Samuel Chew
Riley Harrison
Patrick Huarng
Armand Parajon
Peter Yetti
Nathan Zimmerman
02/26/2017
CS162 400 W2017

Group Project Plan

Identify Problem

Simulate a multi-round game of rock paper scissors between a user and a computer. Track both user and computer score. Allow user to select their tool and strengths for all the tools. Choose a tool for the computer based on the user's past choices. Allow the user to control the game using a menu function.

Identify Inputs

Tool strengths, menu options, tool choices.

Identify Outputs

Win, loss, or tie. Total score.

Computer Choice Algorithm

The computer's choice in tool will be random for the first 3 rounds played. After the first 3 rounds, the computer will select their tool using a char vector recording past human tool choices. The index of the vector will be selected randomly. The choice for the computer will then be the opposite of the value at the selected vector index. An opposite value is simply the tool that beats the selected tool. If the user selects a certain tool more than the others, this method ensures that the computer choice will be weighted towards choosing the winning option vs that choice.

Fight Algorithm

The fight algorithm if called on the computer's tool will take as a parameter the user's tool choice. The function will create temporary strength variables for the function tool and the tool entered as a parameter. Conditional statements will then check the parameter tool type against the function tool type. The temporary strength variables will then be modified per game rules. If the tools are the same, the strength variables will not be modified. The two temporary strength variables are then compared. The higher value results in a win, the lower in a loss, and even strengths in a tie. The function returns a char representing a win, loss, or tie.

Program Design

The Main file will be known as play_game.cpp per project specs. The play_game.cpp file will start by instantiating an RPSGame class object using the classes default constructor. The default constructor will ask the user if they want to change the strength of any of the tools. The constructor will then set the tool strengths to those values. A while loop will then call the RPSGame object's playRound function until the function returns false.

In the playRound function a menu function will be called. The menu will have four options. The first being to select rock, the second to select paper, the third to select scissors, and the last to select exit. A conditional statement will check if the exit value was chosen. If the exit value was not chosen, then a Tool object will be dynamically created using the RPSGame Tool* humanTool data member. The constructor for the tool will contain type and strength as parameters. The computer choice algorithm will then be executed to assign a type value for the computer. The RPSGame Tool* computerTool data member will then be used to create a new object for the computer. The RPSGame class vector data member will record the human's choice as a new element. The object's constructor will contain the result of the computer choice algorithm as well as the appropriate strength value data member from RPSGame.

The Tool class virtual fight algorithm will then be called on either the computer or the human tool. The fight algorithm will return a char representing a win, loss, or tie. Either the human_wins or the computer_wins variable will then be incremented depending on the return value for the fight function. A print statement will display the results to the user. The playRound function will then delete memory associated with the dynamically created objects. The function will by default return true. If the exit value was chosen it will return false. After the while loop in the play_game.cpp a final score will be printed to the screen.

Pseudocode

algorithm: play_game.cpp

Test: If exit character is selected, program exits.

```
Instantiate RPSGame object
Set gameExit variable to false
while gameExit returns false
        gameExit is equal to RPSGame playRound
print message saying the game has ended
print final scores
```

Note: For the fight algorithm, the general structure is represented below. For the rock and scissor version of the function the tool names will need to be switched based on the standard rules of rock paper and scissors.

algorithm: Paper::fight

Test: User choice accurately reflects a win, loss, or tie.

```
Create temp paper strength variable and set to strength data member
Create temp parameter strength and set to strength lookup value
Create outcome variable
If object parameter is of type rock
        Temp paper strength is equal to double that of parameter strength
        Compare the two temp strengths
                Outcome is the higher of the two temp strengths
If object parameter is of type scissors
        Temp paper strength is equal to half that of parameter strength
```

Compare the two temp strengths
Outcome is the higher of the two temp strengths
If object parameter is of type paper
Outcome is a tie
Return outcome

Algorithm: nextAIMove
Test: User input choices are reflected in the AI choices as the game goes on past round 4. Wins, losses, and ties are reflected accurately.

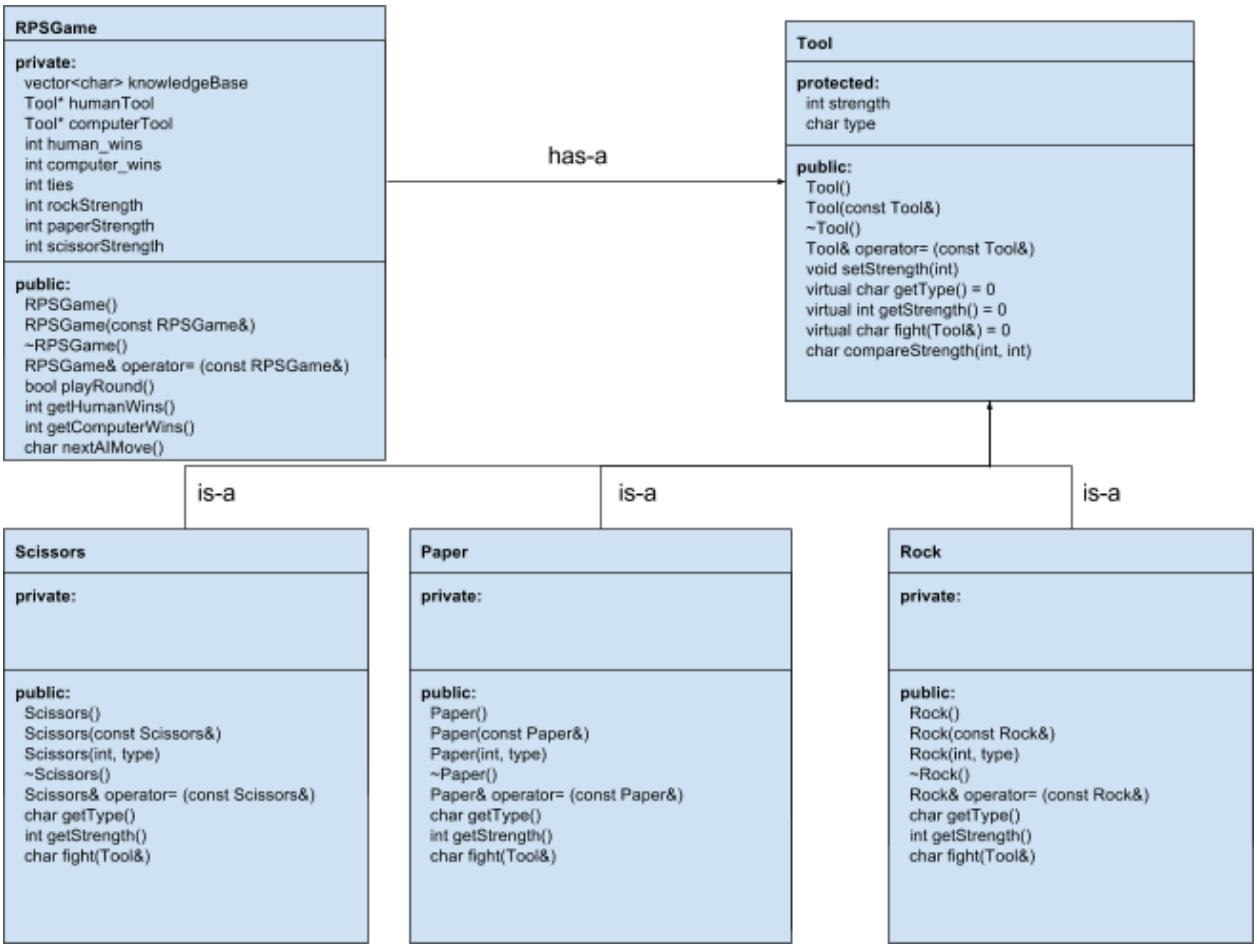
If the current round is less than 4
Pick a random tool
Return random tool
Else
Set index choice to a random number from 0 to array length
Set move equal to value of vector variable at set index
If the move value equals scissors
Set the move value to rock
If the move value equals rock
Set the move value to paper
If the move value equals paper
Set the move value to scissors
Return the move value

Algorithm: playRound
Test: User choice of tool is accurately reflected in the total win, loss, and tie scores.

If user chooses menu option for exit
Return false
else
if user chooses rock
Dynamically create a rock object
set rock strength to RPSGame rock strength
else if user chooses paper
dynamically create a paper object
set paper strength to RPSGame paper strength
else if user chooses scissors
dynamically create a scissor object
set scissor strength to RPSGame scissor strength
add user choice to knowledgebase vector
set a char variable value to result of nextAIMove function
if char variable is equal to rock
Dynamically create a rock object
set rock strength to RPSGame rock strength

else if char variable is equal to paper
    dynamically create a paper object
    set paper strength to RPSGame paper strength
else if char variable is equal to scissors
    dynamically create a scissor object
    set scissor strength to RPSGame scissor strength
call the fight function on the human tool object using the computer tool object as a parameter
if the function returns W
    increment human wins data member
else if the function returns L
    increment computer wins data member
else
    increment the ties variable
print who won and the current score to the screen
return true

Class Hierarchy Diagram



Testing Plan

Test Cases	Input Values	Driver Function	Expected Outcomes	Results
Tool, Rock, Paper, Scissor Unit Testing	Instantiate rock, paper, and scissor objects and test tool and inherited class methods.	(Tool*).getType() (Tool*).getStrength () (Tool*).getLongType() *Replace with Rock, Paper, and Scissor	'r', 'p', 's' 100, 100, 100 "Rock", "Paper", "Scissor"	'r', 'p', 's' 100, 100, 100 "Rock", "Paper", "Scissor"
Tool, Rock, Paper, Scissor Unit Testing	Test setters and getters instantiated tool object and inherited classes.	(Tool*).setStrength(200) *Replace with Rock, Paper, and Scissor	Rock.setStrength(200) Paper.setStrength(200) Scissor.setStrength(200)	200 200 200
Tool, Rock, Paper, Scissor Unit Testing P=Paper R=Rock S=Scissor	Test compare strength and fight methods	Init P/R/S with 100 P/R/S.fight(R/S/P) P/R/S.fight(P/R/S) P/R/S.fight(S/P/R) Init P/R/S with 50 P/R/S.fight(R/S/P) P/R/S.fight(P/R/S) P/R/S.fight(S/P/R) Init P/R/S with 24 P/R/S.fight(R/S/P) P/R/S.fight(P/R/S) P/R/S.fight(S/P/R)	Init P/R/S with 100 = 'w' = 't' = 'l' Init P/R/S with 50 = 'w' = 'l' = 'l' Init P/R/S with 24 = 'l' = 'l' = 'l'	Error: Scissor strength is being set to rock strength. Fix: Changed parameter in build tool function. Init P/R/S with 100 = 'w' = 't' = 'l' Init P/R/S with 50 = 'w' = 'l' = 'l' Init P/R/S with 24 = 'l' = 'l' = 'l'
Game menu testing, tool settings	Test validation of inputs and changes in tool strengths	RPSGame::toolSettingsMenu() From menu, select #2 to enter tool settings. Then test setting human and computer tools.	Human: 20, 30, 40 Computer: 10, 100, 5000	Error: Input validation catches valid input. Fix: Change input

Test Cases	Input Values	Driver Function	Expected Outcomes	Results
	for human and computer			validation parameters. Human: 20, 30, 40 Computer: 10, 100, 5000
Game menu testing, gameplay user interaction	Test validation of inputs and gameplay flow	RPSGame::playGame() From menu, select #1 and then play game to test correct game responses. Check against strings and invalid numbers.	" " Error, re-prompt 5 Error, re-prompt 1 Rock 2 Paper 3 Scissors 4 Exit Sends data to game engine correctly and evaluates win or loss correctly (default strengths)	" " re-prompts; 1 Rock (h), Scissors (c), human wins; 2 Paper (h), paper (c), tie; 3 Scissor (h), Rock (c), computer wins; 4 Exit;
Test artificial intelligence model and algorithm	Validate that frequency based approach to computer's win strategy works.	RPSGame::nextAIMove(int); RPSGame::playGame();	Use rock over and over again and confirm that computer is registering frequency and using scissor (use default strengths).	Error: Computer output scissors instead of paper. Fix: Change assignment in fight function to assign paper. Human -> Computer R -> P R -> P R -> R R -> P R -> P R -> P R -> P R -> P

Reflection

How did your design in this project change during implementation?

The design for our project changed to increase menu functionality and to reduce extraneous files. In regards to menu functionality, our initial design argued for a menu function to be created outside of the RPSGame class. Instead, the group decided to create three main menu functions as part of RPSGame. Menu functions are integral to the class, the group felt that encapsulating the functionality of the menus as member functions would help organize our files and tools in a more effective manner. User control was increased by adding three main menus as opposed to one. Adding menus allowed for custom tool settings of both computer and human tools. Also, a menu was added to allow the user to start a new game, or exit the program.

Making the menu functions a part of the RPSGame class also helped to streamline the groups file management. By incorporating the menu functions we reduced the total number of files to an amount that fit with the requirements outlined by the project spec. Housing the menu function in the RPSGame class made sense as it fit with the general description of what the abstract RPSGame class represented.

Lastly, changes were made to the data members of the RPSGame class, the Tool class, and the Tool derived classes. For the RPSGame class, data members were added to track strength settings for both the human and computer. Tracking the strength settings separately for both the human and computer helped to ensure that the user of the program was given maximum control over tool settings. Also, the playRound function was changed to playGame. Instead of returning a bool type that would serve to signify the end of the game, the playGame function was changed to incorporate a loop that continues until the user selects an exit number. By changing the playRound function, we avoided the need for a while loop in main, while keeping the essential purpose of the function the same. In the case of the Tool class, get methods were made non-virtual so that code was not repeated unnecessarily in the derived classes. In hindsight, our program may have been able to leverage a slightly different implementation of Tool Class methods, to simplify the fight/compare methods, but since we were a group of team members of varying availability, it would have been quite difficult to change from the originally documented plan. Managing these difficulties and differences in design choices has proven to be an integral part in the successful implementation of our project design.

What were the actual values from your testing? Did these match your expected values? What did you do to make sure you get the expected values?

As can be seen in the test table, our actual values matched our expected values in most cases. We did, however, encounter three errors through testing. One error involved incorrect parameters being set for our input validation function. Tests revealed that the input validation was not allowing values within a range that was considered correct. To fix the error we needed to change a parameter to accept a temporary strength value representing user input. Another error that was caught through testing was that our scissor strength value was being initialized with the current rock strength value. This error revealed itself through the output of our fight function. To correct this error, we simply changed a parameter from rock strength to scissor strength. Lastly in our nextAIMove function, one of our conditional statements was incorrectly assigning a Tool choice of scissors as the AI tool choice instead of assigning paper. To fix this we simply changed the value assigned as the result of the conditional statement to paper.

As can be seen, to ensure that our program output met expectations, our group tested each of our main functions. By checking program output we were able to make changes necessary to ensure that our program results met expectations. Planning played a key role in ensuring that we minimized the number of bugs that needed to be caught. Source code management and version control was also critical during the team development processes. Early on we tried to separate work activities as much as possible, but quickly discovered that the assignments of work divided out were dependent on each other. As a result, group members were often concurrently developing different or similar functionality. Adopting a workflow managed through GitHub helped to eliminate these conflicts early in the development process.