Deliverable 2: Design Document

Bhavikkumar Patel, Darren Scarfo, Kevin To, Neil-Bryan Caoile

Faculty of Applied Science & Technology, Sheridan College

SYST17796 Fundamentals of Software Design and Development

Sivagama Srinivasan

July 21, 2021

Contents

Use Case Narrative

The following is the primary Use Case with alternate paths, for a game of War:

1. Launch game

2. Enter Player 1 Name

    2.1 Username invalid

3. Enter Player 2 Name

    3.1 Username invalid

    3.1.1 Choose AI opponent

4. Players "Begin Game"

5. Shuffle deck and distribute cards into two even groups of cards, one for each player

6. Each player draws a card from the top of their deck

7. Compare values of cards

    7.1 If same value, War state begins

    7.2 Additional cards are drawn (one face-down, one-face-up)

8. Player with high-card takes all cards with notification of winner/loser; winning cards are placed on bottom of winner's deck in random order

    8.1 War state ends if currently active

9. Repeat 6-8 until all cards are gone until one player has all 52 cards in their deck

    9.1 A player does not have sufficient cards to enter a War game state

10. Winner of game is declared

11. Game over, game stats printed (i.e., number of rounds, number of war states)

12. Play again

Use Case Diagram

The following is the use case diagram for the game of War:

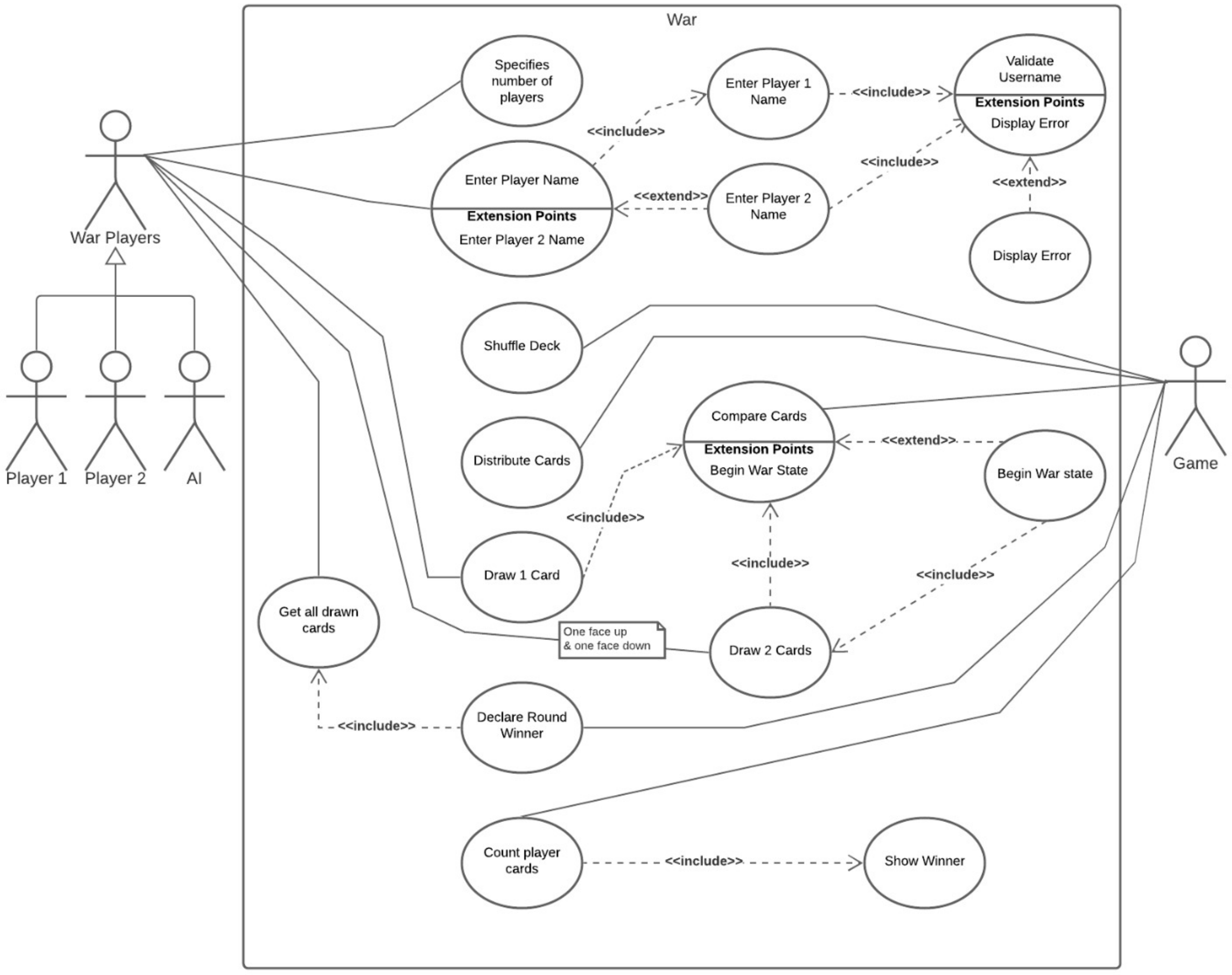
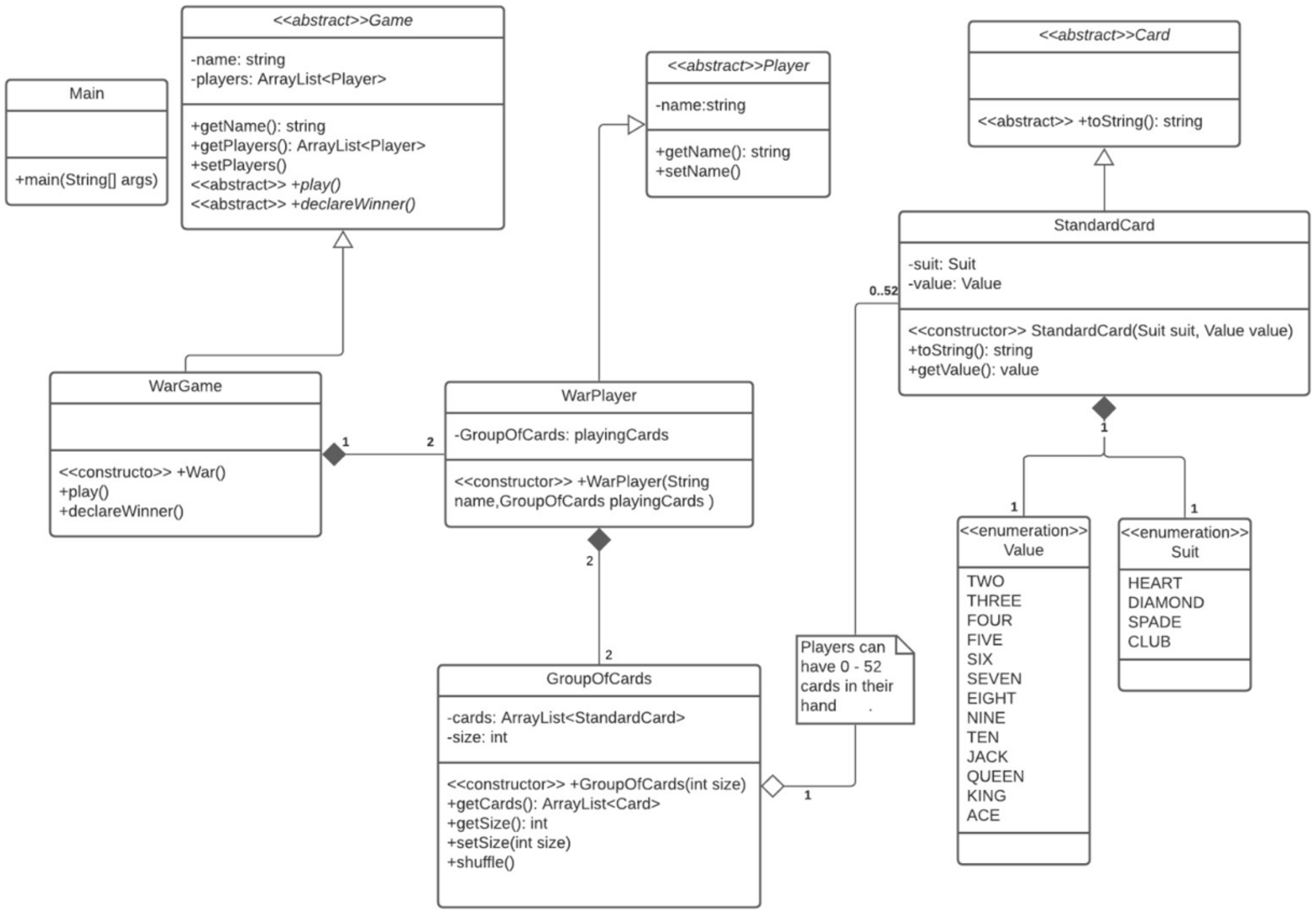


Figure 1: Use Case Diagram

Class Diagram

The following is the planned class diagram for our application:



<<abstract>>Game
-name: string
-players: ArrayList<Player>
+getName(): string
+getPlayers(): ArrayList<Player>
+setPlayers()
<<abstract>> +play()
<<abstract>> +declareWinner()
Main
+main(String[] args)
<<abstract>>Player
-name:string
+getName(): string
+setName()
<<abstract>>Card
<<abstract>> +toString(): string
StandardCard
-suit: Suit
-value: Value
<<constructor>> StandardCard(Suit suit, Value value)
+toString(): string
+getValue(): value
WarGame
<<constructo>> +War()
+play()
+declareWinner()
WarPlayer
-GroupOfCards: playingCards
<<constructor>> +WarPlayer(String name,GroupOfCards playingCards)
GroupOfCards
-cards: ArrayList<StandardCard>
-size: int
<<constructor>> +GroupOfCards(int size)
+getCards(): ArrayList<Card>
+getSize(): int
+setSize(int size)
+shuffle()
<<enumeration>> Value
TWO THREE FOUR FIVE SIX SEVEN EIGHT NINE TEN JACK QUEEN KING ACE
<<enumeration>> Suit
HEART DIAMOND SPADE CLUB
Players can have 0 - 52 cards in their hand
0..52
1
2

Figure 2: Class Diagram

Project Background and Description

As outlined in Deliverable 1, the goal for our group, referred to as the “Warriors”, is to successfully develop a software application that follows the requirements of our SYST 17796 Fundamentals of Software Design and Development. Specifically, our end goal is to develop a functional and enjoyable version of the popular card game “War”.

The game of “War” is known as an “accumulating card game,” where the goal is for one player to “win” the cards of the other player in a series of discrete turns or rounds. More specifically, according to Bicycle Cards (2021), War:

- Is played with a standard 52 deck of cards,
- Is played by only two players or one player versus the artificial intelligence,
- Assumes that aces are ‘high’, meaning they have the highest value in the deck.

The game begins with the program asking the user for Player 1’s username. The program will then provide a prompt for Player 2’s username or, alternately, the user has the option of not entering a Player 2 username and the game will commence with one player versus an artificial intelligence opponent. It should be noted that, given the rules of War, the game will function the same regardless of whether there is a second player or an artificial intelligence opponent. This is because each round of War is driven by one user clicking “Draw”; all card comparisons and winner declarations will be decided automatically by the program. There is no required input from a second user in this game.

Upon entering usernames or deciding on an AI opponent, the game will automatically shuffle the deck before distributing the 52 cards evenly between the two players (kindly note that the artificial intelligence opponent will also be referred to as a “player”).

The players will then draw a single card each from their shuffled deck, with each player being unaware of the value of their card. After, the players will compare their cards and the higher valued card will win. The winner of the round will take both cards and place them at the bottom of their deck (the two cards will be shuffled automatically before they are placed back at the bottom of the winner’s deck). Players are not permitted to shuffle their decks throughout the game.

However, if the two cards that the players (or artificial intelligence) draw are of equal value, the game-state referred to as “War” is declared. During this game-state, each player will draw one card face down and one card face up. The player with the higher valued card will take both piles (totaling six cards). It should be noted that the state of “war” may occur for multiple instances before there is a winner. “War” is only over once there is a declared winner of the cards that were drawn; the same rank cards will always result in “war”. In this case, the winner will take every card that was drawn and place them at the bottom of their deck (the cards will be shuffled before returning to the bottom of the deck). Upon each round, the winner is announced

The game will conclude once one player has won all the cards from the 52-card deck **or** when a player is unable to provide a sufficient number of cards to participate in a War state. There is no set number of rounds. Upon a winner being decided, a declaration of the winner is announced upon the end of the game with relevant statistics being provided (including the total number of rounds and the total number of War states that were played).

The starting code that our group will build upon is provided by our professor, known as “SYST17796 Project Base Code”. The initial authors of the starting code are the following individuals: Dancye, Paul Bonenfant Jan 2020, and Megha Patel. However, Professor Sivagama Srinivasan will overlook the completion of this project.

Our original coding standards will remain unchanged from Deliverable 1, as we will not deviate from these applications and standards.

Design Considerations

The base code included four classes in total, three of which are abstract classes (Card, Game, and Player) and one of which (GroupOfCards) is a concrete class.

Building off of this base of code, as outlined in the class diagram above (Figure 2), we developed three classes that are concrete extensions of the three abstract classes. These classes include "StandardCard", "WarPlayer", and "WarGame".

The StandardCard class is a concrete implementation of (i.e., inherits from) the Card class and will model card objects based on a standard deck of cards. The abstract Card class is defined in the Base Code as "used as the base Card class for the project. Must be general enough to be instantiated for any Card game" and only includes an abstract toString() method with no implementation.

As such, our StandardCard class will include a concrete implementation of the toString() method. Additionally, we will use two enumerations ("enums") in order to represent the suit and value of the StandardCard objects. Enums allow for a variable to be a set of predefined constants, meaning that future variables of this type must match the values predefined for it in the relevant enums. In this case, because there is a limited, set number of suits (Heart, Diamond, Spade, and Club) and a limited number of values (ranging from Two to Ace) it makes sense to delegate these variables to two enums, rather than use an alternate data structure like a String array (on which it is notoriously difficult to perform operations). Enums also allow for increased type safety; the compiler will validate the type of a variable and throw an error at compilation instead of at runtime. These two enums are related to the StandardCard via a composition relationship, given that suit and value are instance variables in StandardCard, that StandardCard functionally owns the suit and value, and that suit and value cannot exist on their own.

The StandardCard class is related to the GroupOfCards class, of which it has multiplicity ranging from 0 to 52, denoting the fact that a GroupOfCards can contain between 0 (no cards) and 52 (all cards) in a given GroupOfCards. The GroupOfCards is a concrete class as provided by the Base Code; it is related to the StandardCard class via an aggregation relationship, as the GroupOfCards class has an arraylist of StandardCard objects as instance variables, indicating ownership, but that a StandardCard objects can be owned by a different GroupOfCards object. We would expect to create two GroupOfCards objects to represent the hands of the two players (as all cards are divided among the two players at all times in the game of War, there are never any unowned cards in a game of War), hence the multiplicity of 2.

The Player class as provided by the Base Code is an abstract class that models each Player in the game. As described above, War is a game played between two players, and again, it should be noted that War is a game that is actually played (or triggered) by one player exclusively; there is no functional difference in terms of how the game is played between a one-player game (versus an AI) or a two-player game (in which two players have entered their own names). As such, the WarPlayer class is a new concrete class that inherits from the abstract Player class, and will, inturn, be composed of GroupOfCards objects; each of the two players (be they two live players or alternately, an AI and a live player) will have their own exclusive GroupOfCards objects.

The Game class is an abstract class from the Base Code that provides multiple methods with no implementation. WarGame is a concrete subclass of this abstract class that we will create, which will implement this abstract class and define the rules required to play the game. The WarGame class will have a composition relationship with the WarPlayer class, with a multiplicity of 1 WarGame based off of 2 WarPlayer objects that represent the two players of the game.

We have also included a Main method, which will actually execute upon launching of the program. We decided to make this class its own separate class in order to separate functions among classes and properly encapsulate our classes and avoid coupling.

Moving forward, as we continue to design and code the game of War, we will focus on having loose coupling and high cohesion. Specifically, we will implement loose coupling via making fields private, using private methods, and limiting interconnection between modules to the minimum number in order for the program to function effectively. We will ensure high cohesion within classes by making sure that classes are as independent as possible and only performing one specific task whenever possible. We will further focus on code maintainability/flexibility by ensuring that we only write code once, structuring our code to avoid repetition.

References

How to play: War. (n.d). Bicycle Cards. https://bicyclecards.com/how-to-play/war/