**CS PROJECT 7 REPORT**

1. Some of the notable obstacles that I overcame while programming this assignment are as follows: the main problem that I had came from the determination of whether the aces should be worth 1 or 11. I was trying to think through the logic of it, and thought I had a solution, but then when I ran the code, the ace wouldn’t be counted as 11 in the situations that it should have been (such as when the player or dealer got a face card and an ace, which should’ve been an automatic blackjack). I looked through the code, and found this following line:

if ((number + 10) < 21 && acecounter == 1)

This line would cause a problem because it says if the total, which I have called number for reasons stated in the comments of the code, is only less than 21, then the aces should be counted as 11. However, this code would need to be changed, because we want to be able to reach 21 because that is the objective of blackjack. As such, my code started working after I switched this statement and similar statements to:

if ((number + 10) <= 21 && acecounter == 1)

Another problem that I ran into was the syntax for calling some of the functions that were stated in the other classes, especially the functions that did not have any arguments. For example, I had written the following line:

while (mDealer.handcount < mPlayer.handcount && mDealer.handcount < 17)

for the game.cpp file. In this case, however, I got a compiler error because this was not proper syntax for calling the functions with the dot methods. This is because even though there are no arguments stated, I still need to include parentheses in order to call the function correctly, so as such, I needed to rewrite the function as:

while (mDealer.handcount() < mPlayer.handcount() && mDealer.handcount() < 17)

This was a simple mistake that probably came from the fact that I was coding very quickly and was not careful when looking at my syntax. Although this was a minor error, it was still significant enough that it caused a compiler error.

The last error that caused a main error also stemmed from the determining whether the aces should be worth 1 or 11. I had written the following code:

if (myCards[k] == ACE)

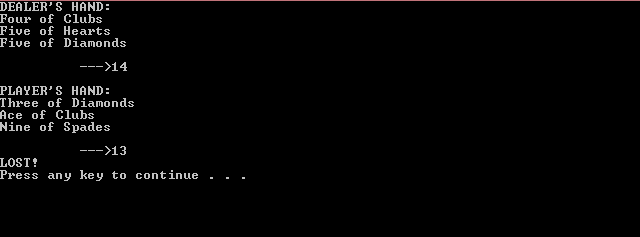
And this had compiled fine and the code had ran, but I wasn’t getting the results that I wanted or needed. After looking around for the source of the error, I realized that you can’t actually write an if statement like that, because you can’t just compare the array value like that. Instead, in order to write this if statement, we need to write:

if (myCards[k].getFace == ACE)

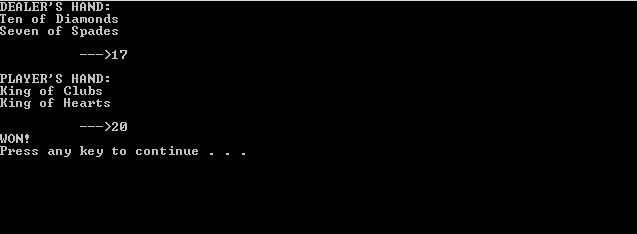
This is the idea that we have to use a caller in order to get the things that we want out of the function and out of the class. As such, this was an error because I couldn’t compare those two things like that with an if statement, even though Visual Studio compiled fine and nothing else would have said that that was an error.

1. In order to test this code, I actually played quite a few rounds of blackjack in order to try to find all of the different combinations of code that could occur. I also found out how bad I was at blackjack, even when I was trying to exploit the rules.

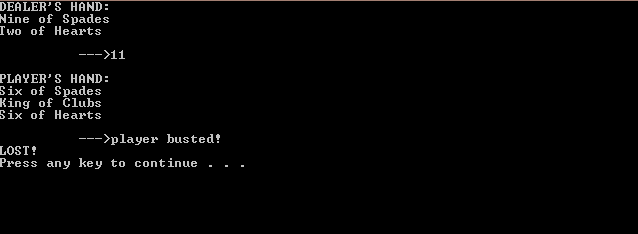
The following are all test cases that I did while I played:



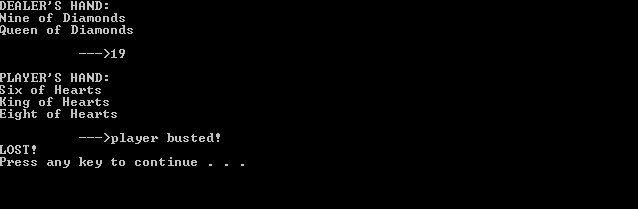
This was a basic functionality test to see if the code would work properly, along with the counters. We can see that I lost here, and that the ace here is counted as 1 because if it were counted as 11, the player would automatically bust at 22. This code was to test if the dealer could win, and that was the case.



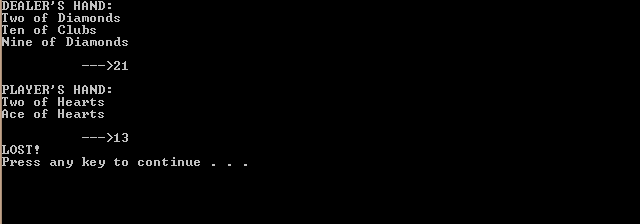
This was to see if the player could win, and if the counters would work properly. Here, we can see that the dealer stopped at 17, like they are programmed to do. The two face cards counted up properly to 20. This tested basic functionality and worked properly.



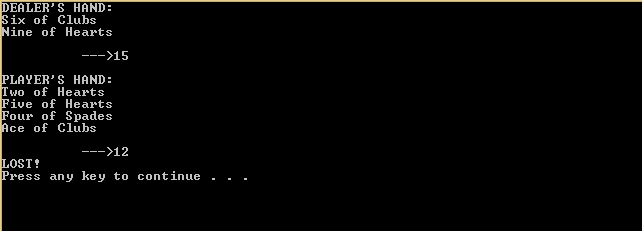
This was to test what would happen if the player busted. We can see that the dealer automatically won, and that the dealer did not even play because they didn’t need to – the player had already lost.



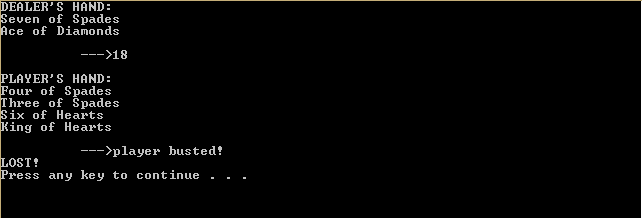
This was again to test what would happen if the player busted, and we see here that the same thing happened as above, as it was supposed to.



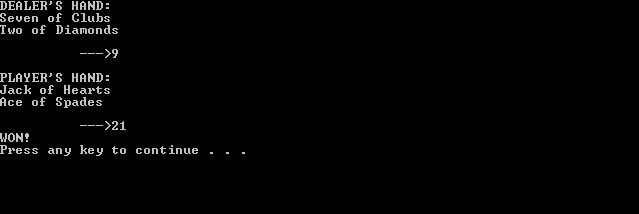
This was to test the functionality of the ace. Here, we have a two of hearts and an ace of hearts. Since the total does not make it so that the player busts, the ace should automatically take a value of 11. We can see here that this was indeed the case, as the total value was 13 rather than 2. We can also see the dealer code working fine, as the dealer was originally at 12 and then got a 9 to make a good value of 21 since the dealer’s hand count was under 17.



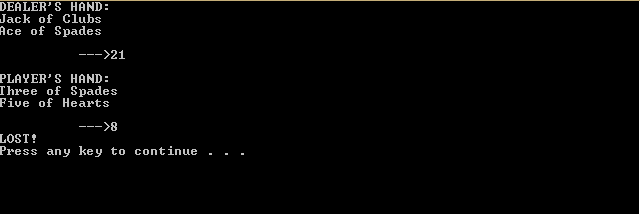
This was to test the ace functionality if the total was too much if the ace was counted as 11. We can see here that this tested fine, and that the value of the hand was 12 and not busted. We can also see the dealer code working well because the dealer didn’t have to play since the player had already busted.



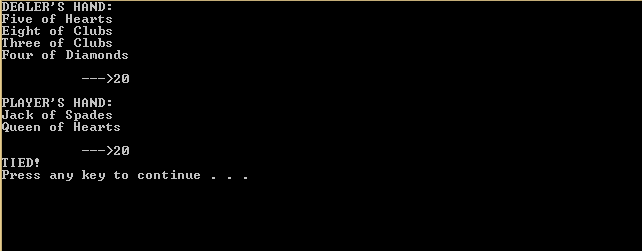
This was again a basic functionality test to see what would happen if the player busted. We see here that everything works well.



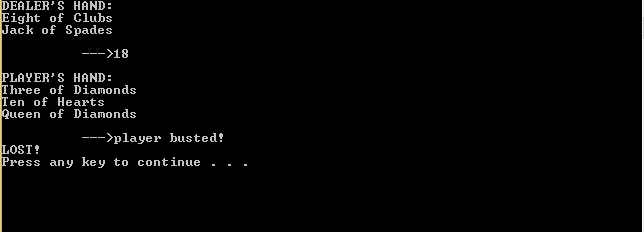
This was a lucky chance to see the automatic blackjack draw. I didn’t even have to play because I had already gotten a blackjack, and this proves that the code works fine, and that the value of the ace was correctly automatically assumed to be 11.



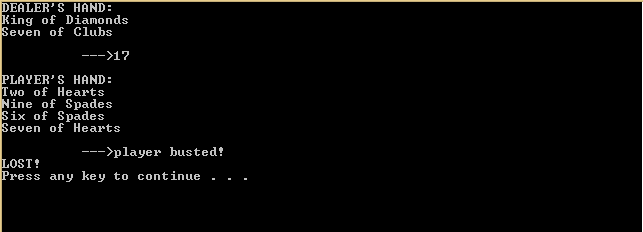
This tested if the game would work correctly if the dealer had automatically gotten blackjack, and this was the case. As such, we can see that the ace mechanic is working well for both the player and dealer.



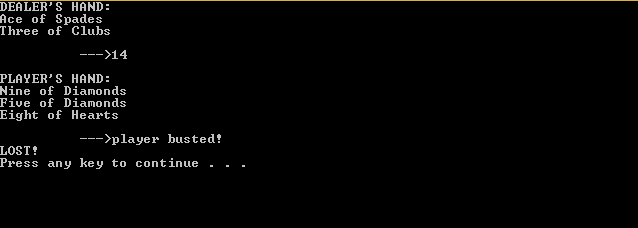
This was a lucky chance, where the player and the dealer actually tied. This was to see if the tie mechanic was coded properly, and it was. The hand counts both add up to 20, and as such a value for tie should be returned, which it was.



This again shows the player bust, and also shows how bad I am at blackjack. This was to test basic functionality, and we can see here that the code works fine. The code works fine for these values.



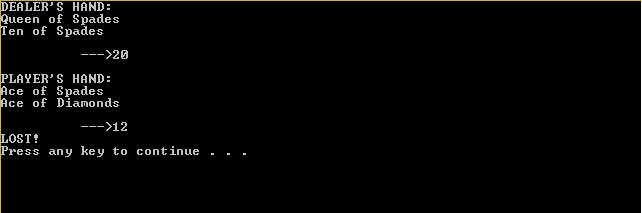
This was another player bust, and the code works fine with these different values.



This was another player bust, and it shows that the code works for these values and that the code can function basically. We also see that the code for the dealer works because the dealer didn’t play, even though 14 is less than 17, because the player already busted.



This was to see if the dealer could possibly bust, and we see here that this is indeed a possibility, at least with these card values.



This was to test the case if there were two aces. This is a very specific case because only one of the aces should be worth 11, and the other ace should be worth 1. We can see that this worked fine because the total value of the player’s hand is 12, instead of 2, as it should be.

In addition to all of these tests, I also ran the basic tests that were given in the assignment specification as a main function in order to test our code. This code is as follows:

#include <iostream>

#include <string>

#include <cassert>

#include "Game.h"

#include "Player.h"

#include "Card.h"

using namespace std;

int main()

{

using namespace std;

using namespace cs31;

    // test code

    Player p;

    Card c( Face::ACE, Suit::DIAMONDS );

    Card d( Face::TEN, Suit::CLUBS );

    Card e( Face::THREE, Suit::SPADES );

    Card f( Face::EIGHT, Suit::HEARTS );

    assert( p.handcount() == 0 );

    assert( p.hasBlackJack() == false );

    assert( p.cardCount() == 0 );

    p.acceptCard( c );

    assert( p.handcount() == 11 );

    assert( p.hasBlackJack() == false );

    assert( p.cardCount() == 1 );

    assert( p.getCard(0) == c);

    p.acceptCard( d );

    assert( p.handcount() == 21 );

    assert( p.hasBlackJack() == true );

    p = Player();

    Player dealer;

    Game g( p, dealer );

    g.playerStands();

    g.dealerStands();

    assert( g.dealerBusted()==false );

    assert( g.playerBusted()==false );

    assert( g.playerTied() == true );

    assert( g.playerWon() == false );

    assert( g.playerLost() == false );

    assert( g.playerHasBlackJack() == false );

    assert( g.dealerHasBlackJack() == false );

    p = Player( );

    p.acceptCard( c );

    p.acceptCard( d );

    dealer = Player( );

    dealer.acceptCard( e );

    dealer.acceptCard( f );

    g = Game( p, dealer );

    g.playerStands();

    g.dealerStands();

    assert( g.dealerBusted()==false );

    assert( g.playerBusted()==false );

    assert( g.playerTied() == false );

    assert( g.playerWon() == true );

    assert( g.playerLost() == false );

    assert( g.playerHasBlackJack() == true );

    assert( g.dealerHasBlackJack() == false );

     cout << "all tests passed!" << endl;

    return 0;

}

Basically, this code established some cards and went through and tested each function. We had an ace, ten, three and an eight of diamonds, clubs, spades, and hearts. It went through and tested the hand count of each iteration, and also whether or not the player had blackjack.

In all, this was just in order to test the completeness of the code. More thorough testing was done through the random numbers generated using the game code from the black console boxes shown above.

We have as such, pretty much thoroughly tested our code for these different situations and iterations, and can say that our code works properly.