Bao Chau, Brendan James, Daniel Maynard, Brock Neidert, Andrew Sturtevant
CS162-400 Group 6 - Reflection Document
30 July 2017

Design Decisions: After some discussion among our group, the project started with a very basic design plan that included completion dates for significant project milestones. The first of those milestones was the Tool class design. The Tool class was necessary for every subsequent step in the design process and also produced the initial functionality to produce a very basic version of the Rock, Paper, Scissors game. Initially, the Tool classes had a virtual fight(Tool) function that would temporarily adjust a particular tool's strength depending on what type other Tool it was fighting. This was determined in each Rock, Paper, Scissor class. With the Tool class complete, our second milestone was a working prototype of the RPSGame class. The RPSGame provided some rudimentary form to the overall progression of the game but lacked validation methods, the setStrength method and contained no menu. The third milestone was the implementation of a menu and an integer validation method. With a largely finished, functional program in place, we were able to move onto the final phases of our design which included implementation of the setStrength method and final testing and debugging.

Testing and results: Much of our testing was planned around the integer validation and setStrength methods. To ensure ease of use, we needed to enforce and standardize what the program accepted as valid input . The menu, tool selection and setStrength method are all reliant on integer values and, with this in mind, we placed significant importance on developing a well rounded integer validation method. We discussed the potential for user input of string values but ultimately decided that the added complexity was both inconsistent with the majority of our program prompts and also introduced complexity that could be easily avoided by enforcing integer input. Message prompts were added to the validation methods to inform the user when input was rejected and, furthermore, what input was allowed in the particular portion of the program. Allowing the user to set the strength of the tool fundamentally altered the expected outcome of the fight method between two objects and required us to ensure the outcome was consistent with the strength values input by the user. While a Rock Tool would typically defeat a Scissor Tool through default strength values, allowing the user to alter this value needed to produce a possible outcome where the opposite was true (Scissors beats Rock).

Design Changes: Several stylistic and optimization changes were made to our project over the of development but the changes listed below are the most significant from a functionality perspective:

-Temporary Strength Adjustment: It was later decided to overload the getStrength function and adjust each tool's temporary strength there instead of inside of the fight(Tool) function. That function was then changed not be virtual.

-User Tool selection: the first iterations of our program contained prompts asking the user to select a tool by the typing the first letter of the Tool name (R for rock, P for paper, S for scissors). This initial choice was somewhat inconsistent with the majority of our other prompts that requested numeric input to make selections so it was changed to bring it in line with other aspects of the program.

-Randomization of computer choice: Moved from slightly weighted choice to evenly distributed chance for all Tool class options.

-Reduced overhead of scoreRound method: Upon testing with some lines of print code, it was discovered that the scoreRound method was actually determining the outcome of the "fight" three different times because there was a method call within each of the "if" statements. It was adjusted to only be called once at the beginning of the scoreRound and stored within a local variable.

Test case #1
SetPlayer1/SetComputer Functions

Test Case	Input Value	Driver Function	Expected Outcome	Observed Outcome
input too low	input < 1	getIntegerFromUser	Your input is invalid. Please enter an integer from 1 to 4.	Looped back to allow the user to enter integers between 1-4
input not a number	input %#	getIntegerFromUser	Your input is invalid. Please enter an integer from 1 to 4.	Looped back to allow the user to enter integers between 1-4
Input is a string	1234	getIntegerFromUser	Your input is invalid. Please enter an integer from 1 to 4.	Looped back to allow the user to enter integers between 1-4
Input too high	Input > 4	getIntegerFromUser	Your input is invalid. Please enter an integer from 1 to 4.	Looped back to allow the user to enter integers between 1-4

Case #2
Main Menu

Test Case	Input Value	Driver Function	Expected Outcome	Observed Outcome
input too low	input < 1	getIntegerFromUser	Your input is invalid. Please enter an integer from 1 to 4.	Looped back to allow the user to enter integers between 1-4
input not a number	input %#	getIntegerFromUser	Your input is invalid. Please enter an integer from 1 to 4.	Looped back to allow the user to enter integers between 1-4
Input is a string	1234	getIntegerFromUser	Your input is invalid. Please enter an integer from 1 to 4.	Looped back to allow the user to enter integers between 1-4

Case #3
Tool Fight Function

Test Case	Input Value	Driver Function	Expected Outcome	Observed Outcome
Rock vs Rock	1	Fight function	Tie	No winner!
Rock Vs Scissor	1	Fight function	Rock wins	You won!
Rock vs Paper	1	Fight function	Paper wins	You lost!
Scissor vs Scissors	2	Fight function	Tie	No winner!
Scissor vs Paper	2	Fight function	Scissor wins	You won!
Paper vs Paper	3	Fight function	Tie	No winner!