Peter Moldenhauer
moldenhp@oregonstate.edu
CS 162 – 400
Lab 2

Lab 2

Program Requirements:

For this lab assignment, the first task is to identify the requirements for the program that I will be building at a later date. It is important to understand a given situation or problem before attempting to code a program to solve the problem. The paragraph below is a written description of the requirements for the program to be built. I will then pick apart this paragraph and look at the specific nouns that are mentioned. This deconstruction of the program requirements will give me a better insight as to how to identify the various classes to be used in the program and an overall plan to solve the situation:

Design a program that will simulate a game of two “players” taking turns to roll a dice. The program user will indicate the number of rounds to play. The user will also specify for each “player” the number of sides on the dice used and if that specific player is using regular or loaded dice. The rules of the game are as follows: For each round, the user will roll a die of the appropriate type for each player. The player with the higher result wins. If the results for both players are equal, then it is a draw. The winner of the game is the player who won the most rounds. When the game is over, the program will output which player won the game.

Identify all of the nouns:

Now that the description of the problem domain has been outlined, I can then identify all of the nouns in the description. It is essential to locate all of these nouns because each one of the nouns are potential candidates to be classes in the program I will write. The following list is all of the nouns from the problem description without duplicating any of them:

Program	Players
Game	Die
Regular Die	Loaded Die
User	Rounds
Sides	Die
Round	Result
Winner	Draw

Refine the list of nouns:

As noted earlier, the nouns that appear in the problem description are just potential candidates for classes to be used in the program. This next step that I will take is to refine the list of nouns. This shortened list of nouns will include only the nouns which will represent classes which are necessary to solve the problem at hand. There are numerous ways to refine the total list of nouns down to the specific nouns to be used for classes. I will be using the following check-list to eliminate the nouns that will not become classes:

- Eliminate the nouns that mean the same thing
- Eliminate the nouns that represent items that I am not concerned with in order to solve the problem
- Eliminate the nouns that represent objects, not classes
- Eliminate the nouns that represent simple values which can become variables and do not require a class

After taking the above check-list into consideration, I have refined the total list of nouns down to three nouns:

~~Program~~	~~Players~~
Game	~~Die~~
~~Regular~~ ~~Die~~	Loaded Die
~~User~~	~~Rounds~~
~~Sides~~	Die
~~Round~~	~~Result~~
~~Winner~~	~~Draw~~

Which nouns are classes:

After refining the total list of nouns, I was left with the three nouns: Die, Loaded Die and Game. These are the three nouns that I will use to name and represent the three classes I will use in my program (Die, LoadedDie and Game). I must note that even though “die” and “loaded die” sound very similar, they will still represent separate classes. Due to the fact that a loaded die is still a type of die, in my final program I will use inheritance and derive the loaded die class from the die class. The third noun (class) is game. This class will be responsible for running the actual dice game.

Which nouns are data members for which classes:

Now that I have identified the classes that I will be using in the program, I now have to identify the things that each class is responsible for knowing. Specifically, I am going to look at which nouns from the total noun list will represent data members for which class. In the process of refining the noun list down to get the three that are now classes, I can go back and assign the nouns that were previously eliminated and reassign them appropriately with the ones that are class data members. Below, I have listed out each of the three classes and the corresponding noun/data members to each class. Note that in addition to the previously eliminated nouns, I have come up with additional data members for each class through my own additional analysis.

Die class:

- The noun “sides” will become the data member numSides

LoadedDie class:

- numSides (inherited from Die class)

Game class:

- The noun “player” will refer to the type of die for each player. These will become two different Die objects as member variables in the Game class. (Die p1, p2)
- The noun “rounds” will become the data member numRounds
- The current scores of each player will be the data members p1Score and p2Score.

Which activities are functions of which classes:

Now that I have identified the things that each class is responsible for knowing, next I will identify the activities that each class is responsible for doing. These activities of each class will become the functions that I will include in each class to make the program. A lot of these class functions will be based upon what data members are in each class. For example, I will need a constructor in the Die class to set the number of sides because I have a numSides data member in that class. I have listed below each of the classes and the corresponding functions that each class will have:

Die class:

- Set up die with specified number of sides. This will become the constructor function Die()
- Roll the die. This will become the function rollDie(). This function will return the result of the roll.

LoadedDie class:

- Set up loaded die with specified number of sides. This will become the constructor function Die() (inherited from Die class)
- Roll the die. This will become the function rollDie() (inherited from Die class). This function will return the result of the roll. Note: this roll function will be more complex than the roll function in the Die class because it needs to set the advantage of having a loaded die.

Game class:

- Set up the game for a specific number of rounds, with two dice that have their own unique number of sides and each die could have either a regular or loaded

die. This will be the constructor function Game(). This constructor will need specific information passed into it.
- Play the game for the specific number of rounds and keeping track of scores at each round. This will be the function playGame()
- Output to the screen the winner of the game (or draw). This will be the printGameResult() function.

Specific actions to take note of:

There are a few key actions that I should reiterate and/or explain more thoroughly that I may not have covered earlier. First, the only action that the Die class will be responsible for is to roll the die and to return the value of what was rolled. Similarly, the LoadedDie class will also only be responsible for the action of rolling the die. However, the die roll action in the LoadedDie class will be more complex than that of in the Die class. This is because the LoadedDie class roll function needs to additionally calculate the roll with a higher probability of landing on the highest number of the die. There will still just be the one roll function in the class, but it will be a little more complex.

In the Game class, two Die objects need to be created to represent the two different players (each potentially having a unique type of die). Due to the fact that each Die object created for each player could be either a regular die or a loaded die, I will use inheritance to allow for this flexibility. Specifically, I will take advantage of the use of inheritance and use a base class pointer. This will allow for the creation of either a regular die or loaded die for each player. Once the Game class is set up with the data that it needs (rounds, dice types), then the only other functions that it will contain are the printGameResult and playGame functions. The printGameResult will simply output to the screen the final score and winner (possible draw) of the game. The playGame function will include a few specific actions that I should make a note of. In this function, there will be a while loop set for the number of rounds initially inputted by the user. Inside of the while loop there will be calls to the different Die object roll functions. Based on the outcome of each type of die rolling, each players' scores will be initialized and updated (also inside of the while loop). The result and scores of this specific round will then be outputted to the screen inside of the loop as well. The number of rounds will be decremented with each iteration of the loop. The playGame function will then exit once the number of rounds in the while loop is equal to zero.

Outline design of main program:

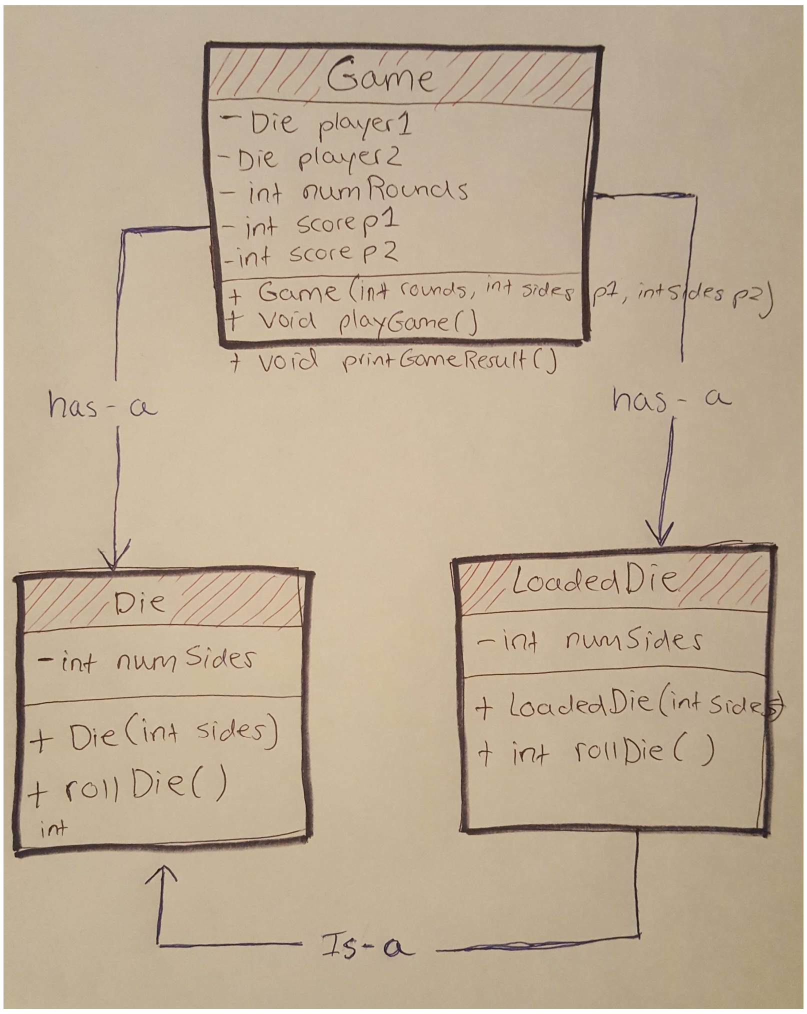
Aside from the three classes, I will build a main.cpp program which will control the flow of the program and allow the game to run. I have drawn out an outline of the design of the main program that I will build (below):

- Display the game title and author of the game
- Display instructions and game rules to the user

- Prompt user for the number of rounds in the game
- Loop to validate user input until an appropriate value is entered (positive integer)
- Prompt user for the number of sides on die for player 1.
- Loop to validate user input until an appropriate value is entered
- Prompt user to see if player 1 will have a regular or loaded die
- Loop to validate user input until an appropriate value is entered
- Prompt user for the number of sides on die for player 2
- Loop to validate user input until an appropriate value is entered
- Prompt user to see if player 1 will have a regular or loaded die
- Loop to validate user input until an appropriate value is entered
- Create a **Game** class object to set up the game.
    - This will involve creating a game object with data passed into the constructor.
    - Player1 and player2 will be set up to either a Die or LoadedDie respectively
    - **Die or LoadedDie** objects will be created in the Game class
    - Number of rounds will be passed into constructor
    - Player1 and player2 score's will both start off at zero
- The playGame function in the **Game** class will be called to play the game
- The printGameResult function in the **Game** class will be called to output the winner
- The program will then exit

Class hierarchy diagram:

The following diagram displays the organization and relationship of the different classes to be included in the program. In the diagram I have specified any "is-a" and "has-a" relations among the classes. Each class is presented in the diagram with its specific name, data and function members corresponding to that class. This diagram helps to give a visual understanding of the problem at hand and how to solve it. [diagram on next page]



Draft test plan:

There are a few places in this program that input will need to be validated, so I will begin by testing these items. If I know that these values are valid, I will be able to ensure that correct values are passed into the Game object constructor.

The user will be prompted for the number of rounds the game will last. The user is expected to enter a positive number here. If the user enters an invalid number, then he/she will be prompted again to enter a positive number. This will happen until a positive number is entered. I can test this by repeatedly entering a negative number and making sure that I am prompted after each input for a correct value. It is also possible that the user may enter a non-numerical character. If this happens, the user should be prompted again for a positive number. I will check this by repeatedly entering non-numerical characters and expecting to be prompted again.

I will use the same approach for ensuring the user enters a valid number of sides per die. The requirement for the input here is the same as the input for the number of rounds: it must be a positive number.

The user will be prompted for whether he/she wants player 1 to use a loaded die or regular die. The valid inputs will be ‘y’ or ‘Y’ for yes, and ‘n’ or ‘N’ for no. If the user enters some character other than those four options, the user is to be prompted again for one of them. This can again be tested by repeatedly entering invalid characters and ensuring that the user is prompted over and over again until a correct input is entered. The same logic applies for testing the prompt for player 2.

To test the actual gameplay, specifically the playGame() function, I will print out a lot of statements to the screen which will display each step of the game. For example, I will use print statements to show the current value of the round, current scores and roll results. These print statements will be used throughout the playGame() function at every spot in which the current data may have been manipulated which results in new data. For example, I need to make sure that the game rounds and players scores are incrementing appropriately so I need to print out before and after each one of these functions manipulate the data. This will show me that the rounds/scores are indeed changing as intended to do so.

The rollDie() function needs to also be tested so that I can be sure that the program is indeed generating a random number within the appropriate range every time the die is rolled. For example, if a regular six sided die is being used, I need test that a random number between 1 and 6 will be generated with each roll. I will again use print statements after every roll of the die to show that each result is an appropriate value. Similarly, with the rollDie() function for the loaded die, I need to test that a random number is generated appropriately but also that the odds are increased for the highest number on the die. In testing this, I will again use print statements to see that a valid number is being returned but I will also have to test this multiple times. The more times I test the rollDie() function for the loaded die, the more accurate I can see that the roll is returning the highest possible result more often than the other numbers.

To give a better understanding of the various inputs, the testing purposes and expected outcomes of each test, I have drawn up numerous charts below. The charts are labeled by the corresponding prompt to the user or program function that would potentially cause an error.

Prompt user for number of rounds (in main function)

Test number	Test	Purpose	Expected Outcome
1	Input < 1	Make sure that the program will not accept an input below 1	Loop to keep prompting user to re-enter a valid input
2	Input > 10^30	Make sure the program will not accept an input that is too large	Loop to keep prompting user to re-enter a valid input
4	Inputs multiple values such as 2 4 67 19	Make sure the program only accepts one value	Loop to keep prompting user to only enter one value
5	Inputs a value other than an integer (‘a’ or ‘&’)	Make sure the program only accepts a valid integer	Loop to keep prompting user to only enter a valid integer

Prompt user for number of sides on die (in the main function)

Test number	Test	Purpose	Expected Outcome
1	Input < 1	Make sure that the program will not accept an input below 1	Loop to keep prompting user to re-enter a valid input
2	Input > 10^30	Make sure the program will not accept an input that is too large	Loop to keep prompting user to re-enter a valid input
4	Inputs multiple values such as 2 4 67 19	Make sure the program only accepts one value	Loop to keep prompting user to only enter one value
5	Inputs a value other than an integer (‘a’ or ‘&’)	Make sure the program only accepts a valid integer	Loop to keep prompting user to only enter a valid integer

Prompt user on whether players should have a loaded die (in main function)

Test number	Test	Purpose	Expected Outcome
1	Input ‘y’ or ‘Y’	Make sure that the program accepts this user input	Accept user input continue with loaded die type
2	Input ‘n’ or ‘N’	Make sure that the program accepts this user input	Accept user input and continue with regular die type
4	Inputs invalid character such as ‘B’ or ‘*’	Make sure the program does not accept this input	Loop to keep prompting user to enter a valid character
5	Inputs an integer	Make sure the program does not accept an integer	Loop to keep prompting user to only enter a valid character

playGame() function activities

Test number	Test	Purpose	Expected Outcome
1	Use cout statements throughout function	Make sure that values are correct and that the calculations are correct	Number of rounds will decrement and player scores will increment accordingly
2	Have the function run every combination of die/loaded die types for each player	Make sure the function works correctly with different types of die objects	The program will run correctly with all die objects.

returnGameResults() function activities:

Test number	Test	Purpose	Expected Outcome
1	Manually keep track of score as game is running	Make sure the returned result from the function is indeed the actual result	The function will return the correct winner, looser and possible draw

rollDie() function activities (for both Die and LoadedDie):

Test number	Test	Purpose	Expected Outcome
1	Use cout statements in function	Make sure the function is returning an appropriate value	The function will return an integer that corresponds to the number of sides to that specific die
2	Run program multiple times, specifically looking at the cout statements in the function	Make sure the function is indeed returning a random value each time	The value returned with each die roll will be different with each time the program is run
4	Manually record the output multiple times with each cout statement of the loaded die rollDie() function	Make sure the function returns the "loaded" number more frequently than the other possible numbers	The function will return the loaded number at least 50% of the time