Group Project: Rock Paper Scissors
Group 15
Emmet Cooke, Neale Mason, Lee Rice, Matt Shim
CS 162
Due: 11/5/2017

Responsibilities

Emmet:

- Additional Input validation

Neale:

- Testing/Debugging
- Augmenting for Flip

Matt:

- Setup Git repo
- Initial class setup
- Menu selection
- Input validations

Lee:

- Initial gameplay function
- Strength logic/comparison
- Win/loss tracking
- Play again function

Planning

Class setup plan here

Lee: The plan for gameplay loop was fairly straightforward and fit the flow of the game. First we planned to have the user be asked if they want to choose the strengths for the various tools. The program would then take a strength for each tool, for both computer and ai, if the user chose to provide different strengths. If a user strength was provided, a custom constructor for that tool instantiates an object. Two tool objects ended up needing to be required due to a possible variation between the user strength and computer strength.

The user then chooses their tool, the computer's tool is chosen at random. After this, the gameloop() is called. A series of if statements provide the logic to set the tool strengths. This is done by comparing the user's tool to the computer's tool and setting temporary strength variables for each.

The adjusted temporary tool strengths are then passed on to the toolFight function. The tool fight function compares the adjusted strengths and increments the win counters based on which tool has the largest strength integer. The player is then given a count of player wins and computer wins. Finally, the player is given the choice to start the game again. Counters are not reset between rounds

TESTING

Testing was a little difficult as each team member implemented their own code. We made sure that anywhere requiring user input was tested. Game logic can always be improved.

Input validation

Test	Input Value	Relevant Methods	Expected	Actual
Entering doubles, characters, negative numbers for ints	12345, pto, Asdfasdfasdf, -15, 1.4345	Selecting menu item, selecting strength,	If double: “Please enter a valid int” If characters: “Please enter a valid int” If negative (for strength): “Please choose a positive strength”	Displayed correctly
incorrect char values, correct char value followed by string	T, U, X, speeding, paperboy	Selecting Rock/Paper/Scissor	“Please only enter a valid character (r = rock, s = scissors, p = paper, e = exit game)”	Input checks for first character only. First character check is correct, but “paperboy” would be read as “p”

Menu Selection

Test	Input Value	Relevant Methods	Expected	Actual
Random integer and double values	6, 4, 9, 1.5	Menu function at start of game	“Please enter a valid int”	Displayed correctly for incorrect ints and for doubles
Random characters and strings	“Checker”, “pleaseFail”	Menu function at start of game	“Please enter a valid int”	Displayed correctly

Rock Paper Scissor selection

Test	Input Value	Relevant Methods	Expected	Actual
incorrect char values, correct char value followed by string	T, U, X, speeding, paperboy	Selecting Rock/Paper/Scissor	“Please only enter a valid character (r = rock, s = scissors, p = paper, e = exit game)”	Input checks for first character only. First character check is correct, but “paperboy” would be read as “p”. However, given that it validates input and checks for the correct value, it does not appear to need changing

‘Combat’

Test	Input Value	Relevant Methods	Expected	Actual
Does the number of wins display correctly based on the # of games?	N/A	toolFight	Wins start at 0 then increment by 1 based on which player wins. If there’s a tie, nobody gets the # of wins incremented	Increments and displays correctly

Reflection

Lee: As mentioned above, overall the final code largely followed the initial plan. There were a few issues I came across when writing the gameplay loop. The first was how to keep track of the strengths. I was at first trying to find a way to create one object per tool and change its strength based upon whether the tool belonged to the player or the AI. This became far too tedious and I eventually realized I only needed to create two objects per round, so I just implemented custom constructors for each tool. Code wise, this was much easier to write and far easier to read.

In retrospect, the strength logic could have been broken out into its own separate function. There was really no need to write it into the game loop itself when a fully separate function, or even a different class would have worked. That said, though the strength logic code is lengthy, it is fairly easy to understand.
Matt - I created the Git repository - linked here and wrote the foundational code for the program - i.e the Menu, some basic validations (leaving the bulk to Emmet), Player class, the Tool class, and the derived

classes, although I left most of the code for the derived classes blank. I also created the initial scaffolding for the 'RPSGame' client which would handle most of the game logic, to satisfy the requirement for 'minimal code in main()'. Other team members handled most of the finer game logic for the actual game implementation.

I also provided some Git basic/introductions to a few team members who were interested in learning, although we decided early on that if Git became too troublesome due to time constraints we could do it the 'old fashioned way' and work off of google docs or sending email files. I was impressed that everyone managed to get some basic git practice in, and I'm proud to have facilitated first pull requests for some. I was initially very overwhelmed with the amount of work due this week but I'm glad that this project went relatively smoothly.

Emmet - My contribution to the project was the input validation of the program. There are two separate boolean validation statements and a getNum function included six other times. The getNum function is included from my validateInput file which I personally have used in previous projects.

The two Boolean validation blocks uses getline to enter the input into a string and then checks that the string is no longer than 1 and that the first character of the input string matches either a 'y' or a 'n'. These allow for the program to loop until the user enters the correct value.

The getNum function is used to gather the strengths of the tools used by both the player and computer. It checks to see that the string entered is a valid int and then it uses the stoi function to change it to an int.

These functions allow the program to run without crashing from invalid input, as well as directing the users towards what the correct input should be.

Neale - I got a late start on the project due to work commitments, so I contributed mostly near the end of the project. I focused on error checking, testing, and ensuring that the code compiled and ran correctly on Flip. I also added and repaired small amounts of code as needed when an error was discovered.

We ran into some issues with the tool strengths (some tools were winning when they weren't supposed to), some tools being chosen incorrectly (ex: entering "p" and getting scissors), and ties not being properly unassigned (our previous functions had ties going to the computer).