**Syst 17796 Deliverable 1**

1. **Background Description and Description of the Project**

**Project Goals and Final Vision:**

This project’s major goal is to develop a working card game based on the old classic ‘War’ using the existing code. Our ultimate vision is to see the full backend for this game that would enable two players compete against each other by drawing cards from their deck and trying to have higher ranks. When a player draws a higher ranked card than his opponent in any round, they win and collect both cards placing them at the bottom of their deck. However, if both players draw cards of same rank, it leads to “war”!

**How to Play War:**

War is essentially a simple card game played with a standard 52-card deck. The process involve. Shuffling the deck before splitting it evenly between both players. Both players turning over their topmost card. Winner takes all (two cards) unless war breaks out because both players turn-up equal or matching cards. In a war, three cards are played face down by each player and then one card is revealed face up. The winner between the two players is the one with a higher ranked card on display. The victor takes all the cards of war from play, consisting of both his own and those of his rival. If the exposed cards have identical ranks, the situation results in another war until there is a winner. This game ends when either someone has collected all cards or when mutually agreed upon by both players to stop it.

It is Java code made, below which includes game classes like:

* Card: An ordinary card
* WarCard: A card in the game of War having rank and suit
* Player: An average player
* WarPlayer: A player of War
* GroupOfCards: Such as deck (a set of cards)
* Game: General game
* WarGame: A logic that deals with dealing out cards to playing rounds and deciding who wins for this war case.

This code follows Java convention by having appropriate class and method names, access modifiers and other necessary elements. Despite its technicality, it has been presented in such a way that it can be easily understood by even an unskilled reader.

1. **Project Scope**

**Team members and Roles:**

Amninder Kaur: Project leader and Developer

Jasleen Virk: Developer and Tester

Wajiha : Developer and Documentation Specialist

Anaya: Developer and Code Reviewer

Our team worked on an update for a card game called War, which involves dealing cards and comparing them until one player emerges victorious. We are responsible for ensuring that the game follows the rules while still being enjoyable to play. Amninder was overseeing this project and will be responsible for coding it. The coding of the game will also be carried out by Jasleen and Wajiha who focus on testing for functionality and providing clear instructions for gameplay respectively. Anaya checks code for errors and ensures its quality.

1. **High Level Requirements**

The new Game System should have the following:

* Player Sign Up: Players can enroll into the game using a unique name or identifier.
* Win/loss Announcer: At end of war, you either win, loss or draw.
* Card Counting: Information about how many cards you have left.

1. **Implementation Plan**

Git Repository Url: [amninderkaur/Jawas: war game deliverable1 (github.com)](https://github.com/amninderkaur/Jawas)

First, we created UML class diagram of the project. After every workday or week ends, we shall pool our code assets into one big cauldron. New things are added using feature branches before merging everything back to main branch during a good code review.

For organizations we make different folders for UML diagrams and documentation.

Coding Rules: We followed SOLID principal.

Tools:

IDE: NetBeans

Version Control: Git

Diagrams: Visual paradigm

1. **Design Considerations**

Encapsulation:

To maintain tidiness, we have separated the WarCard class into a separate entity that saves a card’s ranks and suits. It has access to the data only by its various methods in order to ensure proper control over its usage. The strategy of encapsulating the internal data structure by WarPlayer class is another example.

Delegation:

Delegation is our project’s watchword. For instance, WarGame class delegates cards dealing to GroupOfCards class which is responsible for it. It makes use of dealCards method of this object, so that players receive their cards. In a similar manner, the card-playing logic in WarPlayer class has been delegated to playCard method which takes care of all these aspects without exposing inner workings.

Flexibility/Maintainability:

We’ve written our code in an abstract way that makes it flexible and easy to extend using classes like Card and Player. This means that to add other card games all we need do is inherit from these abstract classes. We also split game logic (WarGame), player logic (WarPlayer), and card logic (WarCard) into separate classes; thus, any changes made within one component would not affect others thereby minimizing system impact.

Potential improvements:

Additionally, instead of dealing with such coupling systems at the game component level, we could go as far as creating interfaces for each individual element or part of the game system.

Overall, our project strives to create a robust and flexible system that can be easily extended and maintained. By following these object-oriented principles and considering potential improvements, we're on the right track!