

## Project Sprint 3 John Chirpich

Github link: <https://github.com/johniscool1/cs-449-project>

### 1. Demonstration

Youtube/Panotopo link: Link

### 2. Sumamry of Source Code

Source Code file name	Production or testcode?	# of lines
main.cpp	pro	16
screen_def.hpp	pro	120
screen_def.cpp	pro	431
game_logic.hpp	pro	102
game_logic.cpp	pro	516
unit_tests.cpp	test	323
	total	745

### 3. Production Code vs User stories/Acceptance Criteria

User Story ID & name	AC ID	Class Name(s)	method Name(s)	Status	Notes
1 Choose a board size	1.1-2	GameBoard	GameBoard::Set-BoardDimen-sions, MM-counter_check	done	Both AC 1,1 & 1.2 Involve the same classes and methods
2. Choose the game mode of a chosen board	2.1-2	GameLogic	GameL-ogic::setGameM-ode, playGame-ButtonCB	done	Both AC 2.1 & 2.2 Involve the same classes and methods
3. Start a new game of the chosen board and game size	3.1	GameBoard, GameLogic	GameBoard::Set-BoardDimen-sions, Game-Board::DrawBut-tons, playGame-ButtonCB, GameL-ogic::setGameM-ode	done	
4. Make a move in a simple game	4.1	GameBoard, GameLogic	Game-Board::Game-BoardButton-Pressed (uses data from game logic)	done	
	4.2	GameBoard	Game-Board::Game-BoardButton-Pressed	done	By using toggle buttons, the method deacti-vates them so that they cant be clicked again

User Story ID & name	AC ID	Class Name(s)	method Name(s)	Status	Notes
5. A simple game is over	5.1	GameBoard, GameLogic	GameLogic::SequenceFinder, CheckOutcome	done	
	5.2	GameBoard, GameLogic	GameLogic::SequenceFinder, CheckOutcome	done	
6. Make a move in a general game	6.1	GameBoard, GameLogic	GameBoard::GameBoardButtonPressed (uses data from game logic)	done	
	6.2	GameBoard, GameLogic	GameBoard::GameBoardButtonPressed, GameLogic::SequenceFinder	done	
	6.3	GameBoard	GameBoard::GameBoardButtonPressed	done	By using toggle buttons, the method deactivates them so that they cant be clicked again
7. A general game is over	7.1-3	GameBoard, GameScreen	GameBoard::SequenceFinder, GameBoardButtonPressed	done	

#### 4. Tests vs User stories/Acceptance Criteria

*Class Names Have been left out of the table because classes were not used in test code.*

Tests 2.1 and 2.2 were ai generated, screenshots are at the end.

User Story ID and Name	AC ID	Method	Description(expected I/O)
1 Choose a board size	1.1	TEST_CASE("ID 1.1: Choose Gameboard Size is > 3")	Takes the counters and sets the value like a user would, and passes it to the Main Menu Counter check that checks if it is valid.
	1.2	TEST_CASE("ID 1.2: Choose Gameboard Size is < 3")	Like the last one, it takes the counters and sets one of them to an incorrect value. (due to how the counter work, only one of them can be set at a time). If a counter's value is < 3, an error will pop up and after the user acknowledges, the counter is set to 3. NOTE: Due to the function calling the popup, this test will display that popup, but you just have to press escape or close the popup
2. Choose the game mode of a chosen board	2.1	TEST_CASE("ID 2.1 User presses Simple Gamemode")	Simulates the radio buttons a user uses on the main menu to select a gamemode. Then Uses a class setter to set the gamemode. I am unable to use the whole function that contains this setter because it also creates the window for the gameboard. Although I created both buttons, the code only checks if the simple GM one is checked or not.
	2.2	TEST_CASE("ID 2.2 User presses General Gamemode")	Same as the last one but checks for the general gamemode.
3. Start a new game of the chosen board and game size	3.1	This one combines both of the two previous tests, as any further functions heavily involve the GUI and I wanted to try to make these as automated as possible.	

User Story ID and Name	AC ID	Method	Description(expected I/O)
4. Make a move in a simple game T}	4.1	TEST_CASE("ID 4.1 & 6.1 Player places a S or O")	This test revolves around the "GameBoardButton-Pressed" callback which runs when a user presses a button on the gameboard. First we setup all the data we need for the CB, create a button, set the piece to S and then run the callback. WE then check the Game-Logic class, which has a vector that stores where the palyers has played, and checks if the piece was registered.
5. A simple game is over	5.1	TEST_CASE("ID 5.1 & 6.1 Simple Game is over, someone wins")	Creates a gameboard and creates a sos sequence and checks if the game ends and player 1 won.
	5.2 & 5.3	TEST_CASE("ID 5.2 Board is filled & 5.3 Declare a draw in simple game")	Fills up the board with all s, creating a filled board and a tie. Check if the game ended and both players points are 0
6. Make a move in a general game TEST_CASE("ID 4.1 & 6.1 Player places a S or O")	6.1 Becuase scoring and sequence finding has not been fully implemented, this AC shares a test with 4.1 becuase the gameplay is the same in both gamemo-des until a sequence is formed. 6.2	TEST_CASE("ID 5.1 & 6.1 Simple Game is over, someone wins")	If the a sequence is formed, than they win, so this test accomplishes both AC.
7. A general game is over	7.1 & 7.3  7.2	TEST_CASE("ID 7.1 Board is filled & 7.3 Declare a draw in general game")  TEST_CASE("ID 7.2 Declare the winner")	Fills up the board with all s, creating a filled board and a tie. Check if the game ended and both players points are 0 Fills up board with S except for the 3rd column, were the an sos sequence is scored. WE then cehck if the sequence was found and if it was scored.

**5. Describe how the class hierarchy in your design deals with the common and different requirements of the Simple Game and the General Game?**

My program currently only uses class hierarchy in one place and that is with the Parent class GameLogic with 2 children: SimpleGamemode and GeneralGamemode. Since the game-play is the same between the 2 gamemodes until someone scores, each of the parent function holds all the data needed to run the game such as what spaces have been played and whos turn it is. It also holds the score, but that is only really important for the general gamemode. The differences between the two gamemodes come when someone forms a sequence. In the simple Gamemode, the game ends, and for the General game, the game continues and the player gets another turn.

In order to accomplish this, the GameLogic class has a virtual function called "CheckOutcome". The function is then defined in the children classes GeneralGamemode and Simplegamemode. For GeneralGamemode, it checks if the board is filled, and for the simple gamemode, it checks if there was a sequence from SequenceFinder() or if the board has been filled. Having this virtual function allows me to call the CheckOutcome in the GameBoardButtonPressedCB callback and not have to check what the gamemode is and rewriting it to check for the gamemode.

You are a c++ developer. I need you write a unit test using catch2 for c++ to test the following user acceptance criteria (is is formatted as a groff table)

```
T} 2.1 T{
```

AC 2.1 User presses the simple game mode

.br

Given a user chooses the simple game mode

.br

When a user presses the simple game mode button

.br

Then initialize the simple game mode

This is the method that calls the radio button:

```
void game_main_menu()
```

```
{
```

```
    Fl_Double_Window* GMainMenu = new Fl_Double_Window(700,  
700, "SOS GAME");
```

```
    Fl_Box* MM_text_box_title = new Fl_Box(50,40, 600, 200, "SOS  
GAME");
```

```
    MM_text_box_title->labelsize(100);
```

```
    //counters for gameboard size
```

1. **Proper Initialization of the GameLogic Class:** Ensure that `GameLogic` has a method `setGameMode()` and a member `GameMode` that you can check after setting it.
2. **Memory Management:** FLTK uses widgets that don't need `delete` if they are placed within a window (FLTK will manage them), but since this is a unit test and you're not attaching the buttons to a window, `delete` is necessary here.

Here's an enhanced version of the test:

```
cpp Copy code

TEST_CASE("ID 2.1 User presses Simple Gamemode", "[GameMode]") {
    // Initialize game logic
    GameLogic* GameData = new GameLogic;

    // Create the radio buttons for game modes
    FL_Round_Button* SimpleGameModeRB = new FL_Round_Button(200, 300, 70, 20, "Simple");
    SimpleGameModeRB->type(FL_RADIO_BUTTON);

    FL_Round_Button* GeneralGameModeRB = new FL_Round_Button(410, 300, 70, 20, "General");
    GeneralGameModeRB->type(FL_RADIO_BUTTON);

    // Simulate user selecting the simple game mode
    SimpleGameModeRB->value(1);
    GeneralGameModeRB->value(0); // Ensure General mode is not selected

    // Call the game logic to set the mode based on the user's selection
    GameData->setGameMode(SimpleGameModeRB->value());

    // Check that the simple game mode was correctly set in GameLogic
    REQUIRE(GameData->GameMode == 1);

    // Clean up
    delete GameData;
    delete SimpleGameModeRB;
    delete GeneralGameModeRB;
}
```



How to test callbacks?

Any other edge cases?

**Key points:**

Now create a test case that detects if the gerneal gamemode has been choosen





To create a test case that checks if the **General Game Mode** has been chosen, we will follow a similar approach to your previous test for the Simple Game Mode. The main difference will be setting the value of `GeneralGameModeRB` to 1 and verifying that the `GameMode` in the `GameLogic` class is correctly set for the general mode.

Here's the test case for detecting the **General Game Mode**:

cpp

Copy code

```
TEST_CASE("ID 2.2 User presses General Gamemode", "[GameMode]") {
    // Initialize game logic
    GameLogic* GameData = new GameLogic;

    // Create the radio buttons for game modes
    FL_Round_Button* SimpleGameModeRB = new FL_Round_Button(200, 300, 70, 20, "Simple Game Mode");
    SimpleGameModeRB->type(FL_RADIO_BUTTON);

    FL_Round_Button* GeneralGameModeRB = new FL_Round_Button(410, 300, 70, 20, "General Game Mode");
    GeneralGameModeRB->type(FL_RADIO_BUTTON);

    // Simulate user selecting the general game mode
    GeneralGameModeRB->value(1);
    SimpleGameModeRB->value(0); // Ensure Simple mode is not selected

    // Call the game logic to set the mode based on the user's selection
    GameData->setGameMode(GeneralGameModeRB->value());

    // Check that the general game mode was correctly set in GameLogic
    REQUIRE(GameData->GameMode == 1); // Assuming GameMode == 1 indicates general

    // Clean up
    delete GameData;
    delete SimpleGameModeRB;
    delete GeneralGameModeRB;
}
```

Key points:

How to test callbacks?

How to mock FLTK components?

1. **General Mode Set:** We set `GeneralGameModeRB->value(1)` and `SimpleGameModeRB->value(0)` to ensure the general mode is selected.