Project 2 - Report

Name: Qin Peiran Student ID: 119010249

Part 1: Design ideas

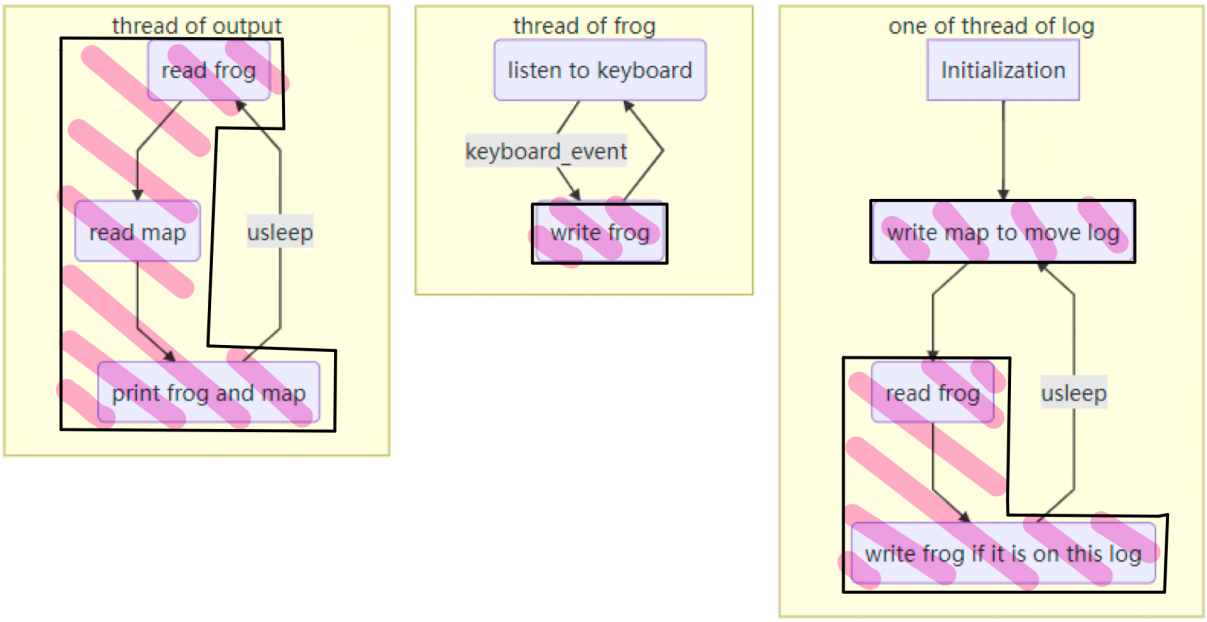
- Overview:
  - Basically this program use multithread programming for the movement of logs, control of frog, and refreshing of screen.
  - There are nine pthreads for each log's movement, one pthread for the control of frog and the main thread is for continuously refreshing the screen.
  - Muxtex locks are used to protect data.
  - For the bonus part, Qt is used to provide a graphical output of the game.

(1) The structure of the program and flow chart

- First, at the beginning of `main( )` function, program will first do the initialization of all the resources, including:
  - The initialization of the graphic window.
  - The initialization of `map`, `pthreads`, and `mutex`.
- Then, totally 11 pthreads will be created and each execute their corresponding function. Pthreads I use in this project are:
  - 9 threads for the movement of logs
  - 1 thread for the frog control
  - 1 thread for the refreshing of screen
- What's more, their are two main public resources, which are `frog` (records the position of frog) and `map`. (`frog` and `map` are independent in my project). To protect these two public resources, two mutex locks are created: `mutex_map` and `mutex_frog`.
- **Threads 1-9: Movement of logs**
  - Threads of logs will execute the function `void *logs_move( void *rank )`, where `rank` is the corresponding number of log, which is passed as argument by `pthread_create(&threads[rank], NULL, logs_move, (void *)&thread_ids[rank])`
  - According to the `rank` of each log, each thread would initialize the direction, length and start point of their corresponding log. The direction is fixed. The length is randomly generized between 10 and 20. The start point is also generized meaning each log would appear at any position at its row. The initialization would also update the `map` regarding to the position and length of logs.
  - After initialization, each thread will update the position of its log in a certain frequency, by using `usleep(speed)`. The frequency can be adjusted by player during the execution.
  - During update process of log's position, two public resources may be read and written. First is `map`, which would be modified because of the log's movement. Second is `frog`, which would be modified only when the frog is lying on the corresponding log. During the modification of `map` and `frog`, `mutex_map` and `mutex_frog` would be invoked.
- **Thread p_frog - Control of frog:**
  - In order to make frog move according to the keyboard input with nearly no latency. The control of frog is implemented by a new pthread.

    - This thread would continuously listen to the keyboard and change the position of frog.
    - Noted that public resource `frog` would be written. Thus `mutex_frog` would be invoked during the process.
    - Except for control of frog, this thread also in charge of changing the speed of logs. (Because player use keyboard to modify the speed)
- **Main Thread - Refresh the sceen**
    - At the beginning, there will a judgement whether there the game comes to end (win, lose, quit).
    - Then, the ouput is completely handled by this thread, which prints out current `map` and `frog` on the window. Since it reads both `map` and `frog`, two mutex locks would be invoked at the same time.
- **Remark: Ensurance of Correctness**
    - As shown in above, there are multiple pthreads and several uses of mutex locks. How to ensure the correctness of functions is one of most significant thing this project should consider. Here are some considerations:
        - There are some section of code must be atomic, like:
            - Judgement of the end of game and output -- we should avoid modification of even one element during this whole process. Otherwise we may print maps of different time (for example, during the process of printing, one log moves, then it will print a wrong map).
            - modification of map or frog while control of frog and movement of logs and frog -- for the element-modification(because elements in map are independent) of map and control of frog, they should be atomic.
        - For each time of refreshing, the logs should move simultanuously. It means that during the moving of logs, there cannot be refreshing of screen.
    - This project use `mutex_frog` and `mutex_map` to lock the whole process of refreshing the screen (print out frog and map). Use either `mutex_frog` or `mutex_map` to lock read or write of frog or element in map.
    - To make logs move simultanuously, this program use a public variable `number_of_moved_log`, if it is not equal to 0 or 9, means the threads of logs are executing movement of log. And refreshing of screen would not be performed until the `number_of_moved_log` turn 9. (However, this approach is different from using one thread executing logs movement and refreshing serially, because before and after all the logs move, there can be high frequency of refreshing screen to make sure control of frog can be shown on the screen immediatly).

To inllustarte my idea, the overall flowchart is provided below:



- Remark: the contiguous areas marked with pink color should be atomic, which means that at least one mutex lock will be added to each of them.

Part 2: Environment

- My project adopts the recommended environment:
  - version of OS:

```
qpr@ubuntu:~/Desktop/Assignment_1_119010249$ cat /etc/issue
Ubuntu 16.04.5 LTS \n \l
```

  - version of kernel:

```
qpr@ubuntu:~/Desktop/Assignment_1_119010249$ uname -r
4.10.14
```

  - gcc version:

```
root:/ $ gcc --version
gcc (Ubuntu 5.4.0-6ubuntu1~16.04.12) 5.4.0 20160609
Copyright (C) 2015 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

- The environment of GUI:
  - This project uses **Qt 5**, the method to set up the environment and compile my code will be shown in next part.

Part 3: Step of executing my code

Install the package of GUI:

- update your local repository index with that of the Internet:

```
sudo apt-get update
```

- install the Build Essential package:

```
sudo apt-get install build-essential
```

- install the Qt Creator package:

```
sudo apt-get install qtcreator
```

Please enter Y when the system prompts you with a choice to continue the installation.

- use the default version:

```
sudo apt install qt5-default
```

- then

```
sudo apt-get install qt5-doc qtbase5-examples qtbase5-doc-html
```

Compile and run my code:

- Please make sure that you are in the terminal of Ubuntu.
- enter into the corresponding directory:

```
cd source
```

- compile the source code:

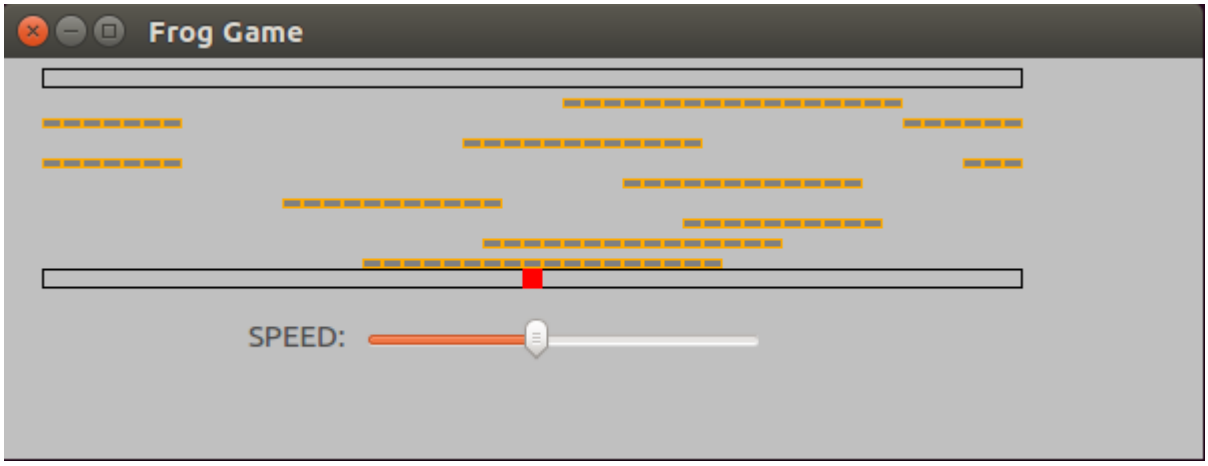
```
qmake source.pro
make
```

- Then run it by:

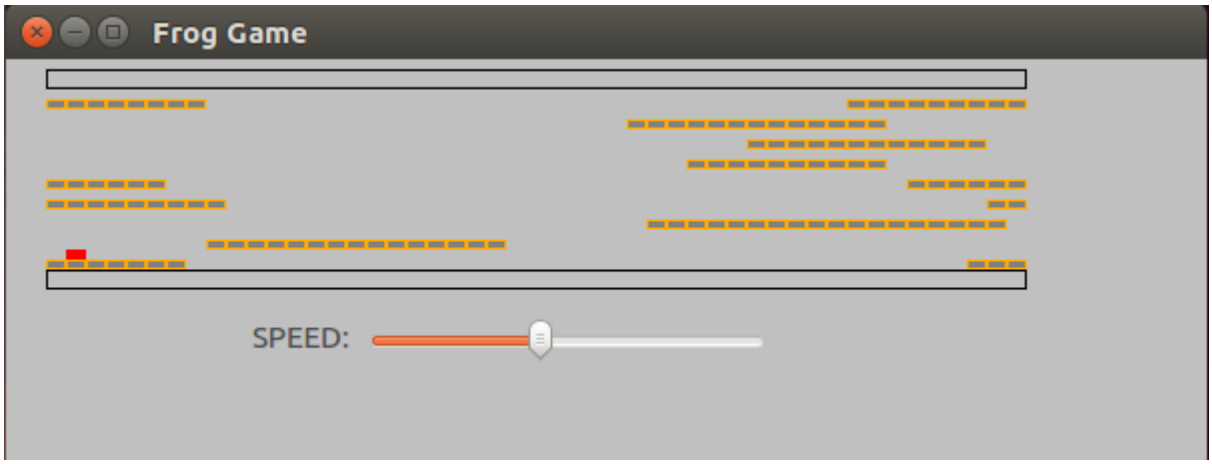
```
./source
```

Part 4: Sample output

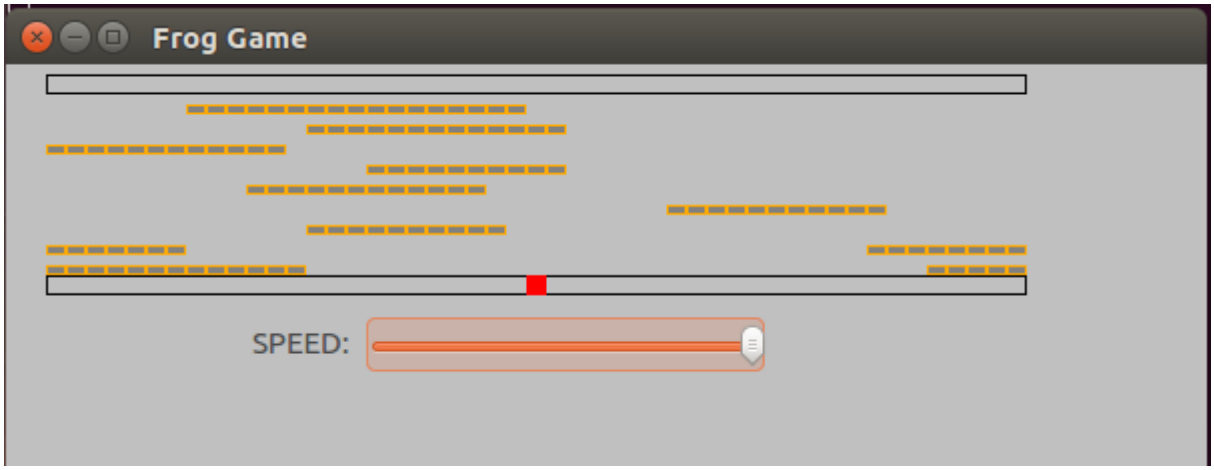
- At beginning, the length and position of logs are randomly generized on GUI.



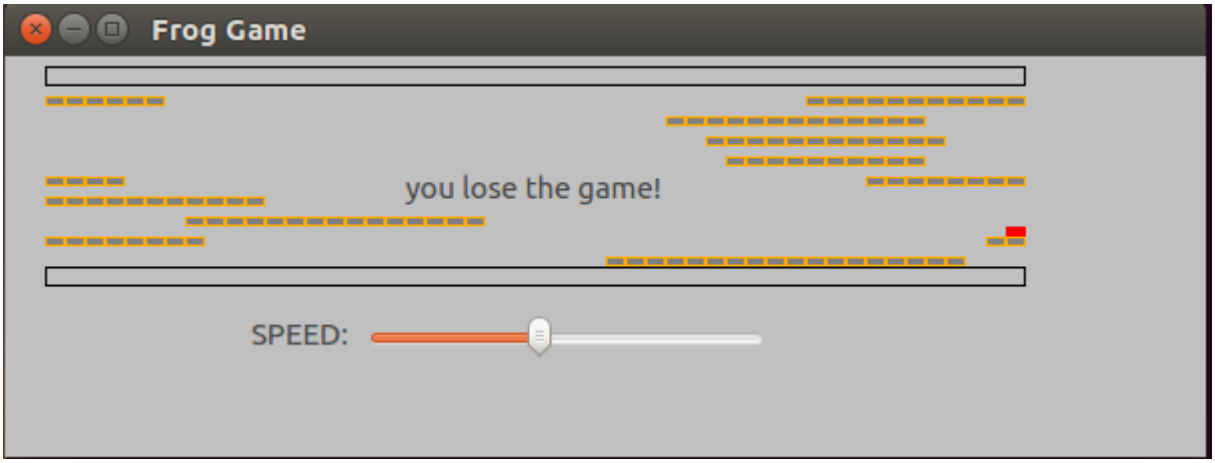
- If frog jumps to the log, it would move with log



- Change the speed of log movement using the slide bar.



- Output when lose the game (may because of dropping to river or reach the left/right boundary)



Part 5: Things I learned

I learned basically two things from this project. The first thing is the basic multithread programming skill, including some useful functions and grammar in Pthread. The second thing is the strategies about how to ensure the correctness of multithread programming.

1. First, I learned about the some useful functions in Pthread and how to use them, including some functions to create thread (also pass argument to thread), create and use mutex lock to protect data, and functions to join and exit pthreads.
2. Secondly, I learned the strategies about how to ensure the correctness of multithread programming, which is the biggest harvest I learned from this project. The key word I learn is: **atomicity**.
    - During coding, I learned that ensure atomicity is not only about making the read-write of public resources(protect data) atomic, but also making some contiguous part of code atomic. For example, the process of refreshing of screen should be mutually exclusive with the movement of logs. Otherwise, logs may not move simultanuously in one time of refreshing.
    - The method to solve this problem is using mutex locks, I use two locks (mutex_frog, mutex_map) to make sure that the process of refreshing is completely atomic. Then I use a public variable `number_of_moved_log` to indicate the thread of refreshing screen whether threads of logs have all finished moving (or all haven't begun moving), which make sure all logs move simultanuously during each time of refreshing screen.
3. Thirdly, I learned how to solve the situation of **dead lock**. I once met dead lock because my program use two mutex locks (`mutex_frog` and `mutex_map`) and I handle them in a wrong way: on thread hold `mutex_frog` and request for `mutex_map` while the other hold `mutex_map` request for `mutex_frog` . Then I re-order the my code and make sure that the situation above would not happen.
4. Fourthly, I tried other kinds of lock like `read-write lock` . The reason I tried rwlock is that I use 9 threads for the movement of logs, however, they modify different resouces (different indexes in map). Since read-lock can be hold by many threads at same time, there is no need for them to compete for the same mutex lock, thus efficiency can raise. And write-lock is used for the refreshing of screen and control of frog, which will also ensure the atomicity mentioned above. This approach can also perform the game correctly(the source code I submiited is still implemented by mutex locks as requirement).