Procedural Programing Project

*“War” Cards Game with C*

Version of “war” card game developed for assignment of procedural programing 1. Highest Card game.

Jose Ignacio Retamal,

3 January 2018,

GMIT.

Procedural Programing Project

“War” Cards Game

Contents

[Introduction 2](#_Toc502833503)

[Structs 2](#_Toc502833504)

[Card 2](#_Toc502833505)

[Player 2](#_Toc502833506)

[Turn 3](#_Toc502833507)

[Game 3](#_Toc502833508)

[Functions 4](#_Toc502833509)

[displayCard(Card card) – void 4](#_Toc502833510)

[createDeck(Card deck[]) - void 4](#_Toc502833511)

[dealCards(Player players[], Card fullDeck[], int numOfPlayers)-void 4](#_Toc502833512)

[displayPlayerCards(Player player)-void 4](#_Toc502833513)

[displayCardsTurn(Card cards[])-void 5](#_Toc502833514)

[calculateWinner(Card turnCardsTemp[], int numOfPlayers)-int 5](#_Toc502833515)

[calculatePoint(Card turnCardsTemp[], int NumOfPlayers)-int 5](#_Toc502833516)

[createNewGame(Game \*game)-void 5](#_Toc502833517)

[displayGameStats(Game game) 5](#_Toc502833518)

[playGame(Game \*game) – void 5](#_Toc502833519)

[saveGame(Game game)-void 6](#_Toc502833520)

[loadGame(Game \*game, char gameNameParameter[])-void 6](#_Toc502833521)

[addGameName(char gameName[])-void 6](#_Toc502833522)

[loadGameNames(char gameNamestrings[]][])-void 6](#_Toc502833523)

[displayGameNames(char gameNamestring[][])-void 6](#_Toc502833524)

[getNamesStringLength(char gameNamesstring[][])-int 7](#_Toc502833525)

[checkIfNameExist(char gameName[],char gameNamestring[])-int 7](#_Toc502833526)

[clearConsole()-void 7](#_Toc502833527)

[displayPointsOrdered(Game game)-void 7](#_Toc502833528)

[checkCards(Player players[],int numberOfPlayers)-int 7](#_Toc502833529)

[main()-void 8](#_Toc502833530)

[Conclusion 9](#_Toc502833531)

# Introduction

I have chosen to solve the problem using structs, I know that is probably not the simplest way to do it, but I think it will help me to understand better how they work. I have use a structure for each card, player, turn and one for the game stats.

All the game is developed in only one file(main.c), I think for this situation this is fine because the program gets well-structured using functions. The order of the function declaration is the same in which I develop the program.

There are three “main” functions: main() ,playGame() and createGame(). In this function is where all the others function get call. On these 3 functions is where most of the GUI happens.

# Structs

## Card

Basic structure that represent a single card. The deck is an array of Cards, the deck contains 52x3 Cards.

Cards are displayed using a function that display a letter in the value when needed.

Members Definition

value(int): value of the card, from 2 to 14.

suit(char): suit of the card. H = hear, D = diamonds, C = clubs, S = spade.

Null Card

A card with value of -1 and a suit of X is used as null card.

## Player

Represent a single player, in the game there is a minimum of 2 and maximum of 10 players.

Members Definition

-number(int): From 1 to 10, each player have a different number and represent the order to play.

-score(int): Total score of the player at a specific time on the game.

-hand(Card[]): Array of Cards that represents the cards that the player have. When a card is used is replaced with the Null Card.

## Turn

Represent 1 turn of the game, there are 13 turns in the game.

Members definition

-turnPoint(int): total disputed point in the turn. Added after each player plays.

-winner(int): Player -> number of the winner of the turn, -1 if there is no winner.

-turnCardsTemp(Card[13]):Cards played in the turn, this array will be use for find the winner, the repeated cards will be removed. Hold 13 cards.

-turnCardsPerm(Card[13]): Cards played in the turn, no Card will be removed from this array, it will be used for show the turn Cards played. Hold 13 cards.

## Game

Store data for 1 game, there is only 1 Game referred on all the time in the program. When creating a new Game, a unique name must be assigned by the user, games are stored in different files where the file name is the name of the Game and there is another file with all the names of the games saved.

Members definition

-name(char[30]): String that hold the name of the game which must be assigned by the user.

-turnNum(int): From 0 to 12, 1 will be added when show turn to user(1-13), is the actual turn on which the game is. Will be 13 when the game is over.

-pointOnHold(int): Rolled over points, if there are any, will be 0 if there are not.

-numberOfPlayers(int): From 2-10, total of players in the game. This value is chosen by the user when a new game is created.

-turns(Turn[13]): Array of Turn, 13 turns with data of each turn.

-players(Player[10]): Array of players, will store data for each player, from 2 to 10, if there are less than 10 players, the remaining players will be left undefined.

# Functions

## displayCard(Card card) – void

Take a card as argument and display that card value and suit in the console using the function “printf”. Card is show as “value-Suit”, example = A-H.

## createDeck(Card deck[]) - void

Takes an array of cards as argument which must be of size 156(pointer ), the same array is filled with the cards.

Fill the deck with 156 cards, first create a subDesck of 52 cards(like a normal deck) and then using that deck fills the deck[].

## dealCards(Player players[], Card fullDeck[], int numOfPlayers)-void

Deal cards from “fullDeck” to players[0-numOfPlayers]->hand. First create “shuffledDeck” using “fullDeck”, then deal the cards to each player in the parameter array players[] using the parameter numOfPlayers which is the number of players in the game.

## displayPlayerCards(Player player)-void

Display the cards of the players excluding those that have been used(null cards with value of 0).Also display the position/value of the card for the user to pick one.

## displayCardsTurn(Card cards[])-void

Display an array of cards, used to display the cards played for each player on each turn. Display cards until Null card is encounter or reach the maximum(13).

## calculateWinner(Card turnCardsTemp[], int numOfPlayers)-int

Used for calculate the winner of each turn, will return the player number of the winner or -1 if there is no winner. First parameter must be Game->Turn[x]->turnCardsTemp[], repeated cards will be replaced by null card and then the higher card value will be found, if found will return the position of that value.

## calculatePoint(Card turnCardsTemp[], int NumOfPlayers)-int

Return the total sum of the values of the card in the first parameter array as an int.

## createNewGame(Game \*game)-void

Create a new game using the game pointer parameter, user inputs (game name and number of players) and functions(“createDeck()”, “dealCards()” and “checkCards()”).

First input game name and number of player from user, then initiate required values and then create a deck using the functions . When creating the deck and dealing card checks it the deal is correct(each player have at least one card of each suit)using the function “checkCards()”.

## displayGameStats(Game game)

Display basic stats of the parameter game: number of players, actual turn, rolled over points and scores (using displayAllPlayersPoints() function).

## playGame(Game \*game) – void

Function where the game happens, is call with the Game pointer that have been load or created, work with new games or with games that are at mid stage .

Use a “big” while loop, one loop represents one turn, will run until the variable “gameIsOn” is modified. Inside this loop there is a for loop which will run one time for each player. After each itineration of the “big” while loop the program will ask the user if they want to play the next turn or save the game or if the game is over will show the game stats.

## saveGame(Game game)-void

Save the game to a file with the same name of the game parameter and using the FIlE\_EXTENSION constant.

First check if the game is all ready saved, if is not add the name to the file that store all games names.

It can be use with games in any stage and can save games that are ready saved, when this happens the file will be re-write.

## loadGame(Game \*game, char gameNameParameter[])-void

Load a game using the game name string parameter(gameNameParameter[])), and modified the Game parameter. Follow the same structure used for the function saveGame().

## addGameName(char gameName[])-void

Add a game name to the file gameStr.dat where all the games names are save. It will append the new name to the file.

## loadGameNames(char gameNamestrings[]][])-void

Load the game names saved on the gameStr.dat file to the array of string parameter.

If the gameStr.dat file do not exist it will create the file, and add a “99999” for initiate the array if there are no names(this is used for show nothing when there are no games saved) and will have no use if there are names.

End of the array of string gameNameString[] will be mark with the value ”99999”.

## displayGameNames(char gameNamestring[][])-void

Display the cards of the array of string parameter, if the first value of the array is the “99999” will prompt that there are no saved games, if is not will display names until “99999” is encounter.

## getNamesStringLength(char gameNamesstring[][])-int

Return the length of the array parameter, that the number of games loaded. Will count the length using the end of array value(99999).

## checkIfNameExist(char gameName[],char gameNamestring[])-int

Check if the first string parameter is on the second string array parameter, if it is will return the position, if there is not will return -1.

## clearConsole()-void

Remove any undesiring character on the stream. Used after read and input.

## displayPointsOrdered(Game game)-void

Display the score of all players after the game is done, will display scores in descending order(Winner at the top).

shuffleCards(Card fullDeck1[], Card sDeck[])-void

Shuffle cards of a deck, distribute in a random way the cards from first parameter(fullDeck1) to the second parameter(sDeck). Both parameters must be and array of Cards of size 156.

## checkCards(Player players[],int numberOfPlayers)-int

Check if each player in the first parameter array have at least one card of each suit, if they have will return 1 if not will return 0.

## main()-void

Starting point of the program, Game structure is declared and initiated and an array with the name of the games saved, that basically of the data that the game need(adding each file with the data of each game saved).

First the game have 3 options: new game, load game and exit, those 3 options are selected using a switch statement and the program run a while loop until the user select exit.

If the user chose the option create new game, the function createNewGame is call and after the function playGame(), both use the Game structure(there is only one Game structure all the time in the program).

If the load game option is selected the user will be prompt with all the games available (using displayGameNames()), then the game will be loaded using loadGame() for after call playGame();

# Game Flow

## Display a card

Cards structs have 2 members (value and suit), they are displayed calling the function displayCard(), which will reamplace the value for the corresponding letter when needed (14 for A).

## Dealing cards

Done using 4 functions: createDeck(), dealCards() ,shoufleDeck() and checkCards().First a deck is created (createDeck()) and after dealCards() is call with that deack. Cards are deal to each player, in the dealCards() function, after calling shoufleDeck() in the same function. After the cards of each player are checked using checkCards(), if the deal is not good all the process is done again.

The dealing is done with one deck of 156 cards for any number of players.

Dealing cards happens only when a new game is created.

## Creating a new game

One function will create the game, initiate the requires values of the game and then create the deck and deal cards.

## Saving game

Games are saved using the function saveGame(), in one file are save the names of each game and one independent file for each game. All the data of the game is saved on that file.

## Loading game

The game are loaded to the one Game struct using the function loadGame() which have as parameter the pointer to that struct.

## Playing the game

The game is played using the function playGame() which is call with the Game struct and can be call with a new game or with a game loaded.

It work with 2 main loop: one for each turn and another for each player.

# Conclusion

I am happy with the solution because it works well and I learn a lot about how to use structures. My estimation is about 40 hours of coding and testing the program which sound a lot, but I don’t feel like I waste my time because I learn many new thinks.

I think all the requirements are met. There are some possible enchantments like the option to delete saved games or play again an AI.