Nasr Alae-eddine
119010531
CSC3150 Operating systems

Assignment 2 report

Development environment

- VM: virtual box, SSH to work locally on VSCode
- OS: ubuntu 20.04.5
- Kernel: 5.15
- GCC: 9.4

Game Design

Before writing any code I try to design how the game would run with a multithreaded program, the main questions were: How many threads are needed ? What global variables are needed ? Which operation would necessite a mutual exclusion lock (mutex) ?

Thread Situation:

With some brainstorming I came to the conclusion that I would need to create 3 threads from the main thread:

- Game_render: responsible for rendering the map to show the game in the terminal at all times
- Logs_move: responsible for the constant movement of the logs, will move frog if on a log
- Game_control: responsible for controlling the keyboard inputs and also checking the game state (lose-win-quit)

Global Variables:

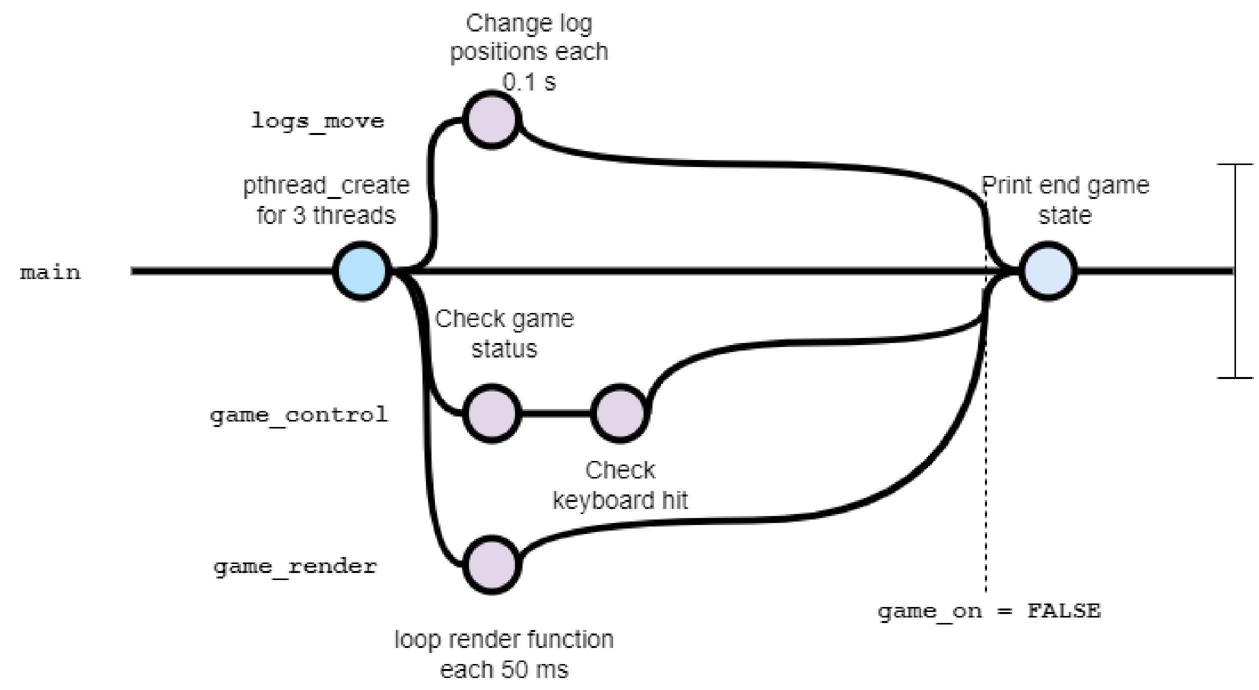
For the design proposed above to work properly a few global variables need to be shared by all threads:

- Map 2D array: contains the current game state as characters that will be printed row by row. Is NOT changed directly by the threads involved in moving objects !
- Frog node: contains frog position at all times, (x = row position, y = column position)
- Logs array: contains information of each row's log starting position, this is the array to be changed for each iteration of logs_move
- Game_on flag: turns to false in either case of the game ending (win/lose/quit)
- Finish_state: to communicate how exactly the game ended (win/lose/quit)

Mutual Exclusion:

Even though the program has numerous main global variables the only one that is susceptible of being altered by 2 threads at the same time is the frog coordinates. That is why we secure each instance that changes it by a mutex_lock.

With that said my program actually uses another mutex_lock to make sure it stays consistent in always displaying the last game state where the user won/lost/quit for half a second before clearing the screen.
To do this each time the game “ends” the responsible thread for ending the game also calls a single instance of the render function.
It has never happened in my tests but in theory even if the window is but a few microseconds the game might end while “render” is still in the critical part of the loop and it would create problems so the render function itself (not the thread!) is secured by a mutex_lock and the last game state will always be the last thing printed on the screen before it is cleared and an end message is shown.



Note: more technical design choices are explained in the code comments
Execution:
I have taken the advice from the “readme” provided:
-compile with g++ hw2.cpp -lpthread
-run with ./a.out

Learning outcomes

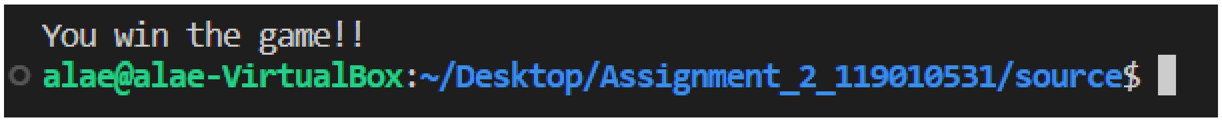
This assignment was very fun to figure out and also a great learning curve into the importance of multithread programming. There may be ways to do this exact task through sequential code and no additional threads but to model the “live objects” like the logs and frog into different threads feels intuitive now. And it feels more lively to have a render function looping always to print the screen without stopping the flow of the game whatsoever.
This was just a prologue to the world of Multi threads, and I am looking forward to making my own projects feel more alive with the introduction of what I learned here

Bonus Task:

My bonus task was definitely left unfinished from a personal time constraint but I had a lot of fun exploring the concept of thread pools through the lecture, the interesting TAs and the internet in general. I hope I can get at least a comment on my code. And an example of a working implementation so I can revisit this subject at a later appropriate time and learn more!

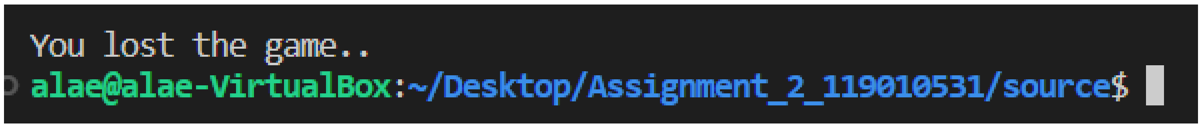
Screenshots (frogger)

Win:



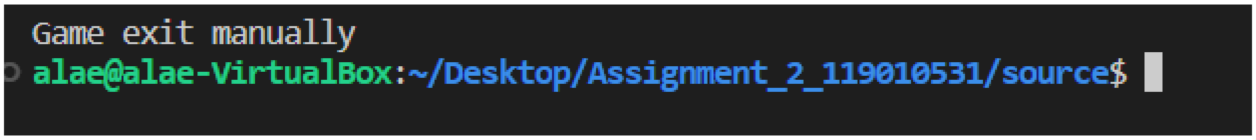
```
You win the game!!
alae@alae-VirtualBox:~/Desktop/Assignment_2_119010531/source$
```

Lose:



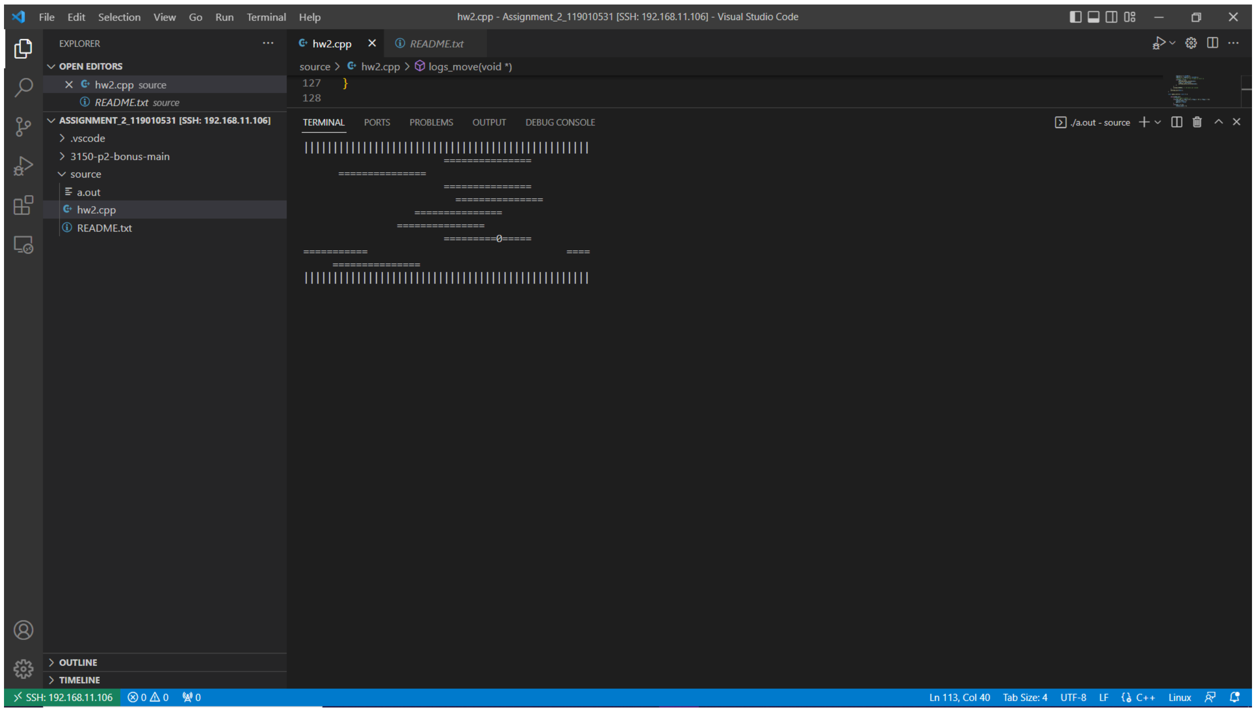
```
You lost the game..
alae@alae-VirtualBox:~/Desktop/Assignment_2_119010531/source$
```

Quit:



```
Game exit manually
alae@alae-VirtualBox:~/Desktop/Assignment_2_119010531/source$
```

In game Screenshot:



```
|||||||||||||||||||||||||||||||||||||||||||||||||
                        ===============
      ===============
                        ===============
                          ===============
                   ===============
                ===============
                        =========0=====
===========                                  ====
     ===============
|||||||||||||||||||||||||||||||||||||||||||||||||
```