Group 21: Samantha Tone, Elliot Lapinel, Alexandra Henley, Patrick Rice , Noah Buchen
CS 162-400
5 Nov. 2017

Incl: Work Division, Reflection, Testing Plan, Design Document

Work Division	
Samantha Tone	Designing, input validation, RPSgame sans Tool assignment and calls, computer AI, main function, testing, reflection
Elliot Lapinel	Set Strength
Alexandra Henley	Designing, fight functions, reflection
Patrick Rice	Group management, managing merges of members work with GitHub, code review, program output
Noah Buchen	designing, Tool Rock Scissor Paper Class, testing, testing plan, reflection

Reflection

After an initial design flowchart was created and our group met for the first time we decided to create detailed pseudocode before any coding would begin. These initial steps laid an excellent ground work for all the work to come. The group also decided to use GitHub to manage the project. Most of the group members had little to no GitHub experience, but Patrick agreed to help everyone learn this important tool. There were some challenges with using GitHub, but with Patrick's help each piece of the project was successfully merged together.

Once the coding began things moved along very quickly thanks to Sam. She created almost the entire RPSGame class, including the computer AI functionality. Additionally she provided input validation for all inputs and started the testing document. Because of our initial discussions (and Sam's skill!) there were no major challenges or changes made to the RPSGame class. Next , Elliott and Patrick worked on the setStrength function while Noah created the basic Tool, Rock, Paper, and Scissor classes minus the fight functionality. Patrick noticed and corrected the fact that Noah had included member variables in the Rock, Paper, and Scissors Class that were the same as the parent Tool class and should instead be inherited. The only challenges with creating the setStrength function were due to not completely understanding the program requirements and were solved by further discussion. Finally, like everyone in the class we needed additional time discussing exactly how and why the strength of each tool would affect the outcome of each fight. Alex completed the fight functions and then the program was tested and complete. Throughout the entire process it should be mentioned again that Patrick was incredibly helpful in managing the GitHub and working through the challenges of everyone learning that tool as we went.

Testing Plan

validChar Testing				
Description: Testing the ability of function validChar to handle a variety of different user inputs for strength "y/n" and tool "r/p/s/e" questions in RPSGame Class. A main function housed in charTest.cpp was used to initially test this separate from the game. The same tests were performed once it was included in the RPSgame.cpp file.				
Test Case	**Input Values**	**Driver Functions**	**Expected Outcomes**	**Observed Outcomes**
Testing the string length if statement				
strIn.length() > 1	2+ char string	validChar()	This should be caught in the if (strIn.length() > 1) statement, should return "String entered is too long. Please try again.", and should read in a new user input for validation.	Strings of all different lengths were tested, including ones that contained only the passable char letters from the yn and rpse arrays. All entered this statement to prompt for a new input to be validated.
strIn.length() = 1	1 char string	validChar()	This should proceed on to the else statement, in which it should either be passed out of the function if the it is within the yn or rpse arrays or prompt for a new user input if it is not.	Strings of length 1 with a variety of different letters and numbers were tested. These all made it to the second loop, in which a new input was only prompted if it did not fall in the yn or rpse arrays.
Testing the array checking for loop				
char != 'y'||'n' or 'r'||'p'||'s'||'e'	numbers and letters outside of input arrays	validChar()	This should be caught in the if statement following the for loop, should return "String entered is not one of the specified char options. Please try again.", and should read in a new user input for validation.	Various numbers, letters, and characters were tested. All entered left the for loop with loopEnder == false and prompted the user for a new input in the if statement following it.
char == 'y'||'n' or 'r'||'p'||'s'||'e'	letters within input arrays	validChar()	For all characters within the yn or rpse arrays, this should end the validation function and return that validated char to the main().	Each possible input was tested, including those in the opposite array. This performed as expected and returned a char value within the array each time.
Testing after a failed input				
strIn.length() > 1 after failed string	2+ char string	validChar()	This should perform exactly like the original test case. Test added to make sure do-while loop is flipping	Tested several times. Performed identical to the test case above.

			loopEnder to restart validation.	
char != 'y'||'n' or 'r'||'p'||'s'||'e' after failed string	numbers and letters outside of input arrays	validChar()	This should perform exactly like the original test case. Test added to make sure do-while loop is flipping loopEnder to restart validation.	Tested several times. Performed identical to the test case above.
char == 'y'||'n' or 'r'||'p'||'s'||'e' after failed string	letters within input arrays	validChar()	This should perform exactly like the original test case. Test added to make sure do-while loop is flipping loopEnder to restart validation.	Tested several times. Performed identical to the test case above.
testing fights				
Description: Testing the abilitys of the fight function using both the setStrength, default and AI				
Test Case	**Input Values**	**Driver Functions**	**Expected Outcomes**	**Observed Outcomes**
one tool strength is set much larger than another	case 1: human tool=20, comp=2 case 2: comp=20, human =2	fight()	case 1: human almost always wins, case 2: computer almost always wins	as expected
both strengths are equal	user chooses not to input strength	fight()	correct tools win when fighting each other, R beat S, S beats P, P beats R	as expected
both strengths are equal	human =3, comp =3	fight()	correct tools win when fighting each other, R beat S, S beats P, P beats R	as expected
testing AI	human always chooses rock, or scissor, or paper	fight()	after 3 fights the computer starts to choose the correct object to always win	as expected

testing AI	human chooses mostly one type	fight()	computer will win most of the fights	as expected

Design Document

Tool Class:

int strength
char type
void SetStrength(int)

Rock Class:

```
Rock()
        strength = 1;
        type = r;
Rock(int)
        strength = int;
fight(char Tool)
        if(Tool == s)
                strength = strength*2;
        else if(Tool == p)
                strength = strength/2;
```

Paper Class:

```
Paper()
        strength = 1;
        type = p;
Paper(int)
        strength = int;
fight(char Tool)
        if(Tool == r)
                strength = strength*2;
        else if(Tool == s)
                strength = strength/2;
```

Scissors Class:

```
Scissors()
```

```
            strength = 1;
            type = s;
        Scissors(int)
            strength = int;
        fight(char Tool)
            if(Tool == p)
                strength = strength*2;
            else if(Tool == r)
                strength = strength/2;
```

RPSGame Class:

char getToolComputer(*char array, int roundInput**)**

```
    int rock = paper = scissor = 0;
    char last = array[roundInput - 2];
    char current = array[roundInput - 1];
```

should probably also include if/else statement to return random selection if less than 3 rounds completed

```
    for (int i=1, i<roundInput-1, i++)
        if (array[i] == current && array[i-1] == last)
            if (array[i+1] == 'r')
                rock++;
            else if (array[i+1] == 'p')
                paper++;
            else if (array[i+1] == 's')
                scissor++;
```

if/else statements to determine largest & return appropriate next char

mainRPS:

Create RPSGame object
Display the menu (which asks about setting strengths, but I think it is optional and would rather not mess with it.)
have computer choose tool
Set computer tool
prompt for user to choose tool (validate input)
Set human tool
call fight using data entered which-returns 'h' for human win, 'c' for computer win, 't' for tie

display fight results
add results to win counters
display win counters
Ask user to play again or quit (validate input)