Project Sprint 2
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	14
screen_def.hpp	pro	78
screen_def.cpp	pro	298
game_logic.hpp	pro	103
game_logic.cpp	pro	109
unit_tests.cpp	test	143
	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::SetBoardDimensions, MMcounter_check	done	Both AC 1,1 & 1.2 Invovle the same classes and methods
2. Choose the game mode of a chosen board	2.1-2	GameLogic	GameLogic::setGameMode, playGameButtonCB	done	Both AC 2.1 & 2.2 Invovle the same classes and methods
3. Start a new game of the chosen board and game size	3.1	GameBoard, GameLogic	GameBoard::SetBoardDimensions, GameBoard::DrawButtons, playGameButtonCB, GameLogic::setGameMode	Inprogress	Only thing left is that the program needs to do something if the board size is too big.
4. Make a move in a simple game	4.1	GameBoard, GameLogic	GameBoard::GameBoardButtonPressed (uses data from game logic)	inprogress	Placing an s or o works, but scoring is not implememented yet or checking for sequences
	4.2	GameBoard	GameBoard::GameBoardButtonPressed	done	By using toggle buttons, the method deactivates them so that they cant be clicked again

User Story ID & name	AC ID	Class Name(s)	method Name(s)	Status	Notes
6. Make a move in a general game	6.1	GameBoard, GameLogic	GameBoard::GameBoardButtonPressed (uses data from game logic)	inprogress	Placing an s or o works, but scoring is not implemented yet or checking for sequences
	6.2	GameBoard, GameLogic	GameBoard::GameBoardButtonPressed, GameLogic::SequenceFinder	InProgress	Sequence finder right now only checks vertically, and the score and line to indicate a score has not been implemented yet.
	6.3	GameBoard	GameBoard::GameBoardButtonPressed	done	By using toggle buttons, the method deactivates them so that they cant be clicked again

4. Tests vs User stories/Acceptance Criteria

Class Names Have been left out of the table beucase 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 whould, 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 hoe 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 acknoledges, 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 contians this setter becuase it also creates the window for the gameboard. Altough 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 hevailiy 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.
6.	Make a move in a general game	TEST_CASE("ID 4.1 & 6.1 Player places a S or O"	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 gamemodes until a sequence is formed. Future tests will test for what happens when sequences are formed

You are a c++ developer. I need to 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

```cpp
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, "Simp
    SimpleGameModeRB->type(FL_RADIO_BUTTON);

    Fl_Round_Button* GeneralGameModeRB = new Fl_Round_Button(410, 300, 70, 20, "Ger
    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

```cpp
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, "Simp
    SimpleGameModeRB->type(FL_RADIO_BUTTON);

    Fl_Round_Button* GeneralGameModeRB = new Fl_Round_Button(410, 300, 70, 20, "Ger
    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-`