Anh Huynh
Danish Khan
Thanh Nguyen
Simon Altamirano

**Assignment 5: GUI-Based RPS Game**

Events in our Game Application

Our game application consists of 2 windows (frames): RPS_setup and RPS_Frame. RPS_setup class is used to retrieve configuration settings such as which CPU mode the player chose and the amount of rounds. For this class, we used an event table that consisted of the menu and buttons as seen in **Table A.1**. In **Table A.2**, we also used the slider to select rounds as an event, however, instead of including it in the event table, we decided to use the different method of "Connect" to connect it to our RPS_Setup frame.

```
wxBEGIN_EVENT_TABLE(RPS_Setup, wxFrame)
    EVT_MENU(wxID_ABOUT, RPS_Setup::OnAbout)
    EVT_MENU(wxID_EXIT, RPS_Setup::OnExit)
    EVT_BUTTON(buttonRandCPU_ID, RPS_Setup::OnClick_RandCPU)
    EVT_BUTTON(buttonSmartCPU_ID, RPS_Setup::OnClick_SmartCPU)
    EVT_BUTTON(buttonGeniusCPU_ID, RPS_Setup::OnClick_GeniusCPU)
    EVT_BUTTON(buttonSubmit_ID, RPS_Setup::OnClick_SubmitInfo)
wxEND_EVENT_TABLE()
```

Figure A.1: RPS_Setup Event table consisting of the top bar menu and buttons

```
Connect(ID_SLIDER, wxEVT_COMMAND_SLIDER_UPDATED, wxScrollEventHandler(RPS_Setup::OnScroll));
```

Figure A.2: RPS_Setup slider Event using "Connect" instead of the Table Method

Moreover, in the RPS_Frame Class, events were only declared in the Event table which consisted of the Menu top bar items (Restart and Exit options) and the Rock, Paper, and Scissors buttons for the User to press (See Figure 2).

```
wxBEGIN_EVENT_TABLE(RPS_Frame, wxFrame)
    EVT_MENU(ID_RestartGame, RPS_Frame::OnRestart)
    EVT_MENU(wxID_ABOUT, RPS_Frame::OnAbout)
    EVT_MENU(wxID_EXIT, RPS_Frame::OnExit)
    EVT_BUTTON(buttonRock_ID, RPS_Frame::OnClickRock)
    EVT_BUTTON(buttonPaper_ID, RPS_Frame::OnClickPaper)
    EVT_BUTTON(buttonScissors_ID, RPS_Frame::OnClickScissors)
wxEND_EVENT_TABLE()
```

Figure 2: RPS_Frame Event table consisting of Menu and Button Items

Callback functions handling the Events

The RPSmain class will create and show the RPS_Setup GUI once the code is compiled and executed. RPS_Setup is responsible for setting the game up with the Player's decision of which CPU mode to compete with and the amount of rounds. Each of these are events, which call a function and make an action such as setting a variable with a CPU Mode number from 1 to 3. Once both options have been set, the Player may press the submit button, which will be handled by the callback function (See **Figure C).** If the player does not choose a CPU mode, it will pop up a window letting them know, otherwise, the function will create the new frame sending the collected data. In addition, the default for rounds is 1 unless the player has chosen a certain amount, which would then call the slider callback function.

```
void RPS_Setup::OnClick_SubmitInfo(wxCommandEvent& event) {
    if(CPUMode == -1){
      wxMessageBox("Please Select CPU Mode!\n",
                   "No CPU Mode Selected", wxOK | wxICON_INFORMATION);
    }else{
      RPS_Game_Frame = new RPS_Frame("Rock-Paper-Scissors", wxPoint(50, 50), wxSize(450,
640));
      RPS_Game_Frame->Center();
      RPS_Game_Frame->Show(true);
      RPS_Game_Frame->set_config(slider->GetValue(), CPUMode);
      Close(true); // closes window
    }
}
```

Figure C: Snippet of the submit button function from RPS_Setup class

Finally, both RPS_Setup and RPS_Frame both have an "About" button on the drop down menu under the submenu "Help". If the above is clicked on, it will trigger the callback function to display the instructions on how to play the game and the three sets of rules as seen in **Figure D**.

```
void RPS_Frame::OnAbout(wxCommandEvent& event) {
    wxMessageBox("Welcome to Rock, Paper, Scissors!\n"
                 "\nThe game is simple, pick rock, paper, or scissors by entering a number
                                        between 1 to 3."
                 "\n\nThere are only 3 rules to win or lose: "
                 "\n\t1) Paper beats Rock"
                 "\n\t2) Scissors beats paper"
                 "\n\t3) Rock beats Scissors",
                 "About Hello World", wxOK | wxICON_INFORMATION);
}
```

Figure D: Snippet of the Callback Function for About

Modified, Refactored, and Reused Code

In order to connect the front end GUI with the back end functionality, the class Game needed some alterations since we personalized with our own version of GUI through the terminal. As seen in **Figure E**, the image demonstrates the printUI class depending on Game class to print our results of the rounds and total match. However, for this assignment we removed the printUI class completely since our GUI front end would be in charge of not only displaying statistics but also receiving user input. Our new UML class (see **Figure F**) demonstrates the new classes replacing the printUI class, which is the GUI presented to the user. For example, we added a new struct named "GameStats" which is metadata that is collected and returned to the RPS_Frame class each round. GameStats supplies the front end with: Round Winner, CPU Prediction, CPU move, CPU wins, Player Wins, and Total ties.

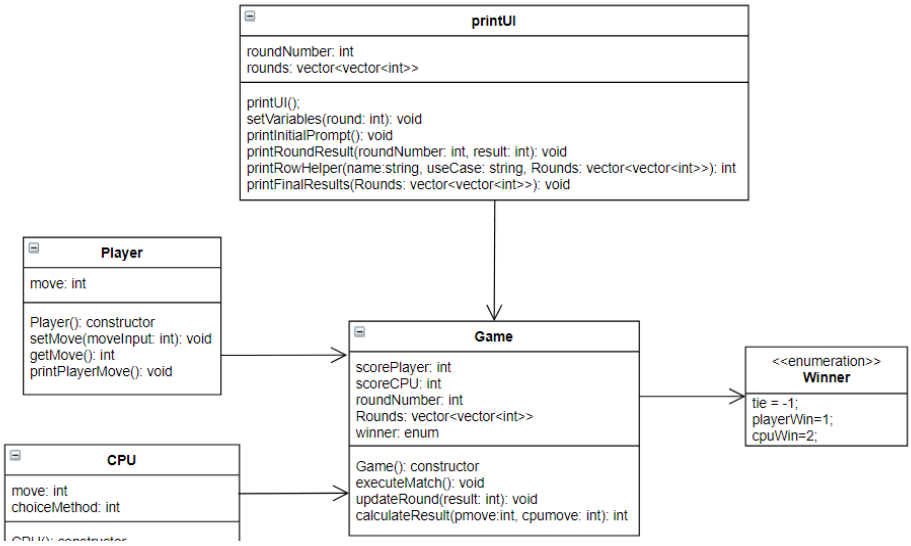


Figure E: Snippet of our old UML Diagram

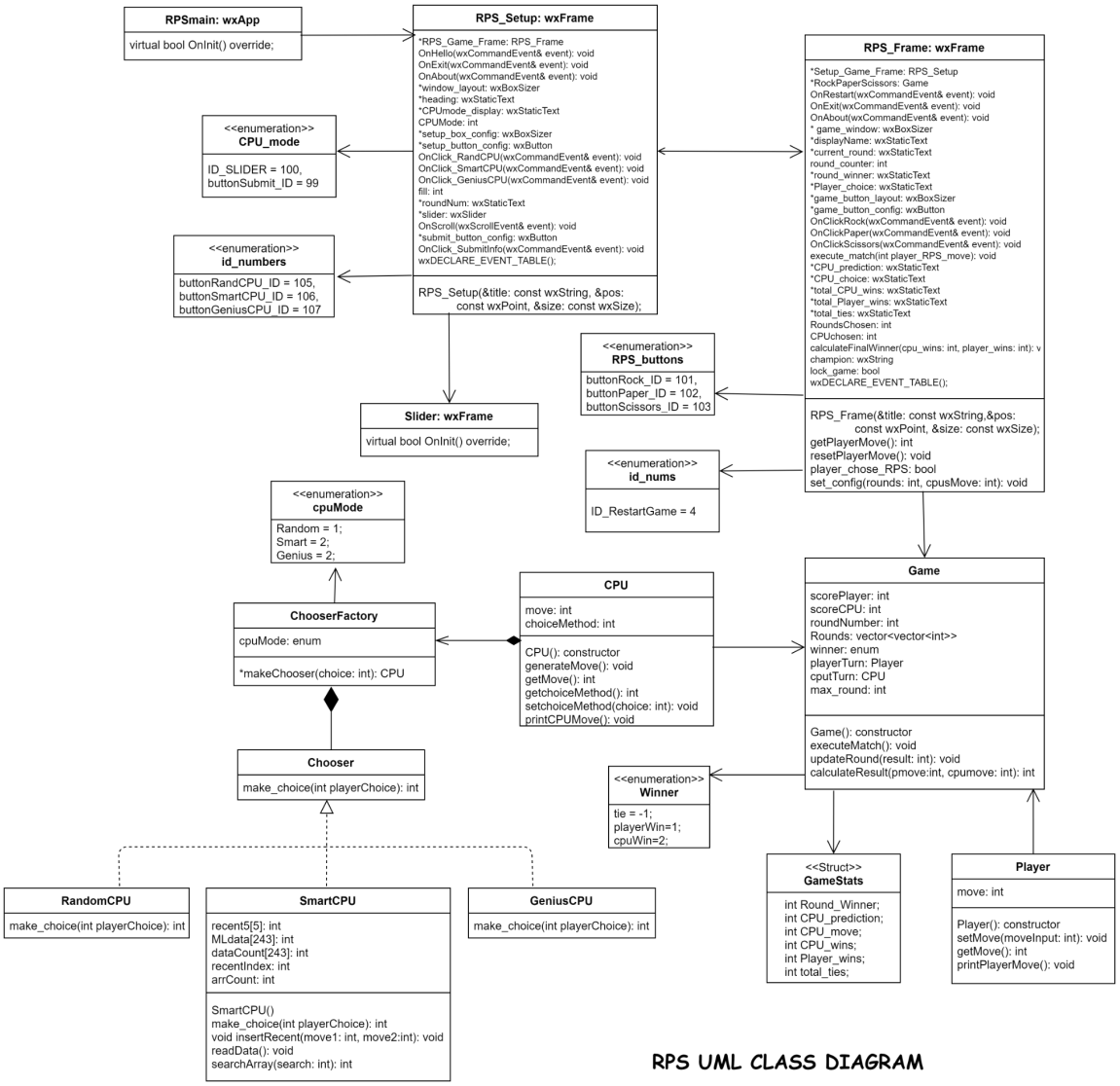


Figure F: New UML Class Diagram with GUI classes

In addition to our printUI class removal, we needed to modify the Game class in order to communicate with the GUI class of RPS_Frame. However, the class Game is not dependent on RPS_Frame. We used to have an inefficient function that did all the gaming work, but was reduced to being called each round while also making 2 helper functions and 1 setup function that establishes the CPU mode. As seen in **Figure G,** our new execute_Match() function not only is shorter and takes the player input as a parameter, but it also returns a metadata struct so that RPS_Frame is able to display all statistics. This change allowed for Game class to become more loosely coupled and organized. RPS_Frame calls each function as needed, and uses our new executeMatch to not only execute one round but also retrieve the statistics of the game at that moment. One we were finished debugging, we also removed all cout statements from Game class since the front end GUI would not display the information in the window the player uses.

```
GameStats Game::executeMatch(int playerInput) {
    GameStats Round_stats = {0};
    ofstream fileptr;

    playerTurn.setMove(playerInput);
    cpuTurn.generateMove(playerInput);

    int pmove = playerTurn.getMove();
    Round_stats.CPU_move = cpuTurn.getMove();
    Round_stats.Round_Winner = calculateResult(pmove, Round_stats.CPU_move);

    updateRound(Round_stats.Round_Winner, pmove, Round_stats.CPU_move);

    for(int i = 0; i < Rounds.size(); i++){
        if(Rounds[i][0] == 1){ //cpu wins
            Round_stats.CPU_wins++;
        }else if(Rounds[i][1] == 1){ //player wins
            Round_stats.Player_wins++;
        } else { //ties
            Round_stats.total_ties++;
        }
    }
    Round_stats.CPU_prediction = return_predict_move(Round_stats.CPU_move);
    return Round_stats;
}
```

Figure G: New refactored function for execute_Match();

Results of our User Interface

GUI implementation in **Figure H** demonstrates the configuration window when the app is first launched. The user must choose a CPU mode, otherwise an event will trigger a callback function to prompt the player with a warning window as seen in **Figure I**. The player is not required to choose a round amount since the default is 1, however, they may choose any round amount between 1 and 20.

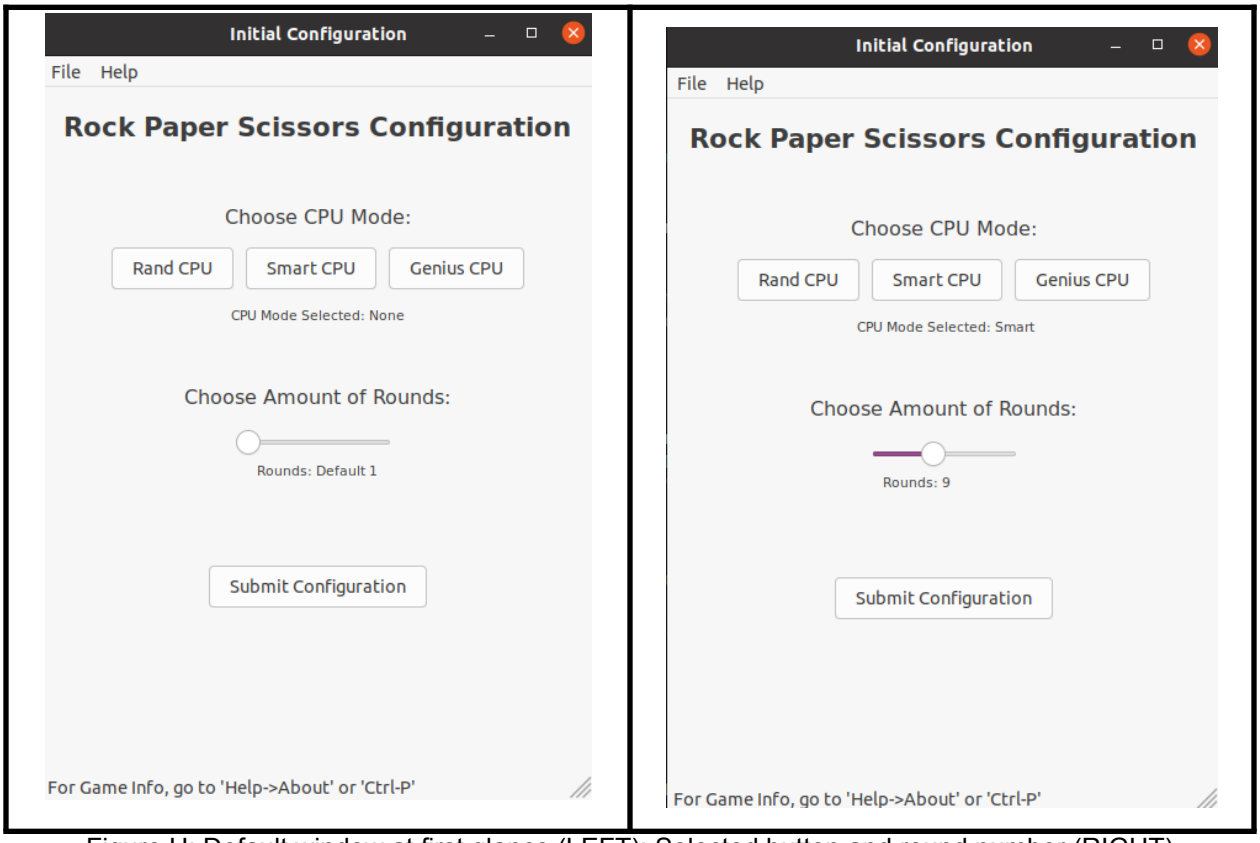


Figure H: Default window at first glance (LEFT); Selected button and round number (RIGHT)

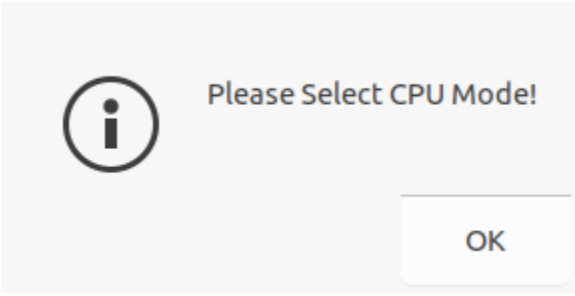


Figure I: Message Box when

Moreover, once the submit button is pressed and CPU mode has been selected, the player will be prompted with a Game window as seen in **Figure J**. Once the player starts the first round by choosing Rock, Paper, or Scissors button, the buttons will activate a callback function to run the round in the Game class which will return metadata of a struct variable to display results as. One the game ends, the player will be displayed with a message box displaying if the player or CPU won the game, or if it ended with a tie as seen in Figure K. The Message box will also allow the User to Restart the game by being rerouted to the Configuration window (see **Figure H**), or completely quit the game app.

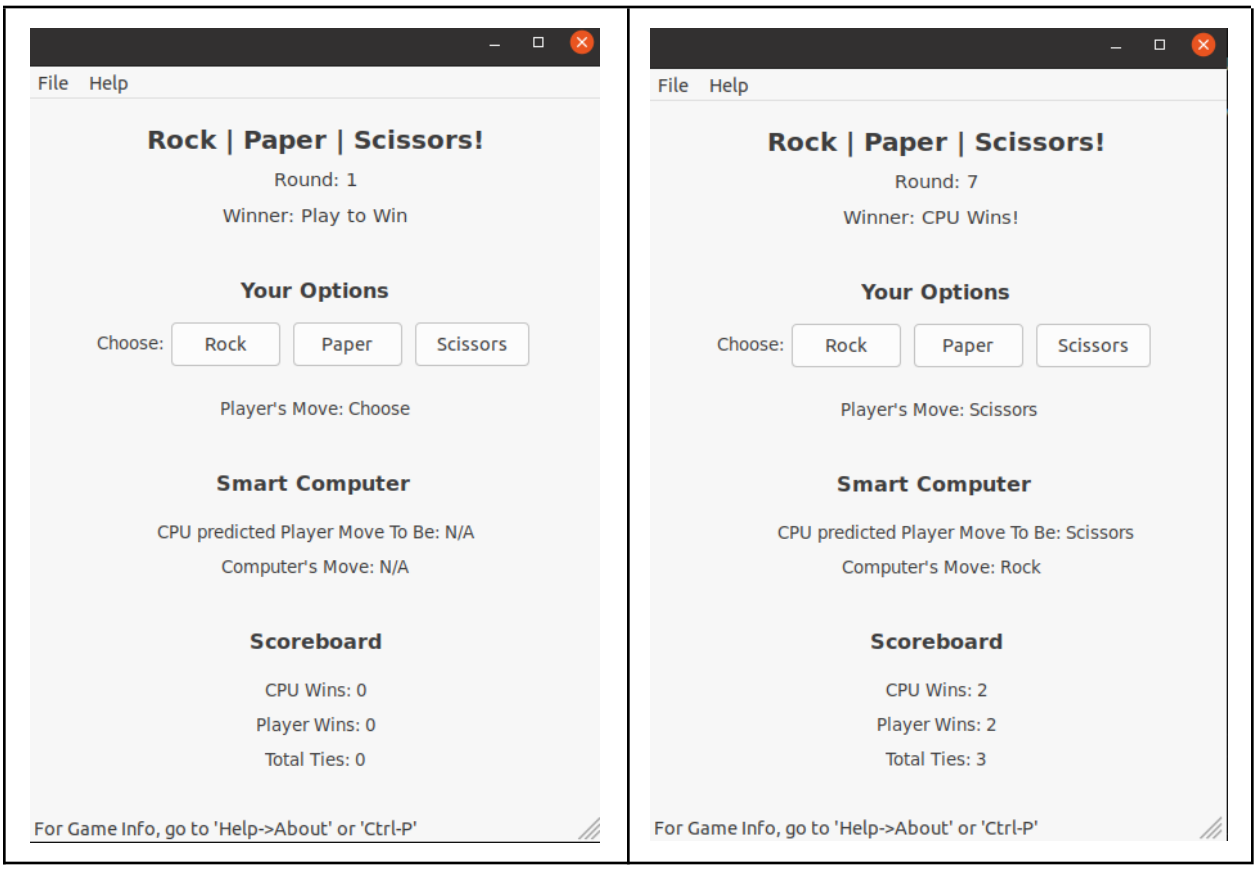


Figure J: Game Window when first prompted (LEFT); At round 7 with statistics and moves (RIGHT)

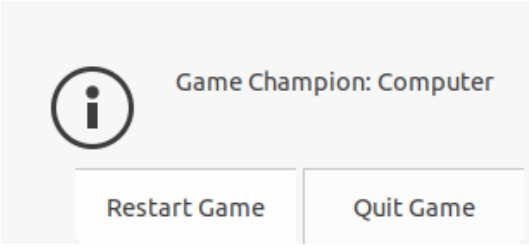


Figure K: Window displaying Game winner followed by 2 options, quit or restart game