User story	Task	Task Assigned To	Estimated Effort per Task (in hours)	Actual Effort per Task (in hours)	Done (yes / no)	Notes
Play Naval Battle against a computer.	Task 1: Generate ships at random positions on the board keeping constraint.	Glenn	1 hour	2,5 Hour	Yes	This should be done during the previous sprint, but we didn't manage to get it done in time. The task was shifted to this sprint, and I've managed to get it to work. Ships are generated at random positions by the computer.
	Task 2: Verify whether ship is touched, if so player can touch another block, else computer turn.	Glenn	0.5 hours	1,5 hours	Yes	This should be done during the previous sprint, but we didn't manage to get it done in time. The task was shifted to this sprint and is now fully working. The functionality of verifying that a ship is touched was already implemented in last sprint, in this sprint I implemented the functionality that the computer shoots when it is his turn.
	Task 3: Verify whether whole ship is touched, if so	Glenn	3 hours	4,5 hours	Yes	If a ship has sunk, the game should show that a ship has sunk, this is done

	blocks around ship must also be marked.					by giving the sunken ship a different colour. When a ship is sunk, the squares around the ship should be marked as well, because there is no possibility that there will be a ship on these squares. I've done this by giving these squares the "shot" property, which means that the squares can't be shot again and they will appear black. This is done for both the computer and player grid.
	Task 4: Implement intelligence computer	Glenn Mahira	3 hours	7 hours	Yes	The computer will shot randomly, until it hits a square with a ship on it. When the computer hits a square with a ship on it, the computer will try to find in what direction the ship is positioned, and shoot all squares that contain that particular ship. The computer will stop when the ship has sunk, in that case, it will go back to randomly shooting again.

						For this tasks, Glenn wrote most of the code, while Mahira wrote most of the tests. Code is very messy and needs to be organised using the design patterns.
ScoreSystem	Task 1: Calculate the score of the player based on the weakness of the ship and amount of misses	Mahira Pravesha	3 hours	2 hours	No	While placing the ships we store the front of each ship in a list. Having the front of each ship we can obtain at which location of the board the weakest spot of the ship is located. For each board we have created a variable counter on the amount of misses. With this the score system could easily be implemented. Testing is not done yet. Therefore this task cannot yet be considered as done.
Store high scores.	Task 1: The game shall allow a new player to register	Jeroen Yash	2 hours	2 hours	Yes	Checks if user exists , if not register.
	Task 2: Player authentication	Jeroen Yash	2 hours	2 hours	Yes	Checks if user details match database data by accessing and verifying

	Task 3: Set up the remote database.	Yash	4 hours	3.5 hours	Yes	Used MySQL workbench and a jdbc driver to connect to the database. For the first sprint the database can register users and authenticate users.
	Task 4: Store new score in database	Jeroen Yash	1.5 hours	-	No	We didn't manage to do this problem and we've decided to push it forward
Graphical User Interface	Task 1: Make the start-up screen.	Yash	40 minutes	40 minutes	Yes	Used JavaFX to create a window that is the first window of the game and allows the user to login, register or close the game.
	Task 2: Make the Login in screen.	Yash	1 hour	1 hour	Yes	Used JavaFX to create the login screen where registered users are asked for credentials and then after verification from database they are logged in.
	Task 3: Make the register screen.	Yash	1 hour	2 hours	Yes	This screen allows a user to register with the database and enter their username and password of their choosing provided that the username does not already exist in the database.
	Task 4: Make the home-screen	Yash	1 hour	1 hour	Yes	After logging in, or registering successfully, the home-screen is

						displayed where a user has multiple options.
	Task 5: Link the screens to the main gameboard screen.	Jeroen, Yash	1 hour	~1.5 Hours	Yes	On choosing option to start a new game the game screen is shown with the game board and the game commences.

Main problems Encountered

Problem 1:
Description: in task 3, Verify whether whole ship is touched, if so blocks around ship must also be marked, I encountered the problem that when a ship has sunk, you have to move to other squares left/right/up/down of the last square shot before the ship has sunken. To accomplish this, I wrote methods to get the square left/right/up/down of a given square, using an arraylist in which every square of the grid is stored, so basically I gave every square an index (for example left upper: 0, left bottom 90) with a total of 99 indexes. So to retrieve a square right of a given square, I use the formula:
$10 * y_{coordinate} + x_{coordinate} + 1$.
When done with this, I used the methods to determine whether or not a square has a ship on it, and when it has, I give the square another colour.

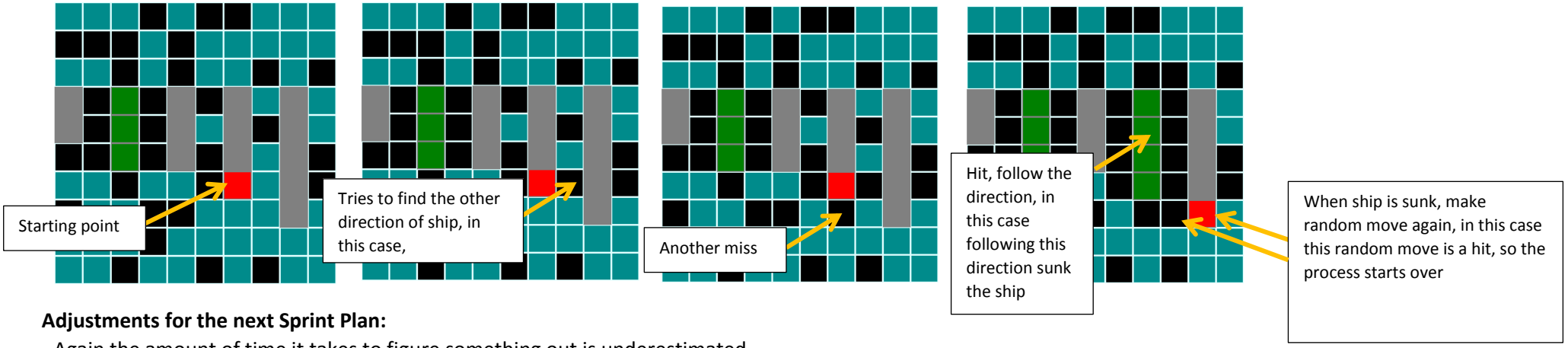
Problem 2:
Description: in task 4: Implement intelligence computer, we encountered a lot of bugs and edge cases. It took a lot of time to fix these bugs. Most of the bugs were stack overflow errors because of an infinite loop or an out of bounds exception because of using a method that would retrieve something that was not inside the board (using get left/right/down/up methods).

Problem 3:
Description: all tasks regarding testing "Play Naval Battle against a computer". While testing we encountered the problem how to test a method that makes use of the java.util.Random. After spending a lot of time figuring this out, we found the solution to pass the randomizer as a parameter to the method, so that while testing we can pass a mocked randomizer of which we can control the output using Mockito.

Problem 4:
Description: all tasks regarding testing "Play Naval Battle against a computer". While testing we encountered the problem that many methods made use of recursive calls, which indicates bad code design. This made testing the code very difficult. For this sprint we did not have the time to properly refactor it, therefore we have tested the opponent functionality as much as we could for now (86%, not bad but could be better).

Problem 5:
Description: all tasks regarding testing “Play Naval Battle against a computer”. While testing we encountered the problem that there is many duplicated code, which indicates bad code design. This made testing the code very extensive. For this sprint we did not have the time to properly refactor it, therefore we have there are multiple tests, testing the same thing, but are needed to test different parts. In the next sprint we will fix this.

Problem 6:
Description: also in task 4: Implement intelligence computer, we also had to find a way to make the computer finish sinking a ship when it hitted a part of a ship before making a random move again. We did this by adding the first square that was hit with a ship on it to an arraylist, and take this square as a start point. Every time it’s the computer turn, the computer will try to shoot right/left/up/down of this start point, when it hits a part of a ship again, it will follow this direction. So for look at the following images. This took us quite some time to get working flawlessly.



Adjustments for the next Sprint Plan:
- Again the amount of time it takes to figure something out is underestimated.

- Testing more thoroughly, since we are not satisfied yet with the amount of code coverage.

- Refactor the code for the functionality of the opponent, since the current code contains many recursive calls and duplicate code which indicates bad code design.