CS 246 Fall 2018 Project - Euchre

M. Godfrey, C. Kierstead, B. Lushman, N. Naeem

Due Date 1: Monday, November 19, 5:00pm
Due Date 2: Monday, December 3, 5:00pm

Project Description

This project is intended to be doable by 2-3 people in 2 weeks. Because the breadth of students' abilities in this course is quite wide, exactly what constitutes 2 weeks' worth of work for 2-3 students is difficult to quantify. Some groups will finish quickly; others won't finish at all. We will attempt to grade this assignment in a way that addresses both ends of the spectrum. You should be able to pass the assignment with only a modest portion of your program working. Then, if you want a higher mark, it will take more than a proportionally higher effort to achieve, the higher you go. A perfect score will require a complete implementation. If you finish the entire program early, you can add extra features for a few extra marks.

Above all, **MAKE SURE YOUR SUBMITTED PROGRAM COMPILES AND RUNS**. The markers do not have time to examine source code and give partial correctness marks for non-working programs. So, no matter what, even if your program doesn't work properly, make sure it at least does *something*.

1. Euchre Game Description

The course project is based on a multi-player version of the card game of euchre (pronounced "YOO-KER"). There are many different versions of the game worldwide, but we will only consider the North American version.

Below are a few web sites that have rules and strategies to the game:

- `http://userpages.bright.net/~double/euchre.htm`
- `https://www.pagat.com/euchre/euchre.html`
- `https://www.thespruce.com/euchre-rules-tricktaking-card-game-409350`

There is also a commercial site devoted to euchre at `http://www.thehouseofcards.com/euchre.html`. At this site, there are books, electronic games, sample software, and a list of online sites devoted to the game.

The game specification follows. If there is something in doubt in this specification, it will be addressed on Piazza. **You are thus required to read Piazza regularly!**

1.1 Objective

Euchre is a card game with two teams of two players using a 24-card deck, running from Aces down to the 9's (named A, K, Q, J, T, 9 in the description to keep things simple) in 4 suits (H = Hearts, D = Diamonds, C= Clubs, S = Spades). The players sit around a common area, such as a tabletop, so that one team-pair (players 0 and 2) faces each other, and the other team-pair (players 1 and 3) faces each other, as in Figure 1.

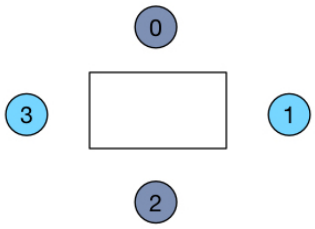

Figure 1

The objective of the game is to be the first team to score 10 (or more) points.

We ***HIGHLY*** recommend that you play this card game within your group ***BEFORE*** trying to implement the game.

1.2. Game Play

Before going into details of the game, there are a few terms that every euchre player ought to know.

1.2.1 Terminology

- A *round* is completed whenever each player has played a card.
- A *hand* consists of five rounds, *i.e.,* when all the cards that have been dealt have been played.
- The term *trump* is probably the most important concept in the game. The trump suit is set at the beginning of the hand by the trump-calling process, and the cards in that suit are the most powerful cards in the game. So, a 9 of the trump suit beats an ace of any other suit, although an ace of trump will defeat that same 9. Note, though, that the ace of trump is not the highest card for that hand (see the following definition of *bower*). This concept is similar to that used in the game of bridge.
- An *off-suit* is any suit but trump.
- A player is considered to be *void* in a particular suit if the player has no cards in that suit.
- The *kitty* is the stack of four cards left over in the deck after the deal is complete.
- The Jack of the trump suit is called the *right bower*. It is the highest card of all of the cards in the round. For example, if the trump suit is hearts, the Jack of hearts is the right bower.
- The Jack of the suit that is the same colour as the trump suit, but not the trump suit, is called the left bower. It is the second highest card of all of the cards in the round. If the trump suit is hearts, the Jack of diamonds is the left bower. After the bowers, the rest of the hearts cards in descending order (A, K, Q, T, 9) are highest. None of the other diamond cards count as trump, and follow the usual rules of play.
- In general, whenever a person wins a *round*, their team gets one *trick*. 1 point is given to the team who receives either 3 or 4 tricks in a hand, and 2 points if the team receives all 5 tricks in a hand. However, the opposing team can instead receive 2 points if a *euchre* is played (more on this later). *Hands* are played until one team (the winner) reaches 10 or more points.
- A player can *order up* the dealer during the calling trump process. This signifies that the dealer **must** pick up the card. This is clarified further in section 1.2.3.
- The player that is calling trump can choose to declare that they are *going alone*. (Note that if the player orders up their partner, they are implicitly choosing to go alone.) A player who does this believes that their hand is strong enough to take all 5 tricks without the assistance of their

partner. **Thus, their partner will not play any cards in the hand**. See section 1.2.5.1 for how this situation is scored.

- In certain circumstances, a player that has a single card in a particular suit can choose to *void* itself in that suit by playing that card when it cannot follow the suit of the led card. For example, if hearts are trump, the Ace of clubs is led and the next player plays the Queen of hearts, a player who has {9H, TH, AD, QS} in their hand may choose to play their Queen of spades so that if spades are led, they might have the opportunity to play one of their trump cards and possibly take the trick if everybody else has to follow suit.

1.2.2 Dealing

One player is chosen, normally at random, to deal the first hand. This player we will refer to as the *dealer*. After each hand is played, the deal passes clockwise around the table. So, as in Figure 2, if player 0 is chosen as the dealer for the first round, the deal passes to player 1 for the next round. From the viewpoint of player 0 sitting at the table, player 1 is sitting to its left.

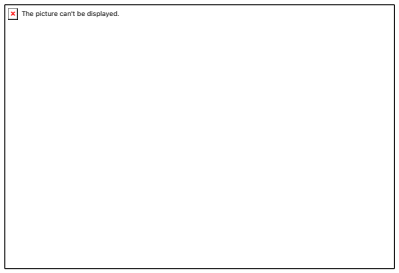

Figure 2

The dealer then deals 5 cards to each of the 4 players. At the end of the deal, 4 cards are left over, forming the kitty.

1.2.3 Calling Trump

The dealer turns over the **top** card of the kitty to reveal the suit proposed for trump. Starting with the first player to the dealer's left e.g. player 1, each player in turn has the option of either declining that suit as trump (called *passing*, and done by saying "pass"), or accepting that suit as trump by *ordering up* the dealer[1]. If all players pass, then the top card of the kitty is turned over to indicate this.

If the dealer is ordered up, they **must** pick up the top card of the kitty. It is up to them, however, as to which card from their hand is discarded, face-down, on the top of the kitty to bring their hand back down to 5 cards in number.

If all players pass, then starting with the player immediately to the dealer's left, each player in turn is given the chance to propose a trump suit. **A player may not propose the same suit as what was turned down!** The first player to declare a suit sets the trump suit for the round, and bidding stops. If nobody is willing to declare a suit trump, then the round is nullified i.e. all cards are returned to the deck and a new round is started.

[1] Note that the dealer can choose to make this suit trump by picking up the top card of the kitty and discarding one of the cards from their hand. There is no specific term for this, but it's not called "ordering up" in this situation.

1.2.4 Game play

The player to the left of the dealer *leads* by playing a card.[2] The card can be any card from the player's hand. Play then proceeds clockwise. The next 3 (2 if the person calling trump is going alone) players **must** play a card of the same suit, if able. If not, they may either *trump* (play a card of the trump suit) or *throw off* (play a card that is neither of the suit of the card played, nor of the trump suit). If one of the players does not follow suit, and that person is able to do so, then that player has *reneged* and an automatic 2 points is awarded to the other team. In real life, reneging is detected by a player noticing that someone played a card in a suit where they had previously not followed suit. In your game, the game logic can see the player's hands and tell if someone had a card in the suit to be followed or not.

1.2.4.1 Trumping

Trumping is accomplished by playing any card of the trump suit. Since even a nine of trump will beat any card of an off suit, this is sometimes a good idea.

1.2.4.2 Throwing off

Throwing off means playing a card that is neither trump nor follows suit. As mentioned above, if you can follow suit and you do not, then you have reneged. If you throw off, you personally cannot win the trick, so this is generally only a good idea when it looks like your partner will take the trick. It is also a good way to void yourself in a particular suit so that you can play trump on that suit if it is led later. For example, if a player has only a single spade card, and no clubs, when clubs are led, they may choose to play the spade if they can't take the trick so that they can later play a trump card if clubs are led again.

2.4.3 Winning rounds and finishing the hand

The person that played the highest trump card takes the trick. If no trump was played, then the trick is awarded to the person that played the highest card of the suit led. So, for example, if hearts are trump and clubs were led, and the cards played consisted of {AC, TC, 9C, 9S}, played in that order, then the Ace of clubs takes the trick. Remember that in off suits, aces are high. The player who took the trick leads the next round. The rest of the rounds are played in a similar fashion, with the winner of each round leading the next.

Once all 5 rounds have been played, each team totals their tricks. The team with the most tricks wins the hand.

1.2.5 Scoring

If the team that called trump takes either 3 or 4 tricks, they earn 1 point, plus 1 bonus point if they take all 5 tricks.

If the team that called trump fails to take at least 3 tricks, then this team is awarded 0 points and the other team is given 2 points. This is called a *euchre*.

1.2.5.1 Going alone

In the trump-calling process, if you think that your hand is really good, you may want to consider *going alone* if you are the one calling trump. This means that your partner does not play in the hand and your partner's cards are discarded face down. Thus, the hand is played with only 3

[2] The only exception to this rule is if the player directly clockwise of the dealer goes alone, then the player opposite the dealer leads; see section 1.2.5.1 for further details.

players. If you get all 5 tricks by yourself, then you score 4 points. If you get only 3 or 4 tricks, then you still only get 1 point. If you fail to obtain at least 3 tricks, the usual euchre scoring applies.[3]

2. Implementing Euchre

The program you are implementing has a text interface i.e. commands are entered from the keyboard, and output is sent to the screen.

Question: One possible additional enhancement is to add a graphical user interface. Explain briefly why, especially if you choose this enhancement, you would want to structure your project to follow the *MVC* design pattern.

Whenever input is required from the user, the program prints the prompt "`> `" (without the quotation marks). The list of valid commands is described in section 2.2.1.

2.1 Running your program

In order to be able to mark your program, your program needs to output the state of the game, such as the cards each player holds. This, however, doesn't make for a very interesting game, so this will be a feature that can be turned on by passing in a flag, `-d`, on the command line.

As well, the execution of the program must be repeatable for testing purposes, so your command-line argument must also be able to pass in an integer seed value, indicated by the `-s` flag, to the (pseudo) random number generator[4]. If no seed is specified on the command-line, the value of 0 is used for the seed. If both the seed and the debug flag are given, you may assume that the debug flag occurs first. You are not required to check that the seed is an integer.

For test purposes, you must also be able to initialize a game from a file. This lets you test very specific scenarios and responses. The name of the file will be provided on the command-line as the parameter following the `-f` flag. You may assume that if the `-f` flag is present, it will follow all previous flags. You may also assume that the specified file exists, is readable, and follows the format you have decided upon. You will thus need to submit your test files to Marmoset along with your solution.

The format of the command line is as below, where the square brackets indicate the optionality of the argument, and are <u>not</u> physically present:

```
./euchre [[-d] [-s seed] [-f filename]]
```

If both a file and a seed are specified, the seed value is silently ignored since it is replaced by the value specified in the file. So, possible command invocations are:

```
./euchre
./euchre -d
./euchre -s 123
./euchre -d -s 123
./euchre -f saved-game.txt
./euchre -s 123 -f saved-game.txt
./euchre -d -s 123 -f saved-game.txt
```

When the game starts without being initialized from a file, a player is randomly chosen as the dealer, the deck is initialized, shuffled and dealt out. If player 0 is the initial dealer, the deal moves clockwise

[3] There are scoring variations to this, but we'll stay with this simple version.
[4] See the description of how to shuffle the cards in section 2.3.2.

around the table, to player 1, then 2, etc. The cards will then be dealt in any reasonable fashion. The remaining 4 cards will be set aside as the *kitty*, and the top card of the kitty is turned over to face up. In debug mode, the contents of each hand and the kitty will be displayed as text to standard output; otherwise, just the top-card of the kitty and the human player's hand will be printed. See the sample executable, and section 2.4 for the suggested output format.

As described in 2.2, initially there will be 1 human player and 3 computer players.

2.2 Implementing the players

Since teaching you how to communicate across a network is beyond the scope of the course, we will instead resort to having 1 player be the human playing the game, and the other 3 players be simple computer players. (If you are sufficiently knowledgeable to know of an easy way to do this, this would make a fine bonus.)

Question: There are several possible approaches to setting up the player classes so that the game can easily replace the human player with a computer player without having to change the game logic or how it interacts with the player class. Briefly explain the design approach you took.

Note: You will be often comparing cards to determine which is the largest, which will depend upon the trump suit, and the context, such as whether or not you must follow suit versus finding the largest card in a player's hand. You will also frequently want to count the number of cards in a suit, or the number of off-suit aces, or find the largest (or smallest) card in a particular suit. Take some care in designing your card class to make this easy. It would also be a good idea to create some helper methods to search the hand right at the beginning, and make sure that they work properly before you implement more complex sections of code.

2.2.1 Human player

The human player takes the role of player 0. Thus, player 2 is, by default, the human's partner while players 1 and 3 are the opposing team.

2.2.2 Computer player

The computer player (CP) that you code is not required to be very smart, though having a good computer player would be a possible enhancement. We strongly suggest that you start by having it select the first valid card from its hand, and leave more complex strategies for later, once you have the basics working.

The implementation, while not difficult, can be somewhat complex. It is strongly suggested that you develop it incrementally, and start with something very simple, such playing/leading the first (valid) card in the hand, and declaring trump as the first suit in the player's hand.

2.3 The deck

The deck will be initialized with the cards consisting of the 9 through Ace of each suit. You are allowed to use any reasonable shuffling algorithm so long as the shuffling is repeatable with a pseudo-random number generator.

Any reasonable dealing strategy may be used; however, because the game play has to be perfectly repeatable, you will need to find a way to remember the order of the cards after the deck is shuffled. Whenever the round is over, before the cards are shuffled the next time, you will have to return the deck's cards to their previous order.

For example, if the cards in the deck have the order:

```
AS QC KC KS JS 9D TS 9S JH JD AH 9H AD QS TC AC KD 9C TD QH QD TH KH JC
```

after being shuffled once with a seed of 0, they will need to be back in this order before being shuffled.

2.4 The Display

2.4.2 Without the debug flag

The game must provide enough output for the flow of the game to be followed. At a minimum, you must output:

- at the start of the hand:
    - the score for the two,
    - the top card of the kitty,
    - who the dealer is, and
    - who starts the decision-making on trump
- each player's decision to either pass or order up the top card of the kitty (if the trump isn't set from the input file),
- whether or not the top card of the kitty is turned down, in which case which suit cannot be trump,
- each player's decision to either pass or declare a suit (if the trump isn't set from the input file),
- an appropriate message if all players pass and the hand is nullified,
- an appropriate message indicating which card is led, and which cards are played in response, including if a player is skipped since their partner is going alone,
- an appropriate message if a player reneges, and the new score,
- an appropriate summary message for who took the trick,
- an appropriate summary message for the hand, once all cards have been played, as to which team won the hand and whether or not they were euchred, and
- an appropriate message once a team reaches 10 or more points.

2.4.2 With the debug flag

Providing the debug flag on the command-line causes additional information to be printed, so that you and the markers can verify what is occurring in your game logic. All of the previously discussed messages are still printed.

Your output now includes:

- the seed value being used,
- the order of the cards in the deck at the start of the round,
- the contents of the kitty,
- what card a player discarded if the player picked up the top card of the kitty,
- the contents of the player's hand at the start of a player's turn (one of choosing trump, discarding a card after being ordered up, leading or playing after the led card), the contents of their hand will be printed, along with the number of tricks they have taken so far in the round.

2.5 The Command Interpreter

There are only a few commands that need to be supported, since the only input comes from the single human player, assuming they haven't rage quit yet. Note that whenever input is required from the human player, the game prints the prompt "`> `" so that the human knows to type a response. All output is sent to standard output, i.e. `std::cout`.

Command	**Explanation**
`q`	Terminates the program immediately.
`r`	Human player "rage quits", and is replaced by a computer player. The game prints "`Player 0 rage quit.\n`". The game play continues until the game is over i.e. one team wins with 10 or more points, and then the program terminates.
`p`	Used in the trump declaration process to indicate that the player is passing. Results in an unrecognized command error message if given in the wrong context, such as when the player needs to discard, lead, or play a card.
`o`	Used to order up the dealer in the trump declaration process while considering the top-card of the kitty as trump. Used even if the human player is the dealer and wants to pick up the top-card of the kitty.
`oa`	Same as before, but now the player is also going alone.
`suit` `n`	One of {`H`, `D`, `C`, `S`}. Used in the trump declaration process after the top-card of the kitty has been turned down. Specified suit is declared trump and the player is not going alone, unless it is an error as discussed in the following section. Results in an unrecognized command error message if given in the wrong context, such as when the player needs to discard a card. *Note that the parameter 'a' or 'n' is not read if the suit choice is invalid.*
`suit` `a`	Same as above, but the player is going alone.
`card`	Rank and suit, as in `9S`, for a card to be discarded/played/led, depending upon the context and prompt. Results in an unrecognized command error message if given in the wrong context, such as when the player needs to declare trump.

2.5.1 Dealing with errors

Being human, mistakes could happen, deliberately or otherwise. It is thus important that your implementation check the validity of all moves. Whenever input is required from the human, you need to produce an appropriate error message if the described error situation occurs.

Type of error
Enters an invalid command. Human is re-prompted for a new command until a valid one is entered.
Declares as trump the same suit as that of the turned-down top-card of the kitty. Human is re-prompted for a new, valid suit until one is entered.
Tries to discard, lead or play a card that isn't in the player's hand. Human is re-prompted for a new, valid card until one is entered.
Enters the order-up command (normal or alone version) after the top-card of the kitty was turned down.
Reneges when playing a card. Player is prompted for a new, valid card. Score for the other team increases by 2 points for every occurrence.

3. Reading in a saved game

In order to keep things simple, you may assume that the format of the input file is always correct. It is up to you to determine a reasonable input file format. At a minimum, it should be possible to start a game at any reasonable point e.g. before deciding what to do about the top card of the kitty, after turning down the top card of the kitty but before choosing a trump suit, after choosing a trump suit but before a round starts, at the end of the round. If trump hasn't been declared yet, assume that the processing of the top-card of the kitty needs to be done. Player 0 is always initially a human player.

Grading

Your project will be graded as follows:

Correctness and Completeness 60%	Does it work? Does it implement all of the requirements?
Documentation 20%	Plan of attack; final design document.
Design 20%	UML; good use of separate compilation, good object-oriented practice; is it well-structured, or is it one giant function?

Even if your program doesn't work at all, you can still earn a lot of marks through good documentation and design, (in the latter case, there needs to be enough code present to make a reasonable assessment).

Question: Many games follow a common set of steps when being played. Explain how you could apply the *Template Method* design pattern to your euchre implementation i.e. what class or classes would be affected.

Question: The *Abstract Factory* design pattern, discussed in Chapter 4 of the "Head First Design Patterns" book, ensures that only related items within a family are created. For example, if you were creating a window displaying system, the look of the buttons on a Mac versus a Windows window are different, and if you were creating a window for a Mac, you would only want a Mac's "look and feel". Similarly, if you were marketing your software globally, you could use the Abstract Factory to ensure that all text and messages were in the appropriate language. If you wanted to provided different euchre variants, discuss how you would apply the Abstract Factory design pattern.

If Things Go Wrong

If you run into trouble and find yourself unable to complete the entire assignment, please do your best to submit something that works, even if it doesn't solve the entire assignment. For example:

- Is the deck correctly initialized and shuffled?
- Are the cards dealt out correctly and the kitty properly formed?
- Does the correct player start?
- Does the trump suit get passed or accepted correctly based upon the top card of the kitty and the player hands?
- If the proposed trump suit is declined, is the correct trump suit declared based upon the player hands?

- If the proposed trump suit is declined, do all computer players pass on declaring trump correctly based upon the player hands?
- Does a computer player lead the correct card based upon the trump suit?
- Do computer players play appropriately, based upon what is led and what is trump?
- Are tricks correctly taken and scored?

You will get a higher mark for fully implementing some of the requirements than for a program that attempts all of the requirements, but doesn't run.

Make sure that your program is at least able to load a pre-saved game and display the game.

A well-documented, well-designed program that has all of the deficiencies listed above, but still runs, can still potentially earn a passing grade.

Plan for the Worst

Even the best programmers have bad days, and the simplest pointer error can take hours to track down. So be sure to have a plan of attack in place that maximizes your chances of always at least having a working program to submit. Prioritize your goals to maximize what you can demonstrate at any given time.

One of the first things you should probably do is write a routine to read in and output a card and a container of cards, such as a hand or a deck. Then work on manipulating your container of cards. You should also develop the ability to read in pre-saved games early on. This will ensure that we are able to test features faster by loading our pre-saved games. You should also do the command interpreter early, so that you can interact with your program. You can then add commands one-by-one, starting with '`q`' and '`p`', and separately work on supporting the full command syntax. Work on your game steps and strategies one at a time. Don't try to build everything all at once! Start with a rough structure and fill in capabilities as you go.

Take the time to work on a test suite at the same time as you are writing your project. Although we are not asking you to submit a test suite, having one on hand will speed up the process of verifying your implementation.

You will be asked to submit a plan, with projected completion dates and divided responsibilities, as part of your documentation for Due Date 1.

If Things Go Well

If you complete the entire project, you can implement extra features for a few extra credits. These should be outlined in your design document, and markers will judge the value of your extra features.

Submission Instructions

See `Guidelines.pdf` for instructions about what should be included in your plan of attack and final design document.

Due Dates

Due Date 1 (November 19, 2018): Due on due date 1 is your plan of attack, `deadlines.txt`, and initial UML diagram for your implementation of euchre, `a5-uml-initial.pdf`.

Due Date 2 (December 3, 2018): Due on due date 2 is your actual implementation of euchre. All `*.h`, `*.cc, *.cpp` and any text files needed for your project to compile and run should be included in `euchre.zip`. All test files required to demonstrate the functionality of your project must also be submitted. The ZIP file must contain a suitable `[Mm]akefile` such that typing `make` will compile your code and produce an executable named `euchre`. In addition, upload `a5-uml-final.pdf` and `a5-design.txt` to the appropriate place on marmoset.