**Bop-It!**

**LAB 6**

**SECTION 2**

**RYAN AUN SHERN CHIN**

**SUBMISSION DATE:**

**3/6/24**

# Problem

In this lab, we were tasked to develop problem-solving skills and develop skills using multiple loops, as well as practice writing whole programs from the ground up. The end goal was to implement a simple “Bop-It”- like game that had specific requirements.

# Analysis

Looking at the requirement for this simple “Bop-It” game, it should have a start menu waiting for an input from the user, preferably a specific button press on the DS4 controller to start the game.

After starting the game, the program should output a line telling the user to press a specific button on the DS4 that was randomly chosen by the program which indicates the usage of a “seed”. The program then waits a certain time for a respond by the user, which preludes to the sense that the previous time value should be kept in track as to compare it with the newly scanned in time. This is required as needed by the program specifications to end the game if time for the user to respond runs out, as well as the need for time decrements per iteration or correct button press from the user.

The game also has to end if the user presses the wrong button, which makes sense to create a safety net to ensure that button presses are only registered once per press. Lastly, the game has to run continuously until the user reaches the aforementioned lose conditions and outputs how many successful actions the player as executed at the end, which gives an idea to use loops to achieve this.

# Design

For this lab, I spearheaded the design in steps that was outlined in the lab manual. I first created a “start menu” that involved two printf() statements that ran outside loops to avoid continuous output of a “start menu” by the program.

For the program to wait for a user input to start, in which in my case was a circle button to start, I used a while loop that had a condition to continuously loop and scan in -t and -b flag values from the DS4 controller if the triangle, circle, cross and square button values were 0. This is achieved by the “initialization” 0 values of the DS4 buttons at the beginning of the program. For example, if the user presses the circle button, the circle button’s value would change to 1 which causes the if statement after the scanf() statement to trigger and break the while loop, advancing to the next stage of the game or (in programming terms) the next loop that dealt with the main mechanisms of the game.

For the loop that dealt with the main mechanisms of the game that allowed the game to run continuously until the user reaches a lose condition, I used a while loop that used a condition that checked if the variable rounds were true or false. I figured that it would easier to break this loop when I set rounds to false when the user inputs the wrong button press. At the end before the loop ends, I implemented a printf() function to output how many successful action the player as executed at the end. This is achieved by declaring a variable called roundCount that increments after each successful button press by the user which can be done inside nested if loops within the while function.

**How did you randomize the buttons that needed to be pressed?**

These buttons that the user were required to press to advance to the next iteration was determined randomly by the program which was done by first creating a ‘seed’ using ‘srand(time(NULL));’ that ensured a random game each time. Next I created a function called ‘gameButton()’ that focused on determining the random button the user was required to press. It involved using ‘rand() % 4 + 1;’ to generate a random number from 1 to 4 that corresponded to cases within a switch statement for which case 1 was associated with the triangle button, case 2 was associated with the circle button, case 3 was associated with the cross button and case was associated with the square button. So if the random number generated was 1, it required the user to press the triangle button, if the random number generated was 2, it required the user the press the circle button, if the random number generated was 3, it required the user to press the cross button, if the random number generated was 4, it required the user to press the square button.

**What mechanism did you use to make sure extraneous button presses were not registered?**

For the above to work and to make sure extraneous button presses were not registered, I called this function in the main function and initialized ‘randNum = gameButton();’. Now the randomized values of randNum that went from 1 to 4 are ‘AND’ed (&&) to the corresponding button variables, which are then ‘OR’ed (||) together to determine if the user had pressed the correct button using an if conditional statement, where ‘if((randNum == 1 && triangle == 1) || (randNum == 2 && circle == 1) || (randNum == 3 && cross == 1) || (randNum == 4 && square == 1))’, which also avoided registering extraneous button presses. This was also coupled by an if statement beforehand that checked whether if buttons were being pressed, and if one was being pressed, then it would straight away go to the aforementioned if statement given above, avoiding registration of extraneous button presses by ‘if (!((triangle + circle + cross + square) == 0))’.

**What game states, if any, did you keep track of?**

I had to keep track of the amount of time given to the user to press a button for each iteration after a successful button press. Within the if statement that determined if the user pressed the correct button, I implemented the decrement of the time allocated to the user after each successful button press using ‘timer -= 100’, where the variable timer is declared and initialized at the beginning of the program to 2500 as per requirement of this program which for every successful button press, the user had 100 milliseconds less than the previous iteration to press the correct button.

Then, I had to keep track of the values from the scanf() statements that continuously scanned inputs from the DS4 controller for the main mechanism of the game (not the start menu). I had to implement another while loop, ‘while (currentTime + 260 > t)’ that forced the program to scan in new values from the DS4 controller, with currentTime being declared and initialized to ‘t’, which was the scanned in time value from the DS4 controller. Without it, the time and button values would only follow the first scanned in values from the scanf() statement in the ‘start menu’ and it serves as some sort of delay after scanning in the first input in the start menu.

Next, I had to keep track of the time the user is taking to press the correct button through a “countdown” of sorts. If the user exceeds the total amount of time allocated to respond, the program will end and output a line saying the user ran out of time. This is achieved by initializing the variable ‘currentTime’ with ‘t’ again, which was the scanned in time value from the DS4 controller. Since ‘t’ is continuously increasing with time scanned from the DS4 controller, it made sense to add currentTime which was initialized to the value of ‘t’ after user was prompted to press a button

with the total time allocated to press a button which was ‘timer’ and check if it was below the more than or equal to ‘t’ which meant that the user still had time to press the button, ‘while (currentTime + timer >= t)’. If the user doesn’t have time left then it would be ‘if (currentTime + timer < t)’ where ‘t’ would have increased by so much since the time the user was prompted added with the buffer of the timer. This served and helped to print out the line “You ran out of time. :( Thanks for playing!” as per requirement of this program.

# Testing

For the testing of this program, I tested each functionality of the game step by step, starting with the start menu and later the main functionality of the game like the function which generated the random button required for the user to press. I compiled and ran the program together with ds4rd.exe using `/ds4rd.exe -d 054c:05c4 -D DS4\_BT -t -b`. I first pressed the Circle button for the start menu to make sure that it was working as intended. Later, I play-tested my program and pressed the correct inputs generated randomly by the program. I then intentionally pressed incorrect options to test whether if my program was handling errors correctly. For another testing session, I allowed the time for me to press a button to run out in order to test whether if my program was correctly handling the two game states which were the time allocated for me to press a button and the “countdown” for me to press a button.

# Comments

In doing this lab, I realized the importance of double checking my code in order to avoid any errors that may appear when compiling my code. Furthermore, I managed to become familiar with writing a program from scratch as well as developed my skills in problem solving.

**Source Code lab06.c**

/\*----------------------------------------------------------------------------

- SE 185: Lab 06 - Bop-It! -

- Name: Ryan Aun Shern Chin -

- Section: 2 -

- NetID: ryanchin -

- Date: 2/28/24 -

-----------------------------------------------------------------------------\*/

/\*----------------------------------------------------------------------------

- Includes -

-----------------------------------------------------------------------------\*/

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

/\*----------------------------------------------------------------------------

- Prototypes -

----------------------------------------------------------------------------\*/

int gameButton**();**

/\*----------------------------------------------------------------------------

- Notes -

-----------------------------------------------------------------------------\*/

// Compile with gcc lab06.c -o lab06

// Run with ./ds4rd.exe -d 054c:05c4 -D DS4\_BT -t -b | ./lab06

/\*----------------------------------------------------------------------------

- Implementation

-

-----------------------------------------------------------------------------\*/

int main**(**int argc**,** char **\***argv**[])**

**{**

int t**;**

int triangle **=** 0**;**

int circle **=** 0**;**

int cross **=** 0**;**

int square **=** 0**;**

int rounds **=** 1**;**

int randNum**;**

int roundCount **=** 0**;**

int timer **=** 2500**;**

int currentTime**;**

srand**(**time**(NULL));** /\* This will ensure a random game each time. \*/

printf**(**"This is a Bop-It Game!\n"**);**

printf**(**"Please press the Circle Button to begin!\n"**);**

**while** **(**triangle **==** 0 **&&** circle **==** 0 **&&** cross **==** 0 **&&** square **==** 0**)**

**{**

scanf**(**"%d,%d,%d,%d,%d"**,** **&**t**,** **&**triangle**,** **&**circle**,** **&**cross**,** **&**square**);**

**if(**circle**)** **break;**

**}**

**while** **(**rounds**)**

**{**

currentTime **=** t**;**

int randNum **=** gameButton**();**

**while** **(**currentTime **+** 260 **>** t**)** //delay after scanning in first input

**{**

scanf**(**"%d,%d,%d,%d,%d"**,** **&**t**,** **&**triangle**,** **&**circle**,** **&**cross**,** **&**square**);**

**}**

printf**(**"You have %d miliseconds to respond!\n"**,** timer**);**

currentTime **=** t**;**

**while** **(**currentTime **+** timer **>=** t**)**

**{**

scanf**(**"%d,%d,%d,%d,%d"**,** **&**t**,** **&**triangle**,** **&**circle**,** **&**cross**,** **&**square**);**

**if** **(!((**triangle **+** circle **+** cross **+** square**)** **==** 0**))**

**{**

**if((**randNum **==** 1 **&&** triangle **==** 1**)** **||** **(**randNum **==** 2 **&&** circle **==** 1**)** **||** **(**randNum **==** 3 **&&** cross **==** 1**)** **||** **(**randNum **==** 4 **&&** square **==** 1**))**

**{**

timer **-=** 100**;**

roundCount**++;**

**break;**

**}**

**else**

**{**

printf**(**"\nWrong Button! :(\nYou lose!\n"**);**

rounds **=** 0**;**

**break;**

**}**

**}**

**}**

**if** **(**currentTime **+** timer **<** t**)**

**{**

printf**(**"\nYou ran out of time. :(\nThanks for playing!\n"**);**

**break;**

**}**

**}**

printf**(**"You made it through %d rounds!\n"**,** roundCount**);**

**return** 0**;**

**}**

/\* Put your functions here, and be sure to put prototypes above. \*/

int gameButton**()** **{**

int randButton **=** rand**()** **%** 4 **+** 1**;**

**switch** **(**randButton**)** **{**

**case** 1**:**

printf**(**"\nPress the Triangle button\n"**);**

**break;**

**case** 2**:**

printf**(**"\nPress the Circle button\n"**);**

**break;**

**case** 3**:**

printf**(**"\nPress the Cross button\n"**);**

**break;**

**case** 4**:**

printf**(**"\nPress the Square button\n"**);**

**break;**

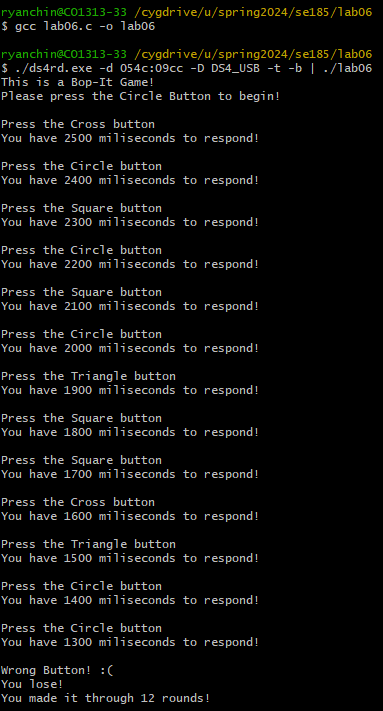
**}**

**return** randButton**;**

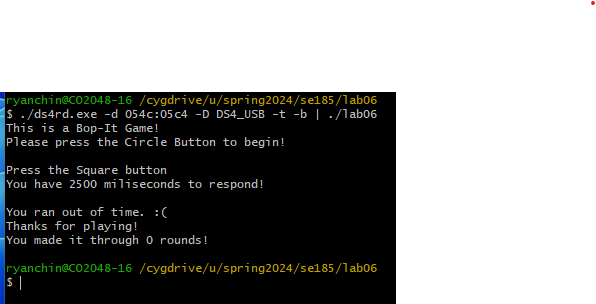
**}**

# Screen Shots

**Wrong Button**



**Ran Out of Time**

****