The Perfect Poker Player?

New Mexico
Supercomputing Challenge
Final Report
April 4, 2012

Team 131
St. Pius X High School

Team Members

James Bunch

Luis Rivera

Thien-Nam Dinh

Ethan Sabay

Teacher(s)

Diana Perea

Project Mentor

Christopher Alme

Table of Contents

Executive Summary:

Poker is usually considered to be a game of luck. However, by examining the probability or likelihood a two card hand has of becoming a valuable five card hand, and by analyzing a player's bet-win ratio at any particular moment in the game, it is possible to turn Poker from a game of luck to a game of intelligence.

In order for this to be properly tested, we wrote our code in the Java programming language and used it to simulate what we wanted done. Our program implements an algorithm that takes in as parameters any given number of poker playing cards (up to a maximum of seven, with only five being used at one time) and calculates the probability of obtaining a good hand once all cards have been dealt. This number can then be compared to the current betting odds to determine the risk of raising the current bet, calling the bet, or folding. This algorithm allows the computer player to determine what the best possible option is at any given point in the game.

Our program also includes a score generating algorithm that ranks hands. The scores that are generated are all relative to each other and have been compared thoroughly against each other multiple times. After comparison, we were able to determine that all scores were being generated properly and that the numerical values that were given to each individual hand as compared to each other corresponded to the proper actual value of ranked hands ranging from a High Card to a Royal Flush.

Our program is separated into several classes that together simulate a poker game by defining what a player, table, deck, and individual cards are. The shuffle algorithm used by the Collections class was also modified to better resemble the Mersenne Twister randomizing algorithm, which has a period of $2^{219,937-1}$ as opposed to the period of 2^{48} guaranteed by the

algorithm used in the Collections class. Even though nothing can be truly random, the Mersenne-Twister random algorithm gave us a more 'random' set of possibilities.

Problem Statement:

To create an artificial intelligence computer poker player that would be able to know it's own chances of winning that round. For instance, if the chances of the player winning with the current hand it is dealt does not have a reasonably good chance of not beating the opposing player, the computer player's programming would make the choice to fold the hand so that it will not lose any more money than it already has. The primary goal of the computer player is to win the game as a whole, not just win the current presented hand. Take for instance, a computer player with a losing hand, and no way to rectify said hand, rather than try to make a good hand out of it, the computer will know to cut its losses, and try to rectify the game as a whole. In short, the problem is to make a player that can know when it is intelligent to quit, bluff, bet, etc, leading to an effective game overall. In order to implement this artificial intelligence it was also necessary to generate a functional Texas Hold'em Poker game engine which would simulate a real Texas Hold'em game.

Method:

From the beginning of the project, we were committed to coding the programs at a relatively basic level; that is, our classes are all originally written in Java using the program "JCreator".

In order to create a functional platform for the premises of our project, it was first necessary to create the fundamental components of a basic poker game. This requires that we

wrote original Java classes which would represent and track relevant data associated with real world poker parts such as single cards, a deck (with capabilities to be shuffled, etc), a table (that is, a platform to hold dealt cards).

Having constructed the necessary parts to be replicated and utilized, we then proceeded to create the framework algorithms which would actually enable us to run the poker game. This means that we created classes which dealt the cards, track bets, etc. Most importantly a player class allows for every possible action by a poker player and displays a text based interface by which a human player can participate in the game. Much of the code is dedicated to solving the myriad of meticulous exceptions and complications in the rules of the game itself, such as problems that arise when a player goes "all in" or when an Ace card has simultaneous values of one and fourteen. We are thus able to troubleshoot the game framework directly and also play rounds of Texas Hold'em between multiple human players.

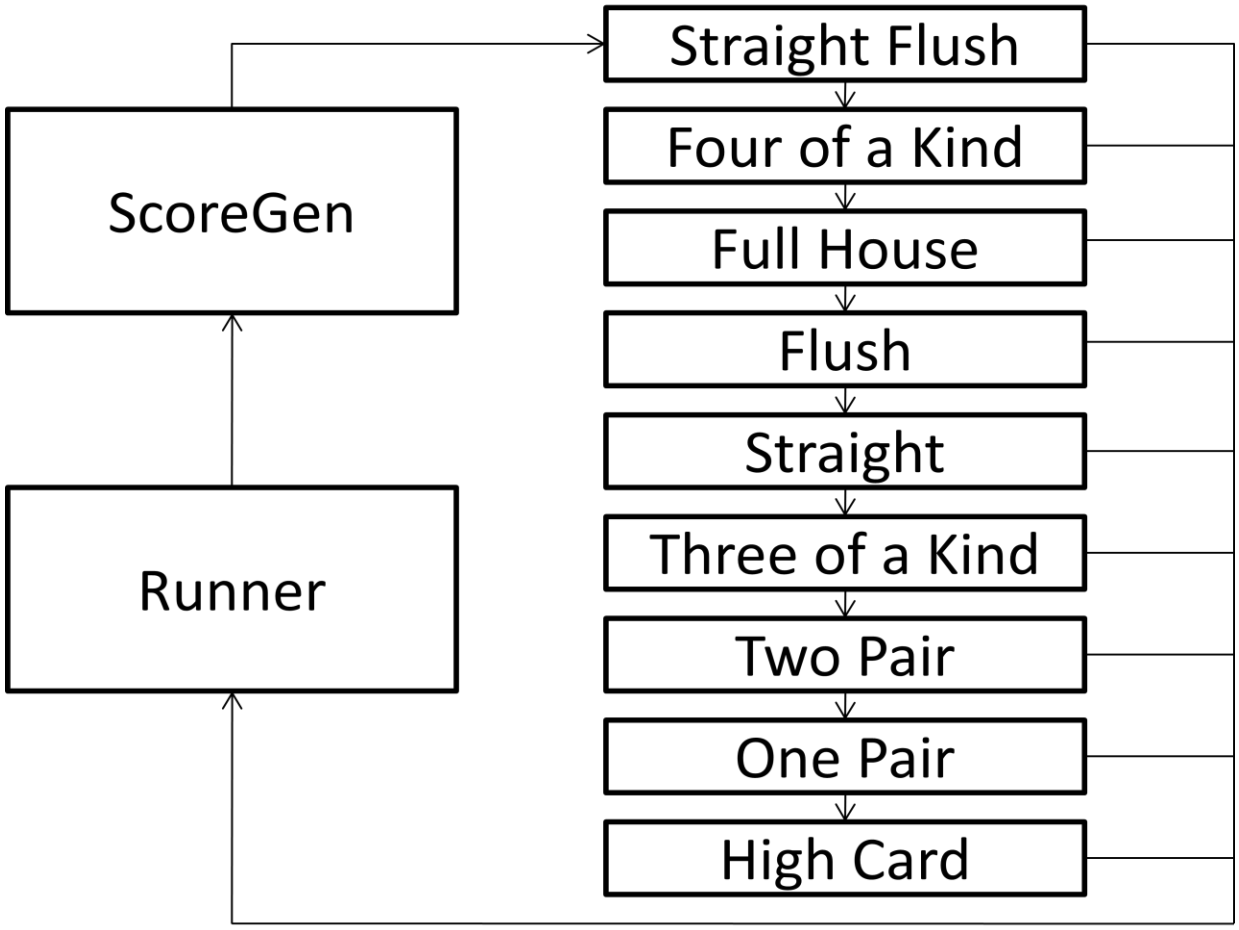
With the capability to play poker fully functional, the next challenge was to develop an artificial intelligence make decisions based on parameters derived from the thousands of unique scenarios in the game. In order to solve the aforementioned problem, our program integrates a complex algorithm which values the AI's hand with the cards that are currently available on the table. This algorithm helps the AI discern what to do every time it is called to perform an action. The basic idea behind the probability program is that it accounts for the current cards that it knows are in play and completes an algorithm in which it is paired with every possibility of the remaining cards in the deck to fill the final seven card set. Using a program called scoregen which generated a numerical score for every seven card set depending on its actually score hierarchy in Texas Hold'em, we can thus take the average of every virtual seven card set and thus generate the projected average score given the original known cards. This allows us to make every necessary decision in the game.

Results:

Based on our data, it is evident that pre-flop combinations of pairs yield the highest projected value with the score escalating along with the face value (number) of the pair. Additionally, there is an apparent trend favoring same suit cards as well as those with adjacent or near adjacent numbers. These parameters aside, it appears, for the most part, true, that combinations with higher face values yield higher projected scores. Another noteworthy variable was that of time. While the computer operated for an average of 34 minutes to generate a projection for seven unknown cards, it averaged 17 seconds for five unknown cards, and an indeterminately shorter time period as the number of unknown cards decrease. As for the structural and interface classes used in our program, our classes did exactly what we wanted them to do and accurately simulated a real poker game by creating the objects that could and would be present in any poker game.

Formula:

Score = [(val1 * 13^4) + (val2 * 13^3) + (val3 * 13^2) + (val4 * 13^1) + val5) * (frequency of hand)] + previous Max Score



Conclusion:

After running our probability program a multitude of times, it became conclusive that the artificially projected values of the probability of having a good hand did, indeed, correspond to the logical actual probability that would result from a human individually analyzing the cards that are available. Since the results were relative, the actual scores were based upon comparison between each and every possible hand that can be created with the cards that are currently available. The values given by the program were returned after every time a card was dealt (Pre-Flop, Flop, Turn, River). During the Pre-Flop, as the value of the card increased, the

average possibility value given by the program followed the general trend, with pairs having higher values that non-pairs. However, the pre-flop projection of the cards is not entirely indicative of a good hand. That is, a high numerical value does not constitute a high probability of winning the round after the flop has been dealt because other cards have to be progressively taken into account until the seven card set if filled. Another factor that was worthwhile noticing was the time that the program took in giving us our data as output. Since computers have a limited amount of Random Access Memory, the computer could only manage a certain amount of mathematical operations at a time and, therefore, took a relatively large amount of time to return the probability values when multiple mathematical operations were being performed. A general, logical, trend was found in our program where the time it took to return the probability during the Pre-Flop round was considerably larger than the time it took to return the probability after the Flop, and continued to decrease as the Turn and the River were dealt. This was of perfect logical sense as the program had to analyze fewer combinations as more cards were being dealt.

In terms of the rest of our goal, the ability of the computer to read and analyze betting patterns is demonstrated purely in a quick runner instance of the patters program. The computer successfully is able to read and keep track of the moves made by various players between rounds. The program currently is able to run a text-based poker game between two players, but can be easily modified to include any number of players so long as there are enough cards. (after more than all the cards are dealt the logic falls apart and the program

would crash) The program is successful overall in that it can play an accurate Hold ‘em game between human players and keep track of their moves.

Acknowledgments:

We would like to offer our sincere gratitude to our Computer Science teacher Diana Perea and our mentor for the project Christopher Alme.

Appendix A

Code in Short

Player Class

- Player(String pname, int Identification, int Balance, String CompOrHuman)
- int getID()
- String getName()
- int updateAccount()
- void wager(int amount)
- void pay(int winnings)
- void emptyHand()
- boolean status()
- boolean updatePlayer()
- boolean endLoop()
- boolean isComputer()
- void fold()
- ArrayList getHand()
- void addCards(PCard newCard)
- void play()
- void bet()
- void call()
- Void raise()
- void raiseOption()
- void checkOption()
- void pfold()
- void check()
- Void showHand()
- void showTable()

- void showPot()
- void showWager()
- void endSequence()
- void Start()
- void getPot()

CardTable Class

- CardTable (int numPlayers, ArrayList players1)
- void FillTable()
- ArrayList getPlayers()
- void Pay()
- void setWinner(int winner)
- void addBets(int wager)
- void addWager(int wag)
- Void addCard(Pcard newCard)
- ArrayList getTable()
- int numCards()
- int getWager()
- int getPot()
- int setWager(int bet)

PCard Class

- public PCard(int newSuit, int newValue)
- public String getSuit()
- public String getValue()
- public String toString()
- public int getNumValue()

Deck Class

- Deck()
- void FillDeck()
- int getDeckSize()
- arrrayList getDeck()

CarDealer Class

- CardDealer(Deck deck1, int nplayers, CardTable nTable)
- Void finishGame()
- Void SecondDeal()
- Void HandDeal()
- Void TableDeal()
- Void Shuffle()
- Static void selectionSort(ArrayList,Pcard>deck)
- ArrayList getCards()
- Pcard DealRandom()

CardTable Class

- CardTable (int numPlayers, ArrayList players1)
- void FillTable()
- ArrayList getPlayers()
- void Pay()
- void setWinner(int winner)
- void addBets(int wager)
- void addWager(int wag)
- Void addCard(Pcard newCard)
- ArrayList getTable()
- int numCards()
- int getWager()
- int getPot()
- int setWager(int bet)

ScoreGen Class

- public double getVal (ArrayList<PCard> arList)
- public ArrayList<PCard> sort (ArrayList<PCard> list, int order)

Probability Class

- Probability()
- public void PossGen(ArrayList<PCard> list)
- public Double getProb()

APPENDIX B

Code in Full

Single Hand Class

```
/**
 * @(#)SingleHand.java
 *
 *
 * @author
 * @version 1.00 2012/1/17
 */
package MerTwist;
import java.util.*;

public class SingleHand
{

        ArrayList<PCard> hands = new ArrayList();

    public SingleHand(ArrayList<PCard> pHands)
    {
        hands = pHands;
    }

    public void AddCard(PCard card)
    {
        hands.add(card);
    }

    public void clearHand()
    {
        hands.clear();
    }

    public void addHand(ArrayList<PCard> list1)
    {
        hands.addAll(list1);
    }

    public ArrayList<PCard> getHand()
    {
        return hands;
    }

    public PCard getCard(int i)
    {
        return hands.get(i);
    }
```

```
    public int size()
    {
        return hands.size();
    }


}
```

Card Dealer Class

```
/**
 * @(#)CardDealer.java
 *
 *
 * @author
 * @version 1.00 2011/8/25
 */
package MerTwist;
import java.util.Collections;
import java.util.ArrayList;
import java.util.Random;

public class CardDealer
{
        Deck deck = new Deck();
        int players1;
        Random generator = new Random();
        private ArrayList<Player> players = new ArrayList<Player>();
        CardTable table = new CardTable(2, players);
   public CardDealer(Deck deck1, int nplayers, CardTable nTable)
        {
                        table = nTable;
                deck = deck1;
                players1 = nplayers;
                players = table.getPlayers();
        }
   public void finishGame()
   {
        ArrayList temp = new ArrayList();
                ArrayList p = new ArrayList();
                temp = table.getTable();

   }

         public void SecondDeal()
         {
                ArrayList<PCard> temp = new ArrayList<PCard>();
                ArrayList p = new ArrayList();
                temp = deck.getDeck();
                PCard tempCard = temp.get(0);
        table.addCard(tempCard);
        deck.getDeck().remove(0);
         }
```

```
public void HandDeal()
{
	for(int i = 0; i < players1; i++)
	{
		for(int yo = 0; yo < 2; yo++)
		{
		ArrayList<PCard> temp = new ArrayList<PCard>();
		ArrayList<Player> p = new ArrayList<Player>();
		temp = deck.getDeck();
		p = table.getPlayers();
		PCard tempCard = temp.get(0);
		p.get(i).addCards(tempCard);
		deck.getDeck().remove(0);
		}
	}
}

public void TableDeal()
{
	for(int yah = 0; yah < 3; yah++)
	{
	ArrayList<PCard> temp = new ArrayList<PCard>();
		ArrayList p = new ArrayList();
		temp = deck.getDeck();
		PCard tempCard = temp.get(0);
	table.addCard(tempCard);
	deck.getDeck().remove(0);
	}
}

/*	for(int i = 0; i < (players1 * 2) + 3; i++)
	{
		if(i < 3)
		{
			ArrayList<PCard> temp = new ArrayList<PCard>();
				ArrayList p = new ArrayList();
				temp = deck.getDeck();
				PCard tempCard = temp.get(0);
			table.addCard(tempCard);
			deck.getDeck().remove(0);
		}
		else if(i < 5)
		{
			ArrayList<PCard> temp = new ArrayList<PCard>();
			ArrayList<Player> p = new ArrayList<Player>();
			temp = deck.getDeck();
```

```
            p = table.getPlayers();
            PCard tempCard = temp.get(0);
            p.get(0).addCards(tempCard);
            deck.getDeck().remove(0);
    }
    else if(i < 7)
    {
            ArrayList<PCard> temp = new ArrayList<PCard>();
            ArrayList<Player> p = new ArrayList<Player>();
            temp = deck.getDeck();
            p = table.getPlayers();
            PCard tempCard = temp.get(0);
            p.get(1).addCards(tempCard);
            deck.getDeck().remove(0);
    }
    else if(i < 9)
    {
            ArrayList<PCard> temp = new ArrayList<PCard>();
            ArrayList<Player> p = new ArrayList<Player>();
            temp = deck.getDeck();
            p = table.getPlayers();
            PCard tempCard = temp.get(0);
            p.get(2).addCards(tempCard);
            deck.getDeck().remove(0);
    }
    else if(i < 11)
    {
            ArrayList<PCard> temp = new ArrayList<PCard>();
            ArrayList<Player> p = new ArrayList<Player>();
            temp = deck.getDeck();
            p = table.getPlayers();
            PCard tempCard = temp.get(0);
            p.get(3).addCards(tempCard);
            deck.getDeck().remove(0);
            }
    else if(i < 15)
    {
            ArrayList<PCard> temp = new ArrayList<PCard>();
            ArrayList<Player> p = new ArrayList<Player>();
            temp = deck.getDeck();
            p = table.getPlayers();
            PCard tempCard = temp.get(0);
            p.get(4).addCards(tempCard);
            deck.getDeck().remove(0);
    }
    else if(i < 17)
```

```
                {
                        ArrayList<PCard> temp = new ArrayList<PCard>();
                        ArrayList<Player> p = new ArrayList<Player>();
                        temp = deck.getDeck();
                        p = table.getPlayers();
                        PCard tempCard = temp.get(0);
                        p.get(5).addCards(tempCard);
                        deck.getDeck().remove(0);
                        }
        }*/

 public void Shuffle()
 {
        OurShuffle.shuffle(deck.getDeck());
 }

 public void ReFill()
 {
        deck.EmptyDeck();
        table.clearTable();
        deck.FillDeck();
 }

 public static void selectionSort (ArrayList<PCard> deck)
{
   int min;
   PCard temp, temp2, temp3, temp4;

   for (int i = 0; i < deck.size(); i++)
   {
      min = i;
      for (int scan = i + 1; scan < deck.size(); scan++)
         if (deck.get(scan).getNumValue() < deck.get(min).getNumValue())
            min = scan;

      // Swap the values
      temp = deck.get(min);
      temp3 = deck.get(i);
      temp = temp3;
      //deck.get(min) = deck.get(i);.
      //temp4 = deck.get(i);
      temp4 = temp;
   }

}
```

```
        public ArrayList getCards()
        {
                return table.getTable();
        }

    public PCard DealRandom()
    {
        ArrayList<PCard> temp = new ArrayList();
        int i = generator.nextInt(52);
        temp = deck.getDeck();
        PCard tempCard = temp.get(i);
        return tempCard;
    }

}
```

Card Table Class

```
/**
 * @(#)CardTable.java
 *
 *
 * @author
 * @version 1.00 2011/8/29
 */
package MerTwist;
import java.util.ArrayList;
import java.util.Scanner;


public class CardTable
{

        final int MAXPLAYERS = 6;
   int currentPot = 0;
        int allInPot = 0;
        int amount = 0;
        int winnerID = 0;
   int callAmount = 0;
        int numP;
        ScoreGen gen = new ScoreGen();
        Scanner scan = new Scanner(System.in);
        private ArrayList<PCard> Table = new ArrayList<PCard>();
        private ArrayList<Player> players = new ArrayList<Player>();
        private ArrayList<Player> winningList = new ArrayList();

   public CardTable(int numPlayers, ArrayList players1)
   {
        numP = numPlayers;
        players = players1;
   }

   public void FillTable()
   {
        String name;
        String type;
        int balance;

        for(int ident = 1; ident < numP + 1; ident++)
        {
        System.out.println("What is player " + ident + "'s name?");
        name = scan.nextLine();
```

```
        System.out.println("What is player " + ident + "'s balance?");
        balance = scan.nextInt();
        System.out.println("Is player " + ident + " a human(H) or a computer(C)?");
        type = scan.nextLine();
        Player player = new Player(name, ident, balance, type);
                players.add(player);
        }
}

public ArrayList<Player> getPlayers()
{
        return players;
}

public void Showdown()
{
        ArrayList<Player> inPlayers = new ArrayList<Player>();

        for(int asdi = 0; asdi < players.size(); asdi++)
        {
                if(players.get(asdi).getFoldStatus() == false)
                {
                        inPlayers.add(players.get(asdi));
                }
        }

        winningList.clear();
        ArrayList<SingleHand> totalHands = new ArrayList();
        SingleHand currentHand;
        ArrayList<PCard> currentH = new ArrayList();
        ArrayList<String> allHands = new ArrayList<String>();
        /*ArrayList<PCard> list1 = new ArrayList();
        ArrayList<PCard> list2 = new ArrayList();
        list1 = (players.get(0).getHand());
        list1.addAll(getTable());
        list2 = (players.get(1).getHand());
        list2.addAll(getTable());
        System.out.println("Trial list1: " + list1);
        System.out.println("Trial list2: " + list2);
        totalHands.add(list1);
        totalHands.add(list2);*/

         ArrayList<Double> totalScores = new ArrayList();
          for(int x = 0; x < inPlayers.size(); x++) //******* Fills ArrayList with Hands *******
                                        {
                                currentH.addAll(inPlayers.get(x).getHand());
```

```
                currentH.addAll(getTable());
                currentHand = new SingleHand(currentH);
                //System.out.println(currentHand.getHand());
                        //totalHands.add(currentHand);

                //          }

                //for(int t = 0; t < totalHands.size(); t++) // Fills an ArrayList with the
scores.
                        //{
                        //      ArrayList<PCard> otherHand = new ArrayList();
                                //for(int h = 0; h < totalHands.get(x).size(); h++)
                        //      {
                        //              otherHand.add(currentHand.getHand());
                        //      }
                                //System.out.println(otherHand);
                                //System.out.println("Individual Hands: " +
totalHands.get(t));
                                System.out.println(currentHand.getHand());
                                totalScores.add(gen.getVal(currentHand.getHand()));
                                allHands.add(gen.getCombo());
                                //System.out.println(totalScores);
                                currentH.clear();
                                        currentHand.clearHand();
                        }

                double win = -1.0;
                int winIndex = -1;



        for(int u = 0; u < totalScores.size(); u++)
                        {
                        if(totalScores.get(u) > win)
                                {
                                        win = totalScores.get(u);
                                        //System.out.println(win);
                                }
                        }

        for(int y = 0; y < totalScores.size(); y++)
                        {
                        if(totalScores.get(y) == win)
                                {
                                //System.out.println(win);
                                winIndex = y;
```

```
                                        winningList.add(inPlayers.get(winIndex));
                                        inPlayers.get(winIndex).setWinner();
                                        }
                                }

                        for(int ddx = 0; ddx < inPlayers.size(); ddx++)
                        {
                                ArrayList<PCard> list1 = new ArrayList();
                                list1 = (inPlayers.get(ddx).getHand());
                        list1.addAll(getTable());
                                System.out.println("Player " + inPlayers.get(ddx) + "'s available
cards: " + list1.toString());
                        }


                if(winningList.size() == 1)
                                {
                                        //Player winner = new Player();
                                //      winner = winningList.get(0);
                        System.out.println("Congratulations " + inPlayers.get(winIndex) + ". You
win this round with a " + allHands.get(winIndex).toString() + ".");
                                }
                else if (winningList.size() > 1)
                                {
                        System.out.println("There is a tie. Congratulations. Players" +
winningList.toString() + " win this round.");
                                }
                                Pay();

   }

   public void Pay()
   {

        //System.out.println("Test 1");
        for(int q = 0; q < players.size(); q++)
        {
                //System.out.println("Test 2");
                if(players.get(q).getWinner() && (winningList.size() == 1))
                {
                        //System.out.println("Test 3");
                        players.get(q).pay(amount);
                }
                else if(players.get(q).getWinner() && (winningList.size() > 1))
                {
                        //System.out.println("Test 4");
```

```
                    amount = amount/winningList.size();
                    players.get(q).pay(amount);
                }
        }



/* if(winnerID == players.get(0).getID())
        {
                players.get(0).pay(currentPot);
        }
 else if(winnerID == players.get(1).getID())
        {
                players.get(1).pay(currentPot);
        }
 else if(winnerID == players.get(2).getID())
        {
                players.get(2).pay(currentPot);
        }
 else if(winnerID == players.get(3).getID())
        {
                players.get(3).pay(currentPot);
        }
 else if(winnerID == players.get(4).getID())
        {
                players.get(4).pay(currentPot);
        }
 else if(winnerID == players.get(5).getID())
        {
                players.get(5).pay(currentPot);
        }*/
 }
 public ArrayList<Player> getInPlayers()
 {
        ArrayList<Player> inPlay = new ArrayList<Player>();
        for(int tre = 0; tre <players.size(); tre++)
        {
                if(players.get(tre).hasFold == false)
                {
                        inPlay.add(players.get(tre));
                }
        }
        return inPlay;
 }
```

```
public void setWinner(int winner)
{
		winnerID = winner;
}

public void addBets(int wager)
{
/*if(players.get(0).status() == true)
	{
		players.get(0).wager(amount);
		currentPot = currentPot + amount;
		callAmount = amount;

	}
else if(players.get(1).status() == true)
	{
		players.get(1).wager(amount);
		currentPot = currentPot + amount;
		callAmount = amount;
	}
else if(players.get(2).status() == true)
	{
		players.get(0).wager(amount);
		currentPot = currentPot + amount;
		callAmount = amount;
	}
else if(players.get(3).status() == true)
	{
		players.get(0).wager(amount);
		currentPot = currentPot + amount;
		callAmount = amount;
	}
else if(players.get(4).status() == true)
	{
		players.get(0).wager(amount);
		currentPot = currentPot + amount;
		callAmount = amount;
	}
else if(players.get(5).status() == true)
	{
		players.get(0).wager(amount);
		currentPot = currentPot + amount;
		callAmount = amount;
	}*/
	amount += wager;
}
```

```
public void addWager (int wag)
{
	callAmount = wag;
}

	public void addCard(PCard newCard)
	{
		Table.add(newCard);
	}

	public ArrayList getTable()
	{
		return Table;
	}

	public void clearTable()
	{
		Table.clear();
	}

	public int numCards()
	{
		return Table.size();
	}

	public int getWager()
	{
		return callAmount;
	}

	public int getPot()
	{
		return amount;
	}

	public int setWager(int bet)
	{
		callAmount = bet;
		return callAmount;
	}
	/*public boolean compareplayers()
	{
		for (int i=0; i< players.getSize(); i++)
		{
			balance1 = players(i).updateAccount();
			balance2 = players(players.getSize()-1).updateAccount();
```

```
            }
            if (balance1 > balance2)
            {
                  return true;
            }
            else
                  return false;
      }*/


}
```

Our Shuffle (Excerpt)

```
public static void shuffle(List<?> list) {
      if (r == null) {
         r = new MTRandom();
      }
      shuffle(list, r);
   }
   private static MTRandom r;

   /**
    * Randomly permute the specified list using the specified source of
    * randomness.  All permutations occur with equal likelihood
    * assuming that the source of randomness is fair.<p>
    *
    * This implementation traverses the list backwards, from the last element
    * up to the second, repeatedly swapping a randomly selected element into
    * the "current position".  Elements are randomly selected from the
    * portion of the list that runs from the first element to the current
    * position, inclusive.<p>
    *
    * This method runs in linear time.  If the specified list does not
    * implement the {@link RandomAccess} interface and is large, this
    * implementation dumps the specified list into an array before shuffling
    * it, and dumps the shuffled array back into the list.  This avoids the
    * quadratic behavior that would result from shuffling a "sequential
    * access" list in place.
    *
    * @param  list the list to be shuffled.
    * @param  rnd the source of randomness to use to shuffle the list.
    * @throws UnsupportedOperationException if the specified list or its
    *         list-iterator does not support the <tt>set</tt> operation.
    */
   public static void shuffle(List<?> list, MTRandom rnd)
         {
      int size = list.size();
      if (size < SHUFFLE_THRESHOLD || list instanceof RandomAccess) {
         for (int i=size; i>1; i--)
            swap(list, i-1, rnd.nextInt(i));
      } else {
         Object arr[] = list.toArray();

         // Shuffle array
         for (int i=size; i>1; i--)
            swap(arr, i-1, rnd.nextInt(i));
```

```
        // Dump array back into list
        ListIterator it = list.listIterator();
        for (int i=0; i<arr.length; i++) {
            it.next();
            it.set(arr[i]);
        }
    }
}
```

Deck

```
/**
 * @(#)Deck.java
 * @author J.E.L.T.
 * @version 1.00 2011/8/23
 */

 /*
  * In: 11:10 8/24/11 (JB/LR) Out: 11:40 8/24/11
  *       In:       12:30 8/25/11 (JB/LR) Out: 1:20  8/25/11
  * In:   11:45 8/26/11 (JB/LR) Out: 12:30 8/26/11
  * In:   12:35 8/29/11 (JB/LR) Out:    1:20 8/29/11
  * In:   11:45 8/30/11 (JB/LR) Out:    12:30 8/30/11
  */
 package MerTwist;
 import java.util.ArrayList;


public class Deck

{
        PCard tempCard;
        ArrayList<PCard> deck = new ArrayList<PCard>();

        //---------------------------------
        //---------------------------------

   public Deck()
   {
   }

   //---------------------------------
   //---------------------------------

   public void FillDeck()
   {
        for(int s=0; s<=3; s++)
        {
                for (int v=1; v<14; v++)
                {
                tempCard = new PCard(s, v);
                deck.add(tempCard);
                }
        }
   }
```

```
    //---------------------------------
    //---------------------------------

        public void EmptyDeck()
        {
                deck.clear();
        }

    public int getDeckSize()
    {
        return deck.size();
    }

    //---------------------------------
    //---------------------------------

    public ArrayList getDeck()
    {
        return deck;
    }
}
```

MT Random (Excerpt)

A Java implementation of the MT19937 (Mersenne Twister)
pseudo random number generator algorithm based upon the
original C code by Makoto Matsumoto and Takuji Nishimura.

Patterns Class

```
//---------------------------------------------------------------------------
//Tracks the tendencies of a player including percent of the time that they fold,
//their average bet, the amount of times the player is defined to "bluff", etc.
//---------------------------------------------------------------------------
package MerTwist;
import java.util.*;

public class Patterns
{
	Player player;
	double bluffFactor;

	int moveNum = 0;

	int bluffChance = 0;
	int bluffNum = 0;
	int foldNum = 0;
	int callNum = 0;
	int betNum = 0;
	double betTot = 0;

	Probability prob = new Probability();

   public Patterns(Player pPlayer, double bluffTendency)
   {
	player = pPlayer;
	bluffFactor = bluffTendency;
   }
   public void TrackMove(String move, double bet)
   {
	moveNum++;

	if(move.equalsIgnoreCase("fold"))
		foldNum++;
	else if(move.equalsIgnoreCase("call") || move.equalsIgnoreCase("bet"))
	{
		betNum++;
		betTot += bet;
	}
	else if(move.equalsIgnoreCase("check")) {}
	else
		System.out.println("ERROR; cannot read move; Patterns Class");
   }
   public double getFold()
```

```
    {
        return foldNum/moveNum;
    }
    public double getBetAvg()
    {
        return betTot/betNum;
    }
        public void bluff(ArrayList<PCard> table, ArrayList<PCard> player)
        {
                bluffChance++;

                prob.PossGen(table);
                double community = prob.getProb();
                table.addAll(player);
                prob.PossGen(table);
                double specific = prob.getProb();

                if(specific < (community * bluffFactor))
                        bluffNum++;
        }
        public double getBluff()
        {
                return bluffNum/bluffChance;
        }
}
```

PCard Class

```
/**
 * @(#)PCard.java
 * @author J.E.L.T.
 * @version 1.00 2011/8/23
 */

 /* In: 11:10 8/24/11 (JB/LR) Out: 11:40 8/24/11
  *       In:       12:35 8/25/11 (JB/LR) Out: 1:20  8/25/11
  * In:   11:45 8/26/11 (JB/LR) Out: 12:30 8/26/11
  * In:   12:35 8/29/11 (JB/LR) Out:    1:20 8/29/11
  * In:   11:45 8/30/11 (JB/LR) Out:    12:30 8/30/11
  */
package MerTwist;

public class PCard
{
                private int suit = 0;
                // SUITS: Spades = 0, Hearts = 1, Clubs = 2, Diamonds = 3
                private int value = 1;
                // Values: Ace = 1, 2 = 2, etc.
                private String Suit;
                private String Val;
                private String spades = "Spades";
                private String hearts = "Hearts";
                private String clubs = "Clubs";
                private String diamonds = "Diamonds";
                private String noOne = "Ether";




   public PCard(int newSuit, int newValue)
   {
        if((newValue < -1 || newValue > 14) || (newSuit < -1 || newSuit > 3))
        {
                System.out.println("ERROR (Class PCard, card values are invalid)");
        }
        else
        {
                        suit = newSuit;
                        value = newValue;
        }
        {
```

```
if (value == 1 || value == 14)
	Val = "Ace";
else if (value == 2)
	Val = "Two";
else if (value == 3)
	Val = "Three";
else if (value == 4)
	Val = "Four";
else if (value == 5)
	Val = "Five";
else if (value == 6)
	Val = "Six";
else if (value == 7)
	Val = "Seven";
else if (value == 8)
	Val = "Eight";
else if (value == 9)
	Val = "Nine";
else if (value == 10)
	Val = "Ten";
else if (value == 11)
	Val = "Jack";
else if (value == 12)
	Val = "Queen";
else if (value == 13)
	Val = "King";
else if (value == -1)
{
	Val = "Ace Clone";
	value = 1;
}
}
{
if (suit == 0)
{
	Suit = spades;
}
else if (suit == 1)
{
	Suit = hearts;
}
else if (suit == 2)
{
	Suit = clubs;
}
else if(suit == 3)
```

```
        {
                Suit = diamonds;
        }
        else if(suit == -1)
        {
                Suit = noOne;
        }
        }
}

        // Returns the suit of the card as a string.

public String getSuit()
{

        return Suit;

}
public int suitVal()
{
        return suit;
}

// Returns the value of the card as a string or int.

public String getValue()
{
        return Val;
        }

        //---------------------------------
        //---------------------------------
        public String toString()
        {
                return (Val + " of " + Suit);
        }
        //---------------------------------
        //---------------------------------
        public int getNumValue()
        {
                if(Val.equals("Ace"))
                        return 14;
                else
                        return value;
        }
        public void setSuit(int newSuit)
```

```
        {
                suit = newSuit;
        }

}
```

Player Class

```
/**
 * @(#)Player.java
 * @author J.E.L.T
 * @version 1.00 2011/8/29
 *
 * In: JB 8/29 12:35 Out: 1:20
 * In: JB/LR 8/30 11:40 Out: 12:30
 * In: JB 9/1 11:40 Out: 1:20
 * In: JB/LR 1:10 Out: 1:47
 */
package MerTwist;
import java.util.ArrayList;
import java.util.Scanner;
import java.util.Random;


   public class Player
{
        protected String name = "Leroy Jenkins";
        protected String CoH = "h";
        protected String tempString = "fold";
        protected String keepPlaying = "y";
        protected int ID;
        protected int wager = 0;
        protected int tempBet =0;
        protected int currentBalance;
        protected int tempPot;
        protected int methodFail = -1;
        protected boolean hasFold;
        protected boolean isWinner = false;
        protected boolean didRaise = false;
        protected boolean isInPlay = true;
        protected boolean isComputer = false;
        protected boolean allIn = false;
        protected ArrayList<PCard> hand = new ArrayList<PCard>();
        protected CardTable table;
        protected PotOdds generator;
        Scanner scan = new Scanner(System.in);

        Random rd = new Random();
        final double WEIGHT = .59;  //Determines how much higher a hand needs to be than the
average in order to bet (lower = aggressive)
   final double RANGE = 10; //Determines how the range of randomness (higher = more
random)
```

```
double randomMod;
	//------------------------------
	//------------------------------

public Player(String pname, int Identification, int Balance, String CompOrHuman)
{
	name = pname;

	ID = Identification;
	currentBalance = Balance;
	CoH = CompOrHuman;
		if (CoH.equalsIgnoreCase("c"))
			{
				isComputer = true;
			}
		else
			{
				isComputer = false;
			}

}

//------------------------------
//------------------------------

public void setWinner()
{
	isWinner = true;
}

//------------------------------
//------------------------------

public boolean getAllIn()
{
	return allIn;
}

//------------------------------
	//------------------------------


	public int getCurrentWager()
	{
		return table.getWager();
	}
```

```
	//------------------------------
	//------------------------------

	public void didBet()
	{
		if(tempBet >= table.getWager())
			{
			didRaise = true;
			}
		else if (tempBet < table.getWager())
			{
			didRaise = false;
			}
	}

	//------------------------------
	//------------------------------

	public boolean betStatus()
	{
		return didRaise;
	}

//------------------------------
//------------------------------

public boolean getWinner()
{
	return isWinner;
}

	//------------------------------
	//------------------------------

public int getID()
{
	return ID;
}

//------------------------------
//------------------------------

public String getName()
{
	return name;
```

```
}

//------------------------------
//------------------------------

public int updateAccount()
{
	return currentBalance;
}

//------------------------------
//------------------------------

public void wager(int amount)
{
	currentBalance -= amount;
}

//------------------------------
//------------------------------

public void pay(int winnings)
{
	currentBalance += winnings;
}

//------------------------------
//------------------------------
public void emptyHand()
{
	hand.clear();
}

//------------------------------
//------------------------------

public boolean status()
{
	return isInPlay;
}

//------------------------------
//------------------------------

public boolean updatePlayer()
{
```

```
        return isInPlay;
}

//------------------------------
//------------------------------

public boolean endLoop()
{
        isInPlay = false;
        return isInPlay;
}

//------------------------------
//------------------------------

public boolean isComputer()
{
        return isComputer;
}

//------------------------------
//------------------------------

public ArrayList getHand()
{
        return hand;
}

//------------------------------
//------------------------------

public void addCards(PCard newCard)
{
        hand.add(newCard);
}

//------------------------------
//------------------------------

public String toString()
{
        return name;
}


//------------------------------
```

```
//------------------------------
public void play()
{
        System.out.println("What would you like to do?");
            System.out.print("Bet || Check || Fold: ");
            tempString = scan.next();
            System.out.println();
}
//------------------------------
//------------------------------
public void bet()
{
    System.out.println("What would you like to do?");
    System.out.print("Call || Raise: ");
    tempString = scan.next();
    System.out.println();
    if(tempString.equalsIgnoreCase("call") == true)
    {
            call();
    }
    else if(tempString.equalsIgnoreCase("raise")== true)
            {
            raiseOption();
            }
}

//------------------------------
//------------------------------
public void call()
{
    tempBet = table.getWager();
    if(tempBet < currentBalance)
    {
    currentBalance-=tempBet;
    keepPlaying = "n";
    isInPlay = false;
    table.addBets(tempBet);
    System.out.println();
    }
    else if(tempBet >= currentBalance)
    {
            isAllIn();
    }
}
//------------------------------
//------------------------------
```

```
public void raise()
{
	if(tempBet < currentBalance)
	{
	table.addWager(tempBet);
	currentBalance-=tempBet;
	keepPlaying = "n";
	isInPlay = false;
	didBet();
	table.addBets(tempBet);
	System.out.println();
	}
	else if(tempBet >= currentBalance)
	{
		isAllIn();
	}
}
//------------------------------
//------------------------------
public void raiseOption()
{
	System.out.println("The current bet is: " + table.getWager());
	System.out.print("How much would you like to bet? (Powers of 10): ");
	tempBet = scan.nextInt();
		if(tempBet < table.getWager())
		{
			System.out.println("Invalid Bet. I will call.");
			call();

		}
		else if (tempBet >= table.getWager())
		{
				raise();

			}
}
//------------------------------
//------------------------------
public void checkOption()
{
	if (table.getWager() == 0)
	{
		check();
	}
	else
	{
```

```
                System.out.print("Invalid Bet. Would you like to bet or fold? ");
                tempString = scan.next();
                if(tempString.equalsIgnoreCase("bet"))
                {
                        bet();
                }
                else
                {
                        pfold();
                }
        }
}
//------------------------------
//------------------------------

public boolean getFoldStatus()
{
        return hasFold;
}


//------------------------------
//------------------------------
public void pfold()
{
        isInPlay = false;
        tempBet = 0;
        keepPlaying = "n";
        hasFold = true;
        //isWinner = false;
        table.addBets(tempBet);
        System.out.println();
}
//------------------------------
//------------------------------
public void check()
{
        tempBet = 0;
        keepPlaying = "n";
        isInPlay = false;
        table.addBets(tempBet);
                System.out.println();
}
//------------------------------
//------------------------------
public void showHand()
```

```
{
	System.out.println("Your hand is: ");
	System.out.println(hand);
	System.out.println();
}
//------------------------------
//------------------------------
public void showTable()
{
	System.out.println(table.getTable());
		System.out.println();
}
//------------------------------
//------------------------------
public void showPot()
{
	tempPot = table.getPot();
		System.out.println("Current Pot: " + tempPot);
		System.out.println();
}
//------------------------------
//------------------------------
public void showWager()
{
	System.out.println("Current wager: " + table.getWager());
		System.out.println();
}
//------------------------------
//------------------------------
public void endSequence()
{
	keepPlaying = "n";
	System.out.println();
}
//------------------------------
//------------------------------
public void start(CardTable cTable)
{
	keepPlaying = "y";
	isInPlay = true;
	table = cTable;
	generator = new PotOdds(table);
	while (keepPlaying.equalsIgnoreCase("y") && isInPlay == true && hasFold == false)
	{
	if(isComputer == false)
	{
```

```
tempString = "";
System.out.println("Player: " + name);
System.out.println("You have: " + currentBalance);
System.out.println("What would you like to do?");
System.out.print("Play || Check_Hand || Check_Table || Check_Pot || Check_Wager: ");
tempString = scan.next();
System.out.println();
if(tempString.equalsIgnoreCase("Play") == true)
{
	if (currentBalance > 0)
	{
			play();
	}
//	else
//  {
//	}
	if (tempString.equalsIgnoreCase("bet") == true && allIn == false)
	{
		bet();
	}
	else if (tempString.equalsIgnoreCase("check") == true&& allIn == false)
	{
		checkOption();
	}
	else if (tempString.equalsIgnoreCase("fold") == true&& allIn == false)
	{
		pfold();
	}
}
else if (tempString.equalsIgnoreCase("Check_Hand") == true)
{
	showHand();
}
else if (tempString.equalsIgnoreCase("Check_Table") == true)
{
		showTable();
}
else if (tempString.equalsIgnoreCase("Check_Pot") == true)
{
	showPot();
}
else if (tempString.equalsIgnoreCase("Check_wager") == true)
{
		showWager();
}
else if (tempString.equalsIgnoreCase("PotOdds") == true)
```

```
	{
		System.out.println(generator.calcPotOdds(table.getWager()));
	}
	else if (tempBet >= table.getWager())
	{
		endSequence();
	}
		}

	}
		returnTable(cTable);
}
	public void isAllIn()
	{
		System.out.println("You have no money left, dunderhead. You are going all in.");
	System.out.println();
	table.addWager(currentBalance);
	currentBalance = 0;
	keepPlaying = "n";
	isInPlay = false;
	hasFold = true;
	table.addBets(currentBalance);
	System.out.println();
	allIn = true;
	//keepPlaying = "n";
	//isInPlay = false;

	}

public int getPot()
{
	return tempPot;
}

 public int getBet()
{
	return tempBet;
}


public CardTable returnTable(CardTable tables)
{
	return tables;
}

public void preFlop(CardTable cTable)
```

```
{
	keepPlaying = "y";
	isInPlay = true;
	table = cTable;
	while (keepPlaying.equalsIgnoreCase("y") && isInPlay == true && hasFold == false)
	{
		if(isComputer == false)
		{
	System.out.println("Player: " + name);
	System.out.println("You have: " + currentBalance);
	System.out.println("What would you like to do?");
	System.out.print("Play || Check_Hand || Check_Pot || Check_Wager: ");
		tempString = scan.next();
	System.out.println();
	if(tempString.equalsIgnoreCase("Play") == true)
	{
			play();

		if (tempString.equalsIgnoreCase("bet") == true)
		{
			bet();
		}

		else if (tempString.equalsIgnoreCase("check") == true)
		{
			checkOption();
		}
		else if (tempString.equalsIgnoreCase("fold") == true)
		{
			pfold();
		}
	}
	else if (tempString.equalsIgnoreCase("Check_Hand") == true)
	{
		showHand();
	}
	else if (tempString.equalsIgnoreCase("Check_Pot") == true)
	{
		showPot();
	}
	else if (tempString.equalsIgnoreCase("Check_wager") == true)
	{
			showWager();
	}
	else if (tempBet >= table.getWager())
	{
```

```
                endSequence();
        }
                }
        }
        returnTable(cTable);
}

public void CompStart(CardTable cTable)
{
        keepPlaying = "y";
        isInPlay = true;
        table = cTable;
        while (keepPlaying.equalsIgnoreCase("y") && isInPlay == true)
        {
                if(isComputer == true)
                {

                }
        }
        returnTable(cTable);
}
//-------------------------------------------------------------
//get table cards from cTable.getTable()
//get hand from array list hand
//get number of players from cTable.getPlayers();
//get the pot total from cTable.getPot();
//get the current wage from cTable.getWager();
//-------------------------------------------------------------
public void CompDecide(CardTable cTable)
{
        Probability prob = new Probability();
        ArrayList<PCard> comp = new ArrayList();

        double decide;
        double foldLim = 1;
        double checkCallLim = 1.5;

                randomMod = (rd.nextDouble() + WEIGHT)/RANGE;

        comp.addAll(cTable.getTable());
        comp.addAll(hand);
        prob.PossGen(comp);
        double compScore = prob.getProb();

        prob.PossGen(cTable.getTable());
        double tableScore = prob.getProb();
```

```
        decide = (compScore/tableScore) * (cTable.getPot()/(cTable.getWager() *
cTable.getPlayers().size() + .00001)) * randomMod;

        if(decide < foldLim)
                pfold();
                else
                {
                        randomMod = (rd.nextDouble() + WEIGHT)/RANGE;
                        decide = compScore/tableScore * randomMod;

                        if(decide < checkCallLim)
                        {
                                if(cTable.getWager() == 0)
                                        check();
                                else
                                        call();
                        }
                        else
                        {
                        //      combet(cTable);
                        }

                }
   }
   //---------------------------------------------------------
   //---------------------------------------------------------
   public void compbet(CardTable cTable)
   {
        final double ECONMOD = .05;

        int totEcon = 0;

        for(int i = 0; i < cTable.getPlayers().size(); i++)
        {
                totEcon += cTable.getPlayers().get(i).updateAccount();
        }
        totEcon += cTable.getPot();
                randomMod = (rd.nextDouble() + WEIGHT)/RANGE;


   }
  }
```

Pot Odds Class

```
/**
 * @(#)PotOdds.java
 *
 *
 * @author
 * @version 1.00 2012/3/9
 */
package MerTwist;
import java.util.Scanner;
import java.util.ArrayList;
import java.text.NumberFormat;
import java.text.DecimalFormat;


public class PotOdds
{
        CardTable table;
        private double handOdds;
        private double potOdds;
        NumberFormat fmt = new DecimalFormat("#0.00");

   public PotOdds(CardTable cTable)
   {
        table = cTable;
   }

   public double calcHandOdds()
   {
        return handOdds;
   }

   public double calcPotOdds(int wager)
   {
        potOdds = (wager + table.getPot()) / wager;

        return fmt.format(potOdds);
   }

}
//--------------------------------------------------------------------------------
//Performs a variety of methods in order to derive the "Gross value index"; a number
//that describes the average value of a hand given the known cards in play and accounting
//for all possible combinations for the remaining empty slots.
//--------------------------------------------------------------------------------
```

```
//The God Number (for this program): 178348.98295992246
package MerTwist;
import java.util.*;

public class Probability {

	Deck standard = new Deck();                                    //Creates deck
	ArrayList<PCard> known = new ArrayList();                      //list of known cards in play
	ArrayList<PCard> left;                                         //list of cards
left in the deck
	ArrayList<PCard> real = new ArrayList();                       //Creates uncompleted combination
of cards based on known
	ArrayList<PCard> possibility = new ArrayList();                //temp Combo object for tracking the
current possibility
	ScoreGen gen = new ScoreGen();                                 //Object for
generating the "score" of a hand
	PCard temp;                                                    //temp
PCard object for pulling out a arraylist index
	ArrayList<Integer> indexList = new ArrayList();                //holds the index of each card in a
compo possibility
	double indexProb = 0;                                          //net probability of
any given hand
	int possNum = 0;                                               //Total number
of possibilities

	int variable = 0;  //not important; delete if you don't know what it is

	//-----------------------------------------------------------------
	//Constructor: fills the deck
	//-----------------------------------------------------------------
  public Probability()
  {
	standard.FillDeck();
  }
  //-----------------------------------------------------------------
  //Makes necessary preparations and calls necessary methods to create
  //an array list of all possible combinations given the current cards
  //in play
  //-----------------------------------------------------------------
  public void PossGen(ArrayList list)
  {
	known = list;
	int unknown = 7 - known.size();
	known = gen.sort(known, 1);
	//gets an arraylist of standard cards
```

```
        left = standard.getDeck();
        //removes the cards in play from the "leftover" deck
        for(int i = 0; i < left.size(); i++)
        {
                for(int j = 0; j < known.size(); j++)
                        if(known.get(j).toString().equals(left.get(i).toString()))
                        {
                                left.remove(i);
                        }
        }

        //transfers all cards in play into a starting "combo" object
        for(int i = 0; i < known.size(); i++)
        {
                real.add(known.get(i));
        }

                //Returns the total average if there are NO known cards
                if(unknown == 8)
                {
                        indexProb = 177571.1052807108;
                        possNum = 1;
                }
                //Utilizes the shortcut method for a pre-flop hand
                else if(unknown == 1412424)
                {
                        indexProb = shortCut();
                        possNum = 1;
                }
                //Returns the certain probability if all cards are known
                else if(unknown == 0)
                {
                        indexProb = gen.getVal(real);
                        possNum = 1;
                }
                //Generates a processed probability value for all other possibilities
                else if(unknown >= 1 && unknown <= 7)
                {
                        Create(0, unknown);
                }
                else
                        System.out.println("ERROR: Unknown must be between 0 and 7
(Probability Class)");
   }
        //-------------------------------------------------------------
        //Recursion method which generates every possible combination of
```

```
//hands given the in play cards and puts them into array list poss
//  -- Exact operations cannot be easily explained
//-------------------------------------------------------------
private void Create(int inTrack, int num)
{
	for(int i = inTrack; i < left.size(); i++)
	{
		if(num == 1)
		{
			indexList.add(i);
			for(int j = 0; j < indexList.size(); j++)
			{
				possibility.add(left.get(indexList.get(j)));
			}
			//-----------------------------------------------
			//Begin operatoins for generating possibilties here
			//-----------------------------------------------
			possNum++;

			possibility.addAll(real);
			double temp = gen.getVal(possibility);
			indexProb += temp;
			//-----------------------------------------------
			//Process ends here
			//-----------------------------------------------

			indexList.remove(indexList.size()-1);
			possibility.clear();
		}
		else
		{
			indexList.add(i);
			Create(i+1, num-1);
			indexList.remove(indexList.size()-1);
		}
	}
}
//-------------------------------------------------------------
//Returns the average probability
//-------------------------------------------------------------
public double getProb()
{
	return (double)indexProb/possNum;
}
//-----------------------------------------------------------------------------------
//Stores pre-generated values of every probability score for the pre-flop round. This is
```

```
        //only for the purpose of reducing processing time.
        //--------------------------------------------------------------------------------
        public double shortCut()
        {
                //Sorts cards in ascending order by card value
                known = gen.sort(known, 1);

                //Shortcut scores for same suit cards
                if(known.get(0).suitVal() == known.get(1).suitVal())
        {
                        if(known.get(0).getNumValue() == 2 && known.get(1).getNumValue()
== 2)
                           return 185466.77867271658;
                        else if(known.get(0).getNumValue() == 2 &&
known.get(1).getNumValue() == 3)
                           return 161092.53888315984;
                        else if(known.get(0).getNumValue() == 2 &&
known.get(1).getNumValue() == 4)
                           return 162923.5906566293;
                        else if(known.get(0).getNumValue() == 2 &&
known.get(1).getNumValue() == 5)
                           return 164753.4135698134;
                        else if(known.get(0).getNumValue() == 2 &&
known.get(1).getNumValue() == 6)
                           return 163859.65686540725;
                        else if(known.get(0).getNumValue() == 2 &&
known.get(1).getNumValue() == 7)
                           return 163217.69743738725;
                        else if(known.get(0).getNumValue() == 2 &&
known.get(1).getNumValue() == 8)
                           return 165032.84723615996;
                        else if(known.get(0).getNumValue() == 2 &&
known.get(1).getNumValue() == 9)
                           return 166203.00815123637;
                        else if(known.get(0).getNumValue() == 2 &&
known.get(1).getNumValue() == 10)
                           return 167472.72089776132;
                        else if(known.get(0).getNumValue() == 2 &&
known.get(1).getNumValue() == 11)
                           return 168336.20681044555;
                        else if(known.get(0).getNumValue() == 2 &&
known.get(1).getNumValue() == 12)
                           return 169499.78507917924;
                        else if(known.get(0).getNumValue() == 2 &&
known.get(1).getNumValue() == 13)
                           return 171190.35268889225;
```

```
                else if(known.get(0).getNumValue() == 2 &&
known.get(1).getNumValue() == 14)
                    return 175758.4053359853;
                else if(known.get(0).getNumValue() == 3 &&
known.get(1).getNumValue() == 3)
                    return 190341.9493662729;
                else if(known.get(0).getNumValue() == 3 &&
known.get(1).getNumValue() == 4)
                    return 166351.872749732;
                else if(known.get(0).getNumValue() == 3 &&
known.get(1).getNumValue() == 5)
                    return 168164.54495162363;
                else if(known.get(0).getNumValue() == 3 &&
known.get(1).getNumValue() == 6)
                    return 167268.35991313533;
                else if(known.get(0).getNumValue() == 3 &&
known.get(1).getNumValue() == 7)
                    return 166621.83390013588;
                else if(known.get(0).getNumValue() == 3 &&
known.get(1).getNumValue() == 8)
                    return 166176.14763104697;
                else if(known.get(0).getNumValue() == 3 &&
known.get(1).getNumValue() == 9)
                    return 167971.3095556509;
                else if(known.get(0).getNumValue() == 3 &&
known.get(1).getNumValue() == 10)
                    return 169233.61307314644;
                else if(known.get(0).getNumValue() == 3 &&
known.get(1).getNumValue() == 11)
                    return 170087.76179540833;
                else if(known.get(0).getNumValue() == 3 &&
known.get(1).getNumValue() == 12)
                    return 171235.81201494392;
                else if(known.get(0).getNumValue() == 3 &&
known.get(1).getNumValue() == 13)
                    return 172904.66071789645;
                else if(known.get(0).getNumValue() == 3 &&
known.get(1).getNumValue() == 14)
                    return 177432.09417867745;
                else if(known.get(0).getNumValue() == 4 &&
known.get(1).getNumValue() == 4)
                    return 195185.5576301969;
                else if(known.get(0).getNumValue() == 4 &&
known.get(1).getNumValue() == 5)
                    return 171511.70661571802;
```

```
                else if(known.get(0).getNumValue() == 4 &&
known.get(1).getNumValue() == 6)
                    return 170613.2478224742;
                else if(known.get(0).getNumValue() == 4 &&
known.get(1).getNumValue() == 7)
                    return 169962.25633490103;
                else if(known.get(0).getNumValue() == 4 &&
known.get(1).getNumValue() == 8)
                    return 169512.12869035796;
                else if(known.get(0).getNumValue() == 4 &&
known.get(1).getNumValue() == 9)
                    return 169119.2898760256;
                else if(known.get(0).getNumValue() == 4 &&
known.get(1).getNumValue() == 10)
                    return 170986.71449796724;
                else if(known.get(0).getNumValue() == 4 &&
known.get(1).getNumValue() == 11)
                    return 171831.51653227193;
                else if(known.get(0).getNumValue() == 4 &&
known.get(1).getNumValue() == 12)
                    return 172964.03129843218;
                else if(known.get(0).getNumValue() == 4 &&
known.get(1).getNumValue() == 13)
                    return 174611.1536895105;
                else if(known.get(0).getNumValue() == 4 &&
known.get(1).getNumValue() == 14)
                    return 179097.87715468596;
                else if(known.get(0).getNumValue() == 5 &&
known.get(1).getNumValue() == 5)
                    return 200001.33962486454;
                else if(known.get(0).getNumValue() == 5 &&
known.get(1).getNumValue() == 6)
                    return 173897.30010044604;
                else if(known.get(0).getNumValue() == 5 &&
known.get(1).getNumValue() == 7)
                    return 173241.98140823076;
                else if(known.get(0).getNumValue() == 5 &&
known.get(1).getNumValue() == 8)
                    return 172787.3753568156;
                else if(known.get(0).getNumValue() == 5 &&
known.get(1).getNumValue() == 9)
                    return 172390.22037660092;
                else if(known.get(0).getNumValue() == 5 &&
known.get(1).getNumValue() == 10)
                    return 172139.51605959612;
```

```
                else if(known.get(0).getNumValue() == 5 &&
known.get(1).getNumValue() == 11)
                    return 173567.13179385892;
                else if(known.get(0).getNumValue() == 5 &&
known.get(1).getNumValue() == 12)
                    return 174684.10160928237;
                else if(known.get(0).getNumValue() == 5 &&
known.get(1).getNumValue() == 13)
                    return 176309.4902842927;
                else if(known.get(0).getNumValue() == 5 &&
known.get(1).getNumValue() == 14)
                    return 180755.41869741832;
                else if(known.get(0).getNumValue() == 6 &&
known.get(1).getNumValue() == 6)
                    return 204455.1780447137;
                else if(known.get(0).getNumValue() == 6 &&
known.get(1).getNumValue() == 7)
                    return 175834.59858876237;
                else if(known.get(0).getNumValue() == 6 &&
known.get(1).getNumValue() == 8)
                    return 175380.32819548788;
                else if(known.get(0).getNumValue() == 6 &&
known.get(1).getNumValue() == 9)
                    return 174983.84356913305;
                else if(known.get(0).getNumValue() == 6 &&
known.get(1).getNumValue() == 10)
                    return 174734.58585883278;
                else if(known.get(0).getNumValue() == 6 &&
known.get(1).getNumValue() == 11)
                    return 174127.5280940186;
                else if(known.get(0).getNumValue() == 6 &&
known.get(1).getNumValue() == 12)
                    return 175800.64243333897;
                else if(known.get(0).getNumValue() == 6 &&
known.get(1).getNumValue() == 13)
                    return 177411.35019398405;
                else if(known.get(0).getNumValue() == 6 &&
known.get(1).getNumValue() == 14)
                    return 179864.71974013542;
                else if(known.get(0).getNumValue() == 7 &&
known.get(1).getNumValue() == 7)
                    return 208993.95924881424;
                else if(known.get(0).getNumValue() == 7 &&
known.get(1).getNumValue() == 8)
                    return 177853.21591417518;
```

```
                else if(known.get(0).getNumValue() == 7 &&
known.get(1).getNumValue() == 9)
                    return 177459.6525308836;
                else if(known.get(0).getNumValue() == 7 &&
known.get(1).getNumValue() == 10)
                    return 177219.37150069486;
                else if(known.get(0).getNumValue() == 7 &&
known.get(1).getNumValue() == 11)
                    return 176632.16558253922;
                else if(known.get(0).getNumValue() == 7 &&
known.get(1).getNumValue() == 12)
                    return 176372.7907907483;
                else if(known.get(0).getNumValue() == 7 &&
known.get(1).getNumValue() == 13)
                    return 178525.11521358244;
                else if(known.get(0).getNumValue() == 7 &&
known.get(1).getNumValue() == 14)
                    return 181518.00679304719;
                else if(known.get(0).getNumValue() == 8 &&
known.get(1).getNumValue() == 8)
                    return 213580.50142536787;
                else if(known.get(0).getNumValue() == 8 &&
known.get(1).getNumValue() == 9)
                    return 179899.10101908588;
                else if(known.get(0).getNumValue() == 8 &&
known.get(1).getNumValue() == 10)
                    return 179661.7062337134;
                else if(known.get(0).getNumValue() == 8 &&
known.get(1).getNumValue() == 11)
                    return 179087.72399151753;
                else if(known.get(0).getNumValue() == 8 &&
known.get(1).getNumValue() == 12)
                    return 178848.32627276634;
                else if(known.get(0).getNumValue() == 8 &&
known.get(1).getNumValue() == 13)
                    return 179166.18581107515;
                else if(known.get(0).getNumValue() == 8 &&
known.get(1).getNumValue() == 14)
                    return 182649.2629950886;
                else if(known.get(0).getNumValue() == 9 &&
known.get(1).getNumValue() == 9)
                    return 218208.37845470035;
                else if(known.get(0).getNumValue() == 9 &&
known.get(1).getNumValue() == 10)
                    return 182016.43177691087;
```

```
                else if(known.get(0).getNumValue() == 9 &&
known.get(1).getNumValue() == 11)
                    return 181455.73553838523;
                else if(known.get(0).getNumValue() == 9 &&
known.get(1).getNumValue() == 12)
                    return 181235.72147278016;
                else if(known.get(0).getNumValue() == 9 &&
known.get(1).getNumValue() == 13)
                    return 181579.8571625935;
                else if(known.get(0).getNumValue() == 9 &&
known.get(1).getNumValue() == 14)
                    return 183349.12454827694;
                else if(known.get(0).getNumValue() == 10 &&
known.get(1).getNumValue() == 10)
                    return 222896.0716998941;
                else if(known.get(0).getNumValue() == 10 &&
known.get(1).getNumValue() == 11)
                    return 184187.3828951667;
                else if(known.get(0).getNumValue() == 10 &&
known.get(1).getNumValue() == 12)
                    return 183984.68609004348;
                else if(known.get(0).getNumValue() == 10 &&
known.get(1).getNumValue() == 13)
                    return 184352.25875628178;
                else if(known.get(0).getNumValue() == 10 &&
known.get(1).getNumValue() == 14)
                    return 186146.25858760226;
                else if(known.get(0).getNumValue() == 11 &&
known.get(1).getNumValue() == 11)
                    return 227467.21302529422;
                else if(known.get(0).getNumValue() == 11 &&
known.get(1).getNumValue() == 12)
                    return 184688.53329069744;
                else if(known.get(0).getNumValue() == 11 &&
known.get(1).getNumValue() == 13)
                    return 185094.5388508394;
                else if(known.get(0).getNumValue() == 11 &&
known.get(1).getNumValue() == 14)
                    return 186926.88211630445;
                else if(known.get(0).getNumValue() == 12 &&
known.get(1).getNumValue() == 12)
                    return 232127.95223669065;
                else if(known.get(0).getNumValue() == 12 &&
known.get(1).getNumValue() == 13)
                    return 185914.6320371751;
```

```
                        else if(known.get(0).getNumValue() == 12 &&
known.get(1).getNumValue() == 14)
                            return 187784.84864517683;
                        else if(known.get(0).getNumValue() == 13 &&
known.get(1).getNumValue() == 13)
                            return 236864.64484498953;
                        else if(known.get(0).getNumValue() == 13 &&
known.get(1).getNumValue() == 14)
                            return 188753.6943435535;
                        else if(known.get(0).getNumValue() == 14 &&
known.get(1).getNumValue() == 14)
                            return 241685.8190954126;
                        else
                                return -2;
        }
        //Shortcut scores for different suit cards
        else
        {
                        if(known.get(0).getNumValue() == 2 && known.get(1).getNumValue()
== 3)
                            return 165825.67905343862;
                        else if(known.get(0).getNumValue() == 2 &&
known.get(1).getNumValue() == 4)
                            return 167589.04249027668;
                        else if(known.get(0).getNumValue() == 2 &&
known.get(1).getNumValue() == 5)
                            return 169351.4602127014;
                        else if(known.get(0).getNumValue() == 2 &&
known.get(1).getNumValue() == 6)
                            return 168565.5162139425;
                        else if(known.get(0).getNumValue() == 2 &&
known.get(1).getNumValue() == 7)
                            return 168017.3291482263;
                        else if(known.get(0).getNumValue() == 2 &&
known.get(1).getNumValue() == 8)
                            return 169769.0149822756;
                        else if(known.get(0).getNumValue() == 2 &&
known.get(1).getNumValue() == 9)
                            return 170920.64740329565;
                        else if(known.get(0).getNumValue() == 2 &&
known.get(1).getNumValue() == 10)
                            return 172164.83084054806;
                        else if(known.get(0).getNumValue() == 2 &&
known.get(1).getNumValue() == 11)
                            return 173031.11107877153;
```

```
                else if(known.get(0).getNumValue() == 2 &&
known.get(1).getNumValue() == 12)
                    return 174175.92502245225;
                else if(known.get(0).getNumValue() == 2 &&
known.get(1).getNumValue() == 13)
                    return 175809.16349523814;
                else if(known.get(0).getNumValue() == 2 &&
known.get(1).getNumValue() == 14)
                    return 182335.34696396056;
                else if(known.get(0).getNumValue() == 3 &&
known.get(1).getNumValue() == 4)
                    return 170849.5916952766;
                else if(known.get(0).getNumValue() == 3 &&
known.get(1).getNumValue() == 5)
                    return 172594.81098957526;
                else if(known.get(0).getNumValue() == 3 &&
known.get(1).getNumValue() == 6)
                    return 171806.58958474084;
                else if(known.get(0).getNumValue() == 3 &&
known.get(1).getNumValue() == 7)
                    return 171253.91429881612;
                else if(known.get(0).getNumValue() == 3 &&
known.get(1).getNumValue() == 8)
                    return 170889.86518235158;
                else if(known.get(0).getNumValue() == 3 &&
known.get(1).getNumValue() == 9)
                    return 172622.91282274926;
                else if(known.get(0).getNumValue() == 3 &&
known.get(1).getNumValue() == 10)
                    return 173860.0252164553;
                else if(known.get(0).getNumValue() == 3 &&
known.get(1).getNumValue() == 11)
                    return 174717.64884780324;
                else if(known.get(0).getNumValue() == 3 &&
known.get(1).getNumValue() == 12)
                    return 175848.08304183726;
                else if(known.get(0).getNumValue() == 3 &&
known.get(1).getNumValue() == 13)
                    return 177461.21862342115;
                else if(known.get(0).getNumValue() == 3 &&
known.get(1).getNumValue() == 14)
                    return 181734.9760080604;
                else if(known.get(0).getNumValue() == 4 &&
known.get(1).getNumValue() == 5)
                    return 175776.92419859604;
```

```
                else if(known.get(0).getNumValue() == 4 &&
known.get(1).getNumValue() == 6)
                    return 174986.5834355211;
                else if(known.get(0).getNumValue() == 4 &&
known.get(1).getNumValue() == 7)
                    return 174429.56705303062;
                else if(known.get(0).getNumValue() == 4 &&
known.get(1).getNumValue() == 8)
                    return 174061.15972529934;
                else if(known.get(0).getNumValue() == 4 &&
known.get(1).getNumValue() == 9)
                    return 173748.30251454024;
                else if(known.get(0).getNumValue() == 4 &&
known.get(1).getNumValue() == 10)
                    return 175547.9902131214;
                else if(known.get(0).getNumValue() == 4 &&
known.get(1).getNumValue() == 11)
                    return 176396.94773868704;
                else if(known.get(0).getNumValue() == 4 &&
known.get(1).getNumValue() == 12)
                    return 177512.99477774126;
                else if(known.get(0).getNumValue() == 4 &&
known.get(1).getNumValue() == 13)
                    return 179106.02006186845;
                else if(known.get(0).getNumValue() == 4 &&
known.get(1).getNumValue() == 14)
                    return 183340.97755493128;
                else if(known.get(0).getNumValue() == 5 &&
known.get(1).getNumValue() == 6)
                    return 178108.21038131142;
                else if(known.get(0).getNumValue() == 5 &&
known.get(1).getNumValue() == 7)
                    return 177546.99442377608;
                else if(known.get(0).getNumValue() == 5 &&
known.get(1).getNumValue() == 8)
                    return 177174.2332238085;
                else if(known.get(0).getNumValue() == 5 &&
known.get(1).getNumValue() == 9)
                    return 176857.1478123002;
                else if(known.get(0).getNumValue() == 5 &&
known.get(1).getNumValue() == 10)
                    return 176678.05427101493;
                else if(known.get(0).getNumValue() == 5 &&
known.get(1).getNumValue() == 11)
                    return 178068.70105914172;
```

```
                else if(known.get(0).getNumValue() == 5 &&
known.get(1).getNumValue() == 12)
                    return 179170.35144449168;
                else if(known.get(0).getNumValue() == 5 &&
known.get(1).getNumValue() == 13)
                    return 180743.2590256367;
                else if(known.get(0).getNumValue() == 5 &&
known.get(1).getNumValue() == 14)
                    return 184939.32987301052;
                else if(known.get(0).getNumValue() == 6 &&
known.get(1).getNumValue() == 7)
                    return 180023.83633850998;
                else if(known.get(0).getNumValue() == 6 &&
known.get(1).getNumValue() == 8)
                    return 179651.3835279485;
                else if(known.get(0).getNumValue() == 6 &&
known.get(1).getNumValue() == 9)
                    return 179334.9015596957;
                else if(known.get(0).getNumValue() == 6 &&
known.get(1).getNumValue() == 10)
                    return 179157.1198257544;
                else if(known.get(0).getNumValue() == 6 &&
known.get(1).getNumValue() == 11)
                    return 178649.093083874;
                else if(known.get(0).getNumValue() == 6 &&
known.get(1).getNumValue() == 12)
                    return 180267.51740605602;
                else if(known.get(0).getNumValue() == 6 &&
known.get(1).getNumValue() == 13)
                    return 181827.02458304496;
                else if(known.get(0).getNumValue() == 6 &&
known.get(1).getNumValue() == 14)
                    return 184166.96798415857;
                else if(known.get(0).getNumValue() == 7 &&
known.get(1).getNumValue() == 8)
                    return 182015.46243471847;
                else if(known.get(0).getNumValue() == 7 &&
known.get(1).getNumValue() == 9)
                    return 181701.69451501084;
                else if(known.get(0).getNumValue() == 7 &&
known.get(1).getNumValue() == 10)
                    return 181532.20838183767;
                else if(known.get(0).getNumValue() == 7 &&
known.get(1).getNumValue() == 11)
                    return 181042.73211803188;
```

```
                else if(known.get(0).getNumValue() == 7 &&
known.get(1).getNumValue() == 12)
                    return 180859.55223669554;
                else if(known.get(0).getNumValue() == 7 &&
known.get(1).getNumValue() == 13)
                    return 182921.95602884318;
                else if(known.get(0).getNumValue() == 7 &&
known.get(1).getNumValue() == 14)
                    return 185762.70055986356;
                else if(known.get(0).getNumValue() == 8 &&
known.get(1).getNumValue() == 9)
                    return 184033.17581726043;
                else if(known.get(0).getNumValue() == 8 &&
known.get(1).getNumValue() == 10)
                    return 183866.3685953733;
                else if(known.get(0).getNumValue() == 8 &&
known.get(1).getNumValue() == 11)
                    return 183389.3097473225;
                else if(known.get(0).getNumValue() == 8 &&
known.get(1).getNumValue() == 12)
                    return 183224.81037534086;
                else if(known.get(0).getNumValue() == 8 &&
known.get(1).getNumValue() == 13)
                    return 183578.8508537763;
                else if(known.get(0).getNumValue() == 8 &&
known.get(1).getNumValue() == 14)
                    return 186874.27829109455;
                else if(known.get(0).getNumValue() == 9 &&
known.get(1).getNumValue() == 10)
                    return 186117.90733374088;
                else if(known.get(0).getNumValue() == 9 &&
known.get(1).getNumValue() == 11)
                    return 185653.3339752102;
                else if(known.get(0).getNumValue() == 9 &&
known.get(1).getNumValue() == 12)
                    return 185506.94442242786;
                else if(known.get(0).getNumValue() == 9 &&
known.get(1).getNumValue() == 13)
                    return 185885.5001815724;
                else if(known.get(0).getNumValue() == 9 &&
known.get(1).getNumValue() == 14)
                    return 187586.54945170935;
                else if(known.get(0).getNumValue() == 10 &&
known.get(1).getNumValue() == 11)
                    return 188253.5029636123;
```

```
                        else if(known.get(0).getNumValue() == 10 &&
known.get(1).getNumValue() == 12)
                            return 188123.13381955549;
                        else if(known.get(0).getNumValue() == 10 &&
known.get(1).getNumValue() == 13)
                            return 188523.35668353442;
                        else if(known.get(0).getNumValue() == 10 &&
known.get(1).getNumValue() == 14)
                            return 190247.10326151713;
                        else if(known.get(0).getNumValue() == 11 &&
known.get(1).getNumValue() == 12)
                            return 188837.8828821941;
                        else if(known.get(0).getNumValue() == 11 &&
known.get(1).getNumValue() == 13)
                            return 189274.97534520275;
                        else if(known.get(0).getNumValue() == 11 &&
known.get(1).getNumValue() == 14)
                            return 191035.28838261156;
                        else if(known.get(0).getNumValue() == 12 &&
known.get(1).getNumValue() == 13)
                            return 190100.17405902842;
                        else if(known.get(0).getNumValue() == 12 &&
known.get(1).getNumValue() == 14)
                            return 191896.6237686013;
                        else if(known.get(0).getNumValue() == 13 &&
known.get(1).getNumValue() == 14)
                            return 192862.47006537596;
                        else
                                return -2;
        }
        }

}

//-------------------------------------------------------------------------------------------------------
//Master code for generating updated list of the shortcut scores
//-------------------------------------------------------------------------------------------------------
/*      for(int i = 2; i < 15; i++)
                {
                        for(int j = (i+1); j < 15; j++)
                        {
                                Probability prob = new Probability();

                                tempCard1 = new PCard(0, i);
                                tempCard2 = new PCard(0, j);
```

```
                                hand.add(tempCard1);
                                hand.add(tempCard2);

                                prob.PossGen(hand);

                                System.out.println("else if(known.get(0).getNumValue() == " + j +
" && known.get(1).getNumValue() == " + i  + ")");
                                        System.out.println("    return " + (prob.getProb() + 140) +
";");

                                hand.clear();


                        }
                }
*/
```

Probability Test Runner Class

```
//-------------------------------------------------------------------------------
//Class utitlized for testing the Probability and ScoreGen classes for logical errors
//based on artificially entered scenarios. Also, it contains the code for generating
//all pre-flop values for the shortcut method in the Probability class.
//-------------------------------------------------------------------------------

package MerTwist;
import java.util.*;

public class ProbabilityTest {


//-------------------------------------------------------------------------------
//Initiates all necessary objects to make the basic poker framework functional
//-------------------------------------------------------------------------------
   public static void main (String[]args)
   {
                        ArrayList<Player> players = new ArrayList();
                CardTable table = new CardTable(1, players);
                Deck deck = new Deck();
                deck.FillDeck();
                        CardDealer dealer = new CardDealer(deck, 2, table);
                        dealer.Shuffle();

                //Creates an arraylist and ScoreGen object for multipurpose use
                ArrayList<PCard> hand = new ArrayList();
                ScoreGen score = new ScoreGen();
//-------------------------------------------------------------------------------
//Allows troubleshoot testing on the ScoreGen class by evaluating artificially
//generated seven card sets for their score
//-------------------------------------------------------------------------------

        //Card values and suits are entered in the object instantiations below
        PCard card1 = new PCard(3, 8);
        PCard card2 = new PCard(2, 8);
        PCard card3 = new PCard(2, 9);
        PCard card4 = new PCard(3, 9);
        PCard card5 = new PCard(3, 3);
        PCard card6 = new PCard(2, 6);
        PCard card7 = new PCard(2, 14);

        //Cards are added into "hand" array list
        hand.add(card1);
        hand.add(card2);
```

```
hand.add(card3);
hand.add(card4);
hand.add(card5);
hand.add(card6);
hand.add(card7);

System.out.println("Hand: " + hand);

//Prints out the calculated numerical score for the given hand
System.out.println("Calculated Score: " + score.getVal(hand));

//Prints out the verbal description of the score (e.g. flush, pair, etc)
System.out.println(score.getCombo());
```

Score Gen Class

```
//-------------------------------------------------------------------------
//Program which accepts a sorted arraylist of PCards to determine which poker hand
//it is and returns a "score" based on how good the hand is
//-------------------------------------------------------------------------
package MerTwist;
import java.util.*;

public class ScoreGen {

   ArrayList<PCard> poss = new ArrayList();              //Main array list that will store the
cards in play
   double score = -1;                                                  //score that will
eventually be returned

   boolean spadeClone;                                                 //Determines
whether there is an ace of spades in play
   boolean heartClone;                                                 //Determines
whether there is an ace of hearts in play
   boolean clubClone;                                                  //Determines whether
there is an ace of clubs in play
   boolean diamondClone;                                               //Determines whether
there is an ace of diamonds in play

         public String combo;

   public final double COMB = 2598960;                       //Constant that holds the total
number of combinatoins in a 52 card deck
   public double highMax = 186083.157;                       //Maximum score that can be
attain with a high card
   public double pairMax = 342979.6564;                  //Maximum score that can be attain
with a pair
   public double twoPMax = 360630.5126;                  //Maximum score that can be attain
with a two pair
   public double threeMax = 368475.3376;                 //Maximum score that can be attain
with three of a kind
   public double straightMax = 369932.5276;              //Maximum score that can be attain
with a straight
   public double flushMax = 370662.2655;                 //Maximum score that can be attain
with a flush
   public double fullMax = 371197.1399;                  //Maximum score that can be attain
with a full house
   public double fourMax = 371286.2856;                  //Maximum score that can be attain
with four of a kind
   public double straightFMax = 371292.0001;             //The score of a straight flush
```

```
   public double getVal(ArrayList<PCard> arList)
   {
        poss = arList;

        spadeClone = false;
        heartClone = false;
        clubClone = false;
        diamondClone = false;

        //Identify an "Ace Clone" for every ace with value 1 for the purpose of indentifying
straights
        int size = arList.size();
        for(int i = 0; i < size; i++)
        {
                //Creates the generic Ace Clone
                if(poss.get(i).getNumValue() == 14)
                {
                        if(arList.size() == size)
                        {
                                PCard aceClone = new PCard(-1, -1);
                                poss.add(aceClone);
                        }

                                //Deterimines which suits the aceclone can have
                        if(poss.get(i).suitVal() == 0)
                                spadeClone = true;
                        if(poss.get(i).suitVal() == 1)
                                heartClone = true;
                        if(poss.get(i).suitVal() == 2)
                                clubClone = true;
                        if(poss.get(i).suitVal() == 3)
                                diamondClone = true;
                }
        }

        //reorders the hand by descending value
        poss = sort(poss, -1);

                //Attempts to find a straight flush; extra operations are needed if aces are present
                score = StraightFlushVal();
        if(spadeClone && score == -1)
        {
                poss.get(poss.size()-1).setSuit(0);
                score = StraightFlushVal();
                poss.get(poss.size()-1).setSuit(-1);
```

```
        }
        if(heartClone && score == -1)
        {
                poss.get(poss.size()-1).setSuit(1);
                score = StraightFlushVal();
                poss.get(poss.size()-1).setSuit(-1);
        }
        if(clubClone && score == -1)
                {
                poss.get(poss.size()-1).setSuit(2);
                score = StraightFlushVal();
                poss.get(poss.size()-1).setSuit(-1);
        }
        if(diamondClone && score == -1)
        {
                poss.get(poss.size()-1).setSuit(3);
                score = StraightFlushVal();
                poss.get(poss.size()-1).setSuit(-1);
        }

                //Systematically sorts through all the other hands; returns -1 if criteria is not fit
        if(score == -1)
                score = FourVal();
        if(score == -1)
                score = FullVal();
        if(score == -1)
                score = FlushVal();
        if(score == -1)
                score = StraightVal();
        if(score == -1)
                score = ThreeVal();
        if(score == -1)
                score = TwoPairVal();
        if(score == -1)
                score = PairVal();
        if(score == -1)
                score = HighVal();

                //Returns the score of the hand minus the lowest possible score
                return score - 122756.4662550045;
}
//-----------------------------------------------------------------------------------------
//Returns a string description of the combination achieved (e.g. "Straight" or "High Card")
//-----------------------------------------------------------------------------------------
public String getCombo()
{
```

```
	return combo;
}
private double HighVal()
	{
		double highScore = 0;
		for(int i = 0; i < poss.size(); i++)
		{
			highScore += (poss.get(i).getNumValue()-1) * Math.pow(13, 4 - i);
		}
		combo = "High Card";
		return highScore * (1302540/COMB);
	}
	private double PairVal()
	{
		int Ptrack = 1;
		int Pindex = 0;
		double pairScore = 0;

		for(int i = 1; i < poss.size() && Ptrack < 2; i++)
		{

			if(poss.get(i).getNumValue() == poss.get(i-1).getNumValue())
			{
				Ptrack++;
				Pindex = poss.get(i).getNumValue();
				poss.remove(i);
				poss.remove(i-1);
			}
		}
		if(Ptrack >= 2)
		{
			pairScore = Pindex * Math.pow(13, 4);
			for(int i = 0; i < 3; i++)
				pairScore += (poss.get(i).getNumValue()-2) * Math.pow(13, 3-i);
			combo = "Pair";
			return pairScore * (1098240/COMB) + highMax;
		}
		else
			return -1;
	}
	private double TwoPairVal()
	{
		int TPtrack1 = 1;
		int TPindex1 = 0;
		int TPtrack2 = 1;
		int TPindex2 = 0;
```

```
                double highOther = 0;
                double twoPScore = 0;

                for(int i = 1; i < poss.size() && (TPtrack1 < 2 || TPtrack2 < 2); i++)
                {
                        if(poss.get(i).getNumValue() == poss.get(i-1).getNumValue() &&
TPtrack1 < 2)
                        {
                                TPtrack1++;
                                TPindex1 = poss.get(i).getNumValue()-2;
                        }
                        if(poss.get(i).getNumValue() == poss.get(i-1).getNumValue() &&
poss.get(i).getNumValue() != TPindex1+2)
                        {
                                TPtrack2++;
                                TPindex2 = poss.get(i).getNumValue()-2;
                        }
                }
                for(int i = 0; i < poss.size(); i++)
                {
                        if(poss.get(i).getNumValue() != TPindex1 && poss.get(i).getNumValue()
!= TPindex2)
                        {
                                highOther = poss.get(i).getNumValue();
                                i = poss.size();
                        }
                }
                if(TPtrack1 == 2 && TPtrack2 == 2)
                {
                        twoPScore = TPindex1 * Math.pow(13, 4) + TPindex2 * Math.pow(13, 3)
+ highOther * Math.pow(13, 2);
                        combo = "Two Pair";
                        return twoPScore * (123552/COMB) + pairMax;
                }
                else
                        return -1;
        }
        private double ThreeVal()
        {
                int Ttrack = 1;
                int Tindex = 0;

                for(int i = 1; i < poss.size() && Ttrack < 3; i++)
                {
                        if(poss.get(i).getNumValue() == poss.get(i-1).getNumValue())
                        {
```

```
                        Ttrack++;
                        Tindex = poss.get(i).getNumValue()-2;
                }
                else
                {
                        Ttrack = 1;
                }
        }

        if(Ttrack == 3)
        {
                combo = "Three of a Kind";
                return Tindex * Math.pow(13, 4) * (54912/COMB) + twoPMax;
        }
        else
                return -1;
}
private double StraightVal()
{
        int Strack = 1;
        int Sindex = 0;
        int Sshift = 1;

        for(int i = 1; i < poss.size() && Strack < 5; i++)
        {
                if(poss.get(i).getNumValue() == poss.get(i-Sshift).getNumValue() -1)
                {
                        Strack++;
                        Sindex = poss.get(i).getNumValue()-2;
                        Sshift = 1;
                }
                else if(poss.get(i).getNumValue() == poss.get(i-1).getNumValue())
                {
                        Sshift++;
                }
                else
                {
                        Strack = 1;
                        Sshift = 1;
                }
        }
        if(Strack == 5)
        {
                combo = "Straight";
                return (Strack + Sindex) * Math.pow(13, 4) * (10200/COMB) +
threeMax;
```

```
        }
        else
                return -1;
}
private double FlushVal()
{
        int spades = 0;
        int clubs = 0;
        int diamonds = 0;
        int hearts = 0;
        String Flush = "";
        ArrayList<Integer> flush = new ArrayList();
        double flushScore = 0;

        for(int i = 0; i < poss.size(); i++)
        {
                if(poss.get(i).getSuit().equals("Spades"))
                        spades++;
                else if(poss.get(i).getSuit().equals("Clubs"))
                        clubs++;
                else if(poss.get(i).getSuit().equals("Diamonds"))
                        diamonds++;
                else if(poss.get(i).getSuit().equals("Hearts"))
                        hearts++;
        }
        if(spades >= 5)
                Flush = "Spades";
        else if(clubs >= 5)
                Flush = "Clubs";
        else if(diamonds >= 5)
                Flush = "Diamonds";
        else if(hearts >= 5)
                Flush = "Hearts";

        if(spades >= 5 || clubs >= 5 || diamonds >= 5 || hearts >= 5)
        {
                for(int i = 0; i < poss.size(); i++)
                {
                        if(poss.get(i).getSuit().equals(Flush))
                                flush.add(poss.get(i).getNumValue()-2);
                }
                for(int i = 0; i < flush.size(); i++)
                {
                        flushScore += flush.get(i)-2 * Math.pow(13, 4-i);
                }
                combo = "Flush";
```

```
                return flushScore * (5108/COMB) + straightMax;
            }
            else
                return -1;
        }
        private double FullVal()
        {
            int FHtrack1 = 1;
            int FHindex1 = 0;
            int FHtrack2 = 1;
            int FHindex2 = 0;
            double fullScore = 0;
            boolean change = false;

            for(int i = 1; i < poss.size() - 1 && (FHtrack1 < 3 || FHtrack2 < 3); i++)
            {
                if(poss.get(i).getNumValue() == poss.get(i-1).getNumValue() && change
== false)
                {
                    FHtrack1++;
                    FHindex1 = poss.get(i).getNumValue()-2;
                }
                if(poss.get(i).getNumValue() == poss.get(i+1).getNumValue() &&
poss.get(i).getNumValue() != FHindex1+2)
                {
                    FHtrack2++;
                    FHindex2 = poss.get(i).getNumValue()-2;
                    change = true;
                }
            }
            if(FHtrack1 >= 3 && FHtrack2 >= 2)
            {
                fullScore = FHindex1 * Math.pow(13, 4) + FHindex2 * Math.pow(13, 3);
                return fullScore * (124/COMB) + flushMax;
            }
            else if(FHtrack1 >= 2 && FHtrack2 >= 3)
            {
                fullScore = FHindex2 * Math.pow(13, 4) + FHindex1 * Math.pow(13, 3);
                combo = "Full House";
                return fullScore * (3744/COMB) + flushMax;
            }
            else
                return -1;

        }
        private double FourVal()
```

```
	{
		int Ftrack = 1;
		int Findex = 0;

		for(int i = 1; i < poss.size() && Ftrack < 4; i++)
		{
			if(poss.get(i).getNumValue() == poss.get(i-1).getNumValue())
			{
				Ftrack++;
				Findex = poss.get(i).getNumValue()-2;
			}
			else
			{
				Ftrack = 1;
			}
		}

		if(Ftrack == 4)
		{
			combo = "Four of a Kind";
			return Findex * Math.pow(13, 4) * (624/COMB) + fullMax;
		}
		else
			return -1;
	}
	private double StraightFlushVal()
	{
		int SFtrack = 1;
		int SFindex = 0;
		int Sshift = 1;

		for(int i = 1; i < poss.size() && SFtrack < 5; i++)
		{
			if(poss.get(i).getNumValue() == poss.get(i-Sshift).getNumValue() - 1 &&
poss.get(i).getSuit() == poss.get(i-Sshift).getSuit())
			{
				SFindex = poss.get(i-SFtrack).getNumValue()-2;
				SFtrack++;
				Sshift = 1;
			}
			else if(poss.get(i).getNumValue() == poss.get(i-1).getNumValue())
			{
				Sshift++;
			}
			else if(poss.get(i).getNumValue() == poss.get(i-Sshift).getNumValue() -1
&& poss.get(i).getSuit() != poss.get(i-Sshift).getSuit())
```

```
                {
                        Sshift++;
                }
                else
                {
                        SFtrack = 1;
                        Sshift = 1;
                }
        }
        if(SFtrack >= 5)
        {
                combo = "Straight Flush";
                return (SFindex) * Math.pow(13, 4) * (40/COMB) + fourMax;
        }
        else
                return -1;
    }
    //-----------------------------------------------------------------------------------
    //Takes array and sorts it; enter positive int for ascending and negative for descending
    //-----------------------------------------------------------------------------------
    public ArrayList<PCard> sort(ArrayList<PCard> list, int order)
  {
    int min;
    PCard temp;

    for(int i = 0; i < list.size(); i++)
    {
        min = i;
        for(int scan = i+1; scan < list.size(); scan++)
            if(order * list.get(scan).getNumValue() < order *
list.get(min).getNumValue())
            {
                min = scan;
            }
        temp = list.get(min);
        list.set(min, list.get(i));
        list.set(i, temp);
    }
    return list;
  }
}
```

Quick Runner Class (Engine)

```
/**
 * @(#)QuickRunner.java
 * @Ethan
 * @version 1.00 2011/8/25
 */
 package MerTwist;
 import java.util.Scanner;
 import java.util.ArrayList;


public class QuickRunner
{

   public static void main (String[] args)
   {

                String name;
                String type;
                boolean nextRound = true;
                boolean preflopRound = true;
                boolean flopRound = false;
                boolean turnRound = false;
                boolean riverRound = false;
                int balance;
                int numP = 2;
                int counter = 0;
                int crazydude = 0;
                int j = 0;
                int jklol = 0;
                int numCards = 0;
                Player tempPlayer;
                Deck deck = new Deck();
                deck.FillDeck();
                Scanner scan = new Scanner(System.in);
                ArrayList<Player> players = new ArrayList<Player>();
                ScoreGen gen = new ScoreGen();


                for(int ident = 1; ident < numP + 1; ident++)
                {
                        System.out.println("What is player " + ident + "'s name?");
                        name = scan.next();
                        System.out.println("Is player " + ident + " a human(H) or a
computer(C)?");
```

```
type = scan.next();
        System.out.println("What is player " + ident + "'s balance?");
balance = scan.nextInt();

Player player = new Player(name, ident, balance, type);
        players.add(player);
        System.out.println();
}
boolean inGame = false;
for(int w = 0; w < players.size(); w++)
{

        if(players.get(w).updateAccount() > 0)
        {
         inGame = true;
        }
}

while (inGame)
{
//Round 1
**************************************************************************************
************************
System.out.println("Test1");
CardTable table = new CardTable(numP, players);
CardDealer dealer = new CardDealer(deck, numP, table);
nextRound = true;
preflopRound = true;
System.out.println();
System.out.println("Dealing the cards now.");
System.out.println();
for (int uh = 0; uh < players.size(); uh++)
{
        players.get(uh).emptyHand();
}
        dealer.Shuffle();
dealer.HandDeal();
        preflopRound = true;
while (nextRound == true)
{

        if (preflopRound == true)
        {
                counter = 0;
                for (int kk = 0; kk < numP; kk++)
                {
```

```
                    if(table.numCards() == 0)
                    {
                        tempPlayer = players.get(kk);
                        tempPlayer.preFlop(table);
                    }
                    }

                    for(int qwe = 0; qwe < players.size(); qwe++)
                    {
                        players.get(qwe).didBet();
                    }
                    for(int lmnop = 0; lmnop < players.size(); lmnop++)
                        {

                            if((players.get(counter).betStatus() ==
false))
                            {
                                tempPlayer = players.get(lmnop);
                                tempPlayer.preFlop(table);

                                for(int qwer = 0; qwer <
players.size(); qwer++)
                                    {

        players.get(qwer).didBet();

        if(players.get(qwer).betStatus() == true)
                                        {

        crazydude++;
                                        }
                                    }
                                        if(crazydude
== players.size())
                                        {
                                            lmnop
= players.size() + 5;
                                        }

                            }
                            counter++;
                        }

                    for(int dis = 0; dis < players.size(); dis++)
                    {
                        int checkdis = 0;
```

```
                                        if(players.get(dis).updateAccount() < 0)
                                        {
                                                checkdis++;
                                        }
                                        if(checkdis == players.size()-1)
                                        {
                                                table.Showdown();
                                        }

                                }
                                table.setWager(0);
                                dealer.TableDeal();
                                preflopRound = false;
                                flopRound = true;
                        }

                        if (flopRound == true)
                        {
                                counter = 0;
                                jklol = 0;
                                for (int kk = 0; kk < numP; kk++)
                                {
                                        tempPlayer = players.get(kk);
                                        tempPlayer.start(table);
                                if(tempPlayer.betStatus() == true)
                                        {
                                                jklol++;
                                        }
                                }
                                for(int qwe = 0; qwe < players.size(); qwe++)
                                {
                                        players.get(qwe).didBet();
                                }
                                for(int lmnop = 0; lmnop < players.size(); lmnop++)
                                        {

                                                if((players.get(counter).betStatus() ==
false))
                                                {
                                                        tempPlayer = players.get(lmnop);
                                                        tempPlayer.start(table);

                                                        for(int qwer = 0; qwer <
players.size(); qwer++)
                                                                {
```

```
        players.get(qwer).didBet();

        if(players.get(qwer).betStatus() == true)
                                                                        {

        crazydude++;
                                                                        }
                                                                }
                                                                        if(crazydude
== players.size())
                                                                        {
                                                                                lmnop
= players.size() + 5;
                                                                        }

                                                        }
                                                        counter++;
                                                }
                                        for(int dis = 0; dis < players.size(); dis++)
                                                {
                                                        int checkdis = 0;
                                                        if(players.get(dis).updateAccount() < 0)
                                                        {
                                                                checkdis++;
                                                        }
                                                        if(checkdis == players.size()-1)
                                                        {
                                                                table.Showdown();
                                                        }

                                                }
                                                table.setWager(0);
                                                dealer.SecondDeal();
                                                flopRound = false;
                                                turnRound = true;
                                }

                                if (turnRound == true)
                                {
                                                counter = 0;
                                                jklol = 0;
                                        for (int kk = 0; kk < numP; kk++)
                                        {
                                                        tempPlayer = players.get(kk);
                                                        tempPlayer.start(table);
```

```
//		if(tempPlayer.betStatus() == true)
//		{
//			jklol++;
//		}
		}
		for(int qwe = 0; qwe < players.size(); qwe++)
		{
			players.get(qwe).didBet();
		}
		for(int lmnop = 0; lmnop < players.size(); lmnop--)
			{
				//System.out.println("1");
				if((players.get(counter).betStatus() ==
false))
				{
					crazydude = 0;
					tempPlayer = players.get(counter);
					tempPlayer.start(table);

					for(int qwer = 0; qwer <
players.size(); qwer++)
						{

	players.get(qwer).didBet();

	if(players.get(qwer).betStatus() == true)
							{

	crazydude++;
							}
						}
							if(crazydude
== players.size())
							{
								lmnop
= players.size() + 5;
							}


				}
				if(counter == players.size()-1)
					{
							counter = 0;
						}
				counter++;
			}
```

```
                        table.setWager(0);
                        dealer.SecondDeal();
                        preflopRound = false;
                        riverRound = true;
                        for(int dis = 0; dis < players.size(); dis++)
                        {
                                int checkdis = 0;
                                if(players.get(dis).updateAccount() < 0)
                                {
                                        checkdis++;
                                }
                                if(checkdis == players.size()-1)
                                {
                                        table.Showdown();
                                }

                        }
                }

                if (riverRound == true)
                {
                        counter = 0;
                        jklol = 0;
                        for (int kk = 0; kk < numP; kk++)
                        {
                                tempPlayer = players.get(kk);
                                tempPlayer.start(table);
                        if(tempPlayer.betStatus() == true)
                                {
                                        jklol++;
                                }
                        }
                        for(int qwe = 0; qwe < players.size(); qwe++)
                        {
                                players.get(qwe).didBet();
                        }
                        for(int lmnop = 0; lmnop < players.size(); lmnop++)
                                {

                                        if((players.get(counter).betStatus() ==
false))
                                        {
                                                tempPlayer = players.get(lmnop);
                                                tempPlayer.start(table);
```

```
                                                    for(int qwer = 0; qwer <
players.size(); qwer++)
                                                            {

        players.get(qwer).didBet();

        if(players.get(qwer).betStatus() == true)
                                                                    {

        crazydude++;
                                                                    }
                                                            }
                                                                    if(crazydude
== players.size())
                                                                    {
                                                                            lmnop
= players.size() + 5;
                                                                    }

                                                    }
                                                    counter++;
                                            }
                                    table.setWager(0);
                                    table.Showdown();
                                    dealer.ReFill();
                                    nextRound = false;
                                    riverRound = false;
                            }

        ////////////////////////////////////////////////////////////////////////////////////////////////////

        ////////////////////////////////////////////////////////////////////////////////////////////////////
                            else if (nextRound == true)
                            {

                    //      for (int i = 0; i < numP; i++)
                    //      {
                            //      int w = 0;
                                    //if(inGame == true)
                            //      {

                                    //      if(table.numCards() < 6)
                                    //      {

                                                    //numCards = table.numCards();
                                    //              tempPlayer = players.get(i);
```

```
                                        //              tempPlayer.start(table);
                                        //              j++;

                                        //      if(((j % 2) == 0) && (j != 0) && (table.numCards()
< 5))
                                        //      {
                                        //              dealer.SecondDeal();
                                        //              table.setWager(0);
                                                //      w++;
                                        //      }
                                        ///}
                                //      if(j == 5)
                                //      {
                                        //numCards++;
                                        //}

                        //      if(table.numCards() == 5 && (j >= (players.size() * 3)))//
                        //      {
                                //      table.Showdown();
                                //      //inGame = false;
                                 //     nextRound = false; //inGame = false;
                                 //     preflopRound = true;
                                 //     dealer.ReFill();
                        //      }
                        //      }

                //      }

                        for(int w = 0; w < players.size(); w++)
                                {
                                        if((players.get(w).updateAccount() > 0) ||
(players.get(w).getAllIn()))
                                        {
                                                inGame = true;
                                                w = players.size()+1;
                                        }
                                        else
                                                inGame = false;
                                }
                }
                }
        }
}
}
```