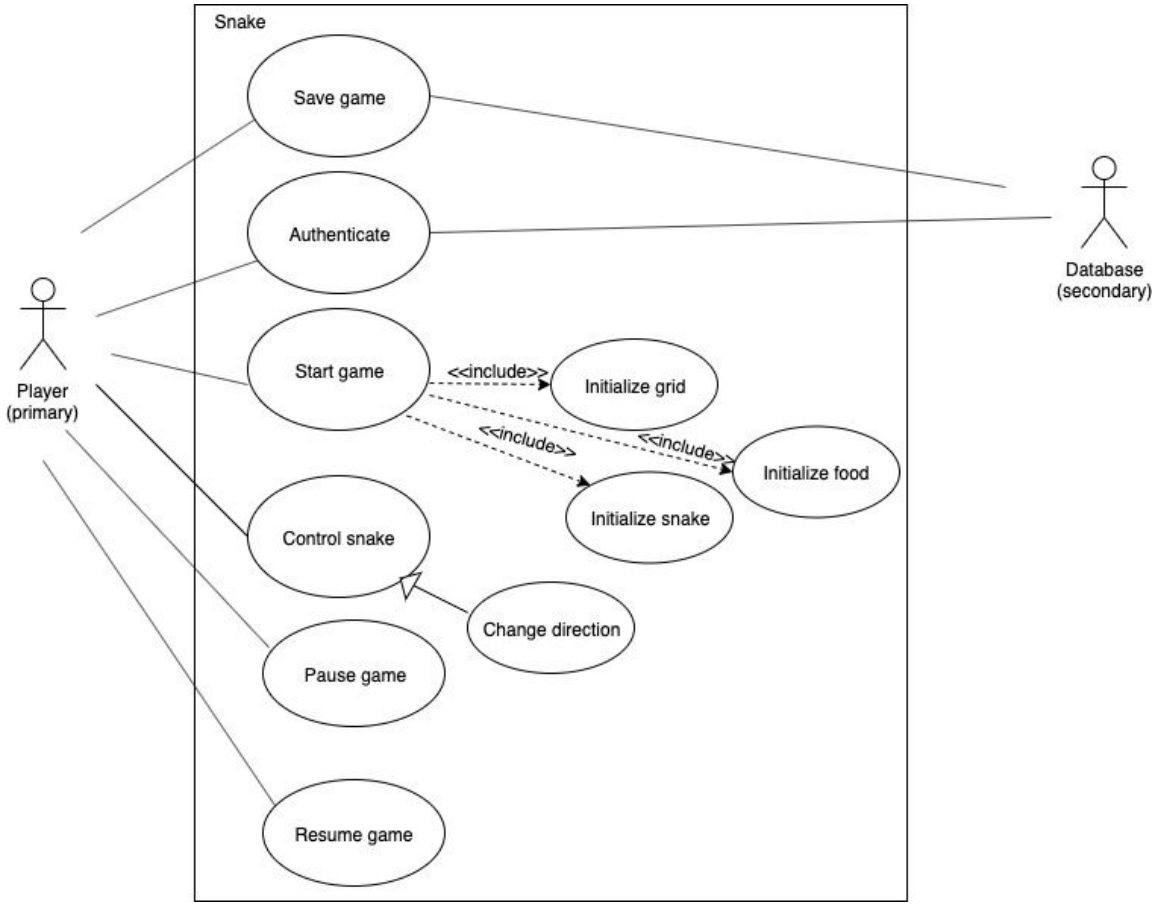
Non-functional Requirements

Besides the provided functionality and services, design constraints need to be included in the requirement specification as well. These requirements do not indicate what the system should do, but instead, indicate the constraints that apply to the system or the development process of the system.

- The game will make use of a MySQL database and the JDBC driver
- The game will use prepared statements in Java to avoid code-injection vulnerabilities

- The game will be playable on Windows (10 or above), Linux (kernel 5.3 or above), Mac OS (10.15 or above)
- The game will be implemented in Java using JavaFX for the GUI parts
- A first working version of the game will be delivered on or before November 29th 2019
- The Scrum methodology will be used for all iterations
- The implementation will be tested with at least 70% meaningful line test coverage (where meaningful means that the tests actually test the functionalities of the game and for example do not just execute the methods involved)

Exercise 2 – Modelling use cases



Use Case: Start game
Date: 25-11-2019
Purpose: Start the game
Actor: Player
Cross-reference: 1 - 5, 22, 23
Overview: When the player hasn't pressed the start button yet (the game is not in progress), he/she is able to start the game which then will be in progress. When the player game starts the game, the player will observe that there has been a 20 x 20 grid initialized. Furthermore, the player will observe that a snake of length 1 has been initialized/spawned in the middle of the grid, with an initial direction to the North. The player also observes a food which has been initialized at a random position of the grid where the snake is not located. The initial food takes 1 point as value.
Pre-conditions: The player hasn't started the game yet.
Post-conditions: Keyboard actions affect the snake, the snake moves immediately to its direction, the food's timer starts.

Use Case: Control snake
Date: 25-11-2019
Purpose: Change the direction of the snake on the board
Actor: Player
Cross-reference: 6 - 10
Overview: When the player is already on the playing screen (the game has started), he/she is able to change the direction to which the snake is heading. If the player is not pressing any key, the snake will move in a certain direction all the time with an initial speed of one movement every 0.7 seconds. When the user presses the 'arrow-up', 'arrow-down', 'arrow-left' or the 'arrow-right' key on the keyboard, the next move of the snake will be North, South, West or East direction respectively. If the snake will hit his own body or the boundary of the game screen, the snake will die. If the given direction is opposite to the direction the snake is moving in before pushing the key on the keyboard, this action will be ignored and the snake will not die.
Pre-conditions: The player has started the game by entering the board and the game is still neither won or lost.
Post-conditions: The score of the player gets incremented when the snake moves to a square which contains a food, the snake's length gets larger when the snake moves to a square which contains a food, the speed of the snake gets incremented when the snake moves to a square which contains a food, the food is placed at another random position on the grid where the snake is not located when the snake moves to a square which contains a food. The game is over when the snake moves to a wall or a square of the snake itself.

Use Case: Pause game
Date: 25-11-2019
Purpose: Pause the game
Actor: Player
Cross-reference: 29
Overview: When the player is already on the playing screen (the game has started), the user is able to press a button to pause the game. When the user presses that button, the screen ‘freezes’. The snake stops moving at all and becomes static, if the higher level is achieved at that time (where the ‘food’ changes its location after 10 seconds) then the food also stops changing its location. The keyboard actions are locked: the player is not able to change the snake’s direction.
Pre-conditions: The player has started the game by entering the playing screen and the game is still neither won or lost.
Post-conditions: The score of the player remains the same, the snake’s length, moving speed, direction is the same, the food’s position stays the same.

Use Case: Resume game
Date: 25-11-2019
Purpose: to resume the game
Actor: Player
Cross-reference: 30
Overview: when the player is already on the playing screen (the game has started), and has paused the game, he/she is able to resume the game by pressing a button. When the user presses that button, the screen ‘unfreezes’. The snake starts moving again to the direction which it had before pausing the game, if the higher level is achieved at that time (where the ‘food’ changes its location after 10 seconds) then the food starts changing its position on the grid. The keyboard actions are unlocked: the player is again able to change snake’s direction by pressing the arrow keys.
Pre-conditions: the game is in the ‘paused’ mode.
Post-conditions: keyboard actions affect the snake, the snake moves immediately to its direction, the food’s timer continues.

Use Case: Save game
Date: 25-11-2019
Purpose: Save the game
Actor: Player, Database
Cross-reference: 18
Overview: The user presses a button that says “Save Game”. The system then shows a screen where the player enters a name for that specific play of the game. The system sends the username and the score of the play together with a given name for the play to the database. The database then stores the data in such a way

that it can be used in leaderboards and other features. If we fail to send the data to the database an error message is shown.
Pre-conditions: The user has just finished his/her game. There is a database set up and running.
Post-conditions: The score is stored in the database.

Use Case: Authenticate
Date: 25-11-2019
Purpose: Authenticate the user that is playing snake
Actor: Player, Database
Cross-reference: 14, 17 - 19
Overview: Before the game starts, the user can pass his username and password. The given username and password will be stored in the database. The database then stores the data in such a way that it can be used in leaderboards and other features. If we fail to send the data to the database an error message is shown. When the game ended (due to the player causing a collision for example), the score can be saved in the database coupled to the given username earlier on.
Pre-conditions: The user is on the starting screen and is not playing a game. There is a database set up and running.
Post-conditions: The username and password are both stored in the database.