

Homework #6

ENGR 0012 – Spring 2007

Due: Thursday, February 22

For this assignment, you are to create an arcade program, which gives the program user the option to play 3 games, you must pick 2 of the following : A) Rock-Paper-Scissors, B) Blackjack, or C) Craps, and the third game can be anything you want.

You can use the programs that you have written during the past week in class. The directions for these programs are included at the end of this assignment document. You will need to change the main program for each of these games so that they are functions with no inputs or outputs that can be called from your main arcade program. The main program should start by displaying a header that explains the arcade, then have a menu open that allows the user to select one of the games. Put a loop in the main program so the user can play a different game when they finish one of the other games.

In each game have a header in the front that explains the rules and the basic concept of the game. Have the program keep track of the win/lose record or the money won/lost. Make sure you indent the code and have comments.

Each of the games must be a function call from the main. Each instructor has the right to modify the basic game rules, but here are some specifications for each game:

Rules of Craps:

- Craps is a dice game played with 2 die
- The player begins by making first roll:
 - *If the roll is a 7 or 11* → **Player wins**
 - *If the roll is a 2, 3, or 12* → **Player loses**
- Otherwise the player continues rolling:
 - *If after the first roll, the player rolls a 7* → **Player loses**
 - *If the player rolls the same total as the first roll* → **Player wins**
- Player continues rolling until the first roll is repeated or a 7 is rolled.
- Player can start game with any amount of money, and can bet any amount (\leq total) for each game, until out of money. No negative bets. Have the pay off 1 to 1 on the bets.

Basic Program Breakdown:

1. Player bets
2. Simulate a random roll of 2 die
3. Evaluate if player wins or loses after first roll based on rules
4. If neither, generate a second roll

5. Evaluate if player wins or loses
6. Repeat 3 and 4 until player wins or loses
7. Output results
8. Update win/lose history
9. See if player wants to play again
10. Add the aspect of betting to the game by:
 - Asking how much money user will start with at beginning of program
 - Asking how much player wants to bet for each game
 - Verifying that bet amount is not more than what the player has
 - Adjusting the player amount of money based on bet amount and whether they won or lost
 - Make this a function called **bet_amount** with what user has as the function input and what user wants to bet as function output
 - This function should tell the user how much (s)he has, ask how much to bet, and check to make sure user does not bet more than they have.
 - If user wins game, add bet amount to total. If user loses game, subtract bet amount from total.

BLACKJACK

You will need to create a function called *blackjack.m*. This will be a single-player game: player vs. dealer. The dealer will have one “hole card” that is hidden from the player while the player decides whether to hit or stand. We will assume that this is a “multiple-deck” game so that we do not have to keep track of which cards were already played (i.e., a card can appear multiple times in one hand.)

Follow these steps to develop your matlab program to play blackjack:

1. Name your blackjack script something like *blackjack.m*. The first line of this main script (excluding comments) should be:

```
rand('state',sum(100*clock));
```

Note: Your program will use Matlab’s random number generator to deal cards. This command “seeds” the random number generator so that you will get a different sequence of random numbers (i.e. cards dealt) each time you run your program.

2. Implement a loop for repeated execution of your blackjack game.
3. Create a function called *get_card.m* which has no input parameters and one output parameters. *Get_card* returns a random integer value from 1 to 52 (there are 52 different cards). You will call this function to deal cards. Try calling *get_card* a few times to verify that it works.

4. Write a function `get_card_string.m` that will return a character string (e.g. “King of clubs” or “Six of diamonds”) for each of the 52 possible cards. So, the input parameter to this function is a single integer value. It’s up to you to decide how `get_card_string` will map the numbers 1 to 52 to the 52 character strings. I recommend something like this:

Integers 1 – 13 are the A,2,3,4,5,6,7,8,9,10,J,Q,K of hearts respectively
Integers 14 – 26 are the A,2,3,4,5,6,7,8,9,10,J,Q,K of diamonds respectively
Integers 27 – 39 are the A,2,3,4,5,6,7,8,9,10,J,Q,K of spades respectively
Integers 40 – 52 are the A,2,3,4,5,6,7,8,9,10,J,Q,K of clubs respectively

Another way to organize the mapping would be:

Integers 1 – 4 are the ace of hearts, diamonds, spades, and clubs, respectively
Integers 5 – 8 are the two of hearts, diamonds, spades, and clubs, respectively
Etc.,etc.,etc.

5. In the main script, deal two cards to the dealer (i.e. call `get_card` twice and store the two integers in a vector called “dealer_cards”). Call `get_card_string` to display one of the dealer cards (the other “hole card” remains hidden from the player) in the command window something like this:

`The dealer has the Ace of Hearts showing.`

6. Similarly, deal two cards to the player (i.e. store the two integers in a vector called “player_cards”) and display both of the player cards in the command window.
7. Prompt the player to hit or stand. Use a while loop to keep dealing cards as long as the player wants to hit. That is, the first hit results in a third number being stored in “player_cards”, and so on. Each time the player is dealt a hit card, display all of the player cards and in the command window.
8. Write a function `add_cards.m` that receives an array of integers (e.g., “player_cards”) as its input parameter and returns the total of the cards. Face cards are worth 10, and an ace is worth 11 unless the resulting total is more than 21, in which case the ace is worth 1. Modify your main script to display the player’s card total each time the player cards are displayed.
9. Add an if-statement to the “hit-or-stand” loop that will break out of the loop if the player has gone bust (i.e. the player’s total exceeds 21).
10. If the player goes bust, display something like “You went bust, you lose” in the command window.
11. If the player has “stood pat” without going bust then display both of the dealer cards and the dealer total in the command window. (This is the first time that the player gets to see the dealer’s “hole card”). Then, automatically loop to “hit” the

dealer with cards as long as the dealer's total is less than 17 and not greater than 21. Each time the dealer gets another card, display all of the dealer cards and the dealer total. If the dealer goes bust, display something like "The dealer went bust, you win". Once the dealer total is between 17 and 21 determine who wins and display a message. If the dealer wins, then display "The dealer wins". And if there is a tie, display "This hand is a draw".

12. Add the aspect of betting to the game by:
 - Asking how much money user will start with at beginning of program
 - Asking how much player wants to bet for each game
 - Verifying that bet amount is not more than what the player has
 - Adjusting the player amount of money based on bet amount and whether they won or lost
 - Make this a function called `bet_amount` with what user has as the function input and what user wants to bet as function output
 - This function should tell the user how much (s)he has, ask how much to bet, and check to make sure user does not bet more than they have.
 - If user wins game, add bet amount to total. If user loses game, subtract bet amount from total.

ROCK-SCISSORS-PAPER

You will need to create a function called `RSP.m` which will be played between the computer and a program user.

Your program must call three (3) functions which do the following:

1. Display the rules of the game.
2. Generate a random number between 1 – 3 which corresponds to rock, scissors or paper, and displays the random selection to the screen.
3. Compare the random selection with a user selection to determine who wins.

Program Requirements:

1. Your main script must contain your name, group number and class in comment lines.
2. Your main script must use comments to detail what you've done.
3. Your program must allow the user the option to play again.
4. Begin the game by asking the user how many times you have to win to be declared the winner. For example, the first person (or computer) to 10 wins is the overall winner.
5. For each game you must display determine what the program user chooses (i.e. paper, scissors, or rock), then what the computer chooses, then display both choices to the screen.
6. For each game you must display if the player won, lost or tied.
7. After each play you must tally the total wins of the computer versus the player. Keep playing until you get to the number of wins required.

- When determining who won the game, use numbers to represent paper, scissors, and rock, NOT strings. Remember, in MATLAB, string comparison can get sticky if the strings are not the same length.

HINTS:

- If you are unable to get a function to run, either comment out the call to the function and/or just assign a possible value in the main program that you expect your function to generate, and continue on to the next part of your program. You can get partial credit if one of your functions doesn't work.
- If you forget how to generate a random number, type "help rand". You can manipulate the random number to be in any desired range by multiplying by the maximum range value, then rounding up.

Sample Program Output:

