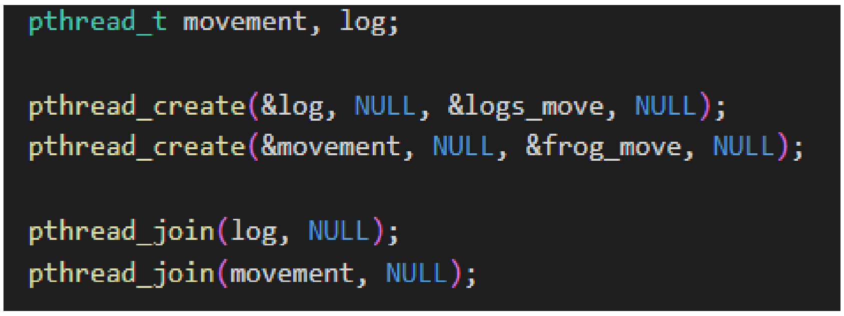
Assignment 2 Project Report

Project Introduction

This second assignment intends to familiarize us with multithreaded programming and thread synchronization. Our job is to create a basic game that involves moving a frog with shifting logs from one end of a riverbank to another. We also gain more knowledge about keyboard inputs and terminal instructions for displaying game results. The Kernel Version used is **Linux 5.10.147** with GCC version **5.4.0**.

Project Implementation

Two threads are used to run the game; one thread controls how the logs travel while the second thread moves the frog. The two public functions facilitate threaded operation of the logs move and frog move methods.



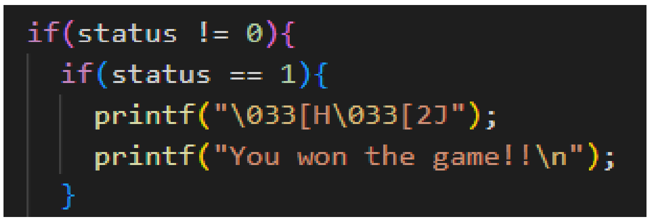
```
pthread_t movement, log;

pthread_create(&log, NULL, &logs_move, NULL);
pthread_create(&movement, NULL, &frog_move, NULL);

pthread_join(log, NULL);
pthread_join(movement, NULL);
```

Figure 2.1 Thread used on Frog and Logs Movement

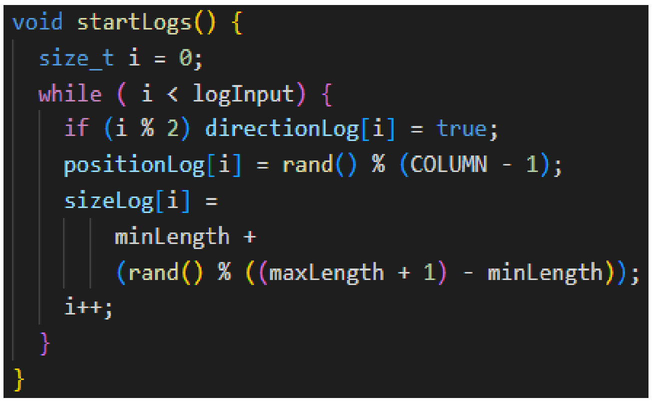
Enumeration from 0 to 4 defines the current game state. 0 is defined if the program still running and the user is still in the game. Number 1 is used if the player has reached the other side of the bank which means the player has won. Number 2 is used if the player loses the game when they fail to jump onto the logs and fall to the river. Finally, game state 3 is used when the player chooses to exit the game.



```
if(status != 0){
  if(status == 1){
    printf("\033[H\033[2J");
    printf("You won the game!!\n");
  }
```

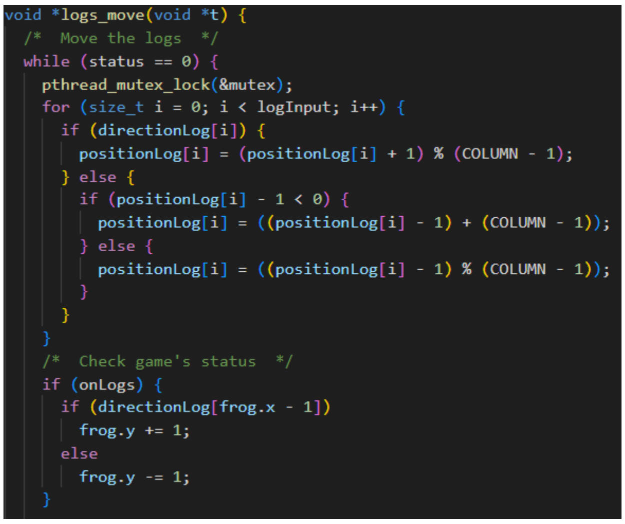
Figure 2,2 Game status example

Two arrays manage the movement of the logs. Two arrays are used: one is called positionLog and saves the starting positions of the logs in the column array, and the other is sizeLog and holds the sizes of each log. Along with having a directionLog array, the logs will also have a direction that they will go in.



```
void startLogs() {
  size_t i = 0;
  while ( i < logInput) {
    if (i % 2) directionLog[i] = true;
    positionLog[i] = rand() % (COLUMN - 1);
    sizeLog[i] =
        minLength +
        (rand() % ((maxLength + 1) - minLength));
    i++;
  }
}
```

Figure 2.3 Arrays for Logs Movement

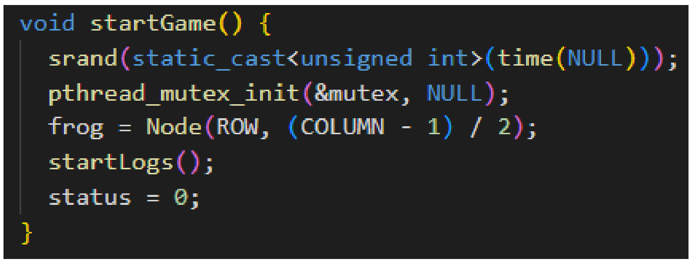


```
void *logs_move(void *t) {
  /*  Move the logs  */
  while (status == 0) {
    pthread_mutex_lock(&mutex);
    for (size_t i = 0; i < logInput; i++) {
      if (directionLog[i]) {
        positionLog[i] = (positionLog[i] + 1) % (COLUMN - 1);
      } else {
        if (positionLog[i] - 1 < 0) {
          positionLog[i] = ((positionLog[i] - 1) + (COLUMN - 1));
        } else {
          positionLog[i] = ((positionLog[i] - 1) % (COLUMN - 1));
        }
      }
    }
    /*  Check game's status  */
    if (onLogs) {
      if (directionLog[frog.x - 1])
        frog.y += 1;
      else
        frog.y -= 1;
    }
```

Figure 2.4 Algorithm for Log Movement

This complies with the assignment's requirements, which state that the first log should move to the left and that subsequent logs should move in alternate directions. By altering the positionLog values, the related logs will move in accordance with the movement of the other logs. Additionally, hit detection throughout the map's borders has been included, and this will update the game's status appropriately.

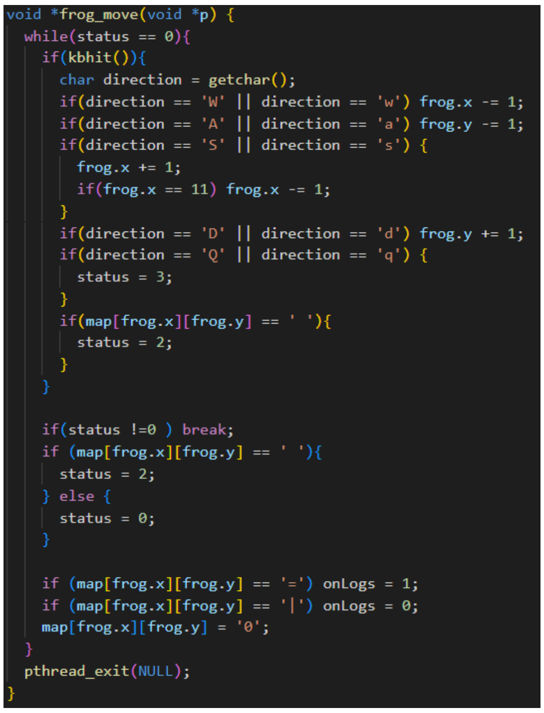
The startGame function will start the game. The starting positions of the frog, logs, and river map will all be initialized by this function. To assist with the initialization of the logs, it calls the startLogs method. This program just creates the sizes and starting positions of the logs at random within a predetermined range. The game class's constructor is where the random seed is created. The user will have the choice of using or not using their own seed.



```
void startGame() {
  srand(static_cast<unsigned int>(time(NULL)));
  pthread_mutex_init(&mutex, NULL);
  frog = Node(ROW, (COLUMN - 1) / 2);
  startLogs();
  status = 0;
}
```

Figure 2.5 startGame Function

The frog movement concept is simple to understand. It directs the movement of the frog using the provided keyboard hit detection. It will also detect the option to exit the game and change the game state appropriately after updating the frog positions. Additionally, this is where the frogs on the logs are discovered. This is kept up by the variable onLogs. This Boolean variable will become true if the frog moves to a position with a log, which will then trigger the frog to follow the log using the logs_move function. The game state will change to lose if the frog lands in the river, and it will cease travelling with the log if it jumps backwards to the river bank. If the frog lands on the river, the game state will also change to lose. The logs_move function conducts the most updates on the shared data variables in the program and will lock the class attribute variables to ensure synchronization with the frog_move method on the position of the logs, game state, and frog position (it will move on top of the log).

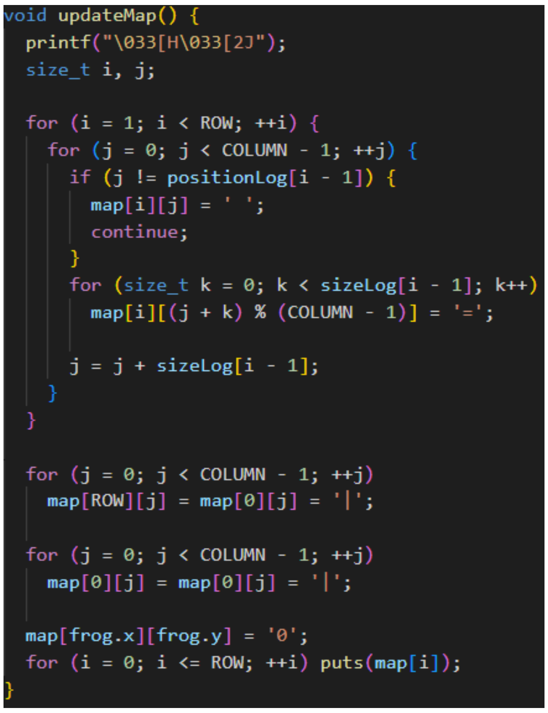


```c
void *frog_move(void *p) {
  while(status == 0){
    if(kbhit()){
      char direction = getchar();
      if(direction == 'W' || direction == 'w') frog.x -= 1;
      if(direction == 'A' || direction == 'a') frog.y -= 1;
      if(direction == 'S' || direction == 's') {
        frog.x += 1;
        if(frog.x == 11) frog.x -= 1;
      }
      if(direction == 'D' || direction == 'd') frog.y += 1;
      if(direction == 'Q' || direction == 'q') {
        status = 3;
      }
      if(map[frog.x][frog.y] == ' '){
        status = 2;
      }
    }

    if(status !=0 ) break;
    if (map[frog.x][frog.y] == ' '){
      status = 2;
    } else {
      status = 0;
    }

    if (map[frog.x][frog.y] == '=') onLogs = 1;
    if (map[frog.x][frog.y] == '|') onLogs = 0;
    map[frog.x][frog.y] = '0';
  }
  pthread_exit(NULL);
}
```

Figure 2.6 Frog Movement Code



```c
void updateMap() {
  printf("\033[H\033[2J");
  size_t i, j;

  for (i = 1; i < ROW; ++i) {
    for (j = 0; j < COLUMN - 1; ++j) {
      if (j != positionLog[i - 1]) {
        map[i][j] = ' ';
        continue;
      }
      for (size_t k = 0; k < sizeLog[i - 1]; k++)
        map[i][(j + k) % (COLUMN - 1)] = '=';

      j = j + sizeLog[i - 1];
    }
  }

  for (j = 0; j < COLUMN - 1; ++j)
    map[ROW][j] = map[0][j] = '|';

  for (j = 0; j < COLUMN - 1; ++j)
    map[0][j] = map[0][j] = '|';

  map[frog.x][frog.y] = '0';
  for (i = 0; i <= ROW; ++i) puts(map[i]);
}
```

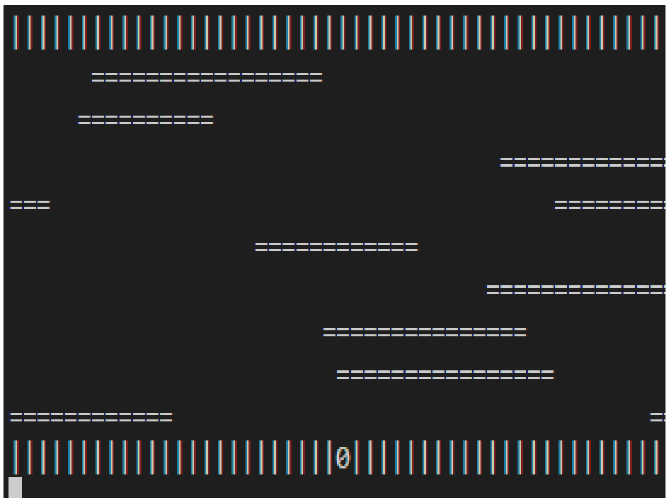
Figure 2.7 Update Map Function

The map and the current game state are printed in the terminal by the updateMap() method in Figure 2.7. Every interval, it cleans the terminal and produces the appropriate map. Additionally, it will identify the logs' present location and print it on the map appropriately. With the modulus (%) operator, it will also wrap around the COLUMN size appropriately.

Project Summary

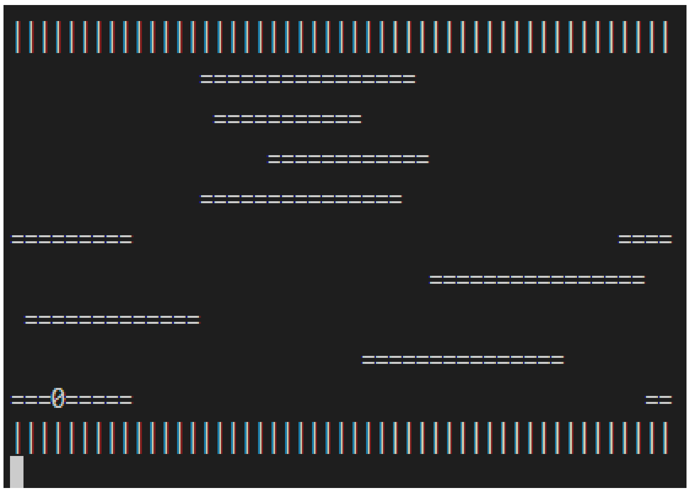
Through this assignment, we gained understanding of thread synchronization in multithreading programs. It aided in my comprehension of how to apply multithreading to a program designed for a single thread and how to securely and effectively construct a mutex lock to prevent racing between the two threads. Additionally, the assignment gave me some pointers and explanations on how to use pseudo-random functions and alter terminal output.

Project Output



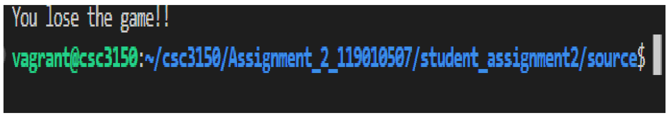
```
||||||||||||||||||||||||||||||||||||||||||||||||
      =================
     ==========
                                    ============
===                                     ========
                  ============
                                   =============
                       ===============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

Figure 4.1 Game Start with random log moving



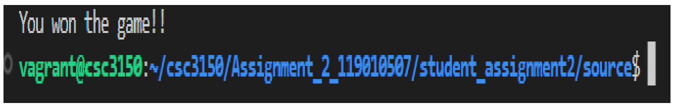
```
|||||||||||||||||||||||||||||||||||||||||||||||||
              ================
               ===========
                   ============
              ===============
=========                                    ====
                               ================
 =============
             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

Figure 4.2 Frog Jumping on Moving Logs



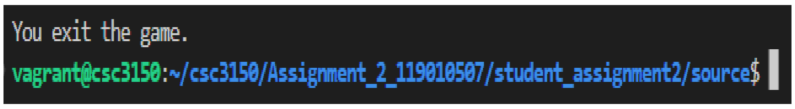
```
You lose the game!!
vagrant@csc3150:~/csc3150/Assignment_2_119010507/student_assignment2/source$
```

Figure 4.3 Lose Statement Output



```
You won the game!!
vagrant@csc3150:~/csc3150/Assignment_2_119010507/student_assignment2/source$
```

Figure 4.4 Win Statement Output



```
You exit the game.
vagrant@csc3150:~/csc3150/Assignment_2_119010507/student_assignment2/source$
```

Figure 4.5 Exit Statement Output