Design Document
Group Project

Casey Bennink
Vinh Dong
Ryan Ellis
Esther Fatehi
Stephanie Gritz
Scott Laverick
Samuel Weinzimer

Requirements:

Tool Parent Class :

Data members: int strength, char type

Functions: void setStrength(int), virtual bool fight(tool)

Sub-Classes:

Rock	Paper	Scissors
Type =r	type=p	type=s
setStrength(1)	setStrength(1)	setStrength(1)
bool fight(tool)	bool fight(tool)	bool fight(tool)

RPSGame Class:

Tool *p1
Tool *p2
Human_wins
Computer_wins
Ties

playRound() //Don't think these functions are requirements
computerTurn()
playerTurn()
printScore()
menu()

Design Overview:

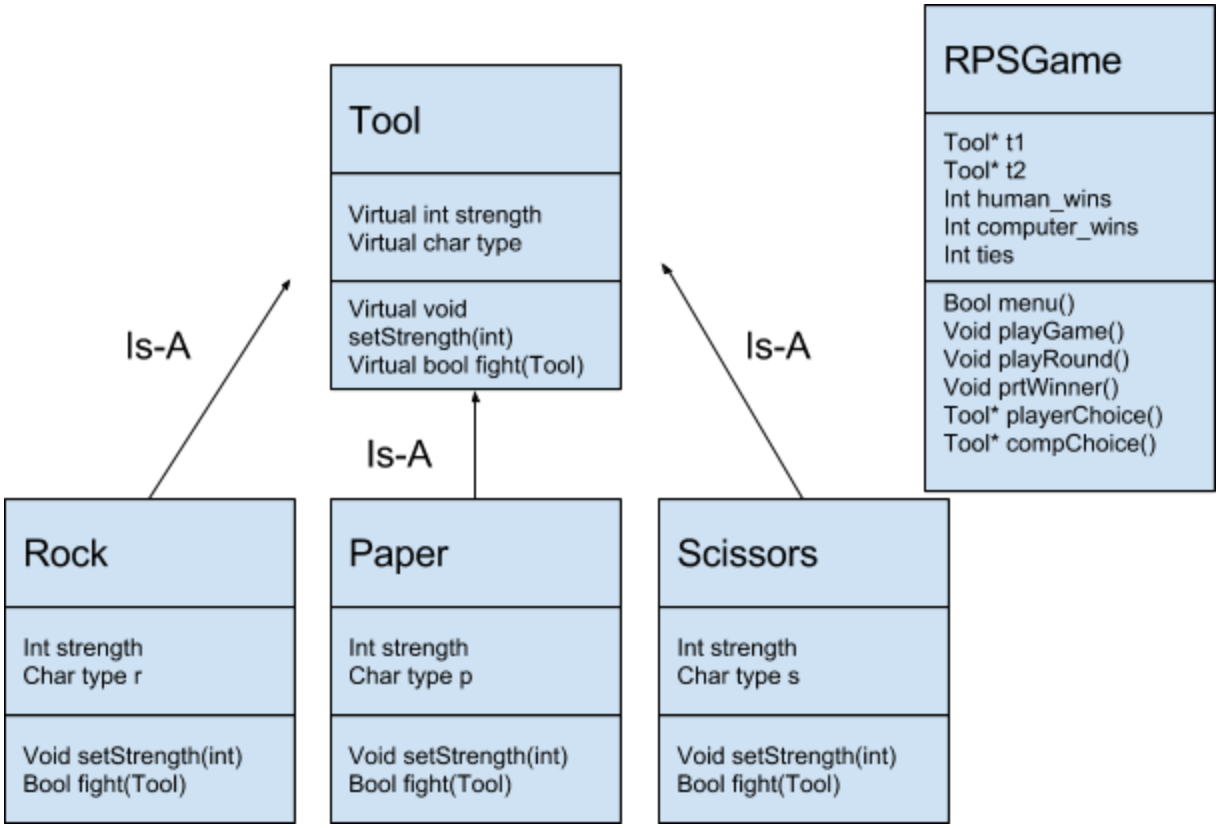
The workload is largely on two fronts: the tool hierarchy and the game class functions.
The tool hierarchy has an abstract tool parent class. Each sub-class will overload the fight(tool) function based on each tool's strengths and weaknesses. Each sub-class requires its own constructor.
The game class functions need to provide a user interface by which to play a game against the computer, probably via a menu function. It will need a function to determine what the computer

is picking (using rand and a switch to select the computer's tool), as well as a function to allow the user to select the tool he wants to use that round against the computer. Another function can facilitate calling the computer and player turn functions, call the fight(tool) function, and record the score of the round. Finally, a print() function would be a reasonable addition. The menu function could have as its options: play a game of RPS, print the current score, or exit the program.

Design Implementation:

- Tool Class (base class) with derived classes:
    - Rock Class
    - Scissors Class
    - Paper Class
- RPS class:
    - playGame function (possibly only a playRound function)
        - Design Choice: Play to a best of n, where n is a val set by the user?
        - How about play one round, then would you like to play another round? y/n
        - Just play one game at a time and keep a tally per instance?
    - compChoice and playerChoice functions
        - Input validation required for user choice
- Main:
    - Game Menu function
        - Input validation required for user choice
        -

Class Hierarchy Diagram:



Pseudo Code design
Tool
protected:
Int strength =1
Virtual int strength = 0
Virtual char type('r', 'p', 's') =0
Virtual void setStrength(int)
Virtual bool fight(Tool) = 0
Rock, Paper, Scissor (int strength)
int strength = 1;
type= 'r', 'p', 's';
Virtual int strength {return strength;}
Virtual char type {return type;}
Virtual void setStrength(int) {set strength to input}
Virtual bool fight(Tool) {
Compare strengths of the Human tool and computer tool
return 0 for loss or 1 for win;}

```
RPSGame(Tool *Human[], Tool *Computer[])
int human_wins = 0;
int computer_wins = 0;
int ties  = 0;
void Menu();
        int human_wins {return ++human_wins; }
        int computer_wins {return ++computer_wins; }
        int ties {return ++ties; }


Main.cpp
Create dynamic array for
Tool *Human[] = {new rock(1), new paper(1), new Scissor(1)}
Tool *Computer[] = {new rock(1), new paper(1), new Scissor(1)}
Create object RPSGame Game();
char type = 'r'   //set as initial value
While(char type !=e)
        Game menu
        Switch
                Case 1 : Ask user to set strength
                Case 2 : Ask user to to set computer strength
        Ask user to input char y
                Cin >> y;
        if y == 'r' {y=0}
                Cout << "User picked rock"
        else If y == 'p' {y=1}
                cout << "User picked paper"
        else If y == 's' {y=2}
                cout << "User picked scissor"
        Computer picks a tool at random,  int x = rand()%2
        bool h = Human[user input y]->fight(Tool Computer);
        bool c = Computer[random input x]->fight(Tool Computer);
If h ==1 && c==0
cout Human win!
        Game.human_wins
Else If h ==0 && c==1
cout Computer win!
        Game.computer_wins
else
cout << "Tie";
        Game.ties
cout
Game.computer_wins
```

Game.human_wins
Game.ties
} end while loop
Deallocate dynamic memory

Testing:

Test Case	**Input**	**Driver functions**	**Expected Outcome**	**Observed Outcome**
Rock (player) v. rock(computer)	Choice = r	game_play(); RPSGame();	Tie	Tie
Rock v. paper	Choice = r	game_play(); RPSGame();	Computer win count incremented	Computer win count incremented
Rock v. Scissors	Choice = r	game_play(); RPSGame();	Player win count incremented	Player win count incremented
Scissors v. paper	Choice = s	game_play(); RPSGame();	Player win count incremented	Player win count incremented
Scissors v. rock	Choice = s	game_play(); RPSGame();	Computer win count incremented	Computer win count incremented
Scissors v. scissors	Choice = s	game_play(); RPSGame();	Tie	Tie
Paper v. paper	Choice = p	game_play(); RPSGame();	Tie	Tie
Paper v. scissors	Choice = p	game_play(); RPSGame();	Computer win count incremented	Computer win count incremented
Paper v. rock	Choice = p	game_play(); RPSGame();	Player win count incremented	Player win count incremented
Player plays with augmented strength	Answer == y; humanStrength = 3; comStrength = 1;	game_play(); RPSGame();	Player will win every round	Player will win every round
Computer plays with augmented strength	Answer == y humanStrength = 1; comStrength = 3;	game_play(); RPSGame();	Computer will win every round	Computer will win every round
Player plays with augmented strength by 1	Answer == y; humanStrength = 6; comStrength = 5;	game_play(); RPSGame();	All normal play except human will win the tie	All normal play except human will win the tie

Both play with equally augmented strength	Answer == y;
	humanStrength = 3;
	comStrength = 3;
		game_play();
		RPSGame();
			Same likelihood of winning as normal powered rounds	Same likelihood of winning as normal powered rounds

User exits program	Choice = e	RPSGame play();	Program ends	Program ends