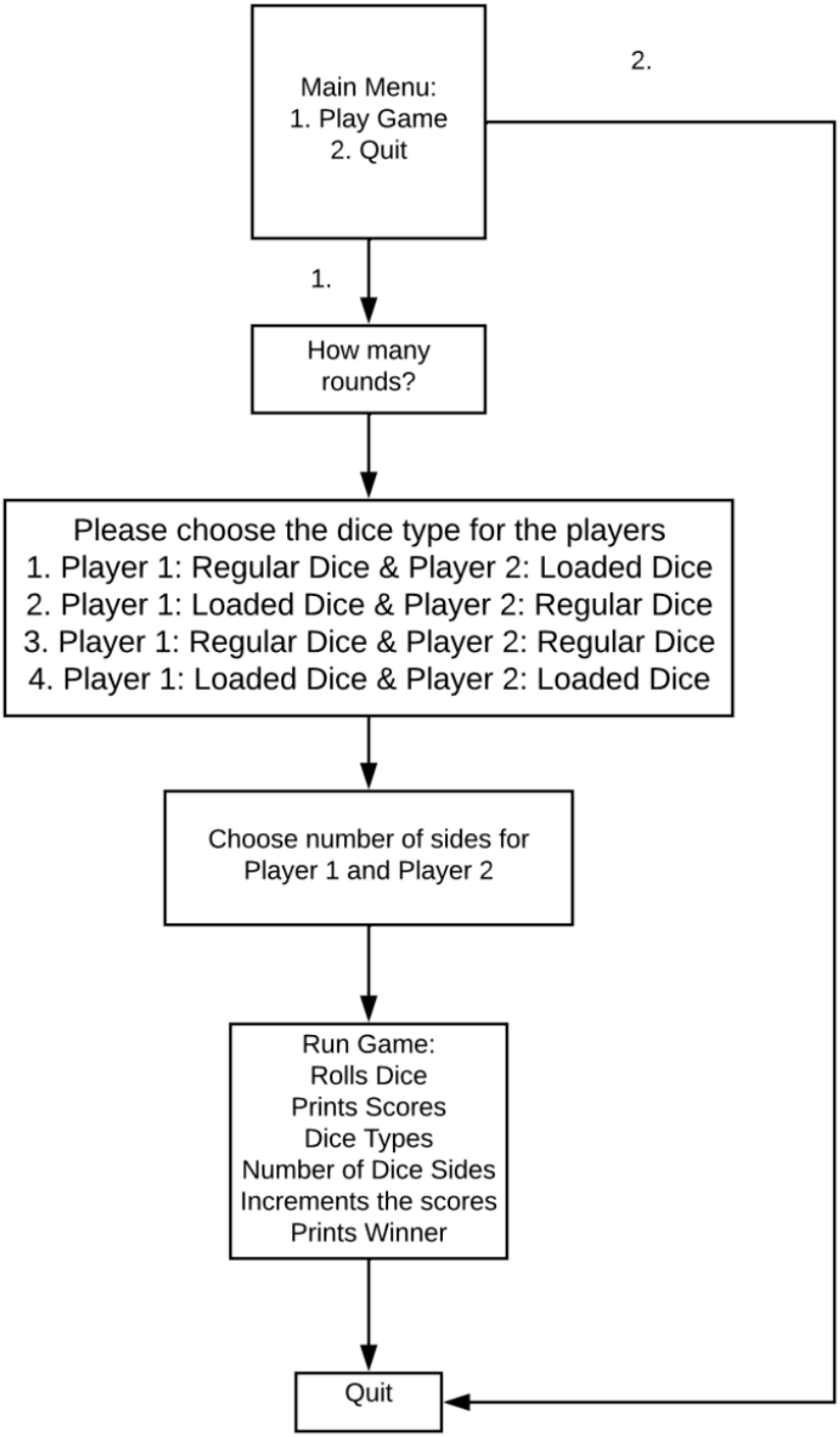
Devin Gendron
Lab 3 – War Game with Dice Design

Flowchart:



PseudoCode:

Main()

```
//set randomizer seed
//run game menu
//enter switch function using menu input
        //case 1 : play game
                //game rounds function
                //die type function
                //enter switch function to set die types in game and execute game
                        //case 1 : die and loaded die
                                //allocate objects
                                //set the players
                                //set the game
                                //run the game
                        //case 2 : loaded die and die
                                //allocate objects
                                //set the players
                                //set the game
                                //run the game
                        //case 3 : die and die
                                //allocate objects
                                //set the players
                                //set the game
                                //run the game
                        // case 4 :  loaded die and loaded die
                                //allocate objects
                                //set the players
                                //set the game
                                //run the game
                //switch ends
        //case 2 : quit game
```

gameMenu()

```
//print statements
//use inputvalidation func to receive proper return
//return value
```

promptPlayer()

```
//using inputval(); code, retrieve rounds
//using inputval(); code, retrieve die type
//using inputval(); code, retrieve die sides
```

dieClass()

```
//private: holds score
//protected: holds number of sides
//public:
    //constructor

    //getters for all private members

    //setters:
        //setter for n
        //setter for score
    //dice roll function
```

loadedDieClass()

```
//private: holds score
//public:
    //constructor

    //getters for all private members

    //setters:
        //setter for score
    //loaded dice roll function
```

loadedDieClass()

```
//private: menudecision, rounds, dice types, dice side numbers, string for winner, and
players

//public:
    //constructor
    //getters for all private members
    //setters for setting players to their die types
    //increment score function
    //run game function
    //functions to retrieve prompts
    //function to set game
    //function to determine winner and quit game
```

incrementScore()

//use conditionals to determine who won round, and increment their score

setGame()

//set player to dice time using setter function

roundWinner()

//uses conditionals to determine who won the roll or if there was a draw

finalWinner()

//use conditionals to determine who won the game

runGame()

//displays game data (rounds, die types, rolls, and scores)
//loop used to run through game
//final scores displayed
//winner displayed

gameQuit()

//quits game

Test Plan:

gameMenu();

Test Case	Expected Outcomes	Observed Outcomes
Input letters	Inputvalidation(); returns "Incorrect entry…". Restarts loop.	Incorrect entry statement printed, loops to restart input request.
Input symbols or space	Inputvalidation(); returns "Incorrect entry…". Restarts loop.	Incorrect entry statement printed, loops to restart input request.
Input too low	badEntry bool sets to true, restarts loop until correct input.	badEntry statement printed, loops to restart input request.
Input too high	badEntry bool sets to true, restarts loop until correct input.	badEntry statement printed, loops to restart input request.
Empty input	badEntry bool sets to true, restarts loop until correct input.	badEntry statement printed, loops to restart input request.
Correct Input	Counter for exit increments, print correct entry statement. Program continues.	Returns value to main to continue with program

game.rounds();

Test Case	Expected Outcomes	Observed Outcomes
Input letters	Inputvalidation(); returns "Incorrect entry…". Restarts loop.	Incorrect entry statement printed, loops to restart input request.
Input symbols or space	Inputvalidation(); returns "Incorrect entry…". Restarts loop.	Incorrect entry statement printed, loops to restart input request.
Input too low	badEntry bool sets to true, restarts loop until correct input.	badEntry statement printed, loops to restart input request.
Input too high	badEntry bool sets to true, restarts loop until correct input.	badEntry statement printed, loops to restart input request.
Empty input	badEntry bool sets to true, restarts loop until correct input.	badEntry statement printed, loops to restart input request.
Correct Input	Counter for exit increments, print correct entry statement. Program continues.	Returns value to main to continue with program

inputValidation();

Test Case	Expected Outcomes	Observed Outcomes
Input Letters	Incorrect entry counter incremented. Restarts loop.	Restarts loop for proper entry
Input symbols	Incorrect entry counter incremented. Restarts loop.	Restarts loop for proper entry
Input too low	badEntry bool sets to true, restarts loop until correct input.	Restarts loop for proper entry
Input too high	badEntry bool sets to true, restarts loop until correct input.	Restarts loop for proper entry
Input empty string	badEntry bool sets to true, restarts loop until correct input.	Restarts loop for proper entry
Correct entry	Counter for exit increments, print correct entry statement. Program continues.	Returns value to main to continue with program

dieType();

Test Case	Expected Outcomes	Observed Outcomes
Input letters	Incorrect entry counter incremented. Restarts loop.	Incorrect entry statement printed, loops to restart input request.
Input symbols or space	Incorrect entry counter incremented. Restarts loop.	Incorrect entry statement printed, loops to restart input request.
Input too low	badEntry bool sets to true, restarts loop until correct input.	badEntry statement printed, loops to restart input request.
Input too high	badEntry bool sets to true, restarts loop until correct input.	badEntry statement printed, loops to restart input request.
Empty input	badEntry bool sets to true, restarts loop until correct input.	badEntry statement printed, loops to restart input request.
Correct Input	Counter for exit increments, print correct entry statement. Program continues.	Returns value to main to continue with program

playerPrompt();

Test Case	Expected Outcomes	Observed Outcomes
Input letters	Incorrect entry counter incremented. Restarts loop.	Incorrect entry statement printed, loops to restart input request.
Input symbols or space	Incorrect entry counter incremented. Restarts loop.	Incorrect entry statement printed, loops to restart input request.
Input too low	badEntry bool sets to true, restarts loop until correct input.	badEntry statement printed, loops to restart input request.
Input too high	badEntry bool sets to true, restarts loop until correct input.	badEntry statement printed, loops to restart input request.
Empty input	badEntry bool sets to true, restarts loop until correct input.	badEntry statement printed, loops to restart input request.
Correct Input	Counter for exit increments, print correct entry statement.	Returns value to main to continue with program

rounds();

Test Case	Expected Outcomes	Observed Outcomes
Input letters	Inputvalidation(); returns “Incorrect entry…”	Incorrect entry statement printed, loops to restart input request.
Input symbols or space	Inputvalidation(); returns “Incorrect entry…”	Incorrect entry statement printed, loops to restart input request.
Input too low	badEntry bool sets to true, restarts loop until correct input.	badEntry statement printed, loops to restart input request.
Input too high	badEntry bool sets to true, restarts loop until correct input.	badEntry statement printed, loops to restart input request.
Empty input	badEntry bool sets to true, restarts loop until correct input.	badEntry statement printed, loops to restart input request.
Correct Input	Counter for exit increments, print correct entry statement.	Returns value to main to continue with program

dieType();

Test Case	Expected Outcomes	Observed Outcomes
Input letters	Inputvalidation(); returns "Incorrect entry…"	Incorrect entry statement printed, loops to restart input request.
Input symbols or space	Inputvalidation(); returns "Incorrect entry…"	Incorrect entry statement printed, loops to restart input request.
Input too low	badEntry bool sets to true, restarts loop until correct input.	badEntry statement printed, loops to restart input request.
Input too high	badEntry bool sets to true, restarts loop until correct input.	badEntry statement printed, loops to restart input request.
Empty input	badEntry bool sets to true, restarts loop until correct input.	badEntry statement printed, loops to restart input request.
Correct Input	Counter for exit increments, print correct entry statement.	Returns value to main to continue with program

Testing Dice Rolls

Test Case	Expected Outcomes	Observed Outcomes	Data Player 1	Data Player 2
Regular Die vs Loaded Die With same sizes	Correct printing of statements and allocation of scores with proper roll functions	Proper roll functions made and scores properly incremented.	2, 2, 5, 5, 5, 1, 4, 3, 6, 3, 4, 4, 3, 5, 6, 1, 1, 1, 3, 4	6, 5, 5, 5, 2, 6, 5, 5, 5, 6, 4, 5, 4, 5, 4, 5, 4, 1, 5, 4
Regular Die vs Regular Die With same sizes	Correct printing of statements and allocation of scores with proper roll functions	Proper roll functions made and scores properly incremented.	2, 4, 5, 5, 1, 5, 3, 5, 2, 5, 3, 2, 1, 5, 1, 4, 4, 5, 2, 3	6, 6, 4, 3 ,1, 1, 3, 2, 6, 1, 3, 2, 3, 1, 1, 2, 5, 3, 1, 1
Loaded Die vs Loaded Die With same sizes	Correct printing of statements and allocation of scores with proper roll functions	Proper roll functions made and scores properly incremented.	6, 6, 4, 6, 4, 6, 4, 6, 6, 2, 6, 3, 5, 5, 1, 6, 1, 6, 5, 1	4, 5, 4, 4, 6, 6, 4, 2, 6, 6, 5, 6, 6, 6, 4, 4, 6, 5, 1, 5
Regular Die(6) vs Loaded Die(12) With different sizes	Correct printing of statements and allocation of scores with proper roll functions	Proper roll functions made and scores properly incremented.	4, 6, 4, 4, 1, 2, 6, 6, 1, 2, 2, 5, 6, 3, 2, 1, 6, 4, 2, 4	2, 7, 7, 7, 11, 12, 11, 7, 9, 12, 10, 3, 6, 9, 2, 8, 9, 2, 12, 12,
Regular Die(6) vs Regular Die(12) With different sizes	Correct printing of statements and allocation of scores with proper roll functions	Proper roll functions made and scores properly incremented.	5, 1, 1, 3, 3, 6, 4, 1, 6, 3, 6, 6, 4, 5, 4, 6, 5, 5, 3, 1	2, 12, 12, 10, 10, 9, 9, 9, 9, 3, 11, 10, 3, 4, 3, 12, 10, 9, 1, 9
Loaded Die(6) vs Loaded Die(12) With different sizes	Correct printing of statements and allocation of scores with proper roll functions	Proper roll functions made and scores properly incremented.	6, 6, 5, 5, 7, 4, 6, 2, 6, 6, 4, 5, 4, 4, 5, 5, 7, 3, 8, 7, 9	1, 12, 12, 7, 9, 10, 10, 12, 12, 8, 8, 10, 8, 2, 4, 7, 3, 8, 7, 9

Reflection:

My designs for my programs have steadily been becoming cleaner (in my opinion) and much easier to understand. I've shifted to using switch statements more often in my menus/main, and I've found that the functionality has increased and the difficulty of constructing the menus has gone down when coupled with my prompt and input validation functions. Now that I've figured out a method to better structure my code, the sequential processing has also been much easier. This leads to my development in constructing classes.

I've always understood how classes worked and how to construct them, but I've always had some issues implementing pointers for them – especially when multiple files are involved. For this reason, I spent a lot of time in my notebook instead of the computer trying to plan my design and have a concrete draft before I started typing it. As I expected, I ran into some difficulty with pointers, but managed to organize myself and get the program compiling properly! However, that wasn't the last obstacle.

My next hurdle was class inheritance. Inheriting classes is simple after spending some time reading up on it, however what it leads to was where I hit my first wall on this lab. Object slicing is what really caught my off guard. My program was running flawlessly, except that it would not use my loaded die's roll function, thus I wasn't getting the proper roll's when a loaded dice was rolled. I then turned my way to researching possible avenues through which I could correct this seemingly small, but difficult error.

After spending some time on the internet, I attempted passing my Die class by reference when used in my run game function. However, this didn't lead to a positive result. I then saw online that those with similar issues, (when passing their classes to a vector to hold them), weren't using a pointer in their class that held the passed class. I found this similar to my code. In my gameClass.cpp file, the variable that held my Die objects (Player 1 and Player 2) was not a pointer. Hoping this would fix my object slicing problem, I made changes to my code and hoped for the best. I managed to get my code working, but it still was not using my loaded roll function. After attending office hours, I learned that when creating a dynamic object, it did not need to be passed by reference since the object persists throughout the program. By adjusting my pointers after removing the ampersand, my program worked perfectly.

Finally, after object slicing was defeated, I was able to run test tables on my program. Specifically, on how my rolls were returned based on the die type and the side number. As can be seen in the table, all regular rolls were rolled properly and all loaded rolls were rolled with a bias towards a higher roll. I had a lot of fun on this assignment, because I'm not the best at class implementation so I took that as a challenge. As I stated earlier, I spent a lot of time on the drawing board and I really think that paid off. I currently have my entire Zoo Tycoon code

planned in my notebook and I’m excited to take what I’ve learned in this lab and transfer it over to the project.