Assignment 5 Rock Paper Scissors GUI

By: Tanishq Gadkari, Christian Consul, Kcirde Santos, Garret Russell

In terms of the general functionality of the assignment, the application was able to:

1) Display which round

2) A way for the user to enter a choice for each round.

3) The computer's choice for the round.

4) Who the winner is (or is it a tie) of the round.

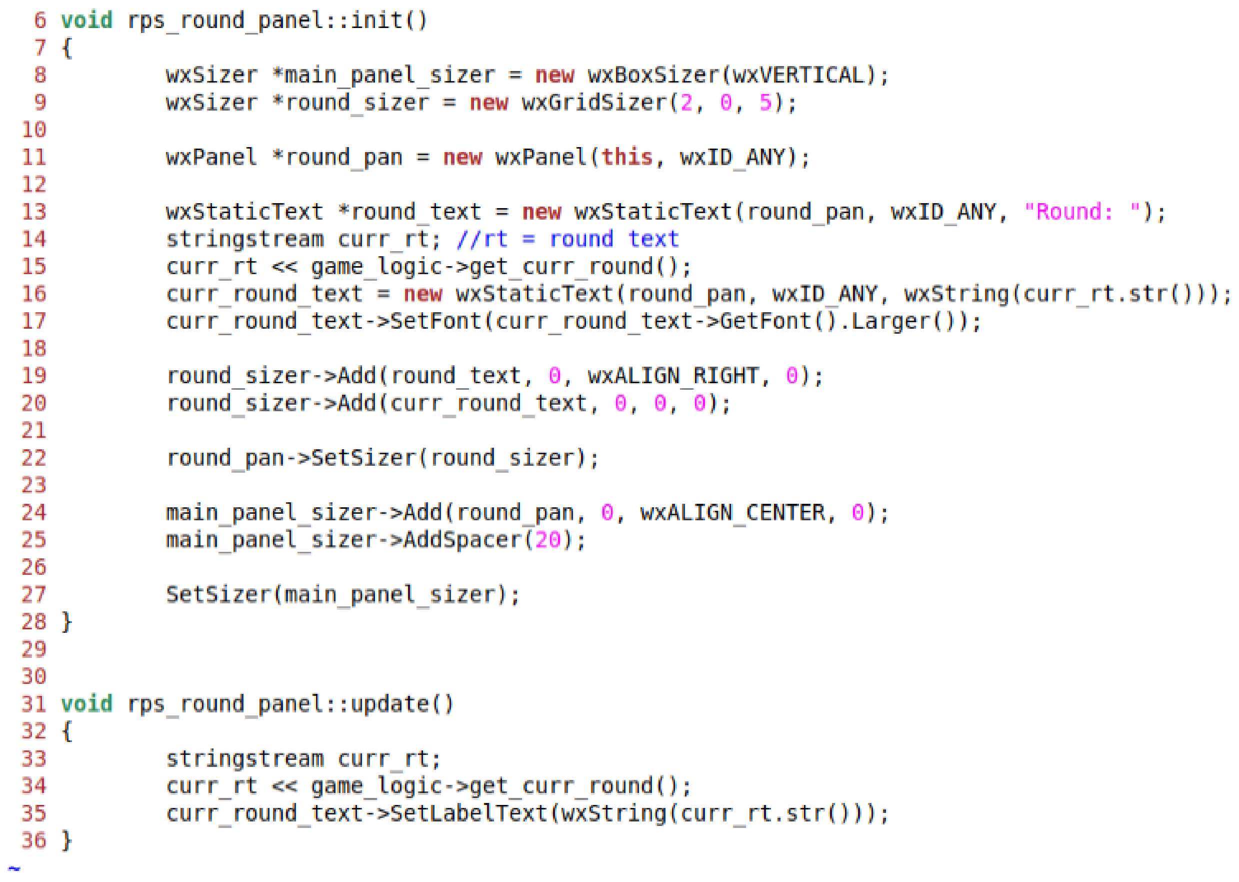
5) The number of human and computer wins, and the number of ties.

What events does your game application generate?

We have two menu events that are the standard implementation expected for about and exit. We also have three more events all related to the button event but the call back is based on the id of the button. These button events are generated when you press the respective button for rock, paper, scissors.

Displaying Rounds

The rounds update correctly, though for now we decided to have infinite rounds. We decided to make a round_panel class that will handle the rounds in itself. On the next page, you'll see the code snippet of our round_panel class. As you can see it's a simple two column grid with the second column being updated by the parent class on EVT_BUTTON assertion and updating the round respectively.

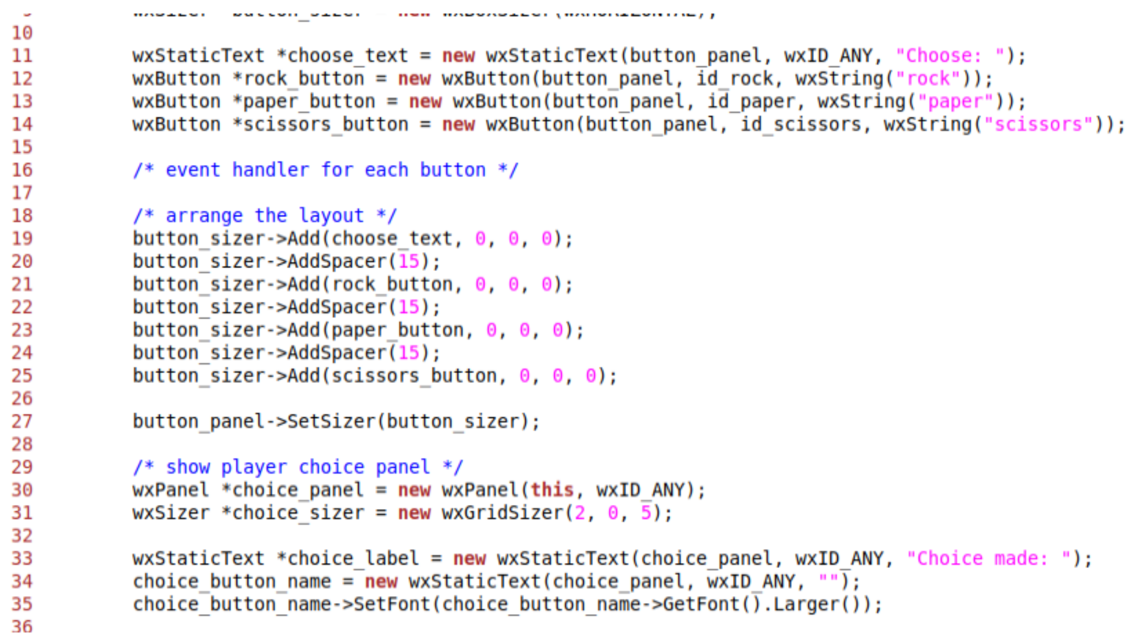


```
void rps_round_panel::init()
{
        wxSizer *main_panel_sizer = new wxBoxSizer(wxVERTICAL);
        wxSizer *round_sizer = new wxGridSizer(2, 0, 5);

        wxPanel *round_pan = new wxPanel(this, wxID_ANY);

        wxStaticText *round_text = new wxStaticText(round_pan, wxID_ANY, "Round: ");
        stringstream curr_rt; //rt = round text
        curr_rt << game_logic->get_curr_round();
        curr_round_text = new wxStaticText(round_pan, wxID_ANY, wxString(curr_rt.str()));
        curr_round_text->SetFont(curr_round_text->GetFont().Larger());

        round_sizer->Add(round_text, 0, wxALIGN_RIGHT, 0);
        round_sizer->Add(curr_round_text, 0, 0, 0);

        round_pan->SetSizer(round_sizer);

        main_panel_sizer->Add(round_pan, 0, wxALIGN_CENTER, 0);
        main_panel_sizer->AddSpacer(20);

        SetSizer(main_panel_sizer);
}


void rps_round_panel::update()
{
        stringstream curr_rt;
        curr_rt << game_logic->get_curr_round();
        curr_round_text->SetLabelText(wxString(curr_rt.str()));
}
```

Enter choice for each round

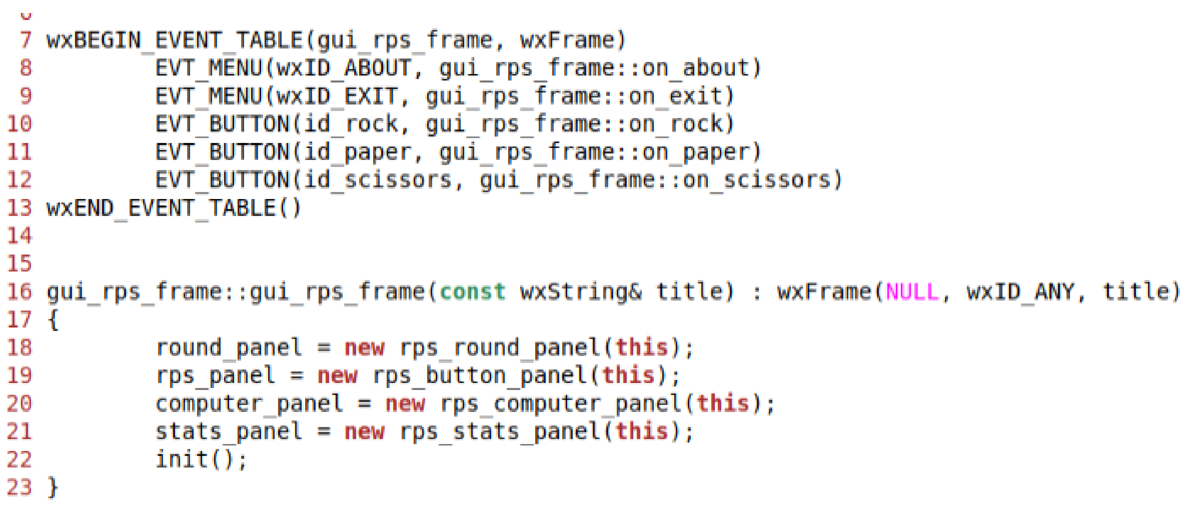
We used a panel to store all the buttons for user selection. Specifically, we created buttons for rock, papers, and scissors and configured the layouts. Note that this button panel is a child of the main frame and will generate a EVT_BUTTON event that will be propagated to our main frame. The way main frame knows when the button was pressed is through the id of that button. For instance, our rock, paper, and scissors have id of id_rock, id_paper, and id_scissors as enum values. When we create an event table in our main frame class the button clicked in button panel class will propagate to the main frame class and service the respective event based on if the player made a choice of rock, paper, or scissors. Below you'll see that these button have their IDs when creating an instance of wxButton and in our main frame table you'll see the same and IDs and their respective call back functions.

Inside of sources/rps_button_panel.cpp



```
        wxStaticText *choose_text = new wxStaticText(button_panel, wxID_ANY, "Choose: ");
        wxButton *rock_button = new wxButton(button_panel, id_rock, wxString("rock"));
        wxButton *paper_button = new wxButton(button_panel, id_paper, wxString("paper"));
        wxButton *scissors_button = new wxButton(button_panel, id_scissors, wxString("scissors"));

        /* event handler for each button */

        /* arrange the layout */
        button_sizer->Add(choose_text, 0, 0, 0);
        button_sizer->AddSpacer(15);
        button_sizer->Add(rock_button, 0, 0, 0);
        button_sizer->AddSpacer(15);
        button_sizer->Add(paper_button, 0, 0, 0);
        button_sizer->AddSpacer(15);
        button_sizer->Add(scissors_button, 0, 0, 0);

        button_panel->SetSizer(button_sizer);

        /* show player choice panel */
        wxPanel *choice_panel = new wxPanel(this, wxID_ANY);
        wxSizer *choice_sizer = new wxGridSizer(2, 0, 5);

        wxStaticText *choice_label = new wxStaticText(choice_panel, wxID_ANY, "Choice made: ");
        choice_button_name = new wxStaticText(choice_panel, wxID_ANY, "");
        choice_button_name->SetFont(choice_button_name->GetFont().Larger());

```

Inside of sources/gui_rps_frame.cpp



```
wxBEGIN_EVENT_TABLE(gui_rps_frame, wxFrame)
        EVT_MENU(wxID_ABOUT, gui_rps_frame::on_about)
        EVT_MENU(wxID_EXIT, gui_rps_frame::on_exit)
        EVT_BUTTON(id_rock, gui_rps_frame::on_rock)
        EVT_BUTTON(id_paper, gui_rps_frame::on_paper)
        EVT_BUTTON(id_scissors, gui_rps_frame::on_scissors)
wxEND_EVENT_TABLE()


gui_rps_frame::gui_rps_frame(const wxString& title) : wxFrame(NULL, wxID_ANY, title)
{
        round_panel = new rps_round_panel(this);
        rps_panel = new rps_button_panel(this);
        computer_panel = new rps_computer_panel(this);
        stats_panel = new rps_stats_panel(this);
        init();
}
```

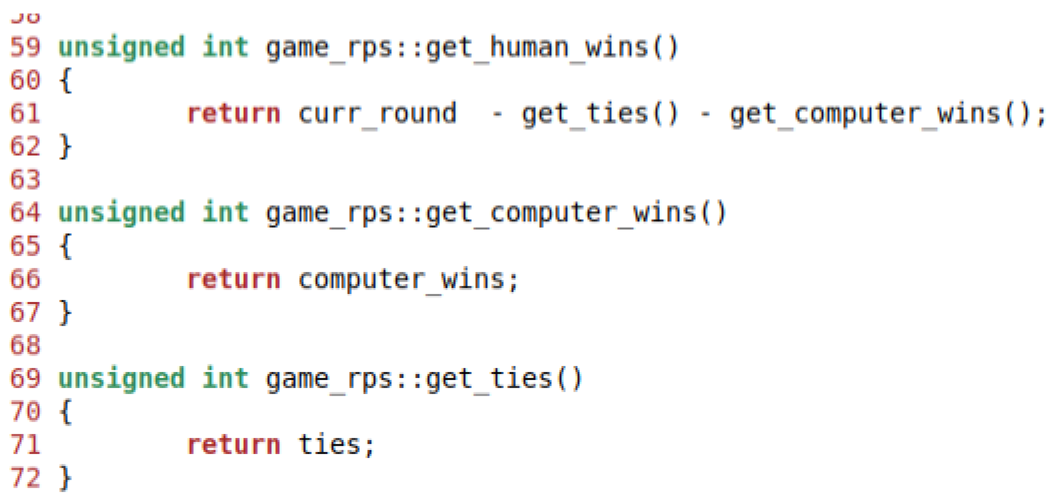
Computer Choice

The main functionality of this section is to simply display the computer choice. As stated before, we treat the computer as if it was a player. Specifically, from our code it is player 2. We are essentially just calling from our computer.h file to determine the choice and get the best choice from previous rounds.

This event is handled from the player choice button. When the button is pressed, it will call to player 2 which is the computer, and it will generate a computer output. Without getting into low level code logic, we have simplified our code enough to just call player functions which then calls to either user or computer. In this case, there is a function call to computer which will generate an output and check the sequence.

Who the winner is

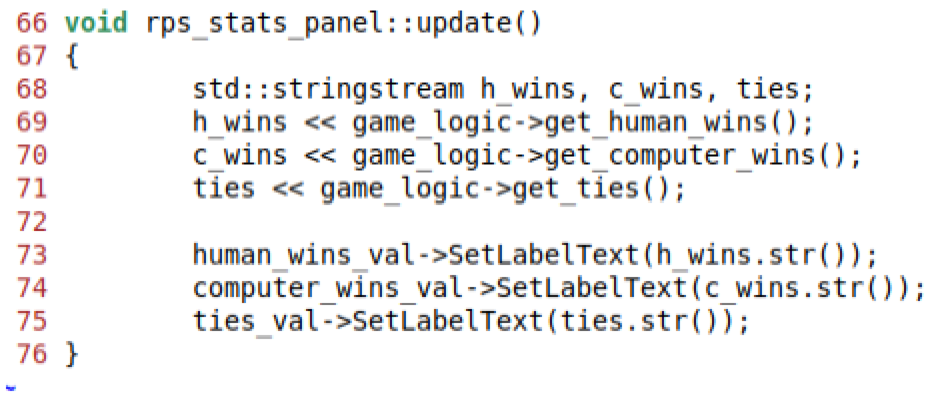
The winner is returned from game.cpp as an int variable, int game_rps::determine_winner(). The value being returned is converted in the GUI directly in stats through other methods such as get_human_wins(), get_computer_wins(), get_ties().



```cpp
unsigned int game_rps::get_human_wins()
{
        return curr_round  - get_ties() - get_computer_wins();
}

unsigned int game_rps::get_computer_wins()
{
        return computer_wins;
}

unsigned int game_rps::get_ties()
{
        return ties;
}
```

Number of human and computer wins

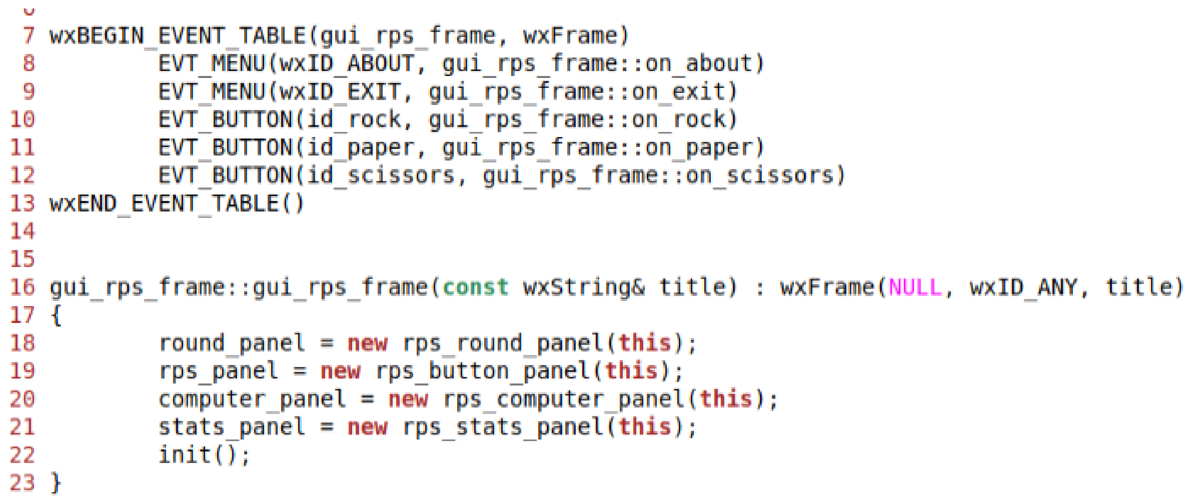
Again, we used methods inside of game_rps class to get the proper respective values and call them in our stats panel class when update is called by our main frame which is triggered on player making a choice by click on one of the buttons for rock, paper, scissors.



```cpp
void rps_stats_panel::update()
{
        std::stringstream h_wins, c_wins, ties;
        h_wins << game_logic->get_human_wins();
        c_wins << game_logic->get_computer_wins();
        ties << game_logic->get_ties();

        human_wins_val->SetLabelText(h_wins.str());
        computer_wins_val->SetLabelText(c_wins.str());
        ties_val->SetLabelText(ties.str());
}
```

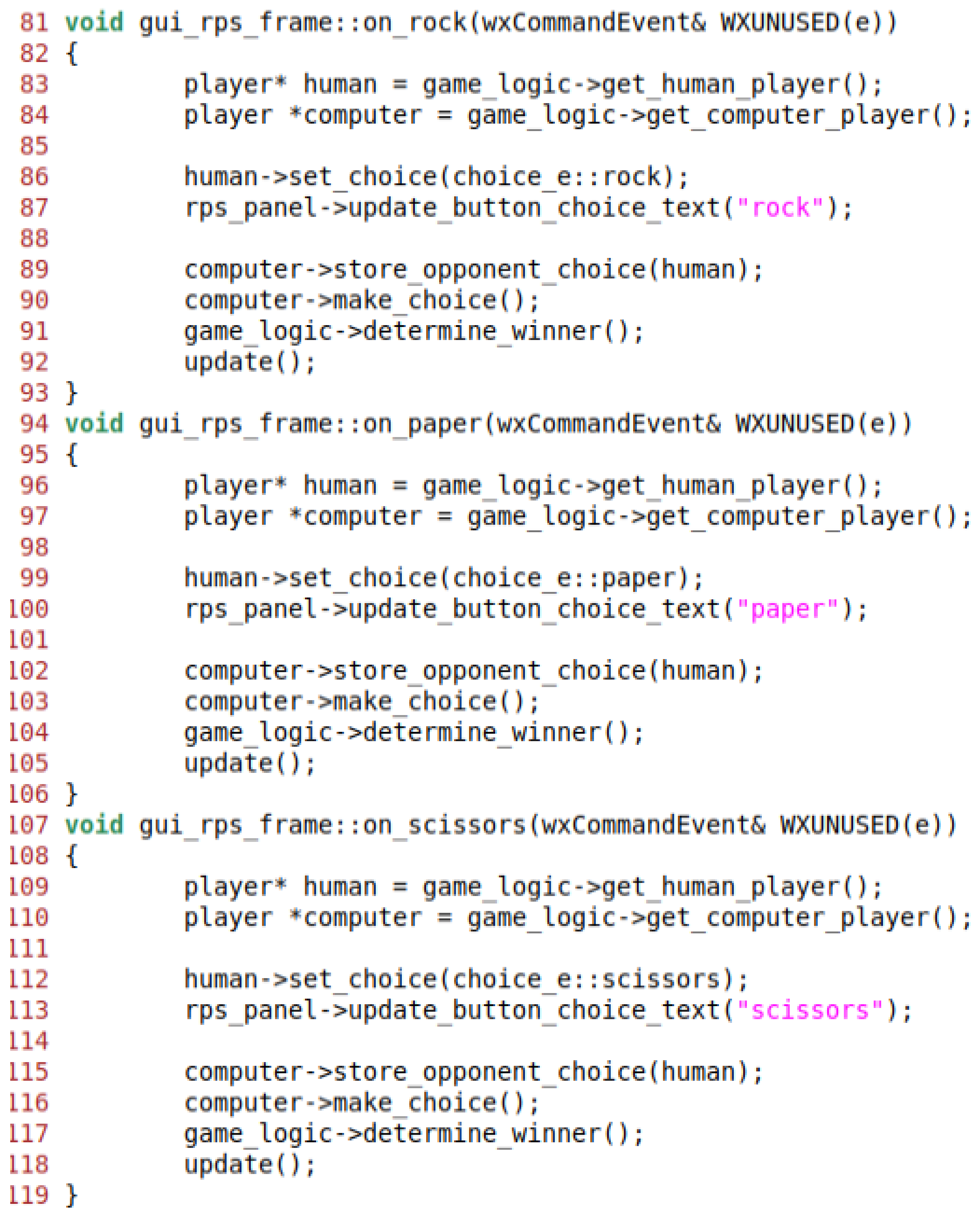
How did you use callback functions to handle the events?

Our events are handled through propagation of events emitted by our button panel to its parent which is the main frame. As shown before, this is reflected in our event table.



```
wxBEGIN_EVENT_TABLE(gui_rps_frame, wxFrame)
        EVT_MENU(wxID_ABOUT, gui_rps_frame::on_about)
        EVT_MENU(wxID_EXIT, gui_rps_frame::on_exit)
        EVT_BUTTON(id_rock, gui_rps_frame::on_rock)
        EVT_BUTTON(id_paper, gui_rps_frame::on_paper)
        EVT_BUTTON(id_scissors, gui_rps_frame::on_scissors)
wxEND_EVENT_TABLE()


gui_rps_frame::gui_rps_frame(const wxString& title) : wxFrame(NULL, wxID_ANY, title)
{
        round_panel = new rps_round_panel(this);
        rps_panel = new rps_button_panel(this);
        computer_panel = new rps_computer_panel(this);
        stats_panel = new rps_stats_panel(this);
        init();
}
```

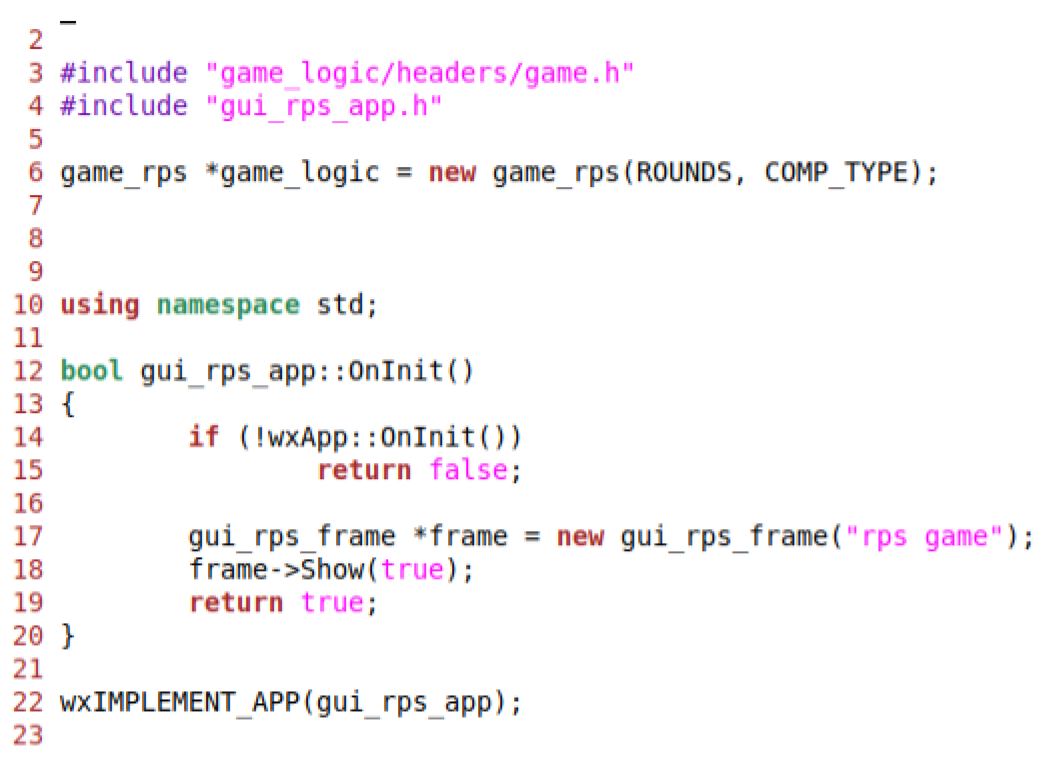
Each call back is similar for the button press event the main difference is the choice being made is reflected in it’s respective call back. See the figure below, on how similar each call is back.



```cpp
void gui_rps_frame::on_rock(wxCommandEvent& WXUNUSED(e))
{
        player* human = game_logic->get_human_player();
        player *computer = game_logic->get_computer_player();

        human->set_choice(choice_e::rock);
        rps_panel->update_button_choice_text("rock");

        computer->store_opponent_choice(human);
        computer->make_choice();
        game_logic->determine_winner();
        update();
}
void gui_rps_frame::on_paper(wxCommandEvent& WXUNUSED(e))
{
        player* human = game_logic->get_human_player();
        player *computer = game_logic->get_computer_player();

        human->set_choice(choice_e::paper);
        rps_panel->update_button_choice_text("paper");

        computer->store_opponent_choice(human);
        computer->make_choice();
        game_logic->determine_winner();
        update();
}
void gui_rps_frame::on_scissors(wxCommandEvent& WXUNUSED(e))
{
        player* human = game_logic->get_human_player();
        player *computer = game_logic->get_computer_player();

        human->set_choice(choice_e::scissors);
        rps_panel->update_button_choice_text("scissors");

        computer->store_opponent_choice(human);
        computer->make_choice();
        game_logic->determine_winner();
        update();
}
```

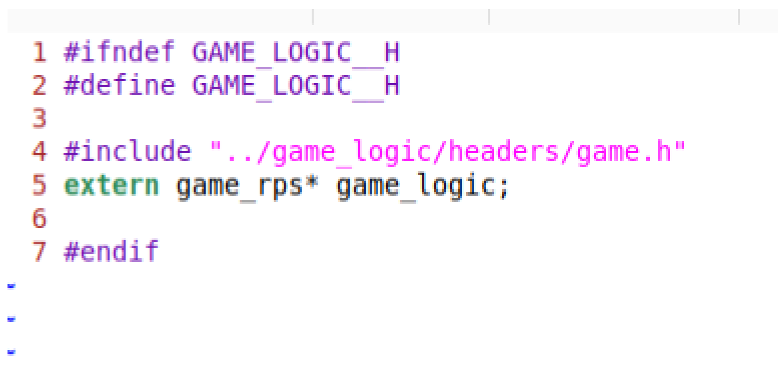
Explain how you were able to reuse code from Assignment #4 now that you have inversion of control. What changes were needed to your design?

We decided to make our game_rps instance a global variable and be shared amongst all panels. This allows for easy access to attributes inside our game_rps which also holds our players. Game_rps itself is instantiated once in our main app file. Note line 6. Then we have a special header file called game_logic.h that is included in any and every class that is dependent on updating respective attributes and calling respective methods. Note the following figure after.



```cpp
_

#include "game_logic/headers/game.h"
#include "gui_rps_app.h"

game_rps *game_logic = new game_rps(ROUNDS, COMP_TYPE);



using namespace std;

bool gui_rps_app::OnInit()
{
        if (!wxApp::OnInit())
                return false;

        gui_rps_frame *frame = new gui_rps_frame("rps game");
        frame->Show(true);
        return true;
}

wxIMPLEMENT_APP(gui_rps_app);

```

Inside of headers/ folder



```cpp
#ifndef GAME_LOGIC__H
#define GAME_LOGIC__H

#include "../game_logic/headers/game.h"
extern game_rps* game_logic;

#endif
~
~
~
```