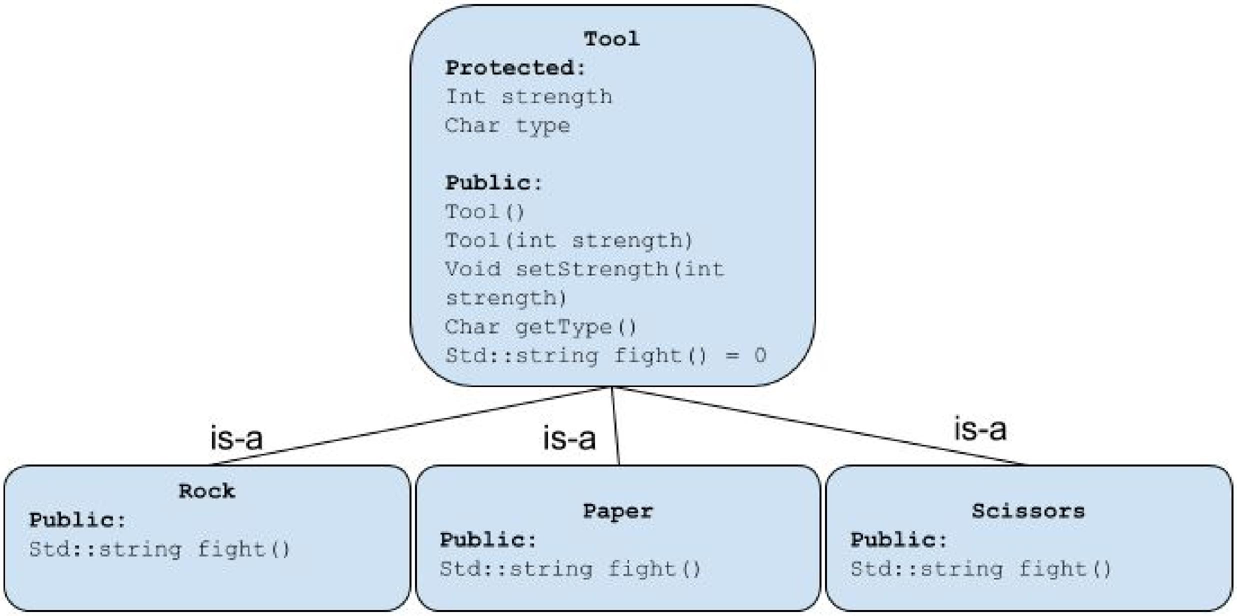
CS-162 Group Project
Group 11:
Charles Chen
Zachary Morrissey
Eve Robitaille
Cody Schmidlin
Beau Shirdavani
Brian Trang
Mark Walker
Max Moffett

Group Project Design Document

We began this project by laying out a basic class hierarchy, with the attributes and methods that we thought would be needed. This helped us break up the project into multiple subcomponents which could be programmed by group members individually.



In addition to the class hierarchy above, we laid out pseudocode for the classes.

Classes

Tool

Attributes: int strength, char type

Methods:

```
        Tool();
        Tool(int strength);
        void setStrength(int strength);
        int getStrength();
        char getType();
```

```
virtual std::string Fight(Tool* opponentTool) = 0;
    // Booleans would be hard to deal with (need to account for win, loss, or tie)
```

Derived Classes

Rock

```
Method: std::string fight(Tool* opponentTool)
    Sets tempStrength = strength
    If opponentTool is scissors
        tempStrength *= 2
    If opponentTool is paper
        tempStrength /= 2

    If tempStrength > opponentTool’s strength
        Return “win”
    Else if tempStrength < opponentTool’s strength
        Return “loss”
    Else if tempStrength == opponentTool’s strength
        Return “tie”
```

Paper

```
Method: std::string fight(Tool* opponentTool)
    Sets tempStrength = strength
    If opponentTool is scissors
        tempStrength /= 2
    If opponentTool is rock
        tempStrength *= 2

    If tempStrength > opponentTool’s strength
        Return “win”
    Else if tempStrength < opponentTool’s strength
        Return “loss”
    Else if tempStrength == opponentTool’s strength
        Return “tie”
```

Scissors

```
Method: std::string fight(Tool* opponentTool)
    Sets tempStrength = strength
    If opponentTool is paper
        tempStrength *= 2
    If opponentTool is rock
        tempStrength /= 2
    If tempStrength > opponentTool’s strength
        Return “win”
    Else if tempStrength < opponentTool’s strength
        Return “loss”
    Else if tempStrength == opponentTool’s strength
        Return “tie”
j
```

RPS Class

Private:

```
Int human_wins;
Int computer_wins;
Int ties;
```

Public:

```
Tool * player1;
Tool * computer;

Void gameSequence()
{
    Do

    //Human selects tool
    cout<< “Please make a selection”<<std::endl;
    Cout << “1. Paper”<<std::endl;
    Cout << “2.Rock”<<std::endl;
    Cout << “3.Scissors<<std::endl;
    std::string humanChoice;
    cin>> humanChoice;

    //Computer selects tool via computerPick()
    Display computer pick

    Computer pick = randomly generated between 0-2

    If 0
        Pick rock
    If 1
        Pick paper
    If 2
        Pick scissors
```

```
If human choice is rock and computer choice is rock
    ties++

If human choice is rock and computer choice is paper
    Computer_wins++;

If human choice is rock and computer choice is scissors
    human_wins++;

If human choice is paper and computer choice is rock
    human_wins++;

If human choice is paper and computer choice is paper
    ties++;

If human choice is paper and computer choice is scissors
    Computer_wins++;

If human choice is scissors and computer choice is rock
    Computer_wins++;

If human choice is scissors and computer choice is paper
    Human_wins++;

If human choice is scissors and computer choice is scissors
    ties++;


Display wins
Display computer wins
Display ties

;

Ask user if he wants to play again

While yes, do human selection and computer selection and determine result again;
```

Testing Plan

Test Input	**What are we testing?**	**Expected Results**	**Observed Results**
Create Rock object	Rock constructor	Object created successfully	Object created successfully
Create Paper object	Paper constructor	Object created successfully	Object created successfully
Create Scissors object	Scissors constructor	Object created successfully	Object created successfully
Create tool object	Tool constructor	Cannot create object	Cannot create object
Create RPSGame object	RPSGame constructor	Object created successfully	Object created successfully
computerPick()	Computer choice of rock paper or scissors	Return int >= 0 and <= 2	
Game checks for valid inputs	Correct data type by user at appropriate times	If the type is incorrect a loop will have the user input the data again until it is the correct type	If the type is incorrect a loop will have the user input the data again until it is the correct type
Play again prompt allows users to run game again	Ensures that play again logic path works correctly, and results are saved properly	Previous results are saved when user chooses to play again, and another round is initiated	Sometimes, game will start automatically without waiting for user prompt, or the game will end without waiting for user prompt
Correct strength levels when playing the game in the default mode	The strength int for rock, paper, and scissor in a normal game	The strength for each should be 1	All tools strength start out at 1
Rock Strength Comparison	When fighting other specific tools the rock's strength	Strength should double when fighting scissors and be	The fight function checks for the opponent's type and

	should vary	halved when fighting paper	adjusts the strength correctly
Paper strength comparison	When fighting other specific tools the paper's strength should vary	Strength should double when fighting rock and be halved when fighting scissors	The fight function checks for the opponent’s type and adjusts the strength correctly
Scissors strength comparison	When fighting other specific tools the scissor's strength should vary	Strength should double when fighting paper and be halved when fighting rock	The fight function checks for the opponent’s type and adjusts the strength correctly
Custom strengths should correctly affect the outcome of the fights	Test high strengths multiple times to see how much the outcome varies	The higher strengths should win more often than not	The higher strength does tend to win more often
Test in isolation - Computer’s pick for rock, paper, or scissors should be reasonable random, given a large number of picks.	Test that computer’s picks are random	Out of 1000 picks, picks should be split evenly amongst rock, paper, and scissors	Out of 1000 picks, picks are split roughly evenly amongst rock, paper, and scissors

Test Results

Test Results			
Default Strength			
Human tool	Computer tool	Result	Total wins
Rock	Paper	Computer won	You:0 , Computer:1 , Ties:0
Rock	Scissors	Human won	You:1, Computer:1, Ties:0
Paper	Paper	Tie	You:1, Computer:1, Ties:1
Paper	Scissors	Computer won	You:1, Computer:2, Ties:1
Scissors	Rock	Computer won	You:1, Computer:3, Ties:1
Scissors	Scissors	Tie	You:1, Computer:4, Ties:2
Non-default strength: Human 2, Computer 1			
Rock	Paper	Tie	You:0, Computer:0, Ties:1

Rock	Paper	Tie	You:0, Computer:0, Ties:2
Paper	Scissors	Tie	You:0, Computer:0, Ties:3
Paper	Paper	Human won	You:1, Computer:0, Ties:3
Scissors	Scissors	Human won	You:2, Computer:0, Ties:3
Scissors	Rock	Tie	You:2, Computer:0, Ties:4
Non-default strength: Human 1, Computer 2			
Rock	Paper	Computer won	You:0, Computer:1, Ties:0
Rock	Paper	Computer won	You:0, Computer:2, Ties:0
Paper	Scissors	Computer won	You:0 , Computer:3, Ties:0
Paper	Paper	Computer won	You:0, Computer:4, Ties:0
Scissors	Scissors	Computer won	You:0, Computer:5, Ties:0
Scissors	Paper	Tie	You:0, Computer:5, Ties:1
Non-default strength: Human 4, Computer 1			
Rock	Paper	Human won	You:1, Computer:0, Ties:0
Rock	Paper	Human won	You:1, Computer:0, Ties:0
Paper	Scissors	Human won	You:3, Computer:0, Ties:0
Paper	Scissors	Human won	You:4, Computer:0, Ties:0
Scissors	Paper	Human won	You:5, Computer:0, Ties:0
Scissors	Rock	Human won	You:6, Computer:0, Ties:0
Non-default strength: Human 1, Computer 4			
Rock	Paper	Computer won	You:0, Computer:1, Ties:0
Rock	Scissors	Computer won	You:0, Computer:2, Ties:0
Paper	Scissors	Computer won	You:0, Computer:3, Ties:0
Paper	Paper	Computer won	You:0, Computer:4, Ties:0
Scissors	Scissors	Computer won	You:0, Computer:5, Ties:0
Scissors	Paper	Computer won	You:0, Computer:6, Ties:0

Design Choices and Changes:

As we implemented the class hierarchy and pseudocode for this project, it quickly became evident that a number of design changes were necessary.

For one thing, the original version of the code did not display what the computer picked. This was because the initial code did not have an easy way to determine what tool a Tool pointer pointed to. To resolve this, a new abstract method, printTool, was added, and Rock, Paper, and Scissors classes all implemented this method such that, when called, a string would print, displaying the type of tool of that specific object.

There was also a bug in the original version of the code, highlighted in the testing plan above. The RPSGame class will ask the user if they would like to play again. The results are saved between rounds, so if the user chooses to play again, the cumulative results would update throughout. While the cumulative results worked as expected, there were occasions when, after the user was prompted on whether or not they wished to play again, the code would start a new round without waiting for user prompt, or the code would simply terminate the game. It was discovered that this was due to the way that input validation was implemented - the code used a while loop on a boolean to determine if the user had input an appropriate input. However, this boolean (isOk) was uninitialized on declaration, resulting in indeterminate results. To fix this, isOk was initialized to false on declaration.

A design choice was made to have the computer simply select a random choice every time, for rock, paper, or scissors. However, because of this choice, it became necessary to run a test to ensure that the computer's picks were truly random. To do so, a separate testing utility was written (compile with “make test”, run with “./test”). This script has the computer pick rock, paper, or scissors 1000 times, and shows the frequency of each result. Running this program, it can be seen that the computer picks each choice about a third of the time.

Finally, a design choice was made to make the strength selections persist throughout the program. The user is given a choice at the beginning as to whether or not they'd like to use non-default strengths, for both the human tools, and the computer tools. However, after this round, the strengths are set for the rest of the game. To implement this, some logic was written to save the strengths for the first time that the game is run.