

## Homework #6

### ENGR 0012/0112 – Spring 2006

**Due:** Tuesday, February 21

For this assignment, you are to create an arcade program, which gives the program user the option to play 3 games: 1) Rock-Paper-Scissors, 2) Blackjack, and 3) Craps.

You can use the Rock-Paper-Scissors and Blackjack programs that you have written during the past week. 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. For example, the main program for Rock-Paper-Scissors will have to be modified so that the first line is *function RPS (or function [ ]=RPS( ) )*, and the file is saved as *RPS.m*.

For the Craps program, you will need to create a new no-input/no-output function called *craps.m*. The specifications for this program are as follows:

#### 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

#### Basic Program Breakdown:

1. Simulate a random roll of 2 die
2. Evaluate if player wins or loses after first roll based on rules
3. If neither, generate a second roll
4. Evaluate if player wins or loses
5. Repeat 3 and 4 until player wins or loses
6. Output results
7. See if player wants to play again
8. 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

### Programming Specifications:

1. Create craps.m file as a function, and set up while loop to allow user to play craps multiple times.
2. Create function with no inputs and one output which randomly generates two separate numbers between 1 and 6 and returns the sum
3. In the main program, call this function for first roll
4. a. If sum = 7 | sum = 11, gameresult = 1 (win)  
b. If sum = 2 | sum = 3 | sum = 12, gameresult = 2 (lose)  
c. Else gameresult = 0, firstroll = sum
5. Set up while loop to keep rolling until player wins or loses (while gameresult = 0)
  - a. If sum = firstroll, gameresult = 1
  - b. If sum = 7, gameresult = 2
6. Output result:
  - a. If gameresult = 1, print "YOU WIN"
  - b. If gameresult = 2, print "SORRY, YOU LOSE"
7. Ask if user wants to play again.
8. To add betting, at beginning of game, give user a total of \$20 for play
9. Prior to each game, ask user how much they want to bet.
  - 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.
10. If user wins game, add bet amount to total. If user loses game, subtract bet amount from total.
  - Make this a function called **bet\_result** with 3 inputs (total, bet amount, game result) and total as the output
  - This function should adjust total based on result of game, then print out the new total.

### Deliverables:

Turn in electronically a folder containing all .m files.

## **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 the player’s total and the dealer has not gone bust. (That is, the dealer cannot lose unless the dealer goes bust). 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”. If the dealer wins, then display “The dealer wins”. And if there is a tie, display “This hand is a draw”.

## 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. 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.
5. For each game you must display if the player won, lost or tied.
6. After each play you must tally the total wins of the computer versus the player.
7. You must use three functions.
8. You may name the functions whatever you choose.
9. The functions must perform the tasks specified above.
10. You may design the functions in a variety of ways. If you find yourself stumped, you may use the function design guidelines suggested below.
11. Save your program and functions to a floppy disk to turn in (***Make sure to check that you have successfully copied the correct files before turning in your disk!!!!***)

### 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.
- 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.
- 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.
- Don't worry about seeding the random number generator.

### Suggested Function Design Guidelines:

1. Function 1: Display rules of game...
  - This function simply displays text, and thus requires no inputs or outputs

- Suggested first line of this function file: *function func\_name\_1*
2. Function 2: Generate random computer play
- This function must first generate a random number between 1 and 3
  - Then display to the screen what the computer has chosen, i.e. if the number is 1, the computer chose PAPER; if the number is 2 the computer chose SCISSORS; if 3, ROCK.
  - The function should return the random number back to the main program to be used to determine who won the game.
  - This function requires no inputs.
  - Suggested first line of function file: *function var\_out\_name = func\_name\_2*
3. Function 3: Determine who won the game
- This function would require two inputs, one being the number corresponding to the computers random game selection from function 2, the second being the selection the program user which could be determined in the main program using the menu command.
  - In the function, the two numbers would be compared to determine if the player won, lost or tied.
  - The function would return a number to the main program corresponding to a win, loss or tie.
  - The function would thus require two inputs and one output
  - Suggested first line of function file:  
*function var\_out\_name = func\_name\_3(in\_var\_1, in\_var\_2)*

### Sample Program Output:

