Lab 3

**Program Flow**

**Main**

Calls Game in a do-while loop, looping until the user decides to quit.

**Game**

(Blue text was added as I expanded the design)

Constructor:

* Displays menu: Play, or quit. Quit returns to main and ends the program immediately.
* If play is selected, prompt for:
  + Number of rounds (up to 10000)
  + Type of die for each player
  + Number of sides of dice for each players (Must be greater than 2. can be different for both)
* Creates the necessary Die/LoadedDie objects
* Loops the number of rounds
  + Plays one round
  + output the detailed result of each round, including:
    - the side and type of die used for each player
    - the number each player rolls
    - the score result
* display the final score and the final winner of the game
* Prompt to play again

Private:

Die for player 1 - pointer

Die for player 2 - pointer

Counter for player 1 score

Counter for player 2 score

MainMenu()

GameSetup()

PlayGame()

**Die**

Protected:

Integer number of sides

Bool loaded or not

Public:

Roll die – returns a random number between 1 and numSides

get sides – returns number of sides

get loaded – returns loaded or not

**LoadedDie**

Public:

Roll die – returns a random number between 2 and numSides

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test case** | **Input Values** | **Affected functions** | **Expected outcomes** | **Observed outcomes** |
| Negative input | Input < 0 | mainMenu()  game setup functions | Reprompt user for positive input | Reprompt user for positive input |
| Input is 0 | Input == 0 | mainMenu()  game setup functions | Reprompt user for positive input | Reprompt user for positive input |
| Input is too high | Rounds > 10000 | mainMenu()  game setup functions | Reprompt user for smaller input | Reprompt user for smaller input |
| User enters float | Input = “1.1” | mainMenu()  game setup functions | Reprompt user for correct input | Reprompt user for correct input |
| User enters letters after numbers | Input = “1a” | mainMenu()  game setup functions | Reprompt user for correct input | Reprompt user for correct input |
| User enters spaces between numbers | Input = “1 1” | mainMenu()  game setup functions | Reprompt user for correct input | Reprompt user for correct input |
| Loaded and unloaded die are created with correct number of sides | Loaded, 100 sides; Unloaded 50 sides | game setup functions | A loaded d100 and a regular d50 are created | A loaded d100 and a regular d50 are created and produce correct roll values |
| Loaded die returns higher than regular | - | rollDie() | Loaded die returns higher than regular | Loaded die returns the same as regular die; solved by making function virtual |
| Loaded die wins every time rounds > 3000ish, even with a d100 | - | - | Loaded die wins every time rounds > 3000ish | Loaded die wins consistently even with a d100 when rounds > 3000ish |
| Loaded die only rolls 2 on a d2 | - | rollDie() | Ant teleports to opposite side | Loaded die only rolls 2 on a d2 |

**Reflection**

This program turned out to be more difficult than I initially anticipated.

I needed to add:

* Die: flag for whether loaded or not, in order to accurately report whether the die was loaded or not in the score printout
* Game: Die to Die pointer. I initially tried instantiating die for each player in the Game constructor and then using setters to set die sides/loaded status, but it got confusing quite quickly and ended up being easier to just use pointers and instantiate the right type of die as it was needed.

I had trouble getting the overloaded function to work with a pointer: die was getting created, and the getLoaded flag was correctly reporting whether it was loaded or not, but the loaded die were still using the regular die roll function. I eventually figured out to “override” a base class function with a derived class’s function, it needs to be declared as virtual.