James Hinson
CS 162
Section 030

Design 1: Go Hooping

Part 1: Understanding the Problem:

(1 pt) [Rephrase the problem statement here]

Our goal is to develop a program in C++ that simulates a “Moneyball” basketball game for two or more players. The game is played with 2 players, using 7 fixed shooting positions for each player, and is won by getting the most points out of a series of “throws” from each position. There is a special “Money-Ball” rack that contains 5 “Money Balls”, which are all worth 2 points, as opposed to the normal 1 point balls. These money balls are also at the end of every regular rack, which has 4 regular balls and 1 money ball. In addition to these 5 racks, there are 2 additional racks with “Starry Balls”. These balls are worth 3 points each, and are located after the 2nd and 3rd racks. Once the game is finished, the players should be asked if they would like to play another round, and the program should keep running until a player wishes to exit.

To ensure that you understand the problem correctly, answer the following guiding questions:

1. **(2 pts) What are the four possible states for each ball shot, and what is the character notation for each state?**

   The four possible states for each ball shot are: missed (X), point (O), moneyball point (M), and starry ball point (W).

2. **(2 pts) For non-money-ball rack positions, state if (a) to (d) is a valid output from random generation. If it is not valid, explain why:**

   a. X X X X X **Valid**

   b. X O O O O **Not Valid - There should be a money ball at the end of the rack, not a regular ball.**

   c. M O O O O **Not Valid - Possible with random generation, but the money ball should be at the end of the rack, not at the start of it.**

   d. X O X O M **Valid**

3. **(2 pts) For the following ball shooting outcomes, fill in the blanks for each rack score, and the total score:**

   a.
   Rack 1: O O O O M | **6** pts
   Rack 2: O O O O M | **6** pts
   Starry : W | **3** pts
   Rack 3: M M M M M | **10** pts
   Rack 4: O O O O M | **6** pts
   Starry : W | **3** pts
   Rack 5: O O O O M | **6** pts

   Total: **40**

   b.
   Rack 1: X O O O X | **3** pts
   Rack 2: O X O O M | **5** pts
   Starry : W | **3** pts
   Rack 3: O O X O X | **3** pts
   Rack 4: X O X O X | **2** pts
   Starry : W | **3** pts
   Rack 5: M X M M X | **6** pts

   Total: **25**

4. **(1 pt) For each execution, what inputs do you need from the user?**

   For each execution, you need input from the user to determine where the moneyball rack is positioned, and whether or not they want to start playing or play again.
5. **(1 pt) For each execution, what outputs do you need to print?**

   For each execution, you need to output a menu for navigation, the different racks and their unique positions, and the total points alongside who won.

(1 pt) [List assumptions that you made]

Assumptions that I made are that the user knows what a moneyball rack is, how to navigate the menus and how to play the game, and that input cleaning/error handling isn't needed (unless stated in the previous questions).

Part 2: Program Design:
To help you break the problem down into smaller subtasks, answer the following guiding questions:

1. **(1 pt) Without error handling, how would you let the user/player choose their "money-ball rack" position?**

   I would ask the user for an integer representing the position of the rack they want to be the moneyball rack. The range is 1-5, representing the 5 racks available.

2. **(1 pt) For a single ball shot, how would you generate the random number to meet the requirement of "50% chance of making the shot"?**

   I would generate the random number by using *rand() % 2*. As the result of this is binary in nature, a zero equals a miss and a one equals a successful shot.

3. **(1 pt) For one rack position (5 ball shots), regardless of it being money-ball rack or not, how would you simulate the result?**

   I would simulate the result by randomly determining (50/50 chance) whether or not each ball is a miss or not, and then printing out the results using the corresponding symbols.

4. **(1 pt) What needs to be done so that your implementation could generate different results every time you run the program?**

   I need to set up the random percentage generator to have a unique seed every time. I can accomplish this by using a for-loop that iterates over every ball shot, using time(0) and the current position in the loop to generate a unique seed.

5. (1 pt) How would you use your design from question 3 above, and expand it to 5 racks?

I would create another simulation - likely using classes - for all 5 racks, and simulate each rack individually.

6. (4 pts) How many arrays would you use to store the ball shooting result? How many dimensions do your array(s) have? What are the sizes of your array(s)? What is the data type of your array elements?

I would need to hard-code three arrays, but might need more depending on the situation. The first array is to hold the location of the racks, primarily where the user-requested money-ball rack is. I can use classes to create new racks (and therefore arrays) for each player. Each array is 5 elements long, unless it is a Starry Ball, in which case it is only 2 balls long. Each array is 1D, of type char, and will be printed out as characters.

7. (1 pt) How would you print out the ball shooting result for 1 rack? 5 racks? Starry balls?

For 1 rack, I would format & print the results directly based on the contents of the array, as they are already in character format. For 5, I would read each rack and do the same steps for 1 rack. For starry balls, I would read the result from the random generator and then print it in a similar format, using characters to show results.

8. (1 pt) Before the scoring calculation, how would you represent your array in your program, so it can tell the difference of regular ball (1 point), missed (0 point), money ball (2 points), and starry balls (3 points)?

I would represent the array with characters. Using the random number generator and modulus, I can assign an integer one or zero to the appropriate character for a regular ball or a missed ball. Using boolean values for starry ball or money ball, I can similarly assign different characters to each generated integer.

9. (1 pt) How would you incorporate the “money-ball” rack chosen by the user into your program?

I would incorporate the money-ball rack into my program by asking the user where they would like their money-ball rack to be, then creating a new object with all the parameters appropriate for a money-ball rack. I can do this by using a boolean value in the object that is specifically for the money-ball rack.

10. (2 pts) How would you calculate the score for 1 rack? 5 racks? What information needs to be stored in this step to help calculate the total score?

For 1 rack, I would count how many balls weren’t misses, then add up the total based on whether or not the ball is a money ball, regular, or starry. The same applies to 5 racks. The information that needs to be stored is whether or not the rack is a money- or starry-ball rack, which ball is money vs normal, and which rack is the money-ball rack.

11. (1 pt) How would you modify your output functionality so that it contains the information of scoring?

I would modify my output functionality so that it contains the scoring information by appending each line with the total score from the rack it was assigned to. After each rack is printed, I would print the total beneath it.

12. (1 pt) How would you alternate the player, i.e., from Player 1 to Player 2?

Player 1 goes first by selecting the location for their money-ball rack, after which the scoring is calculated along with the successful shots. After player 1 goes, player two would then choose and repeat the cycle for their score.

13. (1 pt) How would your program determine the winner (or a tie game)?

The winner has the most total points between the two players by comparing the final scores. A tie game is decided when each player has an equal number of points.

14. (1 pt) How would you implement the "play again" functionality? What kind of loop are you going to use?

I will implement the "play again" functionality by using a do-while loop.

15. (2 pts) Now, considering error handling features, what changes do you need to make when taking user inputs? How would you validate the input and re-prompt until a valid one is provided?

When taking user inputs, I need to make sure that they fall within the acceptable list of inputs needed to play the game, and that there are no errors or unintended functionality by entering an unintended set of inputs. I would validate the inputs by using an if statement, and would re-prompt using either a while or a do-while loop.

(10 pts) [Use your answer above, create pseudocode (or flowchart) for each of the function that you plan to create]
Tip: in order to modularize your program, let each function handle one thing/task. When decompose a program into functions, try listing the verbs/tasks that are performed to solve the problem.

```
void main_menu() {
        cout << "1. Play game        2. Learn about Moneyball        3. Exit" << "\n";
        cin >> choice;

        if(choice == 1) {

                game_loop();

        } else if(choice == 2) {

                learn();

        } else {

                sys.exit();
        }
}

void game_loop() {

        cout <<
        do {
                play_game();
                main_menu();
        } while (choice == 1);

        sys.exit();
}

void play_game () {

        int money_ball_rack;
        cout << "Where do you want to put the money ball rack (1-5): "
        cin >> money_ball_rack;
        gameplay loop, etc;
}
```

Part 3: Program Testing
To help you consider the possible test cases, answer the following guiding questions:
1. (1 pt) What is an example of good input when prompted for the money-ball rack?

Integers 1 through 5 as those are the valid locations for the money-ball rack.

2. (1 pt) How would you tell/test if the generated ball shooting result is valid?

I would tell by checking that the generated values are random every time, and that the only values being passed or printed after generating are within the parameters assigned. For example, only five 1's and 0's are generated, and only X's, O's, M's, and W's are printed or passed to arrays.

3. (1 pt) How would you tell/test if the scores are correctly calculated?

I would manually calculate the initial scores during testing, and would ensure that each type of ball is assigned the correct score value assigned to it at each step of the program.

4. (1 pt) Are there any inputs that would cause your program to crash? If so, what are they?

The only inputs that could cause the program to crash would be when choosing the money-ball location, playing again/exiting, and in a menu. These can all be fixed using good code cleanup and error handling.

(6 pts) [Create a testing table that has representative good, bad, and edge cases, and their expected outputs]

Conditions	**Good Input**	**Bad Input**	**Edge Case Input**
Choosing money-ball location	Player selects an integer from 1-5 for the money-ball rack location	Player selects a value outside of the available range	Player inputs a non-integer value
Playing again/exiting	Player enters y or n in order to play again or quit	Player enters a character or string other than y/n	Play enters a non-string or char value
Menu navigation	Player enters an integer and program continues as expected	Player enters a different integer value outside of range of options	Player enters a non-integer value
Output	No error	All functions should re-prompt until an acceptable value is introduced	All functions should re-prompt until an acceptable value is introduced