Card Game C++ TDD & GDD

v-1.0

Michael Sweeny

# Table of Contents

[**Table of Contents**](#_gt7ae64kmyfp) **2**

[**Introduction**](#_lvso6hpkwq5x) **2**

[**Development environment**](#_nsmzrtbge510) **2**

[**List of Technical Mechanics**](#_v1orruw8li40) **2**

[Enemy A.I.](#_nsa21z28m6zw) 3

[War](#_ewg3z4ybo39i) 3

[Go Fish](#_y1u86ymygp8u) 3

[**Pseudo-Code**](#_z9wqi8mr9s73) **3**

[**Timeline**](#_gx2sx8iwkaus) **4**

[**UML**](#_mnz12fqm6ykh) **5**

# Introduction

This is the technical design documentation for my first full attempt at making a program in C++ that runs as an executable file from the desktop. The goal is to have a welcome screen, a main menu which includes options for 2 card games and to quit. The card games should be fully playable versus the computer.

# Development environment

The project will be written in C++ using Visual Studio 2017. The final program should be delivered as an executable file that does not require opening the solution in an editor to trigger the console to run.

# List of Technical Mechanics

1. **Basic input** - Keyboard:
   * for inputting player name for all fanfare
   * For inputting choices for prompts
2. **Main Menu**
   * Card Games
   * Quit
3. **Card Games**
   * War
   * Go Fish
4. **Welcome Screen**
   * “Press any key to continue”

## Enemy A.I.

### War

Enemy AI will be very simple for the two games. In war, it simply will have to draw a card when the player does.

* The game itself can compare the two and award the player who won the round.

### Go Fish

The AI will have to choose a random card from its hand to ask for when its turn begins.

* The game itself will handle checking either hand for a card that has been asked for, as well as holding the pairs to the side for each player to count score once all cards have been played.

# Pseudo-Code

In this program hierarchy I will need a CardProgram class to house my two games under one parent, the Game class which is to contain all my main game functions. The CardProgram class will hold my WelcomePlayer() function which will play a welcome screen slow-text write out message and then clear the console. Followed by a Menu() function which will show the option to Quit or to SelectGame().

SelectGame() will launch the War class functions or the Go Fish class functions (as well as their parent class: Game) by inputting the correct integer. Passing 0 to Menu() will quit the program. The game class will also store the array for the deck of cards since both games only use a single deck. This means game can also include the Shuffle() function which will be accessed by Play() - a pure virtual function defined in each game class. I believe the game class can also hold the NextPlayer() function since both card games will need this functionality and can be friend classes of Game to gain access to this member. Both games will also need a pure virtual Deal() which we can define later. Each game will need its own attribute to store the players’ hands.

The war class will need functions to Deal() cards to the player and the AI. This game doesn’t require displaying the cards dealt, so next we have a function to Flip() the top card from both hands which will access the function Compare() to decide the winner and push the winning pair to the winning player’s sideHand[]. Flip() will start by asking if both players have cards, if either player is out of cards, the CheckCount() function will see if the player has cards in their sideHand and will shuffle them, add them back to the players hand and continue to execute Flip(). I will also need a GoToWar() function for when the user and AI pull the same card. It will automatically pull three cards from each players hand and then Flip() the top card of each deck, repeating until the conflict is resolved. If CheckHand() finds no cards in the players reserve, the AnnounceWinner() function will execute for the other player.

The GoFish class will need functions to Deal() cards to the player and AI. This time we will need a Display() function which will only show what the player’s hand is, as well as a function to Draw() a card from the remaining deck. There will be a function for Ask() which will allow the current player to request cards from the opposing player (the player must have the card they are requesting), this function will call the CheckEnemy() boolean function to see if they have the card requested. If it returns true, the MakePair() function will take that card from both players hands and add 1 pair to the players’ score. Since we only need to keep score like this in this game, we can just make 2 private ints in this class, one for each players’ score. When there are no cards left in the deck when Draw() is called on the Stack, the function will tell the player that there are no cards left and continues the game. Before Ask() calls HandValidate() to determine if you are asking for something you actually have one of, it will call HandCount() to see if the game has ended. If there are no cards left in the deck && the player has nothing in their hand, that means the game has ended, HasWon() can determine a winner based on pair count and AnnounceWinner() can print the fanfare.

# Timeline

I am expecting to begin programming this during the week of March 25th. That leaves 3 weeks to work on the program, with the final week being almost entirely dedicated to completing the necessary components (I hope to spend at least the last day or two only working on debugging things like input control).

# UML

## 