bg016931
Jounaid Ruhomaun
Programming Report

Initial Design

We were initially instructed to design, implement and test either a cataloguing system or a card based game- of which I chose the latter. In my opinion even though a cataloguing system would be much easier to implement, I didn't find this interesting and thought that it would be difficult to add complexity to this type of project. I have a personal interest in TCG's (Trading Card Games), specifically Magic: The Gathering, a fantasy style card game where players build a deck of cards and duel with usually one winner depending on the game type (more information on this can be found here: https://magic.wizards.com/en/gameplay/how-to-play) . Magic: The Gathering and similar TCG's are generally played with paper (physical) versions of the cards however there are a plethora of online versions of the game that allow players to duel though the internet or over a LAN. I had the idea of creating a similar program to the popular open source platform known as Cockatrice (https://cockatrice.github.io), a free TCG simulator that allowed players to set up matches and such. However, after delving deeper and researching how this program was made, I realised that it was much too complex of a program to create within the time constraints. Therefore as a backup option I decided to create a version of Poker. I decided on this because I am somewhat familiar with the game and knew implementation would be possible, but still knew I would be able to add sufficient complexity to the program.

The first marked piece of work for this project was the 'Project Design Sheet', in which I reviewed several already established online versions of poker. I decided that my game would be a console application game, playable against the dealer where the player would get two cards and would have to decide how much to bet (with a starting amount of £1000) depending on the community cards that would be revealed consecutively after each round. If the player were to beat the dealer, they would receive double the amount they bet and vice versa if the dealer would happen to win the player would lose the total amount of money they bet (with a draw resulting in no change).

Program Development

The first and most important step in developing a poker style game would be to have some sort of card class. Using an online tutorial (https://www.youtube.com/watch?v=NAAEMILMt-Q) as a guideline I created two classes, 'cards.cpp' and 'deck.cpp'. To implement a system that would generate cards, I split the card class into having both a 'suit' and 'face' variable. Due to there being certain cards which have faces that are not integer values (for example 'Ace of Spades') I decided both the face and suit would use the string data type. This did mean I wouldn't be able to use any mathematical operators on the face of the card (for example I would not be able to compare face values), however due to the nature of my program this wouldn't be an issue. I also added functions that would allow for the card to be displayed to the console and for either the face or suit of a card to be returned separately (as this is needed in order to check and compare hands).

In the deck class, I added two arrays, one for the face of the card and one for the suit (as seen in the image below). Using a combination of the values from these two arrays would allow me populate a deck efficiently without having to physically have each possible card typed out in a single array of size 52.

```
deck::deck()
{
    string face[] = { "Ace", "Two", "Three", "Four","Five","Six" ,"Seven" ,"Eight" ,"Nine" ,"Ten" ,"Jack" ,"Queen" ,"King" };
    string suit[] = {"Hearts", "Diamonds","Clubs","Spades"};
    decks = new card[cardsPerDeck];
    currentCard = 0;
    for (int i = 0; i < cardsPerDeck; i++)
    {
        decks[i] = card(face[i % 13], suit[i / 13]);
    }
}
```

The for loop shown above will populate a declared deck with all 52 cards (the limit set by the constant ‘cardsPerDeck’).

The next function in the deck class is called shuffleDeck() is used to, you guessed it, shuffle the deck! Using the ‘rand’ and ‘time’ functions included in <cstdlib> and <ctime>, the ShuffleDeck() function goes through each card and switches it ‘randomly’ creating a deck of cards that is shuffled.

The final function is used to deal cards. The function will return the first indexed card from the deck (i.e the top card of the deck) unless the ‘current card’ is greater than the constant ‘cardsPerDeck’ in which the deck will be shuffled otherwise the function will try and deal a card past the last card resulting in error.

Since in the game, there would be three rounds of betting I decided to create a reusable function for this process (see image below).

```
void playerBet()
{
    do {
        bet = 0;
        errorCheck = false;
        cout << "Enter your bet (max bet 9999, 0 to call) , you currently have " << cash << " pounds to play with" << endl;
        cin >> bet;


        if (bet > cash)
        {
            cout << "Sorry, you do not have enough money to bet that much" << endl;
            errorCheck = false;
        }
        else if (bet > 9999)
        {
            cout << "Maximum bet exceede, please enter a new bet" << endl;
            errorCheck = false;
        }
        else
        {
            cash = cash - bet;
            pot = pot + bet;
            bet = 0;
            errorCheck = true;
        }

    } while (errorCheck == false);


}

int main()
```

This function basically tells the user how much they have to play with, and has an if statement for error checking in case the players bet exceeds the amount they have in the pot. There is also a maximum bet I added later on as when betting large numbers the program would loop infinitely not allowing the player to continue.

The next stage was creating the menu. I used a while loop with the exit case being ‘x’ so that the menu would loop infinitely. The two options were either a description of the game with instructions on how to play it, and the actual game itself. If any input other than either 1, 2 or x were to be typed in the menu would simply loop again (the data type used is string to allow for error checking).

At the start of the game, two cards are dealt to the player, two cards are dealt to the dealer and 5 cards are dealt as community cards (see image below).

```
{
    pot = 0;
    deck d;

    card playerCardOne = d.dealCard();
    d.shuffleDeck();
    card playerCardTwo = d.dealCard();
    d.shuffleDeck();

    card playerCards[2] = { playerCardOne, playerCardTwo };

    card dealerCardOne = d.dealCard();
    d.shuffleDeck();
    card dealerCardTwo = d.dealCard();
    d.shuffleDeck();

    card dealerCards[2] = { dealerCardOne, dealerCardTwo };

    card poolCardOne = d.dealCard();
    d.shuffleDeck();
    card poolCardTwo = d.dealCard();
    d.shuffleDeck();
    card poolCardThree = d.dealCard();
    d.shuffleDeck();
    card poolCardFour = d.dealCard();
    d.shuffleDeck();
    card poolCardFive = d.dealCard();
    d.shuffleDeck();

    card poolCards[5] = { poolCardOne, poolCardTwo, poolCardThree, poo
```

The players hand is then revealed to them in the console and they have the first opportunity to bet (using the betting function described earlier). After the player has chosen an amount to bet, the first three community cards are revealed, again allowing the player to bet. This process of revealing community cards and then betting based on said cards continues until all five community cards are revealed. The dealers hand is then shown and a message indicated who won the game will appear in the console (see image below)

```
if (playerPoints > dealerPoints)
{
    cout << "Congratulations! you win " << pot * 2 << endl;
    cash = cash + (pot * 2);
}
else if (dealerPoints > playerPoints)
{
    cout << "Sorry, you loose" << endl;
}
else
{
    cout << "Its a draw!" << endl;
    cash = cash + pot;
}
```

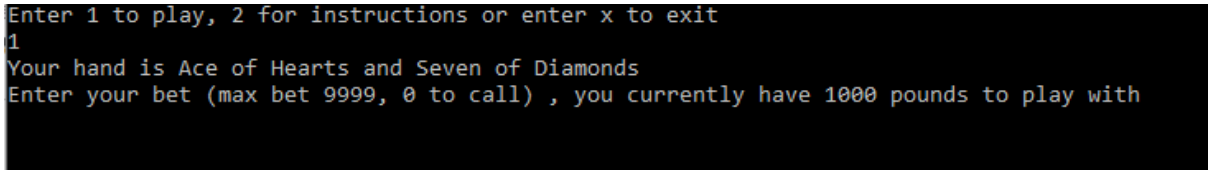
If the player wins the amount bet is doubled and added to the player's total cash. This money can then be used in future games as its value is retained until the program is exited. If the player happens to lose all their money, they will not be able to bet but will still be able to play (by betting 0).

In order to check the players hand against the dealers, I set up a points system depending on what each party has. Ideally this would have been in a separate function outside the main function (or even in a separate class) however in my program I just wrote the code in-between the end of the game and were the winner is declared. There were three winning sets that I implemented, a 'pair' which was valued at 1 point, a 'two pair' which was valued at 2 points, and a 'flush' which was valued at three points. The pair and two pair checking function checks the player's two hand cards against the each card in the pool. If there is a match the temp variable (used to keep track of pairs) increments. This way the program can check how many pairs there are. The flush checking function essentially uses the same code as the previous function, but checks the suits of the players hand against the community cards. If 5 or more match then that means the player has a flush.

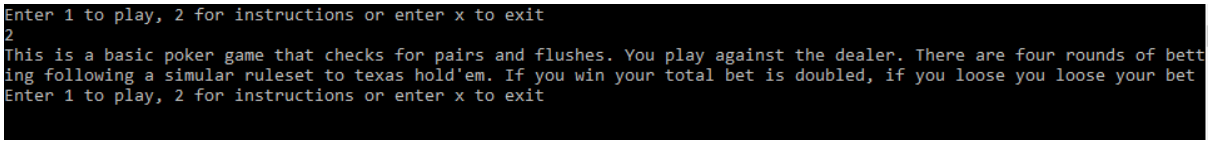
Testing

In order to test the program, I have decided to try every possible input to see how the program handles it.

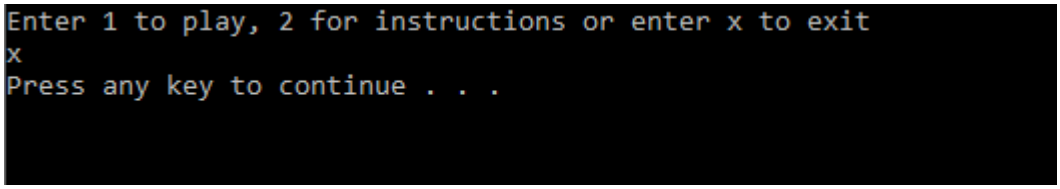
Menu testing:



Here I input 1, and as expected the game starts.



Here I input 2, and as expected the instruction text is output. Also as expected the menu text is repeated allowing for the user to enter another option.



Here I input x, and as expected the program ends (pressing any key after this closes the console window).

```
Enter 1 to play, 2 for instructions or enter x to exit
7
Enter 1 to play, 2 for instructions or enter x to exit
8
Enter 1 to play, 2 for instructions or enter x to exit
9
Enter 1 to play, 2 for instructions or enter x to exit
a
Enter 1 to play, 2 for instructions or enter x to exit
b
Enter 1 to play, 2 for instructions or enter x to exit
c
Enter 1 to play, 2 for instructions or enter x to exit
!
Enter 1 to play, 2 for instructions or enter x to exit
"
Enter 1 to play, 2 for instructions or enter x to exit
£
Enter 1 to play, 2 for instructions or enter x to exit
qqqqqqqqqq
Enter 1 to play, 2 for instructions or enter x to exit
qweqwrqwtrqw
Enter 1 to play, 2 for instructions or enter x to exit
asgf3rf13fasc3qf33fvaev 123rf13vqwev q r123r f
Enter 1 to play, 2 for instructions or enter x to exit
```

In this next test I input numbers, letters, symbols and strings that are not correct input values and as expected the menu re-loops allowing for the user to enter the option again.

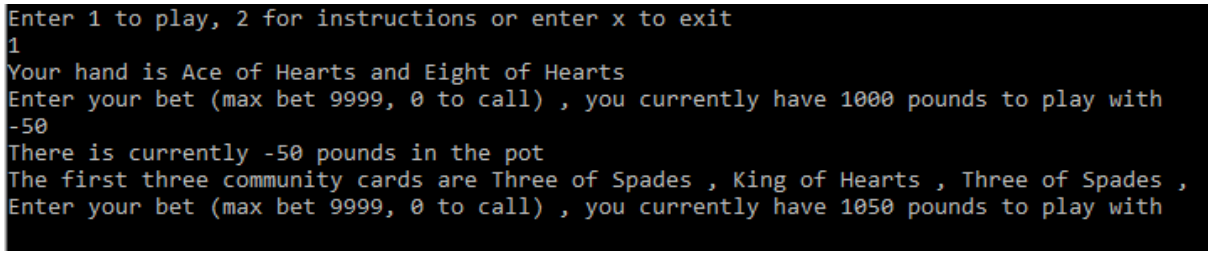
Betting function test:

```
Your hand is Ace of Hearts and Seven of Hearts
Enter your bet (max bet 9999, 0 to call) , you currently have 1000 pounds to play with
50
There is currently 50 pounds in the pot
The first three community cards are Eight of Hearts , Queen of Hearts , Two of Hearts ,
Enter your bet (max bet 9999, 0 to call) , you currently have 950 pounds to play with
```

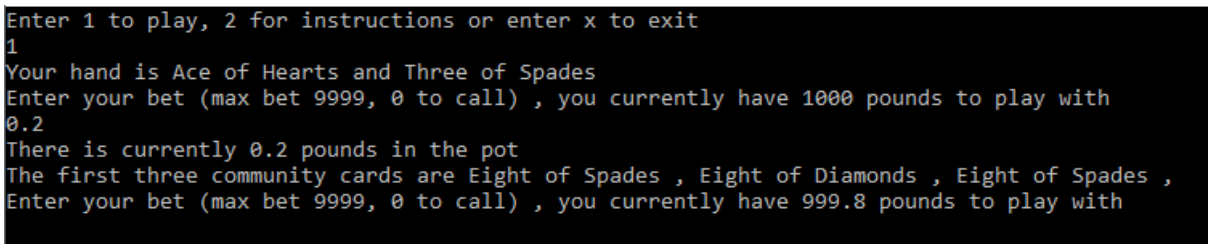
Here I bet 50. As you can see the program correctly handles this value and deducts it from the player's cash.

```
Your hand is Ace of Hearts and Nine of Hearts
Enter your bet (max bet 9999, 0 to call) , you currently have 1000 pounds to play with
1001
Sorry, you do not have enough money to bet that much
Enter your bet (max bet 9999, 0 to call) , you currently have 1000 pounds to play with
```

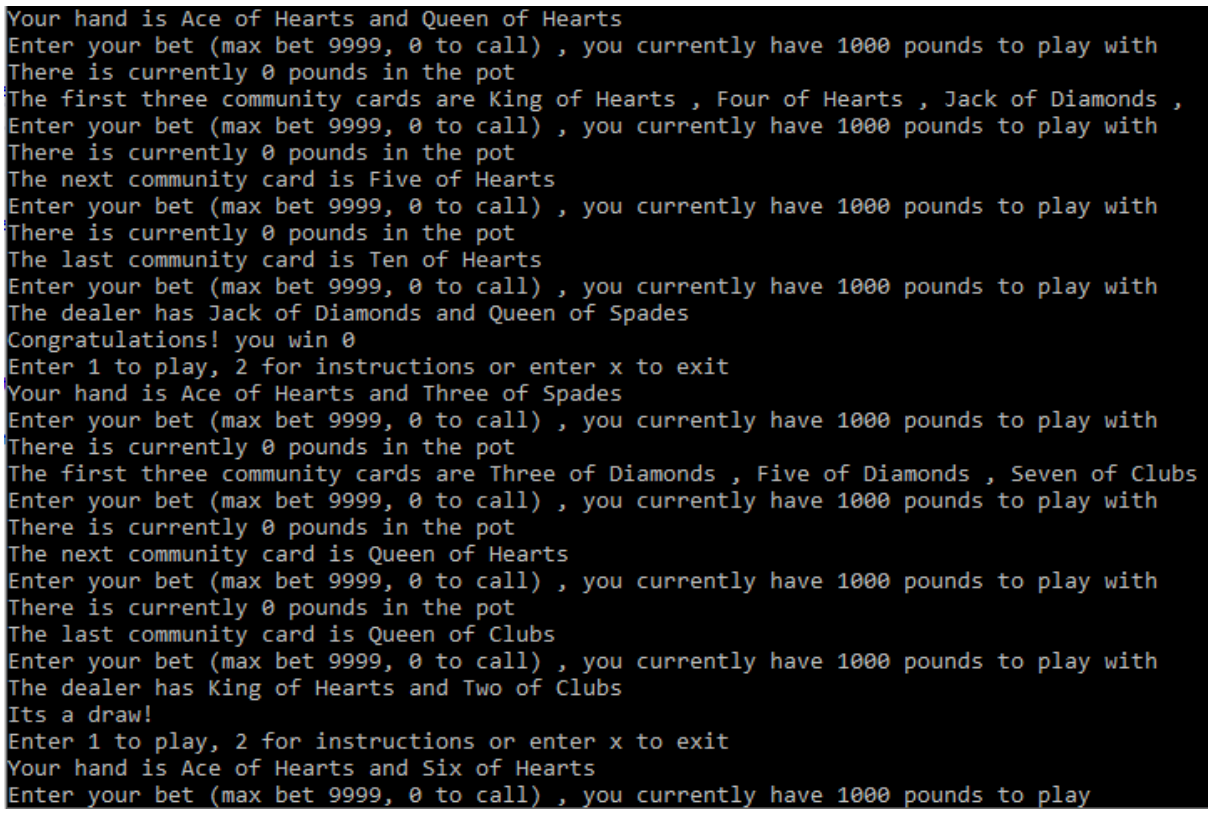
Here I bet 1001. Since this exceeds the players amount of money, the message 'Sorry, you do not have enough money to bet that much' appears as expected and the program prompts the user to re-enter their bet.



Here I bet -50. As you can see above the bet is taken and added onto the player's total cash. This is a bug and is not how the program is intended to run. To fix this I would simply need to add another 'elseif' to the error checking if statement in the betting function which will check for negative numbers, display an error (like for when the bet exceeds the players amount of cash) and prompt the user to bet again.



Here I bet 0.2. As you can see the program correctly handles this value and deducts it from the player's cash.



The above image is a screenshot of what happens when any non-numeric character is input as a bet. The program infinitely loops playing infinite games with bet 0 each round. In order to fix this bug I would need to include error checking against non-numeric values in the betting function. A way I could implement this would be to use the 'if(cin.fail())' statement to check if the console input

created a fail bit. Alternatively I could use the ‘isdigit’ function however this would mean changing the betting system to an integer datatype which means a decimal value (like the 0.2) before would not work and would result in an error.

```
Enter 1 to play, 2 for instructions or enter x to exit
1
Your hand is Ace of Hearts and Six of Clubs
Enter your bet (max bet 9999, 0 to call) , you currently have 1000 pounds to play with
50
There is currently 50 pounds in the pot
The first three community cards are Ace of Diamonds , Two of Diamonds , Queen of Diamonds ,
Enter your bet (max bet 9999, 0 to call) , you currently have 950 pounds to play with
50
There is currently 100 pounds in the pot
The next community card is Four of Diamonds
Enter your bet (max bet 9999, 0 to call) , you currently have 900 pounds to play with
50
There is currently 150 pounds in the pot
The last community card is Six of Clubs
Enter your bet (max bet 9999, 0 to call) , you currently have 850 pounds to play with
50
The dealer has Five of Diamonds and Queen of Clubs
Congratulations! you win 400
Enter 1 to play, 2 for instructions or enter x to exit
```

Here I ran through one entire game and the results are as expected. Since the user has Six of Clubs, and there is also Six of Clubs in the community card pool, the user has a pair. Since the dealer does not have a pair or higher (two pair or flush), the player wins. Since I bet 50 on each of the four rounds, I bet a total of 200 which is doubled then added to my cash.

Conclusion

In conclusion I believe my project is successful as it fulfils the parameters that are in my design document. Using my program the user is able to play a full poker like game against the computer with minimal bugs.

If I had more time to work on this program I would have liked to have transferred my current code into a graphical version of the game with the cards being displayed on a table. I would also have liked to have ‘tidied’ up my code, adding comments and splitting certain parts into functions (as explained earlier) however due the program hand in deadline being earlier than this write up deadline there is little point in editing the code now.