Assignment 2: Frogger

Edris L

July 2024

1 Introduction

Welcome. In this document, you can learn how to play the frogger game I have created. Also, I will explain the changes I implemented to the original code that Dr. Pasandide had provided us, and I will show my code to you as well (in Appendix).

2 Getting Started

2.1 What you need

To compile and play the Frogger game, the following is required...A Unix based operating environment (such as Linux, Ubuntu, etc.). A terminal within the environment. A compiler in C (such as 'gcc').

2.2 How to Compile and Run the Frogger game

First save the C code that is given in the appendix as "frogger.c". You can save the code by going to file¿save, or by pressing ctl+S (if on windows). Then, within the terminal find where "frogger.c" is located, and go to that directory. Then, compile the code in "frogger.c" by putting the following command in the terminal: `gcc frogger.c -o frogger` . Lastly, to run the code, put the following command in the terminal: `./frogger`

2.3 How to Play the Game

The purpose of the game is to move the the frog ("F") up until you reach the highest lane without hitting an obstacle. If the frog hits an obstacle ("x") then you lose. However if the frog goes on a "." then it is safe. To move up, press "W". To move to the right, press "D". To move down, press "S". To move to the left, press "A". And, to exit the game, press "Q", and confirm it by pressing "y" followed by the enter key.

3 Format Changes Compared to the Original Code

The changes I had made improved the structure, efficiency, convenience, readability, and overall functionality of the code. It made the game more user friendly and easier to play and understand.

3.1 Added Additional Libraries

- `#include <unistd.h>` : This provided access to use operating system functions like `usleep()` (it pauses the game for a bit).
- `#include<termios.h>` : This controls terminal input settings so keyboard inputs can be read without pressing "Enter".
- `#include <fcntl.h>` : This manages the file descriptors and allows for non-blocking inputs.
- `#include <stdio.h>` : This handles the standard input and output functions like `printf()`.
- `#include <stdlib.h>` : This was placed to use the `exit` command in the code.

3.2 Better Input Handling

I added non-blocking input handling with "termios" and "fcntl" (that is listed above) to make sure the game is more responsive to user inputers. I used these commands by using the kbhit() function.

3.3 More Clear Drawing Logic

I put the draw() function to clear the creen and redraw the lanes and frog. With the draw() function I also implemented system("Clear") to clear the terminal screen before redrawing the state of the game.

3.4 Better Games State Update

I put a temporary array "temp" to update the lanes based on their speeds more efficiently.This makes sure the lanes move in the correct way and th efame state is updated without directly modifying the original array when the code is iterating.

3.5 Losing ("Game Over") and Winning Conditions

I made if structures to check if the frog hits an obstacle or not, and if it reaches the top lane and wins the game.

4 Appendix

Frogger.c

```
#include <unistd.h>  // Provides access to the POSIX operating system API.
#include <termios.h> // Used for controlling terminal I/O characteristics.
#include <fcntl.h>   // Provides an interface for file descriptor manipulation.
#include <stdio.h>   // Standard input-output header.
#include <stdlib.h>  // Standard library for memory allocation, process control, etc>

// Game constants
#define WIDTH 64
#define HEIGHT 10

// Game global variables
int frogX, frogY;

char lanes[HEIGHT][WIDTH + 1] = {
    "xxx..xxx..xxx..xxxxxxxxxxxxxxx..xxxxxxxxxxxxxxxxxxxxxxxx..xxxxx", // l a n e 0
    "...xxxx..xxxxxx.......xxxx.....xx...xxxx.....xxxxxx...xxxxx.....", // l a n e 1
    "....xxxx.....xxxx.....xxxx.......xxxxxxx.....xx....xxxxxx.......", // l a n e 2
    "..xxx.....xxx.....xxx.....xxx...xxx....xx....xxxx....xx......xx." , // l a n e 3
    "............................................................." , // l a n e 4
    "....xxxx.......xxxx.........xxxx.......xxxx......xxxx....xxxx..." , // l a n e 5
    ".....xx...xx...xx......xx....xx.......xx..xx.xx......xx.......xx", // l a n e 6
    "..xxx.....xx......xxxx..xx......xxxx......xxxx.......xxx...xxx.." , // l a n e 7
    "..xx.....xx.......xx.....xx.....xx..xx.xx........xx....xx......." , // l a n e 8
    "............................................................."  // l a n e 9
};


int speeds[HEIGHT] = {0, -2, +1, -1, 0, +2, -1, -1, +1, 0};

// declaring functions (if key is pressed, handle draw, handle updates)

int kbhit(void);
void draw();
void update();

int main() {
    // setting first position of frog to middle bottom of screen
    frogX = WIDTH / 2;
    frogY = HEIGHT - 1;

    while (1) {
        if (kbhit()) {
```

```
            // setting conditions regarding what key is pressed
            switch (getchar()) {
                case 'w': frogY--; break;
                case 'a': frogX--; break;
                case 's': frogY++; break;
                case 'd': frogX++; break;
                case 'q': exit(0);
            }
        }
        update(); // updates game state such as moving lanes, checking for crash with c
        draw();
        usleep(900000); // slowing down the game a bit

        // if frogger hits top lane and wins

        if (frogY == 0)
        {
            printf("\nYOU WIN!!!CONGRATS!\n");
            exit(0);
        }
    }
    return 0;
}

int kbhit(void) {
    struct termios oldt, newt;

    // variables to store character input and old file statues flags
    int ch;
    int oldf;

    //get current terminal settings and storing them to oldt (searched this up)
    tcgetattr(STDIN_FILENO, &oldt);
    // copy the old setting to newt so we can modify later
    newt = oldt;
    newt.c_lflag &= ~(ICANON | ECHO);
    tcsetattr(STDIN_FILENO, TCSANOW, &newt);
    oldf = fcntl(STDIN_FILENO, F_GETFL, 0);
    fcntl(STDIN_FILENO, F_SETFL, oldf | O_NONBLOCK);

    // trying to read character from input
    ch = getchar();

    tcsetattr(STDIN_FILENO, TCSANOW, &oldt);
    fcntl(STDIN_FILENO, F_SETFL, oldf);
```

```
    // if read character then put it back and return 1 (indicating key pressed)
    if (ch != EOF) {
        ungetc(ch, stdin);
        return 1;
    }

    return 0;
}

void draw() {
    system("clear"); //clears terminal screen

    // loop through each row and columin in row, and check if current position
    // is where frog is, if it is then print 'F', or else print the character
    // from the lanes array
    for (int y = 0; y < HEIGHT; y++) {
        for (int x = 0; x < WIDTH; x++) {
            if (x == frogX && y == frogY) {
                putchar('F');
            } else {
                putchar(lanes[y][x]);
            }
        }
        putchar('\n');
    }
}

void update() {
    // Check if the frogger hits an 'x'
    if (lanes[frogY][frogX] == 'x') {
        // Game over scenario
        printf("\nGame Over! You hit an 'x'.\n");
        exit(0);  // quit the game
    }

    // Update the lanes array based on speeds
    for (int y = 0; y < HEIGHT; y++) {
        if (speeds[y] != 0) { // only updating row is speed isnt zero
            char temp[WIDTH + 1]; // temporary array to hold updated row
            // if speed positive then move right
            if (speeds[y] > 0) {
                for (int x = 0; x < WIDTH; x++) {
                    // calculating new position and moving element there
                    temp[(x + speeds[y]) % WIDTH] = lanes[y][x];
                }
            } else {
```

```
                // if speed negative then move left
                for (int x = 0; x < WIDTH; x++) {
                    // calculating new position and move element there
                    temp[x] = lanes[y][(x - speeds[y] + WIDTH) % WIDTH];
                }
            }
            // copying updated row back to original
            for (int x = 0; x < WIDTH; x++) {
                lanes[y][x] = temp[x];
            }
        }
    }
}
}
```