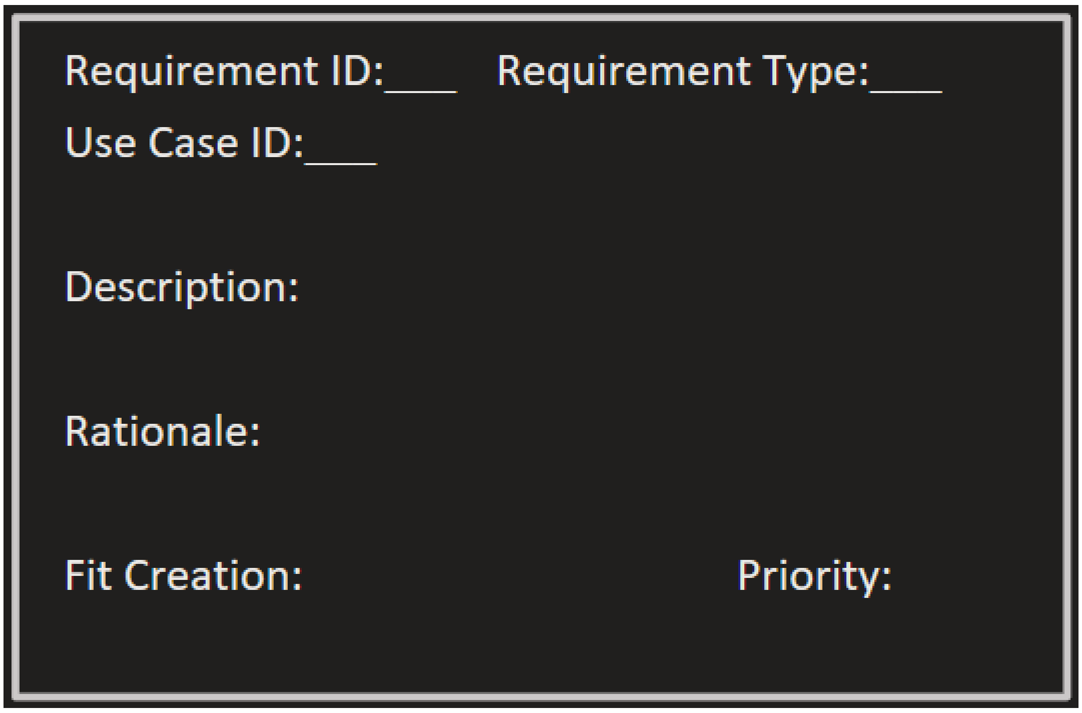
CPSC 3720 Project

Requirements Specification

Team Members: Jacob L, Jagjit S, Liam T

Atomic Requirements Shell



Requirement ID:____ Requirement Type:____

Use Case ID:____

Description:

Rationale:

Fit Creation: Priority:

1. The Purpose of the Project

We want to create a text-based application that implements the draw-three variation of the classic card game “War”.
We will measure our progress through completion of use-cases of each possible move a user can make in War as well as feedback from peer code reviews.

2. The Stakeholders

New Player

- Name: Jimothy
- Age: 19-30
- New to text-based games.
- Knows the basic rules of War.
- Unfamiliar with common War strategies.
    - May just choose the highest card every time.

Sophisticated Player

- Name: Mitchard
- Age: 24
- IQ: 260
- Used to text-based games.
- Knows the rules of Warwell.
- Passing knowledge of Warstrategies.
    - May consider what cards exist in their opponents deck when choosing a card to play.

CPU Player

- May adopt one of many different strategies
    - Choose the highest card.
    - Choose random.

    - Choose’s leftmost card.

User Priority

1. Sophisticated Users
2. New Users
3. CPU Players

Development Team

- Designers
    - Need a robust set of requirements to ensure they can properly design the product to satisfy user needs.
- Developers
    - Need clear enough design instructions to create a robust piece of software. Need a realistically feasible plan from the designers.
- Testers
    - Need to be delivered a working piece of software that can be played repeatedly to thoroughly test all use-cases.
- Maintainers
    - Need consistently formatted code with comments where necessary to make maintaining, refactoring, and extending easier.

3. Constraints

Solution Constraints

The game shall run on the Linux environments present in the Uleth Lab.

Rationale: The project can not be accessed for testing or marking if it does not compile properly on the lab servers.

Fit criterion: The project will be uploaded to Gitlab frequently to ensure it passes the pipelines required to run in the lab. Additionally, it shall be written in C++.

The game shall be implemented in a text-based format.

Rationale: The game must be playable in a console window so that maintainers do not have to work with complex GUI libraries.

Fit criterion: The project will be written with a text-based format in mind.

Budget Constraints

The game must be able to be written in 15-30 man-hours.

Rationale: The project will be handed off to a development team who will only have approximately 15-30 man-hours to complete the implementation phase

Fit criterion: Requirements will be written clearly to ensure no scope creep occurs in the design and implementation phases of the project.

Naming Conventions and Terminology

4a. Glossary of All Terms, Including Acronyms, Used by Stakeholders Involved in the Project

- Deck
  - A standard deck of 52 playing cards
- Stock
  - A player's allotted portion of the Deck
- Hand
  - The three cards a player draws to each turn
- War
  - An event that occurs when both players play a card with the same rank. Outcome of the war is decided in a second round in which each player places 2 cards face down preceded by one card face up. The outcome is checked once again and if both cards share the same rank another war is held, if not the player with the higher ranked card takes all the cards.

Assumptions

Assumptions:

1. Everyone should have access to the repository when working on the product. whether it be in the labs or on personal machines.

2. Each player/a.i. receives 26 cards and knows what cards are in their hand (upon request from the system for the player).

3. The card game will not have any betting or gambling system associated with it.

4. Group members will meet up either daily or weekly through whichever means they deem to discuss project and sprint info.

5. Use whichever IDE you would like to use, make sure that the code works on the lab machines at the University.

6. The war scenario will only place two cards below the new face up card.

7. The Ace card is the highest ranked card, all of the face cards have a higher rank in order of their face (The king card is a higher ranked card than the queen).

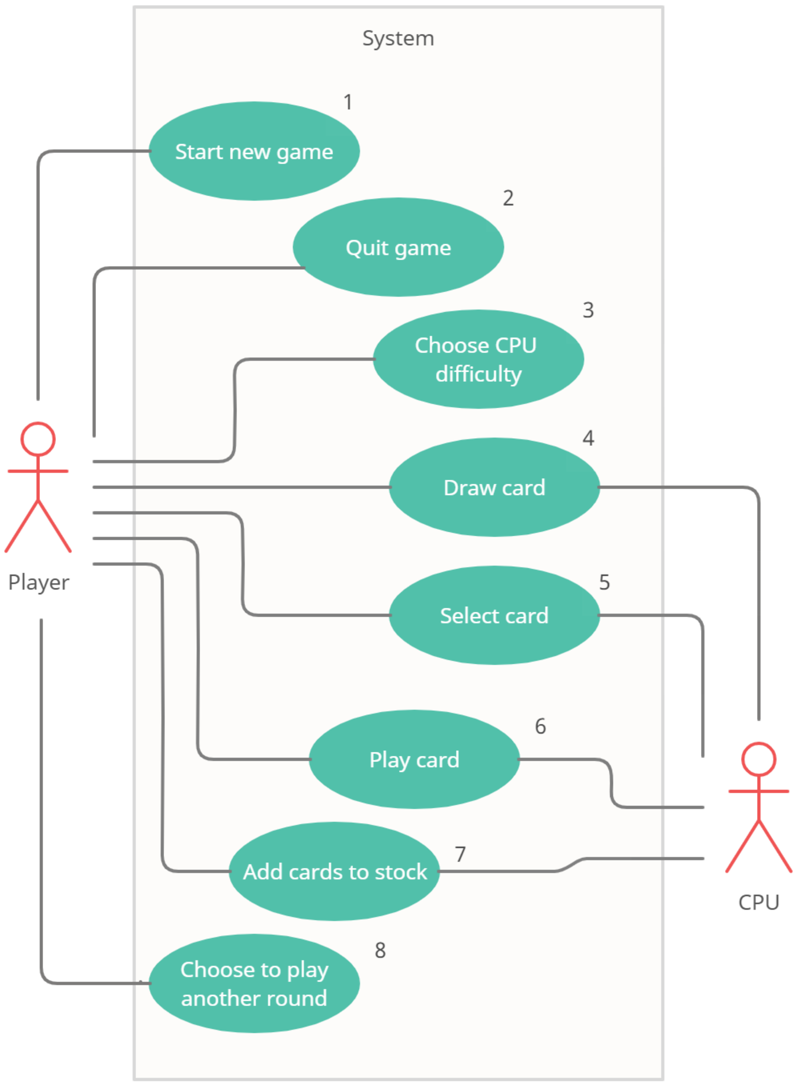
8. When a war scenario happens the players will take cards off the top of their stock leaving their hands separate from the war scenario, unless their stock is empty. Only then will the cards be taken from their hands. When taking cards from the player's hand, choose randomly which cards will be facedown and the last card will be placed face up.

9. If a player's stock and hand run out of cards they lose whether it be due to a war or just regular play.

10. When a player wins a regular or a war scenario the cards will be placed on the bottom of the stock in a random order.

6. The Scope of the Product

Use Case Diagram



Use Cases

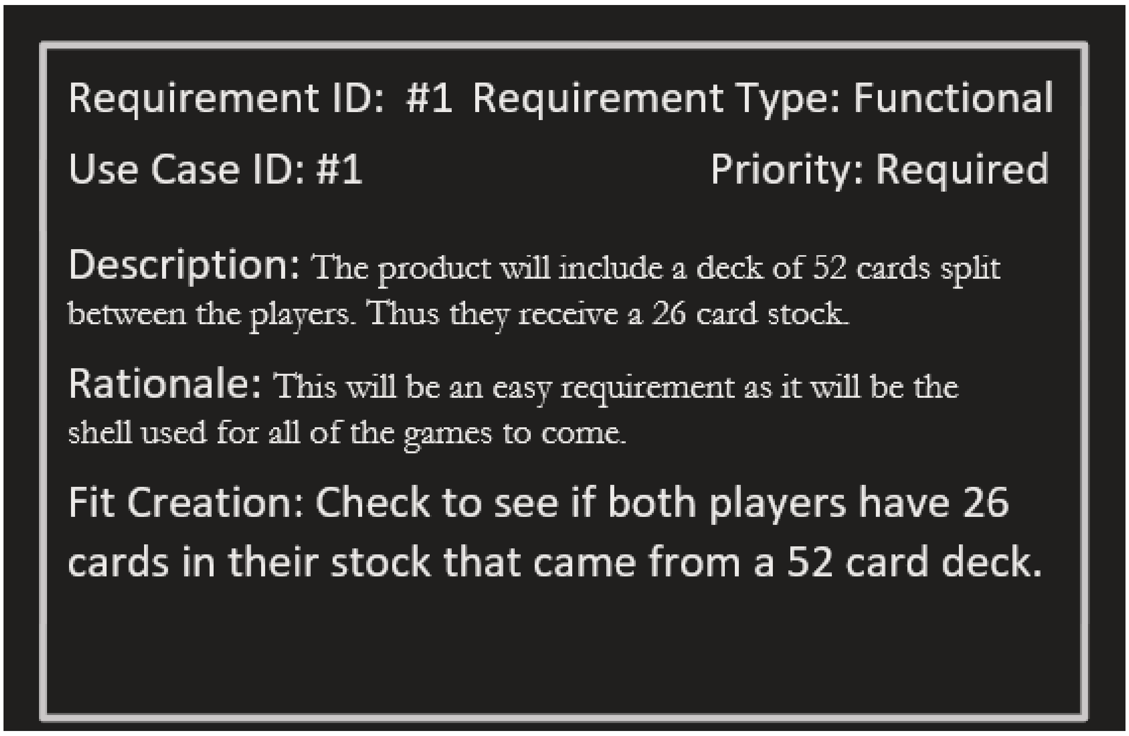
- #1 - Start New Game
    - The player is welcomed on the main menu of the game. The player must then enter a command to start the game.
- #2 - Quit Game
    - The player is able to enter a command on the main menu to exit the application.
- #3 - Choose CPU Difficulty
    - Upon starting a round, the player will choose between an Easy or Hard computer opponent.
- #4 - Draw Card
    - At the start of their turn, each player will draw until their hand contains 3 cards.
- #5 - Select Card
    - On their turn, a player will be prompted to choose a card to play against their opponent.
- #6 - Play Card
    - When both players are ready, the players will reveal their cards to their opponent and the system will evaluate who is the victor.
- #7 - Add Cards to Stock
    - When a player wins a turn, they will add the cards that participated in the turn to the bottom of their stock

- #8 - Choose to Play Another Round.
  - After a round is completed, the player will be prompted to play again or return to the main menu.

7. Functional Requirements

In this section, we rank our requirements on a story point system. Our story points range from 1 to 10, with 10 being the most difficult to implement.

1. The product will include a deck of 52 cards split between the players. Thus they receive a 26 cardstock.
    - Priority: Required
    - Pertinent Use Cases: #1
    - Estimated cost: 2 story points
    - Rationale:This will be an easy requirement as it will be the shell used for all of the games to come.



Requirement ID: #1 Requirement Type: Functional

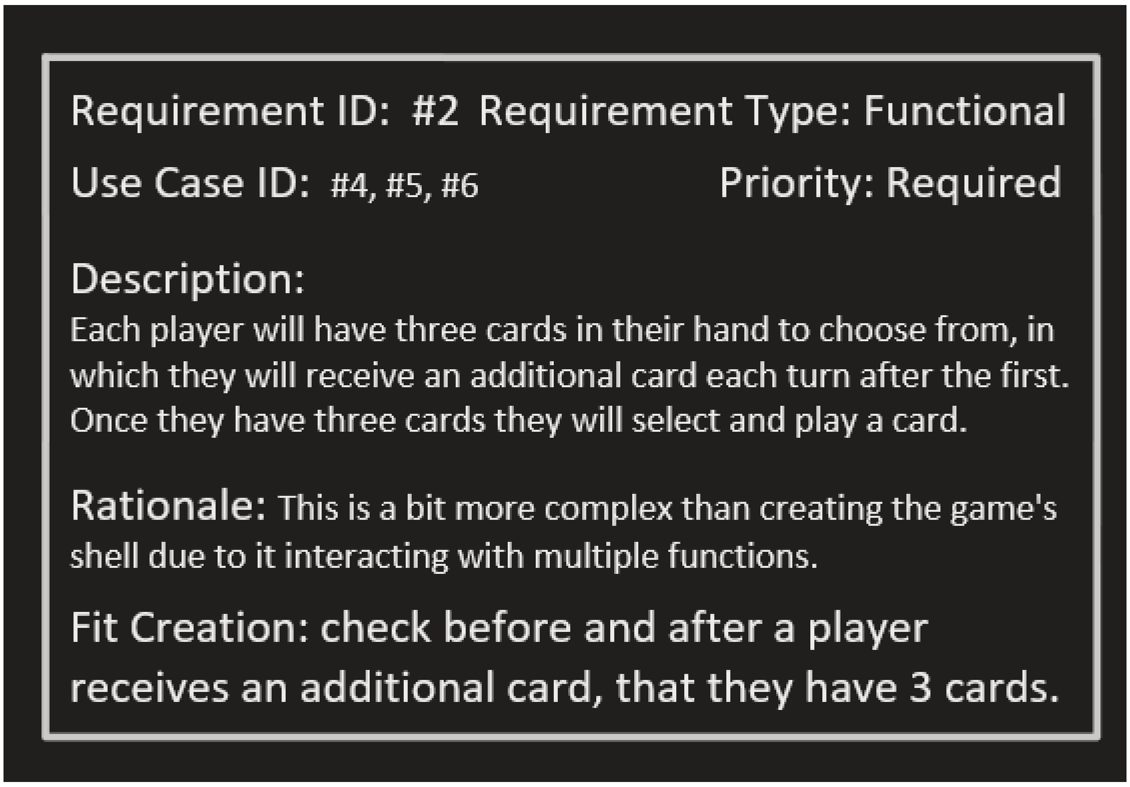
Use Case ID: #1 Priority: Required

Description: The product will include a deck of 52 cards split between the players. Thus they receive a 26 card stock.

Rationale: This will be an easy requirement as it will be the shell used for all of the games to come.

Fit Creation: Check to see if both players have 26 cards in their stock that came from a 52 card deck.

2. Each player will have three cards in their hand to choose from, in which they will receive an additional card each turn after the first. Once they have three cards they will select and play a card.
    - Priority: Required
    - Pertinent Use Cases: #4, #5, #6
    - Estimated cost: 4 story points
    - Rationale: This is a bit more complex than creating the game's shell due to it interacting with multiple functions.



Requirement ID: #2 Requirement Type: Functional

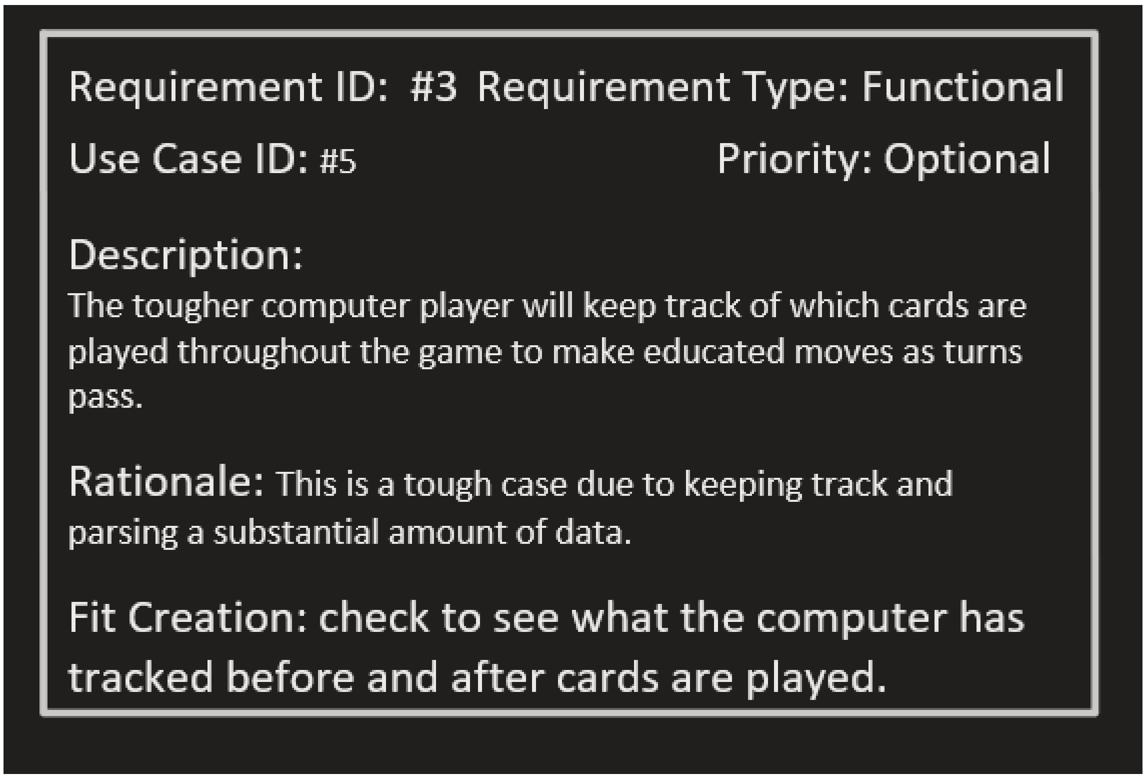
Use Case ID: #4, #5, #6 Priority: Required

Description:
Each player will have three cards in their hand to choose from, in which they will receive an additional card each turn after the first. Once they have three cards they will select and play a card.

Rationale: This is a bit more complex than creating the game's shell due to it interacting with multiple functions.

Fit Creation: check before and after a player receives an additional card, that they have 3 cards.

3. The tougher computer player will keep track of which cards are played throughout the game to make educated moves as turns pass.
    - Priority: Optional
    - Pertinent Use Cases: #5
    - Estimated cost: 8 story points
    - Rationale: This is a tough case due to keeping track and parsing a substantial amount of data.


Requirement ID: #3 Requirement Type: Functional

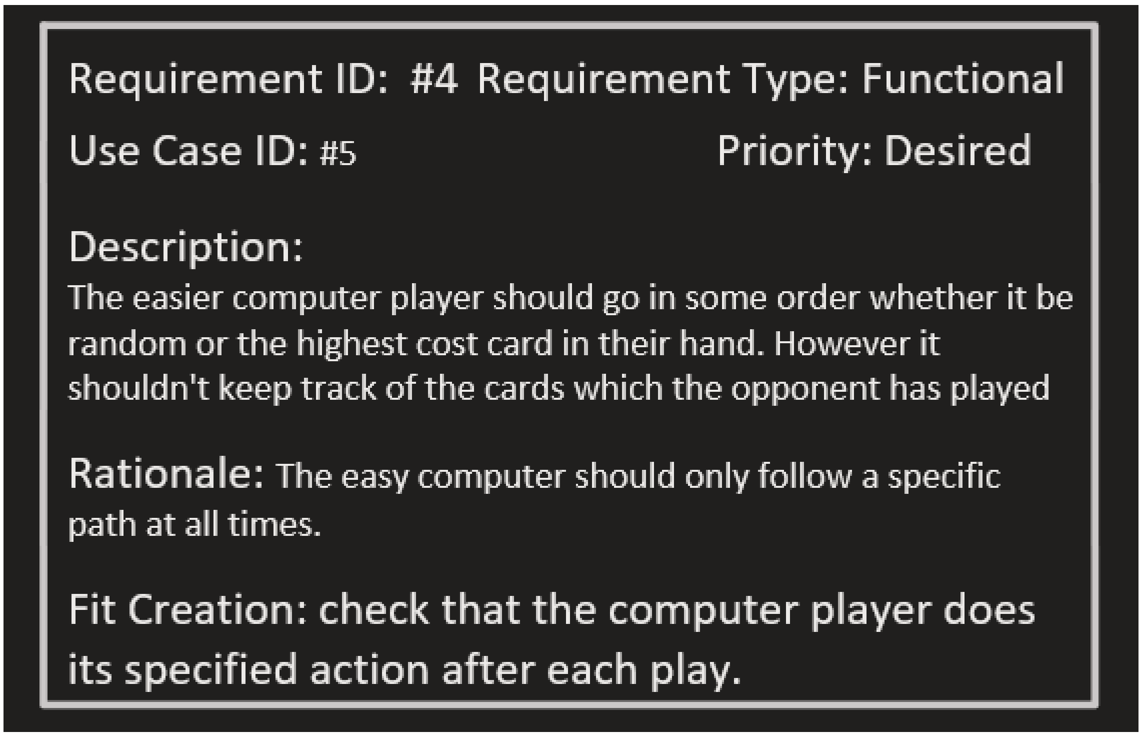
Use Case ID: #5 Priority: Optional

Description:
The tougher computer player will keep track of which cards are played throughout the game to make educated moves as turns pass.

Rationale: This is a tough case due to keeping track and parsing a substantial amount of data.

Fit Creation: check to see what the computer has tracked before and after cards are played.

4. The easier computer player should go in some order whether it be random or the highest cost card in their hand. However it shouldnt keep track of the cards which the opponent has played.
    - Priority: Desired
    - Pertinent Use Cases: #5
    - Estimated cost: 2 story points
    - Rationale: The easy computer should only follow a specific path at all times.


Requirement ID: #4 Requirement Type: Functional

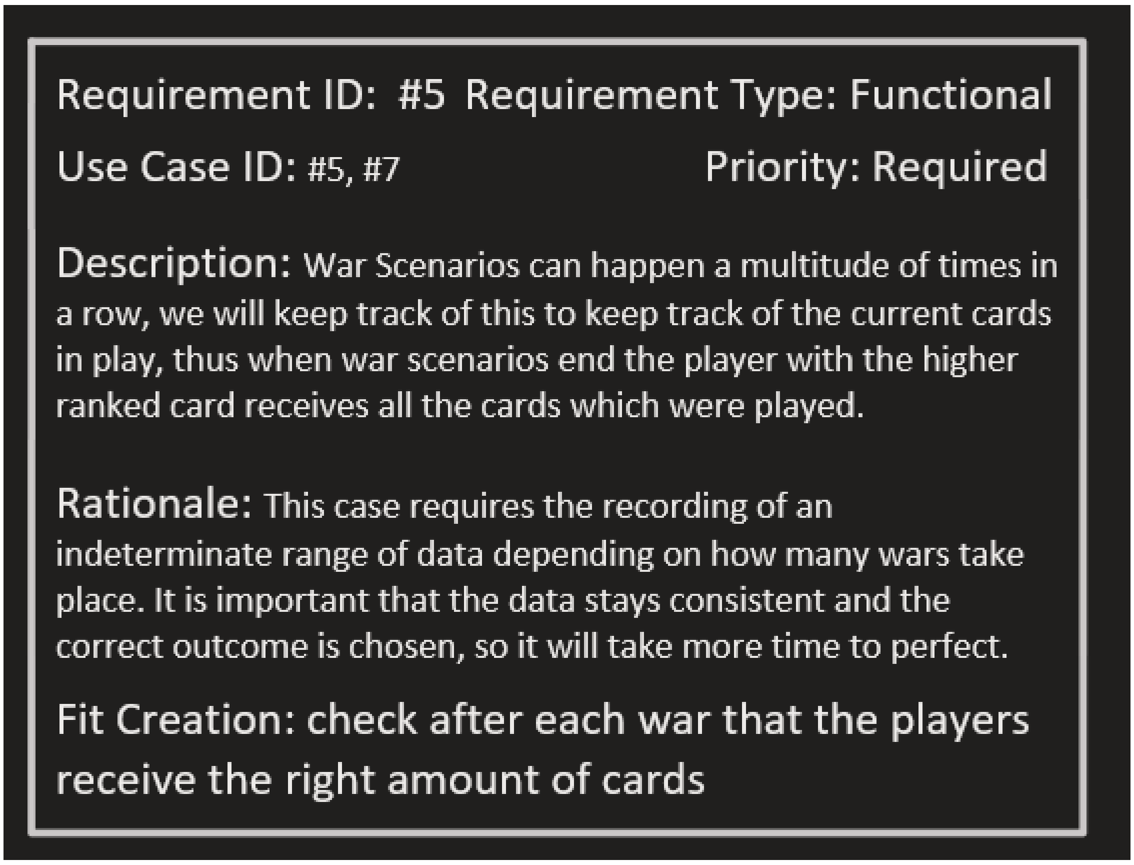
Use Case ID: #5 Priority: Desired

Description:
The easier computer player should go in some order whether it be random or the highest cost card in their hand. However it shouldn't keep track of the cards which the opponent has played

Rationale: The easy computer should only follow a specific path at all times.

Fit Creation: check that the computer player does its specified action after each play.

5. War Scenarios can happen a multitude of times in a row, we will keep track of this to keep track of the current cards in play, thus when war scenarios end the player with the higher ranked card receives all the cards which were played.
    - Priority: Required
    - Pertinent Use Cases: #5, #7
    - Estimated cost: 8 story points
    - Rationale: This case requires the recording of an indeterminate range of data depending on how many wars take place. It is important that the data stays consistent and the correct outcome is chosen, so it will take more time to perfect.



Requirement ID: #5 Requirement Type: Functional

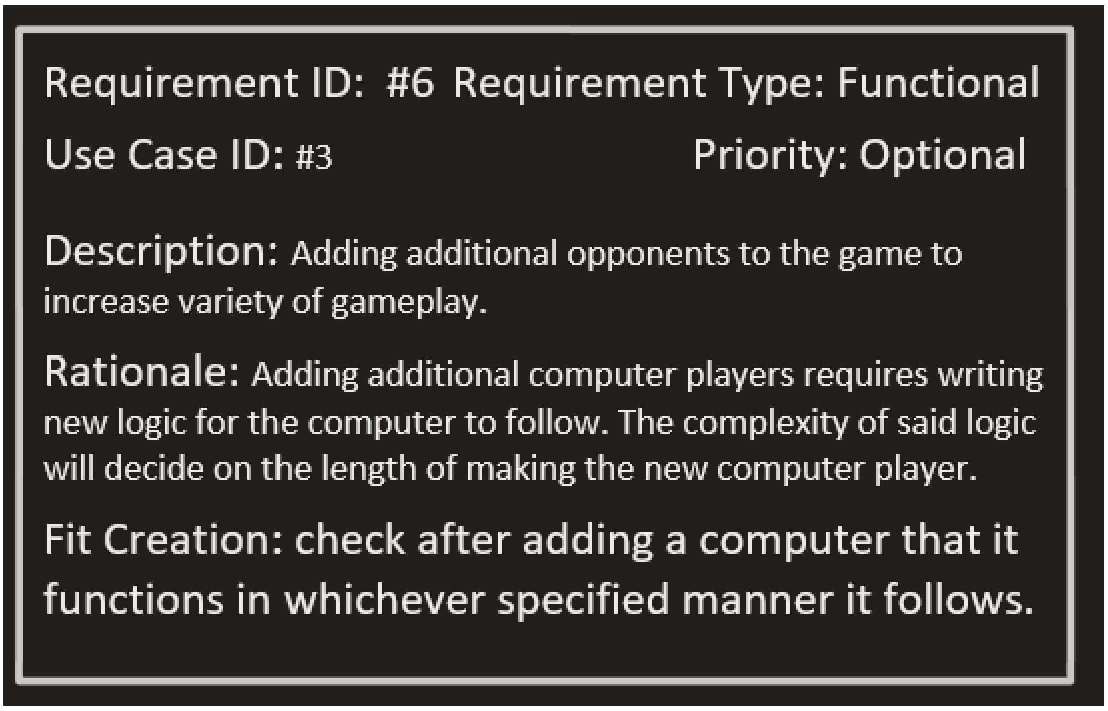
Use Case ID: #5, #7 Priority: Required

Description: War Scenarios can happen a multitude of times in a row, we will keep track of this to keep track of the current cards in play, thus when war scenarios end the player with the higher ranked card receives all the cards which were played.

Rationale: This case requires the recording of an indeterminate range of data depending on how many wars take place. It is important that the data stays consistent and the correct outcome is chosen, so it will take more time to perfect.

Fit Creation: check after each war that the players receive the right amount of cards

6. Adding additional opponents to the game to increase variety of gameplay.
    - Priority: Optional
    - Pertinent Use Cases: #3
    - Estimated cost: 5 story points
    - Rationale: Adding additional computer players requires writing new logic for the computer to follow. The complexity of said logic will decide on the length of making the new computer player.


Requirement ID: #6 Requirement Type: Functional

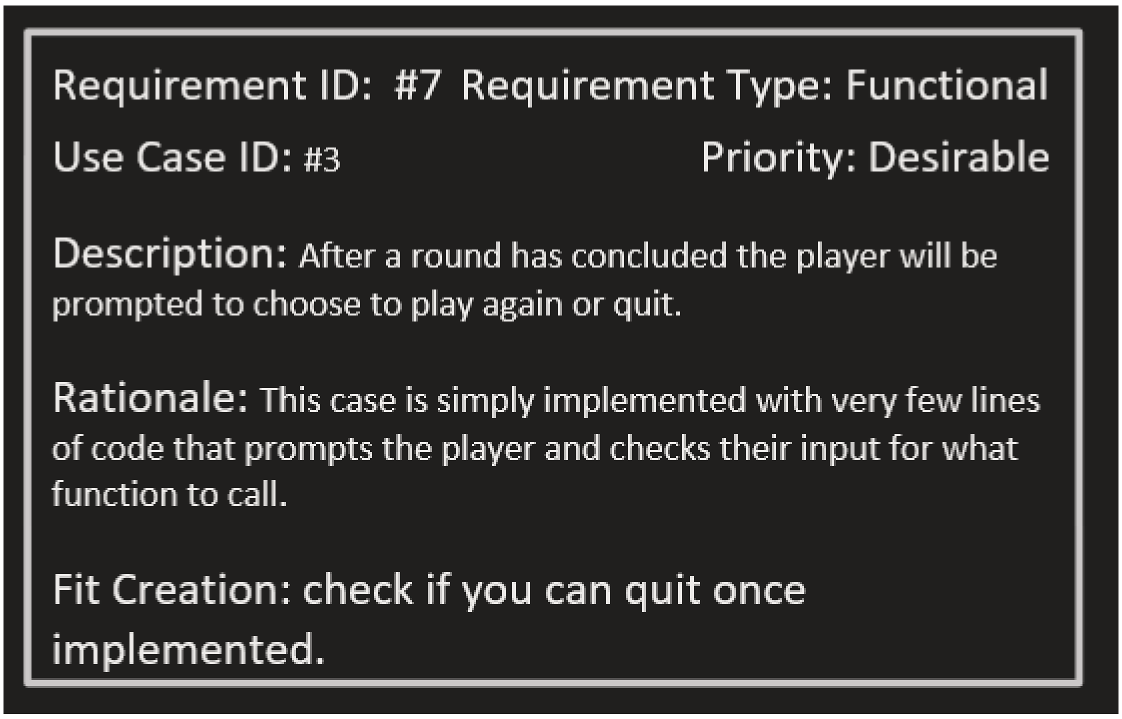
Use Case ID: #3 Priority: Optional

Description: Adding additional opponents to the game to increase variety of gameplay.

Rationale: Adding additional computer players requires writing new logic for the computer to follow. The complexity of said logic will decide on the length of making the new computer player.

Fit Creation: check after adding a computer that it functions in whichever specified manner it follows.

7. After a round has concluded the player will be prompted to choose to play again or quit.
    - Priority: Desirable
    - Pertinent Use Cases: #1, #8
    - Estimated cost: 1 story point
    - Rationale: This case is simply implemented with very few lines of code that prompts the player and checks their input for what function to call.


Requirement ID: #7 Requirement Type: Functional

Use Case ID: #3 Priority: Desirable

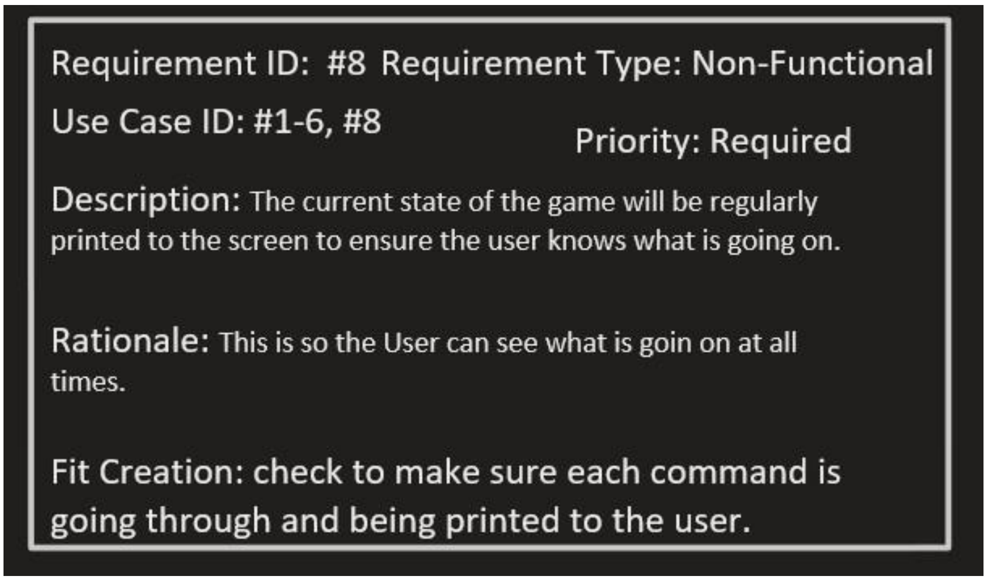
Description: After a round has concluded the player will be prompted to choose to play again or quit.

Rationale: This case is simply implemented with very few lines of code that prompts the player and checks their input for what function to call.

Fit Creation: check if you can quit once implemented.

8. Non-functional Requirements

1. The current state of the game will be regularly printed to the screen to ensure the user knows what is going on.


Requirement ID: #8 Requirement Type: Non-Functional

Use Case ID: #1-6, #8

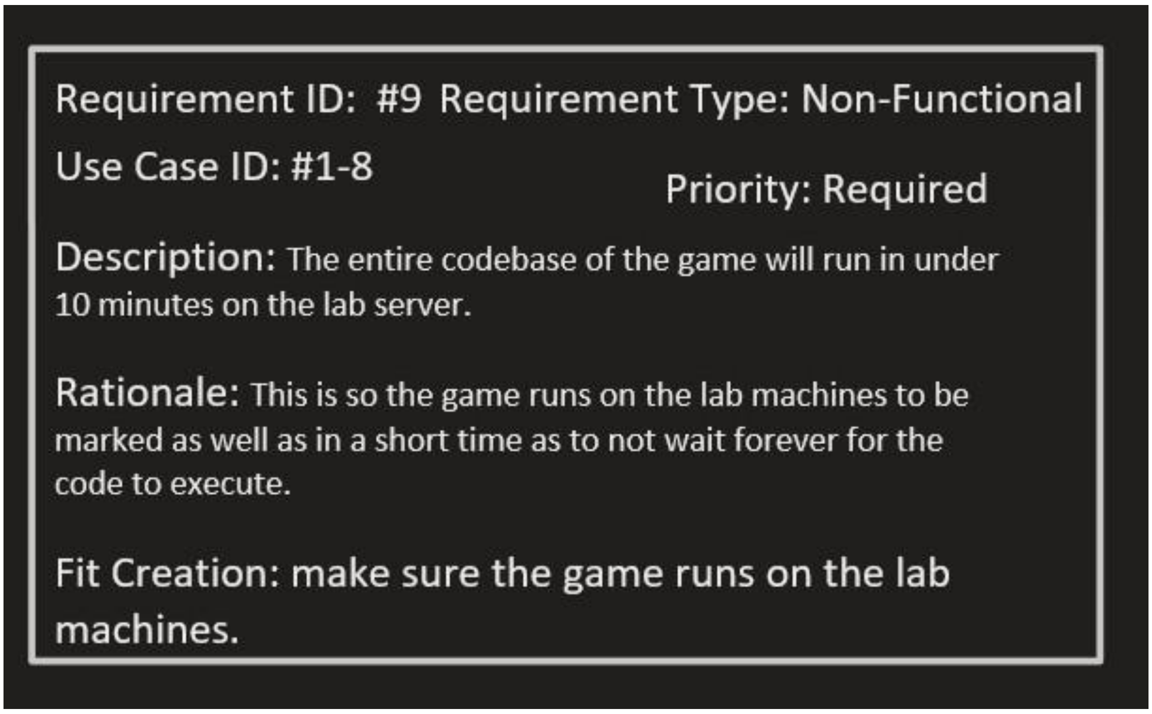
Priority: Required

Description: The current state of the game will be regularly printed to the screen to ensure the user knows what is going on.

Rationale: This is so the User can see what is goin on at all times.

Fit Creation: check to make sure each command is going through and being printed to the user.

2. The entire codebase of the game will run in under 10 minutes on the lab server.


Requirement ID: #9 Requirement Type: Non-Functional

Use Case ID: #1-8

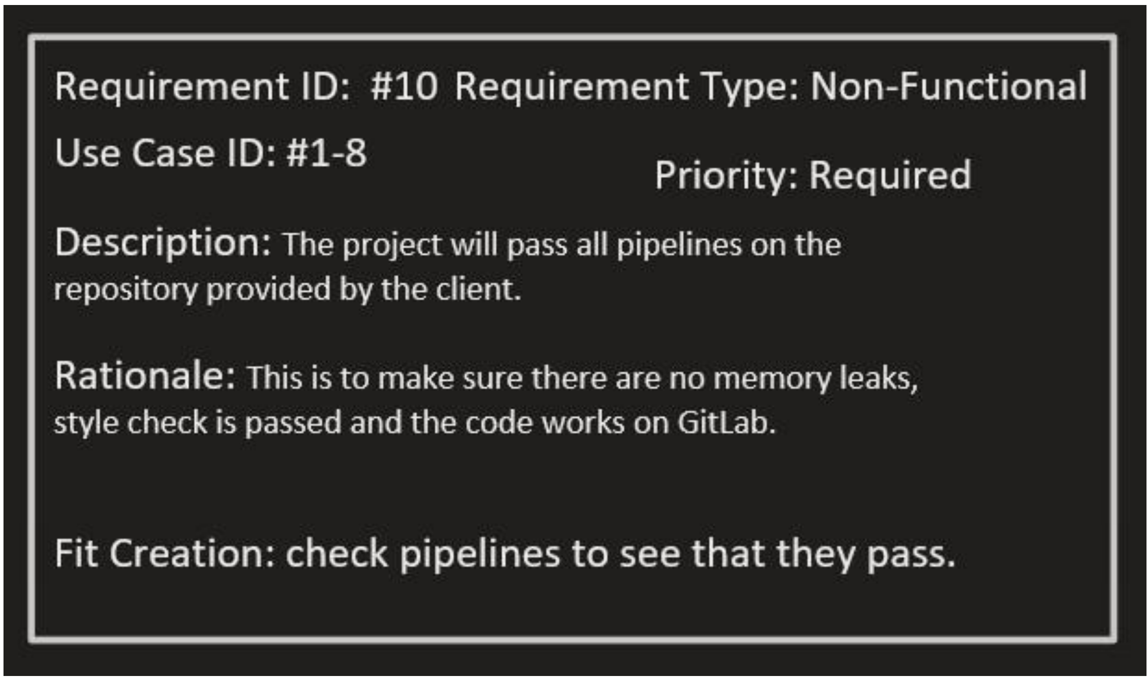
Priority: Required

Description: The entire codebase of the game will run in under 10 minutes on the lab server.

Rationale: This is so the game runs on the lab machines to be marked as well as in a short time as to not wait forever for the code to execute.

Fit Creation: make sure the game runs on the lab machines.

3. The project will pass all pipelines on the repository provided by the client.


Requirement ID: #10 Requirement Type: Non-Functional

Use Case ID: #1-8

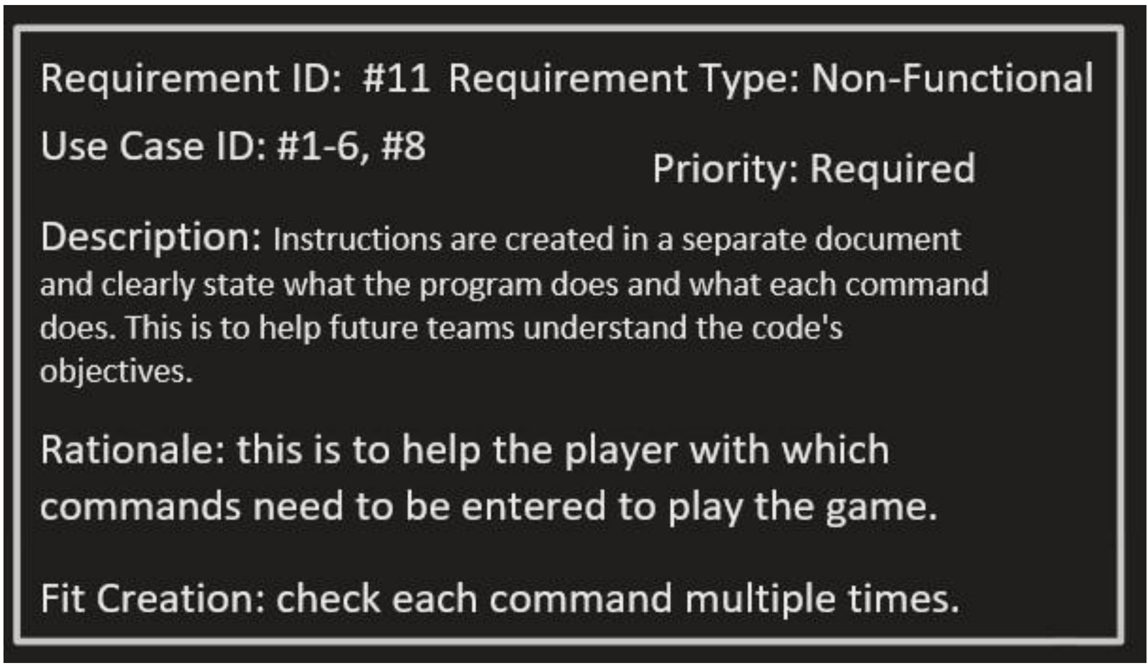
Priority: Required

Description: The project will pass all pipelines on the repository provided by the client.

Rationale: This is to make sure there are no memory leaks, style check is passed and the code works on GitLab.

Fit Creation: check pipelines to see that they pass.

4. Instructions are created in a separate document and clearly state what the program does and what each command does. This is to help future teams understand the code's objectives.


Requirement ID: #11 Requirement Type: Non-Functional

Use Case ID: #1-6, #8

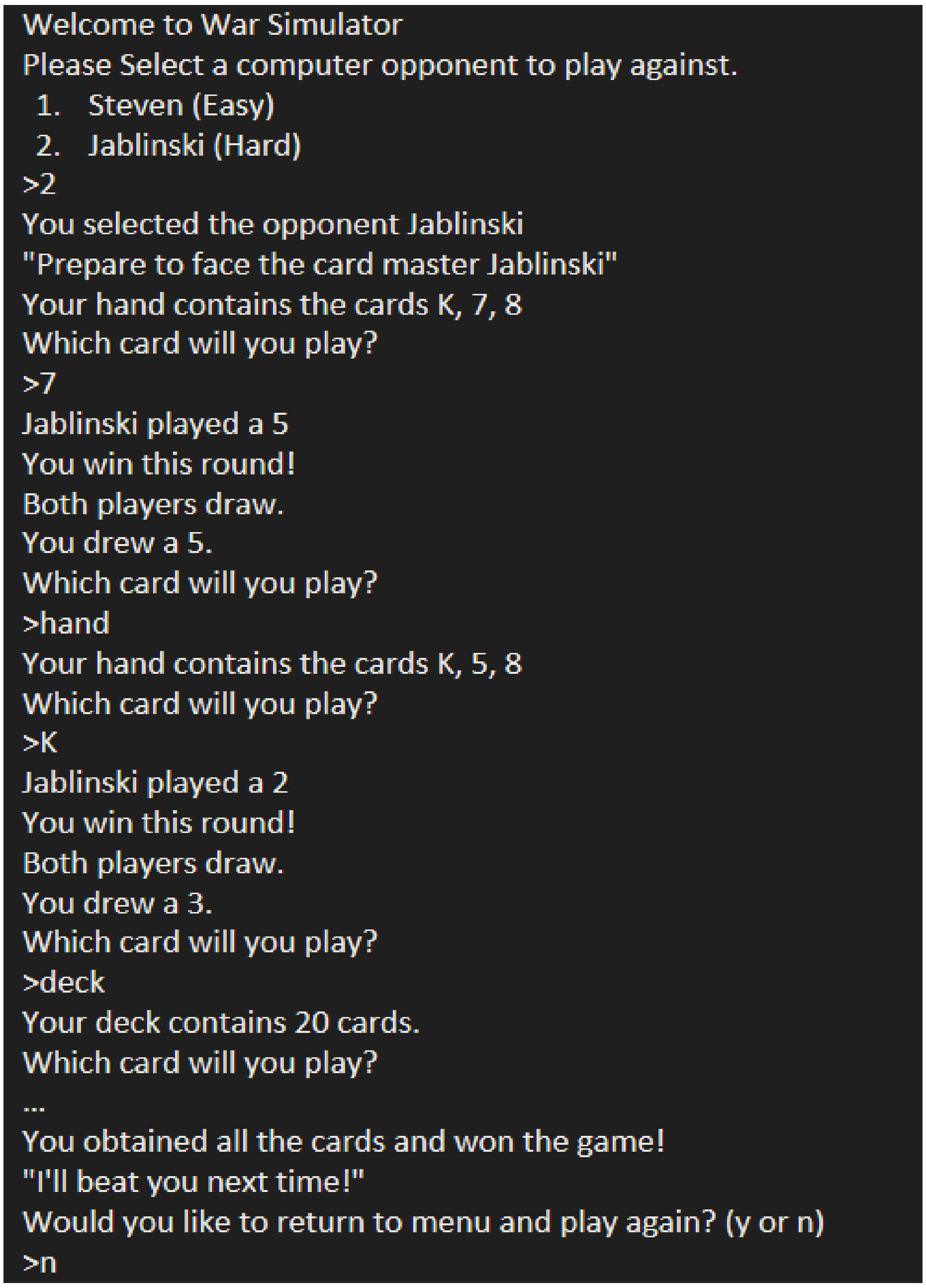
Priority: Required

Description: Instructions are created in a separate document and clearly state what the program does and what each command does. This is to help future teams understand the code's objectives.

Rationale: this is to help the player with which commands need to be entered to play the game.

Fit Creation: check each command multiple times.

9. Look and Feel Requirements



```
Welcome to War Simulator
Please Select a computer opponent to play against.
  1.  Steven (Easy)
  2.  Jablinski (Hard)
>2
You selected the opponent Jablinski
"Prepare to face the card master Jablinski"
Your hand contains the cards K, 7, 8
Which card will you play?
>7
Jablinski played a 5
You win this round!
Both players draw.
You drew a 5.
Which card will you play?
>hand
Your hand contains the cards K, 5, 8
Which card will you play?
>K
Jablinski played a 2
You win this round!
Both players draw.
You drew a 3.
Which card will you play?
>deck
Your deck contains 20 cards.
Which card will you play?
...
You obtained all the cards and won the game!
"I'll beat you next time!"
Would you like to return to menu and play again? (y or n)
>n
```

The deck command prints the amount of cards in the player's deck whenever they would like to know. Similarly the hand command prints out the players cards in their hand.

The player can issue these commands at any point during the game and after it will ask which card they would like to play.

The letters symbolize the face values (K, Q, J, A - King, Queen, Jack, Ace)

The words between the “ ”s are lines the opposing computer would say to give it a bit of personality.

10. Risks

1. The team underestimated how long specific parts of the project would take. This is unlikely due to the project having a primary focus on getting the card game war done with a few changes. if this issue does happen however please consult the team member about why it’s taking too long and if they need assistance to complete the task.

2. Team members might be busy with unexpected life events, or might have issues understanding parts of the design or code. This issues likeness depends on your group members and their lifestyle, nonetheless it is bound to happen as the groups are made of university students with multiple courses and different livelihoods. To prevent this, discuss with your group members on meetup times and see when people can work on the project. if a group member has an issue, be willing to take on some more of the work between other members.

3. At any point in the semester a loss of a team member could happen. This is unlikely, however if it does happen, notify Dr. Anvik about the issue, but proceed on course with the assignment and attempt to produce what's needed for that segment of the assignment.

4. Team members could not understand new concepts as well as others, thus feeling underproductive in certain areas. This is likely as we are all learning new concepts during the semester. Make sure to discuss these concepts with all group members and make sure you all understand them before proceeding. In the event that you can't discuss with a group member then do a small write up to help them understand and if they have additional questions to ask their fellow team members.

5. Team members may be using different IDE’s, which can cause compatibility faults. This is likely as every IDE is different from others. Thus team members should discuss what IDE’s they are using and if conflicts come up you could either learn a bit about other IDE’s to help one another, or go to the labs at the University to see these issues on a centralized machine.