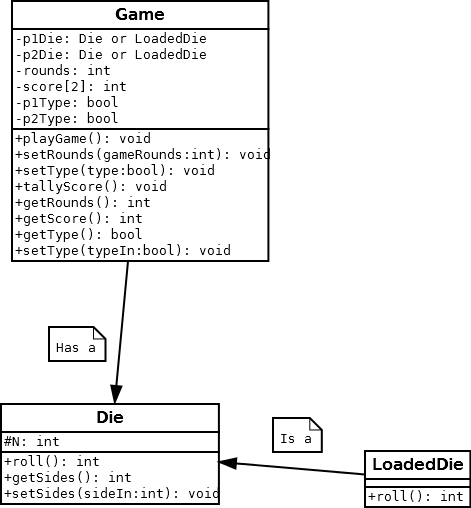
1. Define what the program is to do.
   1. Purpose: Play a game of war using dice between two players.
   2. Input: die sides per player, loaded die or not for each player, number of rounds to play
   3. Output: menu for playing the game, after menu options selected output the results of the game indicating sides and type (loaded/normal) of die, results of each player’s roll per round, and final winner of the game
2. Model the program

* Program starts displaying menu for the game,
  + Menu options are start game, setup player 1 die, setup player 2 die, set number of rounds, or exit
* Setup player 1 die selected
  + prompts user to enter die size for player 1, p1DieSize
  + Prompt user to enter type of die for player 1, p1Type
  + Create p1Die object
    - If p1Type == true then create LoadedDie instead of Die object
* Setup player 1 die selected
  + Prompt user to enter die size for player 2, p2DieSize
  + Prompt user to enter type of die for player 2, p2Type
  + Create p2Die object
    - If p1Type == true then create LoadedDie instead of Die object
* Set number of rounds selected
  + Prompt user to enter rounds to play, gameRounds
* Start game selected
  + Use get functions (getRounds, getSides) to validate that parameters for game are set up prior to running
  + Use Game class to create game object, currentGame
  + Output the sides and type of each players dice
  + playGame method passed number of rounds and controls game
    - loop until required number of rounds go by
    - roll each die once per round using regular or loaded function based on bool type of die
      * loaded function fills array with values corresponding to possible die rolls, plus an extra entry for each of the values that are greater than the mean die roll
      * random number generator returns value corresponding to the possible index spots of the array
      * regular roll function returns random value based purely on the available results from diedie
    - output the results of each roll to the screen each round
    - increment counter to control game loop
    - use tallyScore method to adjust score based on die roll
      * if p1 wins add 1 to score[0]
      * if p2 wins add 1 to score[1]
      * if draw, no score added
    - at end of the game output winner based on score[]
      * if score[0] > score[1] p1 wins
      * if score[1] > score[0] p2 wins
      * else game is a draw
* After game is complete return to the main menu

1. Class hierarchy diagram



1. Testing Plan

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test | Input Values | Driver Functions | Expected results | Observed outcomes |
| No input1 | None | main()  while getRounds, getSides ==0 | Testing input validation to prevent game from running with no data | Program functions as expected. Does not run without die objects created. |
| Integer input validation | spaces  “asdf”  1234asdf | main()  InputValidation function | When creating Die object, or setting rounds from main menu should loop until valid integer input received. | All input validation functions as expected. Created new input validation to check for even numbers that was also implemented and checked out ok. |
| scoreTally test | 5 rounds with same types of dice  both fair 4 sided  both fair 6 sided  fair 4 sided vs fair 10 sided | main()  playGame()  roll()  tallyScore() | Should have proper output in the event of either player winning, or a draw. | After initial test scores were not tallying properly.  Tracked down problem seemed to be with not having score initialized to 0.  All other tests worked after this. |
| Loaded roll output | 15 rounds with loaded and regular 6 sided dice | main()  playGame()  roll()  LoadedDie roll() | Expect to be able to see divergent averages between loaded and regular roll showing a bias toward the numbers greater than the average. | Running with 15 rounds caused seg fault.  Reworked program using shared\_ptr and a vector of shared\_ptr to dice. |

1. Reflection

My plan for this program ended up changing quite a bit from my initial design over the course of this lab. Most of the design changes I implemented had to do with having more time to study and review the course materials over the past week.

Originally my plan was to use standard pointers like I have been doing since CS-161 and try to clean up afterwards. After reviewing more of the text I decided to try and implement some of the newer information from this week regarding smart pointers to help with memory management. So far, I have not been able to fully get things working, but I feel like I am on the right track.

I know that a big part of my problem currently is from passing the shared\_ptr into the vector of shared\_ptr<Die>. Somehow, even though I feel I am following the material in the book somewhat closely my virtual functions are still not being called.

Part of my debugging with trying to track down what is wrong seems to point to losing the pointers once they go into the vector. Therefore, I think my loaded die rolls are always so high because it tries to reference a location that is no longer allocated for the pointer for some reason. With a little more time studying the material I think I could get this working as expected.