Bucket of Luck

PROJECT DUE
TUESDAY
Mar. 29th, 2022
@ end of class

Welcome to the dice game of the century!

Bucket of Luck is played with three standard dice that are kept in a device shaped somewhat like an hourglass that resembles a wire-frame bird cage and pivots about its centre. The dealer rotates the cage end over end, with the dice landing on the bottom.

Wagers are placed based on possible combinations that can appear on the three dice. See the following table for how the payouts work:

Type	Description	Payout
Single Die Bet	Betting how many of a given roll will appear	1 die, 1 to 1 2 dice, 2 to 1 3 dice, 10 to 1
Over 11	The total of the dice will be 11 or higher	1 to 1
Less than 10	The total of the dice will be 10 or lower	1 to 1
Outside 8-12	The total of the dice will be outside the range of 8 to 12 (inclusive). Meaning less than 8 or greater than 12	1 to 1

You are to create this game. This game should be in a loop that repeats until a total of 5 rounds have been completed, or the money has run out.

- The program should be user friendly and straightforward to use.
- There should be a constantly updating display of the user's wallet.
- The user's wallet should start at $100.
- The values of the three dice being rolled each round should be stored in an array.
- All money should be tracked as integers (not dealing with pennies at all)
- The user should not be able to make bets of zero or less, or to bet more than their wallet
- There are at least two functions that are required. And they must be used as defined below

Function Definitions (mandatory)

```
/* prints out the list of rules and explanations for how the game works */
void rules();

/* returns a random result from 1-6 to simulate rolling a die */
int rollDie();
```

See my example program to test logic and function.
http://dev.emmell.org:8888/

Marking Rubric (/28 points)

- **Program Correctness**: You program should work correctly on all inputs. Meaning, it should satisfy all the requirements listed above and not crash.
- **Readability**: Variables and functions should have meaningful names. Code should be organized into functions/methods where appropriate. There should be an appropriate amount of white space so that the code is readable, and indentation should be consistent.
- **Documentation**: Your code and functions/methods should be appropriately commented. However, not every line should be commented because that makes your code overly busy. Think carefully about where comments are needed.
- **Code Elegance**: There are many ways to write the same functionality into your code, and some of them are needlessly slow or complicated. For example, if you are repeating the same code, it should be inside creating a new method/function or for loop.

Program Correctness	**0 Points**	**5 points**	**10 points**	**15 points**
	Program does not compile, or errors occur on input similar to sample.	Significant details of specification are violated, or the program often exhibits incorrect behavior.	Minor details of the program specification are violated, program functions incorrectly on some inputs.	Program always works correctly and meets the specifications
Readability	**0 points**	**2 points**	**4 points**	**6 points**
	Several major issues that make it difficult to read.	At least one major issue that makes it difficult to read	Minor issues such as inconsistent indentation, variable naming, general organization	Code is clean, understandable, well-organized
Documentation	**0 points**	**1 points**	**2 points**	**3 points**
	No comments	Major lack of comments make it difficult to understand code.	One or two places could benefit from comments, or the code is overly commented	Code is well commented.
Code Elegance	**0 points**	**2 points**	**4 points**	
	Many instances where code could have used easier/faster/better approach.	Code uses a poorly chosen approach in at least one place, for example, hard coding something that could be implemented through a for loop	Code appropriately uses for loops and methods for repeated code, and there is minimal hard-coding.	