MVC Raylib Frogs and Toads Assignment

Assignment A03

Welcome to your third C++ assignment. If you find this assignment confusing, please reach out to me and discuss it with your peers. This assignment is brand new. As such, it may not be as clear as it should be. DM me with your questions or book some time for a chat.

Please track how long each part takes to write for your reflection. Also, please submit a Visual Studio Project or CMake project for your program, not just the source code files.

Frogs and Toads

In class, your instructor reviewed a C++ implementation of the Frogs and Toads single-player puzzle game. This game was invented by a British Canadian mathematician with epic eyebrows named Richard K. Guy, who lived to the age of 103 and discovered the "Glider" in Conway's Game of Life. What a Guy!

Game Rules

The game is played on a linear board with N squares, where N is an odd number greater than three. In the variant we will be working with, N is equal to 7. All but the center square are filled with Frogs (F) to the left and Toads (T) to the right. The goal of the game is to get all the frogs and toads to switch places. You can **play** an online version of the game here.

The game is played by moving the frogs and toads, one at a time, into an empty square either by sliding into an adjacent square, or by hopping over a single occupied square into an empty one. Frogs can only move to the right. Toads can only move to the left. The player does *not* need to alternate between Frogs and Toads.

Here are the first few moves in a possible game.

The game begins in the starting state, frogs on the left, toads on the right:

F	F	F		T	T	T

Move 1: The first toad slides into the empty square in the middle:

F	F	F	T		T	T

Move 2: The first frog jumps over the first toad:

F	F		T	F	T	T

Play continues in this manner until all the toads are on the left and all frogs on the right.

The winning state of the board looks like this:

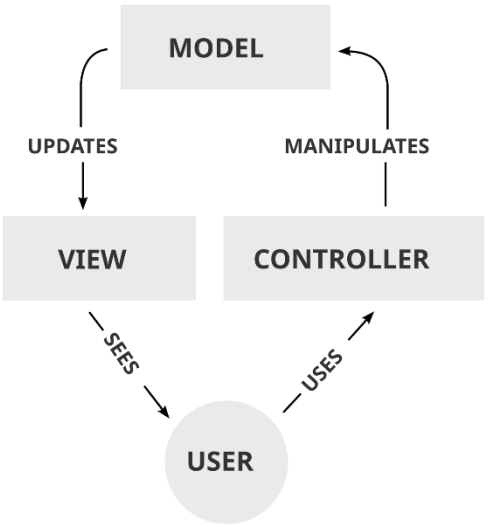
T	T	T		F	F	F

If no further moves are possible, and the winning state has not been reached, the player losses. An example of a losing state:

	F	F	T	F	T	T

Model View Controller

*From [the Wikipedia entry](): *



"**Model–view–controller (MVC)** is a software architectural pattern commonly used for developing user interfaces that divides the related program logic into three interconnected elements.

These elements are:

- the **model**, the internal representations of information
- the **view**, the interface that presents information to the user
- the **controller**, the software linking the two

One of the seminal insights in the early development of graphical user interfaces, MVC became one of the first approaches to describe and implement software constructs in terms of their responsibilities."

C++ Frogs and Toads

Your instructor has coded an MVC implementation of Frogs and Toads in C++ that includes:

- A single model class `BoardModel` to hold the board state, the move logic, and the win-state evaluation.
- An abstract class `IView` to act as an interface for views of the game.
- Two view classes that inherit from `IView`:
    - A console view that outputs text using standard output.
    - An ANSI view that outputs coloured text and redraws the board in place.
- An abstract class `IController` to act as an interface for game controllers.
- A single keyboard-input console controller that inherits from `IController`.
- Stubs for Raylib-based controller and view.
- A `main()` entry point that either runs the text-based game or runs a Raylib version of the game, depending on a `RAYLIB_BUILD` #define.
- My apologies for the fact that all the *.hpp and *.cpp files are currently all in the project root. They should really be organized into separate "include" and "src" sub-folders.

Getting Started

- Download the Visual Studio project here.
- Unzip the project to your computer.
- Open the RaylibFrogsAndToads.sln solution file.
- You may need to refresh the Raylib package using NuGet:
    - Open the "Tools" menu.
    - Select "NuGet Package Manager".
    - Select "Manage NuGet Packages for Solution...".
    - In the "Installed" tab select "raylib".
    - On the right-hand side click the checkbox beside "Project".
    - Click "Uninstall" and then "Apply".
    - In the "Browse" tab search for "raylib".
    - Select the "raylib" package below.
    - Ensure that all checkboxes are enabled on the right.
    - Click "Install" and then "Apply".
    - *Is there a better way than this to refresh the package?!?*
- Uncomment "#define RAYLIB_BUILD" near the top of the frog_and_toads.cpp file.
- Click the Visual Studio "Local Windows Debugger" play button.

- If everything goes well, you should see a terminal open with Raylib logging details along with a blank white GUI window.

Part 1 – A Raylib Controller

For this part of the assignment, you will begin adding Raylib support to the game, starting with a Raylib Controller. Begin by exploring the code for the console controller in `console_controller.hpp` and `console_controller.cpp` to see how it works. Like my console controller, your Raylib controller needs only support keyboard input. Specifically, keys "1" through "7" for moves and the letter "r" for resetting the game. Unlike the console controller, your Raylib controller should be non-blocking, meaning it shouldn't wait on input blocking the next frame from arriving.

Start by researching Raylib's Input Handling Functions in the Raylib Cheatsheet.

To test your new controller without an associated Raylib view, you can temporarily use the ANSI view that already exists. To do this, you will need to temporarily add the ANSI view header to the `RAYLIB_BUILD #ifdef` at the top of the `frog_and_toads.cpp` file:

```
#ifdef RAYLIB_BUILD
    #include "raylib.h"
    #include "raylib_controller.hpp"
    #include "raylib_view.hpp"
    #include "ansi_view.hpp"
#else // etc etc.
```

After that you can temporarily change the code in the `RAYLIB_BUILD` version of `main()` to start with:

```
    FrogToad::BoardModel model;
    FrogToad::AnsiView view;
    FrogToad::RaylibController ctrl;

    // Temporarily hard-coding the window height and width.
    InitWindow(300, 300, "Toads & Frogs");
```

Now you can test your controller with the Raylib GUI window focused, hitting keys while watching the text console window for output.

NOTE: Be sure to click on the Raylib GUI window before testing out your controller, otherwise your input will be ignored.

The class definition for the Raylib controller has provided for you in `raylib_controller.hpp`. Your job is to implement the `nextMove` method in `raylib_controller.cpp`. Remember that the model considers moves to be zero-indexed (0..6) but the user will be using 1-indexed positions (1..7).

In my version of the console and ANSI controllers, invalid moves are simply ignored.

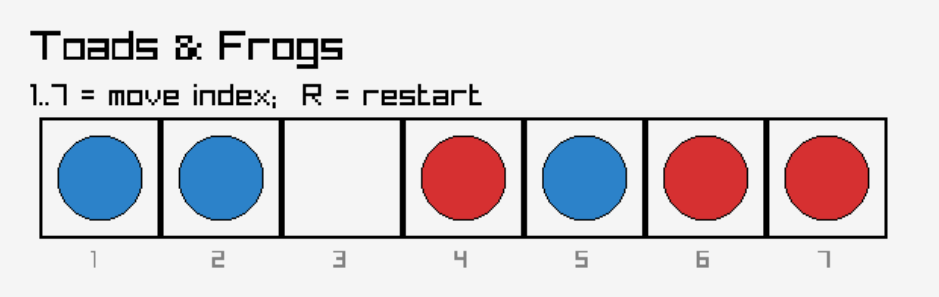
Part 2 – A Raylib View

It's now time to remove the ANSI view from the `RAYLIB_BUILD` version of `main()` and create your own Raylib view:

```
FrogToad::BoardModel model;
FrogToad::RaylibView view;
FrogToad::RaylibController ctrl;

// We are now querying the view for the window dimensions.
InitWindow(view.windowW(), view.windowH(), "Toads & Frogs");
```

Here's an example of what a very basic Raylib view might look like:



As with the Raylib controller, the class definition for the Raylib view has been made for you in `raylib_view.hpp`. Your job is to implement the `draw` method in `raylib_view.cpp`.

Note that the view has `windowW()` and `windowH()` methods that are used by `InitWindow` in `main()` to set the size of the GUI window. Currently these methods return hardcoded values, but you may change their implementations. You may also add additional methods to the `RaylibView` class.

There's an additional mark for an aesthetically pleasing and/or particularly elegant view.

Part 3 – A Way Out

Currently there's no way to end the game using the keyboard. You will now work on adding code to allow the user to quit the game by hitting the letter q on the keyboard. This functionality needs to work with both the console controller and the Raylib controller.

You may need to alter the signature of `nextMove` in the `IController` class such that it returns either a Boolean or a Class Enum, to give implementations of `nextMove` a way to communicate back to the `main()` loop.

Part 4 – Yikes a Mouse!

This part of the assignment might be tricky. I haven't coded it myself yet. As such, this should be considered a "Hard Mode" question. It is, however, worth marks. That said, it's still possible to get an A on the assignment if you do not implement mouse support.

For this part of the assignment your goal is to add mouse support to the program when using the Raylib view and Raylib controller. At a minimum, mouse support means that the user can click on a Toad or Frog to request a move. A more advanced implementation would also support the highlighting of squares hovered by the mouse.

This might require you to rearchitect some of the code, as the controller may now need information from the view (or vice versa). For example, the controller may need to request information about the positions of elements on the screen from the view. There are other ways to accomplish this feature (example: a layout service object shared by the view and controller), so try to think through a few possible scenarios. We'll review your various implementations in class.

The changes you make to support mouse input should not require you to make changes to the console controller, the console/ANSI views, or to the abstract IView and IController classes.

Part 5 – Reflection

Write a short reflection on this assignment. One or two paragraphs will be sufficient. Please include the following:

- What problems you encountered and/or discoveries you made while working on this assignment.
- Which do you like better:
    - Frogs

    - Toads
    - Gizzards
    - Codes
- Personal difficulty rating for each problem: Easy, Medium, Hard

Marking Rubric (16 Marks)

Part 1 – A Raylib Controller (4 marks)

One point each for:

- Implements `RaylibController::nextMove(...)` and compiles under `RAYLIB_BUILD`.
- Number keys map to board positions. Keys outside range are ignored safely.
- Pressing "r/R" resets the game state via the model.
- Follows the `IController` interface; no global/static state introduced (private instance state and/or private instance methods allowed).

One point deducted for missing each of these:

- Keyboard input is non-blocking.
- Key mapping derived from model's BoardSize (no hard coded "7"s).
- Works when paired with the existing ANSI view for testing.

Part 2 – A Raylib View (4 Marks)

One point for each:

- Implements `RaylibView::draw(const Model&)` and properly renders the board state held in the model.
- Visuals clearly indicate frogs, toads, and the empty square.
- Visuals clearly show the index number of each square.
- Your view includes images instead of 2d primitives for the frogs/toads, and/or it is beautiful/elegant compared with the example view presented earlier.

One point deducted for missing each of these:

- Raylib window size comes from view's `windowW()`/`windowH()` (not hard-coded in `main()`).
- Board state updates correctly after moves.
- A win is clearly indicated visually when the winning board state is achieved.
- View is read-only: no model mutation; all moves still go through a controller.

- Temporary ANSI/testing code removed from the RAYLIB_BUILD path before submission.
- `#define RAYLIB_BUILD` toggles text vs raylib builds as intended.

Part 3 – A Way Out (3 Marks)

One point for each:

- Both controllers (console and raylib) support quitting with keyboard input of "q/Q".
- `IController::nextMove` returns a status (i.e. enum class or a bool with clear meaning).
- `main()` loop exits cleanly when alerted by the controller that the user wishes to quit.

One point deducted for missing each of these:

- On the raylib path, resources are released (i.e. properly closing and window and releasing textures). No hangs/crashes on exit.

Part 4 – Yikes a Mouse! (3 Marks)

One point for each:

- Mouse clicking a frog/toad attempts a move consistent with the game rules.
- Click hit-testing matches the drawn board (no misaligned regions).
- Hover highlight for the focused cell.

One point deducted for missing each of these:

- MVC separation preserved:
    - Console controller and console/ANSI views remain unchanged.
    - Raylib controller/view coordinate via a minimal interface (e.g., controller queries view for cell bounds) or shared layout helper. No tight coupling between view and controller.
- Invalid clicks/moves do not alter state and do not crash.

5 - Reflection (2 marks)

Overall	Excellent (2)	Acceptable (1)	Poor (0)
Reflection	Reflection is insightful, clearly written, and	Reflection is complete but may lack depth or	Reflection is missing or fails to address many of the required points.

	addresses all required points.	miss some of the required points.	

Code Formatting, Naming, & Distribution (Marks deducted for infractions)

- **Consistent Formatting**: The code follows consistent indentation and spacing rules, making it easy to read.
- **Descriptive Naming**: Variables, functions, and classes are named descriptively, reflecting their purpose or use.
- **Source Code Packaged Project**: Source code should be easy to open in Visual Studio. It should be packaged as a Visual Studio project with a *.sln file, or if you are not using Visual Studio yourself, as a CMake project with a CMakeLists.txt file.

Code Commenting (Marks deducted for infractions)

- **Function Documentation**: Each function includes a comment describing its purpose, input parameters, return values, and any side effects. These comments should match the doxygen style comments that are already included in the provided code.
- **Purposeful Comments (When Code Isn't Obvious)**: Comments explain the why, not just the what. They describe the intent or reasoning behind complex or non-obvious code.
- **Avoids Over-Commenting**: The code isn't cluttered with obvious comments that state the obvious (e.g., int x = 10; // Assign 10 to x).

Special Note on Generative AI

Like all our Programming 3 assignments, this is a "no ai" assignment, meaning that your source code should be a genuine reflection of your personal effort and understanding.

You are, however, free to "discuss" this assignment with a system like ChatGPT or Claude to better understand the project architecture and the possible approaches to the four parts.

Just make sure your prompts specify that you don't want it to code a solution for you, and that instead you wish to better understand the material at hand.