```
> restart;
```

Jason Burke
6/13/2002

BlackJack analysis and Simulator final project for Math 107B at UCSD with Professor John Wavrik and TA Stefan Erickson.

Warning: This maple worksheet takes about 50 seconds to execute on my Athlon XP 1700+ with 256 MBs of memory. If you have a much slower computer it may take a long time to execute.

Procedures used for BlackJack:

Create global variables

```
> randomize():
> shoe := 1:
> deck := [seq(i mod 13 + 1,i=0..(51 * shoe))]:
> card_value_table := table([(0)=0,(1)=1,
                                (2)=2,(3)=3,
                                (4)=4,(5)=5,
                                (6)=6,(7)=7,
                                (8)=8,(9)=9,
                                (10)=10,(11)=10,
                                (12)=10,(13)=10]):
> doubling := 1:
> stand := 0:
> split := 0:
> num_of_blackjacks := 0:
```

basic strat table is a look up table to find the best move to take depending on the player's 2 cards and the dealer's
up card. The 2 digits represent the player's 2 cards, and the 3rd digit is the dealer's card. 0 represents 10 or a face card and 1 represents an ace. The other numbers are the same as the card. If the value in the table is 0, that means stand, 1 that means hit, 2 means double and 3 means split. If the hand total is less than 8 or greater than 16 it is not in this chart, because you should always hit when you have less than 8 and stand when you have greater than 16. Also note that in order to save space the hands are organized so that your low card is always first, for example (262) is in the chart but not (622) since these are the same.

```
> basic_strat_table := table([(262) = 1, (263) = 1, (264) =
1,
                              (265) = 1, (266) = 1, (267) =
1,
                              (268) = 1, (269) = 1, (260) =
1,
```

(261) = 1,
(352) = 1, (353) = 1, (354) =
1,
(355) = 2, (356) = 2, (357) =
1,
(358) = 1, (359) = 1, (350) =
1,
(351) = 1,
(122) = 1, (123) = 1, (124) =
2,
(125) = 2, (126) = 2, (127) =
1,
(128) = 1, (129) = 1, (120) =
1,
(121) = 1,
(132) = 1, (133) = 1, (134) =
2,
(135) = 2, (136) = 2, (137) =
1,
(138) = 1, (139) = 1, (130) =
1,
(131) = 1,
(142) = 1, (143) = 1, (144) =
2,
(145) = 2, (146) = 2, (147) =
1,
(148) = 1, (149) = 1, (140) =
1,
(141) = 1,
(152) = 1, (153) = 1, (154) =
2,
(155) = 2, (156) = 2, (157) =
1,
(158) = 1, (159) = 1, (150) =
1,
(151) = 1,
(162) = 2, (163) = 2, (164) =
2,
(165) = 2, (166) = 2, (167) =
1,
(168) = 1, (169) = 1, (160) =
1,
(161) = 1,
(172) = 0, (173) = 2, (174) =
2,
(175) = 2, (176) = 2, (177) =
0,

(178) = 0, (179) = 1, (170) =
1,
(171) = 0,
(182) = 0, (183) = 0, (184) =
0,
(185) = 0, (186) = 2, (187) =
0,
(188) = 0, (189) = 0, (180) =
0,
(181) = 0,
(192) = 0, (193) = 0, (194) =
0,
(195) = 0, (196) = 0, (197) =
0,
(198) = 0, (199) = 0, (190) =
0,
(191) = 0,
(112) = 3, (113) = 3, (114) =
3,
(115) = 3, (116) = 3, (117) =
3,
(118) = 3, (119) = 3, (110) =
3,
(111) = 3,
(222) = 1, (223) = 3, (224) =
3,
(225) = 3, (226) = 3, (227) =
3,
(228) = 1, (229) = 1, (220) =
1,
(221) = 1,
(332) = 1, (333) = 1, (334) =
3,
(335) = 3, (336) = 3, (337) =
3,
(338) = 1, (339) = 1, (330) =
1,
(331) = 1,
(442) = 1, (443) = 1, (444) =
1,
(445) = 2, (446) = 2, (447) =
1,
(448) = 1, (449) = 1, (440) =
1,
(441) = 1,
(552) = 2, (553) = 2, (554) =
2,

```
                                (555) = 2, (556) = 2, (557) =
2,
                                (558) = 2, (559) = 2, (550) =
1,
                                (551) = 1,
                                (662) = 3, (663) = 3, (664) =
3,
                                (665) = 3, (666) = 3, (667) =
1,
                                (668) = 1, (669) = 1, (660) =
1,
                                (661) = 1,
                                (772) = 3, (773) = 3, (774) =
3,
                                (775) = 3, (776) = 3, (777) =
3,
                                (778) = 1, (779) = 1, (770) =
0,
                                (771) = 1,
                                (882) = 3, (883) = 3, (884) =
3,
                                (885) = 3, (886) = 3, (887) =
3,
                                (888) = 3, (889) = 3, (880) =
3,
                                (881) = 3,
                                (992) = 3, (993) = 3, (994) =
3,
                                (995) = 3, (996) = 3, (997) =
0,
                                (998) = 1, (999) = 1, (990) =
0,
                                (991) = 0,
                                (002) = 0, (003) = 0, (004) =
0,
                                (005) = 0, (006) = 0, (007) =
0,
                                (008) = 0, (009) = 0, (000) =
0,
                                (001) = 0]):
```

hand_total: Computes the value of a hand

```
> hand_total := proc (hand::list) local total; global
card_value_table;

total := 0;
```

```
total := sum('card_value_table[hand[k]]', 'k' =
1..nops(hand));

if(is_an_ace(hand)) then
  if(total + 10 <= 21) then
    total := total + 10;
  end if;
end if;


return total;
end proc:
```

is_an_ace: checks the hand for an ace. Returns true if there is at least 1 ace

```
> is_an_ace := proc(hand::list) local i;
  for i from 1 by 1 while i <= nops(hand) do
    if(hand[i] = 1) then return true; end if;
  end do;

  return false;
end proc:
```

players_turn_simple: executes the player's turn with simple strategy

```
> players_turn_simple := proc (my_hand::list,
dealers_hand::list) local new_my_hand; global deck, double;
new_my_hand := my_hand;
double := 1;
while(hand_total(new_my_hand) < 16) do
      new_my_hand := deal_card(new_my_hand);
  end do;

return new_my_hand;
end proc:
```

players_turn_basic: executes the players turn with basic strategy

```
> players_turn_basic := proc (my_hand::list,
dealers_hand::list)
                       local new_my_hand::list, dcard, temp,
what_to_do;
                       global deck, stand, double, split,
                       basic_strat_table, card_value_table;

new_my_hand := my_hand;
```

```
dcard := dealers_hand[1];
# first check that we are not handling a split
stand := 0;
double := 1;

while(stand = 0 and double = 1) do

#check 1 card case, we should always hit after a split
if(nops(new_my_hand) = 1) then
   new_my_hand := deal_card(new_my_hand);
end if;

#checking specific 2 card cases
if(nops(new_my_hand) = 2) then

    if(hand_total(new_my_hand) > 7 and
       hand_total(new_my_hand) < 17) then

      if(new_my_hand[1] > new_my_hand[2]) then
        temp := new_my_hand[1];
        new_my_hand[1] := new_my_hand[2];
        new_my_hand[2] := temp;
      end if;

      #generate value for basic_strat_table
      temp :=    100 * card_value_table[new_my_hand[1]] mod
10 +
                 10 * card_value_table[new_my_hand[2]] mod
10 +
                 dcard;
      what_to_do := basic_strat_table[temp];
      if(what_to_do = 1) then
            new_my_hand := deal_card(new_my_hand);
      elif(what_to_do = 2) then
            double := 2;
            new_my_hand := deal_card(new_my_hand);
      elif(what_to_do = 3) then
            if(split = 0) then
               split := 1;
               new_my_hand := [new_my_hand[1]];
               stand := 1;
               else new_my_hand := deal_card(new_my_hand);
end if;
            elif(what_to_do = 0) then
               stand := 1;
            end if;
      end if;
```

```
   end if;


#check hand total cases
if((stand != 1) and (double != 2)) then

    elif((hand_total(new_my_hand) = 9) and
                     (dcard = 7 or dcard = 8 or dcard = 9 or
card_value_table[dcard] = 10 or dcard = 1)) then
          new_my_hand := deal_card(new_my_hand);

    elif(hand_total(new_my_hand) = 9) then
          new_my_hand := deal_card(new_my_hand);
          double := 2;

    elif((hand_total(new_my_hand) = 10) and
                     (card_value_table[dcard] = 10 or dcard
= 1)) then
          new_my_hand := deal_card(new_my_hand);

    elif(hand_total(new_my_hand) = 10) then
          new_my_hand :=  deal_card(new_my_hand);
          double := 2;

    elif(hand_total(new_my_hand) = 11) then
          new_my_hand := deal_card(new_my_hand);
          double := 2;

    elif((hand_total(new_my_hand) = 12) and (dcard = 4 or
dcard = 5 or dcard = 6)) then
          stand := 1; #stand

    elif(hand_total(new_my_hand) = 12) then
          new_my_hand := deal_card(new_my_hand);

    elif((hand_total(new_my_hand) = 13 or
hand_total(new_my_hand) = 14 or
                         hand_total(new_my_hand) = 15 or
hand_total(new_my_hand) = 16) and
                         (dcard = 2 or dcard = 3 or dcard =
4 or
                          dcard = 5 or dcard = 6)) then
          stand := 1; #stand

    elif(hand_total(new_my_hand) = 13 or
hand_total(new_my_hand) = 14 or
         hand_total(new_my_hand) = 15 or
```

```
hand_total(new_my_hand) = 16) then
         new_my_hand := deal_card(new_my_hand);

    #now check if finished

    elif(hand_total(new_my_hand) > 16) then
        stand := 1;

    elif(hand_total(new_my_hand) < 12) then new_my_hand :=
deal_card(new_my_hand);

    else stand := 1;
  end if;

end do;

#Note: if there is a split just return the 1 card that we
are splitting with
#then let the blackjack function take care of handling both
cases.

return new_my_hand;
end proc:
```

dealers_turn: executes the dealer's turn (stand on soft 17)

```
> dealers_turn := proc(dealers_hand::list) local
new_dealers_hand; global deck;
new_dealers_hand := dealers_hand;
while(hand_total(new_dealers_hand) < 17) do
  new_dealers_hand := deal_card(new_dealers_hand);
end do;

return new_dealers_hand;
end proc:
```

deal_card: deals one card at random from the deck into the players hand

```
> deal_card := proc(hand::list) local rand1, deck_length, i,
new_hand::list; global deck;

   deck_length := nops(deck);
   rand1 := (rand() mod deck_length) + 1;


   new_hand := [op(hand),deck[rand1]];
```

```
   deck := subsop(rand1=NULL,deck);

   return (new_hand);

   end proc:
```

winner: determines the winner, assuming hand2 wins on both bust

```
> winner := proc(hand1::list, hand2::list)
global num_of_blackjacks;
```

first check if the dealer has won:

```
if(hand_total(hand1) > 21 or ((hand_total(hand2) <= 21) and
(hand_total(hand2) > hand_total(hand1)))) then    return -
1; end if;
```

now check if the player has won:

check for blackjack (when you have an ace and a 10 card)

```
if((hand_total(hand1) = 21) and (hand_total(hand2) != 21)
and (((card_value_table(hand1[1]) = 10 and hand1[2] = 1) or
card_value_table(hand1[2]) = 10 and hand1[1] = 1)))  then
return (1.5); end if;
```

check to see if the player should win without a blackjack

```
if((hand_total(hand1) > hand_total(hand2)) or
   (hand_total(hand1) <= 21) and (hand_total(hand2) > 21))
then return 1; end if;
```

now check for a tie

```
if(hand_total(hand1) = hand_total(hand2)) then return 0;
end if;

end proc:
```

blackjack: plays 1 game of blackjack! returns 0 if dealer wins, 1 if player wins, 2 for a tie

```
>
> blackjack := proc (strategy)
            local my_hand, my_hand2, dealers_hand;
            global shoe, deck, split, double,
num_of_blackjacks;

  shoe := 1;
    deck := [seq(i mod 13 + 1,i=0..(51 * shoe))];
    split := 0;
    num_of_blackjacks := 0;
```

Now I will deal the initial hand for the player:

```
    my_hand2 := [];
    my_hand := [];
    my_hand := deal_card(my_hand, deck);
    my_hand := deal_card(my_hand, deck);
```

Now the dealer will receive it's hand:

```
    dealers_hand := [];
    dealers_hand := deal_card(dealers_hand, deck);
    dealers_hand := deal_card(dealers_hand, deck);

#printf("My hand: ");
#print(my_hand);
#print(dealers_hand);
```

Now do player's turn...

```
    if(strategy = 1) then my_hand :=
players_turn_simple(my_hand, dealers_hand); end if;
    if(strategy = 2) then my_hand :=
players_turn_basic(my_hand, dealers_hand);

    if (split = 1) then
          my_hand2 := my_hand;
          my_hand := players_turn_basic(my_hand,
dealers_hand);
          my_hand2 := players_turn_basic(my_hand2,
dealers_hand);
       end if;
    end if;
```

Now do dealers turn...

```
    dealers_hand := dealers_turn(dealers_hand);
```

debugging information

```
#printf("My hand: ");
#print(my_hand);
#if(split = 1) then printf("Splitting: "); print(my_hand2);
end if;
#print(dealers_hand);
```

Now check who has the better hand

```
if(split = 1) then return (winner(my_hand, dealers_hand) +
winner(my_hand2, dealers_hand)); end if;
return(winner(my_hand,dealers_hand) * double);

end proc:
```

Blackjack - Start of Analysis

The goal of this program is to determine the effectiveness of using basic strategy in the card game, Blackjack. Blackjack seems like a fairly simple game at first, but optimum strategies for the game can get very complicated. 'Basic' strategy is the term used to mean that the player will make the best possible decision assuming the player does not take into account what cards have already been played. I will not cover card counting strategies here, because these strategies require special card counting skills (which isn't practical for the average player) and will only slightly increase the players advantage (although sometimes increase it enough so that it is in the players favor).

Here is a short description of how Blackjack is played. The object of the game is to try get your cards to add up to 21 but without going over 21. The card values are as follows: all face card are worth 10, an ace is worth 1 or 11 (whatever you choose), and number cards are worth their

number. Each player and the dealer receives 1 card face up and 1 card face down. The dealer always plays last. Starting with the first player, each player decides whether or not he/she should get another card ('hit') or stop receiving a card ('stand'). A player can keep hitting until he/she stands at which point the player's turn is over. Each player is only competing against the dealer, so it doesn't matter what the other players have. In fact, in my simulation I assume there is 1 dealer and 1 player. The dealer's strategy is known, and is usually something similar to 'stand on soft 17'. Which means the dealer will hit when the total of his/her entire hand is anything lower than 17. If the dealer's strategy is 'stand on hard 17' that means if the dealer's hand total is 17 but with an ace valued as 11 (such as having a 6 and an ace) then the dealer will continue to hit, otherwise it is the same as before and the dealer will hit on anything less than 17. There are additional rules which help the player. The player can decide to double his/her bet before he/she hits, at which point the player will only get 1 more card (no more hitting is allowed). Also if you are dealt a pair (both cards the same but different suits) then you can split the hands into 2 hands and double your bet (1 bet for

each hand). For example if you are dealt a pair of eights, you always want to split this into 2 hands each hand starts with 1 card (the eight) and you play each hand seperatly. Most casinos do not allow you to double after you split. For simplification of my program I don't allow splitting after the player has already split. This is a rare occurence and shouldn't affect my results.

In order to determine the effectiveness of basic strategy used in the game of Blackjack I decided to do this I run many simulations of playing Blackjack with basic strategy and with a simple strategy. In fact, I've decided to run each strategy a total of 10,000 times in blocks of 100. I needed to run the simulation many times to get fairly accurate results, but I couldn't run too many times or my program would take too long to calculate the results. I have executed my Maple worksheet many times and I seem to get pretty similar results which means 10,000 times in blocks of 100 seems to work out. This maple worksheet takes about 50 seconds to execute on my Athlon XP 1700+ with 256 MBs of memory.

.

Strategy 1, always hit (get another card) if you have less than 17:

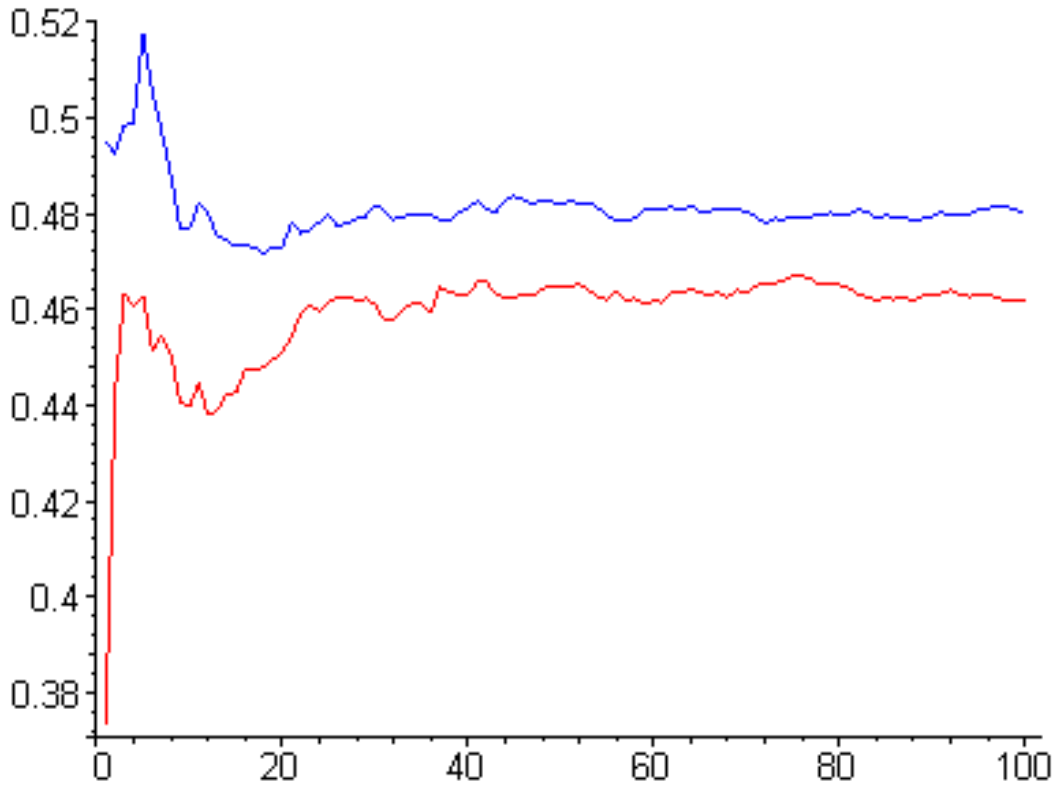
```
> win_percentage_list1 := [seq(i, i = 1..100)]:
```

```
> wins := 0: ties := 0: losses := 0:
> strategy := 1:
> for j to 100 do
  for i to 100 do
   result := blackjack(strategy):
   if (result>=1) then wins := wins + result; end if;
   if (result=0) then ties := ties + 1; end if;
   if (result<=-1) then losses := losses - result; end if;
  end do:
win_percentage_list1[j] := wins/(wins+losses):
end do:
```

Strategy 2, basic strategy. Determines the best action to take depending on your cards and the dealers upturned card.

```
> win_percentage_list2 := [seq(i, i = 1..100)]:
> wins:= 0: ties := 0: losses := 0:
> strategy := 2:
> for j to 100 do
  for i to 100 do
   result := blackjack(strategy):
   if (result>=1) then wins := wins + result; end if;
   if (result=0) then ties := ties + 1; end if;
   if (result<=-1) then losses := losses - result; end if;
end do:
win_percentage_list2[j] := wins/(wins+losses):
end do:
> with(plots):
Warning, the name changecoords has been redefined

> L := listplot(win_percentage_list1, color=red):
> L2 := listplot(win_percentage_list2, color=blue):
> display({L, L2});
```



The blue line represents Basic strategy and the red line represents Simple strategy

You can see that the majority of the time using basic strategy give a better wins/(wins + losses) average. The fact that the graph levels off is a good sign, it shows that the winning percentage is fairly accurate. Next I will calculate after every 100 game block whether or not we come out better by using basic strategy rather than simple strategy:

```
> total:= 0:
> for i to 100 do
if(win_percentage_list2[i] > win_percentage_list1[i]) then
total := total + 1; end if;
end do;
>  printf("%f%%", evalf(total/100) * 100);
100.000000%
```

This is a high percentage, but it still doesn't tell us how much less we would be losing money over a certain amount of time. If I take the average of each of the 100 blocks of games played and computer the total basic average - the total simple average, I should get a positive value that represents the overall gain in win average.

```
> total_avg1 := evalf(win_percentage_list1[100]):
> total_avg2 := evalf(win_percentage_list2[100]):
> avg_diff := total_avg2 - total_avg1:
> printf("%f%%", evalf(avg_diff) * 100);
1.778842%
```

This means that if I played blackjack 100 times that I would win on average:

```
round(avg_diff * 100);
```

2

more times by applying basic strategy over simple strategy.

If you were to play blackjack for 2 hours and bet $5 every 2 minutes, for a total of $300, then if you used simple strategy on average you'd be down: **(.5-total_avg1)*300;**

11.3930348

dollars.
but if you were to use basic strategy you'd be down: **(.5-total_avg2)*300;**

6.0565074

Since my program works by being a simulation, my results vary every time I run it. Basic strategy consistantly comes out ahead of simple strategy. After running my program many times the majority of time basic strategy comes out with losing only about half as much money as simple strategy.
>

Conclusion: While learning basic strategy and applying it helps you win more often, keep in mind that you are losing either way. Like all casino games, Blackjack should be played for fun not for money. It is possible to get the odds in your favor by learning card counting techniques in Blackjack, but professional gamblers who use these techniques are quickly identified by casinos and forced out of the casino. Part of the fun of playing Blackjack has to do with the somewhat complex strategy; people enjoy learning how to get better at something. For example, after learning the best strategy for tic-tac-toe, the game no longer becomes enjoyable to play because you always now the best move. While a game like chess has such complex strategy people can continuosly enjoy learning how to get better at it.