Stephen Townsend

CS 162 – Lab 3

Design Changes

The main change that I made from my original design was to not include objects for loaded dice in the Dice class. Instead, I have the Game class handle them. I originally had one object for dice and one for loaded dice. I split this up into two objects for both classes. Each object is for player 1 and 2. I wanted to have a way to handle if there were no loaded dice or if there were two loaded. The only way I could handle this was to have a Loaded Dice Player 1, Normal Dice Player 1, Loaded Dice Player 2, and Normal Dice Player 2.

The next design change was to utilize the inheritance of Loaded Dice. Before programming, I was unsure how the inheritance would function. As a result, I had some duplicate member variables and member functions that the Dice class already contained. Once I started debugging, I realized how the inheritance worked and how well it worked with objects. As a result, I was able to remove most of what LoadadDice was going to use.

The last change was changing functions in my classes, but mostly changing them in my Game class. The best explanation is to show the difference. Below is Table 1. Table 1 is what I predicted for Lab 2.

Table 1: My Layout for Lab 2

Class Die	Class LoadedDice	Class Game

Data:
LoadedDice loadDice;
int diceSide;
int diceNumbers;
	Data:
	int loadSide;
	int loadNum;
		Data:
		Dice dice;
		int rounds;
		int score;
		int playerTurn;

Members:
int setDieSize()
int getDieSide()
int dieNumbers()
	Members:
	int getLoadSide()
	int LoadNum()
		Members:
		char diceType();
		int getTurn();
		void setTurn();
		int getRounds();
		void setScore();

Next is Table 2. Table 2 is the design that I finished my project with.

Table 2: My Final Layout

Class Die	Class LoadedDice	Class Game

Data:
int diceSides;
	Data:	Data:
		LoadedDice loadedPlayer1;
		LoadedDice loadedPlayer2;
		Dice normalPlayer1;
		Dice normalPlayer2;
		int rounds;
		int player1Score;
		int player2Score;
		int player1;
		int player2;
		int playerTurn;
		int loadPlayer;

Members:
void setDiceSides(int);
int roll();
void getDiceSides();
	Members:
	int loadWeightRoll();
		Members:
		void setTurn(int);
		int loPlayer(int);
		void setScore(int);
		void getOutcome();
		void play();
		void gameSettings();
		int inputValidation(int);
		Game();

Problems

The biggest issue I had was with the inheritance. My compiler kept giving me an error stating that my base class was undefined. I did not understand this error because my class syntax looked completely fine. I then looked at the inheritance class and did not see any issues there either. After a few hours of modifying code, trying to figure out the problem, I finally found the reason. The reason was not the syntax of my class, but it was the header file that I included with my base class.

The header file that I had was for my Game class. I thought that Dice would need access to Game. As a result, I included the header. This apparently caused a circular reference because Game included the Dice header. Thus, Dice was undefined because the compiler went in a loop through the two headers. I realized that Dice and Loaded Dice did not need access to Game because Dice and Loaded Dice are only holding data to be accessed by Game. They did not need to access any data from Game itself.

Stephen Townsend

Analysis

Rounds	Loaded Dice Choice	Dice 1 Sides	Dice 2 Sides	Outcome
20	1 - Player1	6	6	Player 1 will win overall due to weighted Dice
20a	-	-	-	The program will ignore the char and set rounds to 20
20a	0 - No Loaded	6	6	The game will be pretty close in score
A20	-	-	-	The program will ask for a integer
20	5	-	-	The program will ask for a integer between 0 and 3
20	3 – Both Loaded	6	6	The game will be pretty close in score
20	1	6	10	Player 2 may win due to higher die numbers then loaded 6 die

Rolling Overall

If the user uses normal die, the scores are usually pretty close or even tied.

If the user uses a loaded die, the score is skewed to the loaded player.

If both dice are loaded, the scores are similar to normal dice.