# Lab 3 Design + Reflection

## Program Design

### War Game Requirement

### Setting

This is a 2-player game. It is played through dice.

### Rule for scoring

The player who rolls higher number gets one point. If both players roll the same number, it is considered a draw and no one gets a point.

### Dice Specification

There are two kinds of dice:

- normal die, represented by Die class.
- loaded die, represented by the LoadedDie class.

Input Validation: I will recycle my InputValidation class used in Project 1. It has an overloaded function called getInt(). I chose to make InputValidation its own class so that it could be used in future programs. getInt() receives input from the user, converts it into a stringstream, which then checks to see if there is anything beyond an integer that is left over. If there is, it makes the user try to input an integer again. getInt(int min, int max) serves the same function as getInt() but also runs another validation checking input to see if it lies between min and max. getInt(min, max) will be useful for specifying minimum and maximum values for the following inputs:

- How many rounds will be played
- The type of die for each player menu
- The number of sides for dice of both players

I am limiting the number of rounds played to 10 and the number of sides for dice of both players to 20. For the type of die for each player menu, there will only be 2 choices (1 & 2) that the getInt() will limit the user to.

Menu: I will recycle my Menu class used in Project 1 to display menu options, get input from the user, and store that input as a user choice variable. I chose to make Menu its own class so that it could be used in future programs. It references my Input Validation class to ensure the user is entering a valid input from the menu. There are functions to add an option to the menu, print the menu options, ask the user for his/her choice, and return the user choice.

The menu will provide 2 options as specified in a start menu: “Play Game” and “Exit Game”.

There will also be a 2-option menu as specified in order to get the type of die for each player. Those 2 options are: “Regular Die” and “Loaded Die”.

Both menus will be initialized in the Game class.

Die: Die class will have a member integer variable called numSides in order to keep track of the number of sides of each die object. It will also have 2 member functions called getSides() and getRandomInt(). getSides() will return numSides for use in the getRandomInt() function. getRandomInt() returns a random integer between 1 and the number of sides that the die has in order to represent the roll of a die.

LoadedDie: LoadedDie class inherit the behavior and elements of Die class. However, for the die rolling member function, getLoadedRandomInt(), it will randomly add a zero or one to the die roll except for the case when the roll is equal to the maximum number of sides of the loadedDie object. The result will be a loadedDie that has a higher average output for its die rolls. I chose to pick add a 0 or 1 randomly because I still wanted it to be possible, although less likely, to roll the minimum number of sides on the loadedDie.

Game: Game class implements the dice-rolling war game. It first creates a Start menu using a createStartMenu() function that includes the following options: “Play game” or “Exit game”. The Start menu will be displayed with a separate startMenu() function. The main engine for the game will come from a function called playOrQuit() which runs the Dice War game if the user has selected “Play game” from the Start menu and quits the game if the user has selected “Exit game”.

Assuming the user has selected “Play game” a function called getSettings() will ensue that asks the user for the following information:

- How many rounds will be played
- The type of die for each player
- The number of sides for dice of both players

These choices will be stored in the following member variables: rounds, p1DieChoice, p2DieChoice, p1Sides, and p2Sides. As mentioned earlier, the rounds and number of sides will utilize the InputValidation class getInt(min,max) function. A separate menu object for the type of die for each player must be created as well, which will be called dieMenu.

Once all the settings have been stored from the user, another meaty function called getResults() will be called in the playOrQuit() class which rolls the dice as per the user specifications, displays the results of each round, and ultimately displays the winner of the game.

## Reflection

I am new to inheritance and struggled with how to implement the LoadedDie rolls. I initially thought of creating an overloaded getRandomInt() function, but decided the name would be a bit of a misnomer and created a new function altogether called getLoadedRandomInt(). Part of this was having a more descriptive name, but the other part was just struggling how to create the overloaded function in a derived class. It took me a bit to figure out how to implement the getLoadedRandomInt() so that the die would have a higher average output. I went down the rabbit hole of using a weighted distributions, and actually got pretty close. After I walked away from the program for a night, I had the much simpler idea

of randomly adding a 0 or 1 to the die roll except in the maximum case. I decided to stick to this because I was spending too much time on a more complicated topic of distributions that the project did not require.

I think my biggest struggle on this program was creating either a LoadedDie or Die based on the user's inputs. My initial thought was that I would have to use pointers, and I still think I was probably right. However, I couldn't wrap my mind around how to create pointers for 2 different classes and then the possibility of the user wanting both the die to be the same class. I opted for the most likely more inefficient approach of creating the classes in if statements based on the user's choice, storing the die roll information in member variables, and then the Die/LoadeDie classes being destroyed when the if statements went out of scope. I am fully aware that there is probably a much more efficient solution out there, but it did not come to me, and I was ready to get the assignment completed. I think the important thing is that I satisfied all of the requirements on this program, albeit leaving room to improve the efficiency on the table.

## Test Tables

### For InputValidation Class – This class was already tested in Project 1

| Test Case | Input Values | Driver Functions | Expected Outcomes | Observed Outcomes |
|---|---|---|---|---|
| input too low or high | input < 2<br>input > 200 | getInt(2, 200) | You must select an integer equal to or between 2 and 200. | You must select an integer equal to or between 2 and 200. |
| input in correct range | 2 < input < 200 | getInt(2, 200) | Returns input | Returns input |
| input extreme low | input = 2 | getInt(2, 200) | Returns 2 | Returns 2 |
| input extreme high | input = 200 | getInt(2, 200) | Returns 200 | Returns 200 |
| input not an integer | input = a | getInt(2, 200) | You must enter an integer value. | You must enter an integer value. |
| input not an integer | input = 123abc | getInt(2, 200) | You must enter an integer value. | You must enter an integer value. |
| input not an integer | input = enter key | getInt(2, 200) | You must enter an integer value. | You must enter an integer value. |
| input not an integer | input = asdf 123 | getInt(2, 200) | You must enter an integer value. | You must enter an integer value. |

For Menu Class - This class was already tested in Project 1

| Test Case | Input Values | Driver Functions | Expected Outcomes | Observed Outcomes |
|---|---|---|---|---|
| input option string | “Start Langton\’s Ant simulation” | addOption(string)<br>printOptions()<br>main() | 1. Start Langton’s Ant simulation | 1. Start Langton’s Ant simulation |
| input option string | “Quit” | addOption(string)<br>printOptions()<br>main() | 1. Start Langton’s Ant simulation<br>2. Quit | 1. Start Langton’s Ant simulation<br>2. Quit |
| input too low or high | input < 1<br>input > 2 | getInt(1, 2)<br>promptUserChoice()<br>main() | You must select an integer equal to or between 1 and 2. | You must select an integer equal to or between 1 and 2. |
| input not an integer | input = a | getInt(1, 2)<br>promptUserChoice()<br>main() | You must enter an integer value. | You must enter an integer value. |
| input valid | input = 1<br>input = 2 | getInt(1, 2)<br>promptUserChoice()<br>getUserChoice(); | returns 1<br>returns 2 | returns 1<br>returns 2 |

For Game/Die/LoadedDie Class

| Test Case | Input Values | Driver Functions | Expected Outcomes | Observed Outcomes |
|---|---|---|---|---|
| number of rounds not between 1 and 10 | rounds < 1<br>rounds > 10 | getInt(min, max)<br>main() | You must select an integer equal to or between 1 and 10. | You must select an integer equal to or between 1 and 10. |
| number of rounds between 1 and 10 | rounds = 5<br>rounds = 10<br>rounds = 1 | getInt(min, max)<br>playOrQuit()<br>getResults()<br>main() | Dice war game displays results for 5 rounds.<br>Dice war game displays results for 10 rounds.<br>Dice war game displays results for 1 rounds. | Dice war game displays results for 5 rounds.<br>Dice war game displays results for 10 rounds.<br>Dice war game displays results for 1 rounds. |
| Players have equal sided Die/LoadedDie | numSides = 2<br>numSides = 10<br>numSides = 20 | getRandomInt()<br>playOrQuit()<br>getResults()<br>main() | Dice war game displays die rolls 1 or 2.<br>Dice war game displays die rolls between 1 and 10.<br>Dice war game displays die rolls between 1 and 20. | Dice war game displays die rolls 1 or 2.<br>Dice war game displays die rolls between 1 and 10.<br>Dice war game displays die rolls between 1 and 20. |
| Player 1 has less sides on Die/LoadedDie than Player 2 | P1 numSides = 2<br>P2 numSides = 20 | getRandomInt()<br>playOrQuit()<br>getResults()<br>main() | P1 only rolls 1 or 2 whiles P2 rolls 1 through 20. | P1 only rolls 1 or 2 whiles P2 rolls 1 through 20. |
| Player 2 has less sides on Die/LoadedDie than Player 1 | P2 numSides = 2<br>P1 numSides = 20 | getRandomInt()<br>playOrQuit()<br>getResults()<br>main() | P2 only rolls 1 or 2 whiles P1 rolls 1 through 20. | P2 only rolls 1 or 2 whiles P1 rolls 1 through 20. |
| Player 1 & 2 have regular dice | 2 Die objects | getRandomInt()<br>playOrQuit()<br>getResults()<br>main() | About equal rates wins/losses over numerous iterations. | About equal rates wins/losses over numerous iterations. |
| Player 1 has a regular die & Player 2 has a loaded die | P1 Die object<br>P2 LoadedDie object | getRandomInt()<br>getLoadedRandomInt()<br>getResults()<br>main() | P2 should have more wins than P1 over numerous iterations. | P2 should have more wins than P1 over numerous iterations. |
| Player 2 has a regular die & | P2 Die object<br>P1 LoadedDie object | getRandomInt()<br>getLoadedRandomInt() | P1 should have more wins than P2 | P1 should have more wins than P2 |

| | | | | |
|---|---|---|---|---|
| Player 1 has a loaded die | | getResults()<br>main() | over numerous iterations. | over numerous iterations. |
| Score count is accurate after each round | p1Score = 0<br>p2Score = 0<br>rounds = 1<br>rounds = 10 | getRandomInt()<br>getLoadedRandomInt()<br>getResults()<br>whoWonRound()<br>main() | After each round score goes up by the appropriate amount on the appropriate player, including the case of a draw. | After each round score goes up by the appropriate amount on the appropriate player, including the case of a draw. |
| Score count at the end of the game is accurate and correct player wins. | p1Score = 0<br>p2Score = 0<br>rounds = 1<br>rounds = 10 | getRandomInt()<br>getLoadedRandomInt()<br>getResults()<br>whoWonGame()<br>main() | At the end of the game, the player with the higher score is shown to be the winner, including a case for a draw between the 2 players. | At the end of the game, the player with the higher score is shown to be the winner, including a case for a draw between the 2 players. |