Pedro Torres Picón
CS 162 Lab 3 (Updated from Lab 2)

For this lab, we are designing a program that plays a simplified version of the card game War, using dice instead of cards. At a minimum, the program should be able to complete the following tasks:

- Ask the user for the number of rounds to play
- Ask the user for the number of sides for player 1's die, and whether it is loaded
- Ask the user for the number of sides for player 2's die, and whether it is loaded
- Generate dice with any number of sides, and generate dice that are loaded
- Roll the dice and show the user the results
- Determine which player won the round, or if it is a draw, and communicate it to the user
- Keep track of the score after each round
- Determine a winner (or draw) after the number of rounds specified, based on the number of rounds won by each player

With these requirements in mind, we will develop three classes: a Game class, a Die class and a LoadedDie class. Below are descriptions of these classes and the data and methods they will contain.

<u>Game Class</u>

- Data (Private Member Variables):
    - roundCounter: an int that is initialized to the total number of rounds to be played in this game
    - p1die: a Die that holds the die for player one
    - p2die: a Die that holds the die for player two
    - p1score: an int that holds the score for player one
    - p1score: and int that holds the score for player two

- Methods (Public Member Functions):
    - constructor: takes an int number of rounds to be played, and sets the roundCounter to that number
    - getRoundCounter: this method returns the value of the roundCounter variable
    - decRoundCounter: deducts 1 from the roundCounter variable
    - generateDice: this method asks the user to enter a number of sides for the p1die and whether it is loaded, generates a new Die with this info, asks the user for the same information for p2die, generates another new Die, and saves both dice to their respective member variables.
    - playRound: this method rolls both dice, shows the results of both rolls to the user, determines the winner of the round, communicates it to the user, and increments the appropriate score variable.

- endGame: this method compares p1score to p2score, and communicates to the user who the winner was, or whether there was a draw.

Die

- Data (Private Member Variables):
  - sidesArr: an array that holds a number of ints, each representing a side of the die
  - sidesNum: an int that represents the number of elements in the sidesArr

- Methods (Public Member Functions):
  - constructor: takes an int number of sides and generates a 'sides' array that holds the ints from 1 to the max number of sides. also sets the sidesNum variable to the total number of sides.
  - roll: generates a random number from 0 to (sidesnum - 1) that represents an array element, and returns the value of that array element.

LoadedDie (subclass of Die)

- Data (Private Member Variables):
  - The same member variables as Die

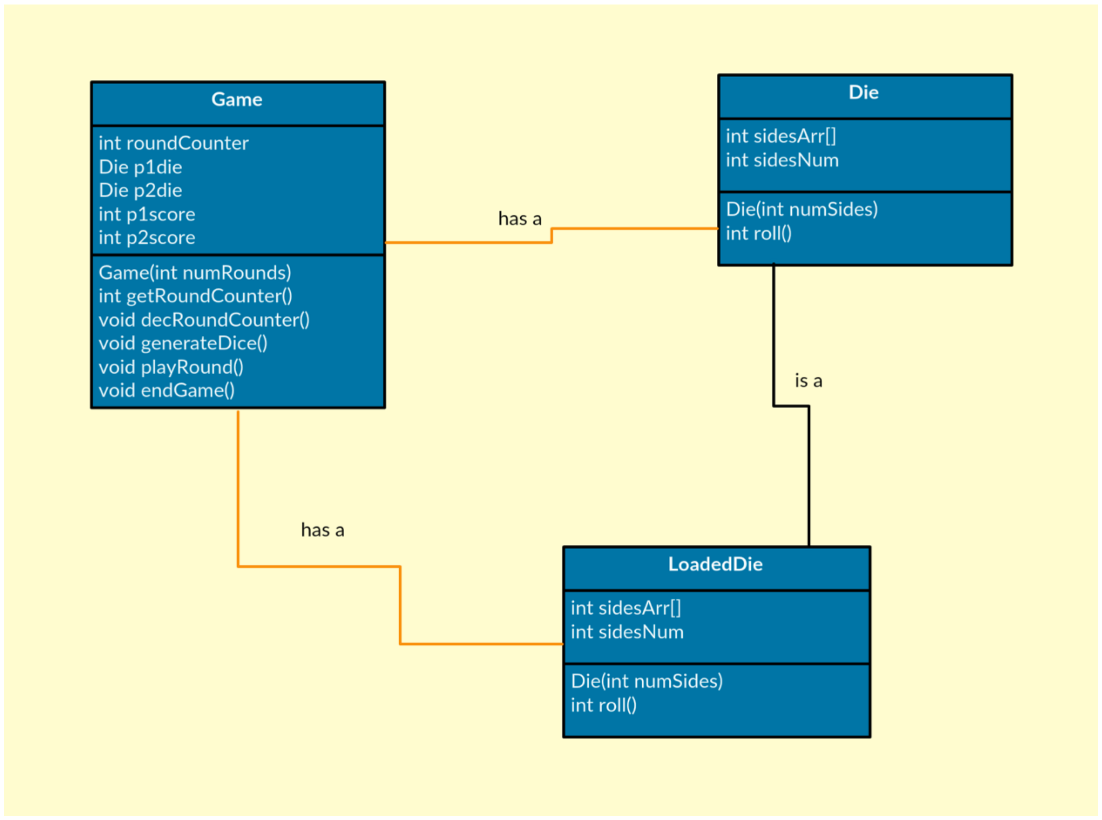
- Methods (Public Member Functions):
  - the same member functions as Die, with the difference that when the sidesArr array is created in the constructor, three instances of the three highest numbers are placed in the array to "load" the die

Note: In the Lab 2 requirements sheet, the definition of a "loaded die" was vague and left to our own imagination. Thus, for the purposes of this project, I made the design decision to define a loaded die as a die that has triple the odds of landing on the three highest numbers in the die. This is why each of the three highest numbers is represented three times in the sidesArr.

Using the classes outlined above, we would then develop a main() function that would run the program. A pseudocode design for that function is outlined below:

```
// welcome the user and explain the rules of the game
// ask the user to input the number of rounds
// create a new instance of the Game class with the number of rounds
// call the Game's generateDice function to generate the players' dice
// loop while the Game's roundCounter variable is larger than zero
       // call the Game's playRound function
       // call the Game's decRoundCounter function
// call the Game's endGame function
```

Class Hierarchy Diagram



Proposed Initial Test Set

Test / Input	**Expected Outcome**
Zero rounds entered	Input validated (illegal)
Negative number of rounds entered	Input validated (illegal)
Large number of rounds entered (100,000)	Input validated (set max)
n is entered as the round number	Game goes on for *n* rounds
Characters entered for number of rounds	Input validated (illegal)
Loaded die vs not loaded with same number of sides	Loaded die wins more
Zero number of sides for a die entered	Input validated (illegal)
Negative number of sides for a die entered	Input validated (illegal)
Large number of sides entered for the die (100,000)	Input validated (set max)
n is entered for the number of sides of an unloaded die	Array of *n* items is created
n is entered for the number of sides of an loaded die	Array of *n+6* items is created
Roll the dice	Random results produced
Results of roll are produced	Winner of round announced
End game with one score larger than the other	Winner is correctly named
End game with two equal scores	Draw is announced

Design Changes and Analysis (Lab 3)

During the implementation phase of this program there were substantial changes in relation to the original design plan. The most significant changes are documented below.

- The p1Die and p2Die variables ended up being pointers so both could be dynamically allocated to be Die or LoadedDie objects depending on the user's choices
- Included a destructor to deallocate the memory that was dynamically allocated for both dies
- The roll() function in the Die class had to be made virtual so the function could be redefined in the LoadeDie subclass
- The implementation of the subclass also changed, where I chose to redefine the roll class in the subclass, which was not in the original design
- In the original design I had thought about using an array to hold all the sides of the array. After I started building the program I realized that was just overcomplicating things and it should just be a random number generator from 1 to the size of the die
- The definition of "loaded" changed from a higher probability for the largest 3 sides, to just 50% probability for the largest side, and equal probability for the rest. This is implemented by getting a random number from 1 to double of numSides, and choosing numSides if the chosen number is larger
- The getRoundCounter and decRoundCounter functions were removed in favor of directly accessing the numRounds variable
- The generateDice function was removed because the Die and LoadedDie constructors have the same utility
- The play() function took the place of the playRound() function, because now the loop lives within the Game class and not in main() like in the original design
- Reused (and improved) two helper functions from the previous activity, one to get an integer from the user in a given range, and one to get a yes/no answer

Actual Test Results

Test / Input	**Actual Outcome**
Zero rounds entered	Input validated (illegal)
Negative number of rounds entered	Input validated (illegal)
Large number of rounds entered (100,000)	Input validated (1000 max)
n is entered as the round number	Game goes on for *n* rounds
Characters entered for number of rounds	Input validated (illegal)
Loaded die vs not loaded with same number of sides	Loaded die wins ~75% rounds
Zero number of sides for a die entered	Input validated (illegal)
Negative number of sides for a die entered	Input validated (illegal)
Large number of sides entered for the die (100,000)	Input validated (1000 max)
n is entered for the number of sides of an unloaded die	Random number 1 to *n* generated

n is entered for the number of sides of a loaded die	Random number 1 to *n* generated
Roll the dice	Random results produced
Results of roll are produced	Winner of round announced
End game with one score larger than the other	Winner is correctly named
End game with two equal scores	Draw is announced