Jesus Sandoval – Draft Report

RedID: 820688304

Declaration

This is me, Jesus Sandoval, declaring that the work I am submitting is my original work except where explicitly marked otherwise.

Program Description

My game is like the original snake game released in 1997 by Nokia. There are key differences between my Snake Bonanza game and Nokia’s. First of them being the fact that my game is called Snake Bonanza. My game also works with other items and not only the food item. The snake can now interact with other items including, a food item, a wall item, and a speed item. The biggest difference between my game and the original is that the snake doesn’t grow and although you may think that will make the game easier, it’s not a bug it’s a feature, the game gets progressively harder as every time you eat a food item a new wall randomly appears further obstructing the player’s path. There will be many things the player will have to keep track of as the game progresses that will keep them on their toes making this game exciting.

C Implementation

#include <conio.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <time.h>

#include <windows.h>

#define rows 20

#define columns 40

// Used integers for loops

int i, j;

// For Map grid

int Map[rows][columns];

// Used integers for Snake movement and length and Food and Poison positions

int snakex, snakey, length, head, tail, food, poison;

// Used for key input logic and randomizing food, poison, and wall positions

int a, b, c, d, e, f, dir, diff, wall;

// Used to determine when game ends

int life;

// Creates my array and assigns initial values of variables

void Initialize()

{

// Assigns every field in my array to 0

for (i = 0; i < rows; i++)

{

for (j = 0; j < columns; j++)

{

Map[i][j] = 0;

}

}

// Snake will initialize in the center of the map

// Logic NOT MINE (atleast some)

snakex = rows / 2;

snakey = columns / 2;

head = 5;

tail = 1;

length = snakey;

// Initial values of food, poison, and wall

food = 0;

poison = 0;

wall = 0;

// Initializes the value of life

life = 1;

// Snake will initialize with a length of 5 with values, 1-2-3-4-(5 head)

for (i = 0; i < head; i++)

{

break;

length++;

Map[snakex][length - head] = i + 1;

}

}

// Food and Poison Positions

void Logic()

{

// Prints the borders, snake, food, poison, and walls

for (i = 0; i <= columns + 1; i++)

{

printf("%c", 254);

}

printf("\n");

for (i = 0; i < rows; i++)

{

printf("%c", 219);

for (j = 0; j < columns; j++)

{

if (Map[i][j] == 0) // Empty Space = 0

printf(" ");

if (Map[i][j] > 0) // Snake > 0

printf("O");

if (Map[i][j] == -1) // Food = -1

printf("%c", 248);

if (Map[i][j] == -2) // Poison = -2

printf("x");

if (Map[i][j] == -3) // Wall = -3

printf("%c", 254);

if (j == columns - 1)

printf("%c\n", 219);

}

}

for (i = 0; i <= columns + 1; i++)

{

printf("%c", 254);

}

}

/\*/ Points the cursor back to (0,0) to reprint updated Logic

//////////////////////////////////////////////////////////// NOT MINE

void Reset()

{

COORD coord;

coord.X = 0;

coord.Y = 0;

SetConsoleCursorPosition(GetStdHandle(STD\_OUTPUT\_HANDLE), coord);

}

\*/

// Random food generator

void Frandom()

{

srand(time(0));

// Location of food cannot be 0 or row size

a = 1 + rand() % (rows - 2);

// Location of food cannot be 0 or column size

b = 1 + rand() % (columns - 2);

// If there is no food on the screen and if empty space, runs if statement

if (food == 0 && Map[a][b] == 0)

{

Map[a][b] = -1;

food = 1; // Food is on screen now

}

}

// Random poison generator

void Prandom()

{

srand(time(0));

// Location of poison cannot be 0 or row size

c = 1 + rand() % (rows - 2);

// Location of poison cannot be 0 or column size

d = 1 + rand() % (columns - 2);

// If there is no poison on the screen and if empty space, runs if statement

if (poison == 0 && Map[c][d] == 0)

{

Map[c][d] = -2; // Assigns poison location to -2

poison = 1; // Poison is on screen now

}

}

// Random wall generator

void Wrandom()

{

// A maximum of 5 random walls will spawn

while (wall < 5)

{

// Location of wall cannot be 0 or row size

e = 1 + rand() % (rows - 2);

// Location of wall cannot be 0 or column size

f = 1 + rand() % (columns - 2);

// Checks if location is empty, if empty, spawns wall

if (Map[e][f] == 0)

{

Map[e][f] = -3; // Assigns wall location to -3

}

wall++;

}

}

//////////////////////////////////////////////////NOT MINE

// For Keyboard Input

int getchsnake()

{

if (\_kbhit())

return \_getch();

// else

// return -1; // return nothing

}

// Snake movement and interaction with the world

void Movement()

{

// Variable difference is assigned to key inputs

diff = getchsnake();

// Ascii value of w(up) = 119, a(left) = 97, s(down) = 115, d(right) = 100

// Snake cannot go from left to right or vice versa because their difference is not greater than 5

// Snake cannot go from up to down or vice versa because their difference is not greater than 5

if ((diff == 'w' || diff == 'a' || diff == 's' || diff == 'd') && abs(dir - diff) > 5)

{

dir = diff;

}

// Logic for up movement

if (dir == 'w')

{

// Snake hits top border, game ends

snakex--;

if (snakex == 0)

{

life = 0;

}

// Snake eats food, adds length to snake and updates food state

if (Map[snakex][snakey] == -1)

{

food = 0;

tail -= 1;

}

// Snake eats poison, removes length from snake and updates poison state

if (Map[snakex][snakey] == -2)

{

poison = 0;

tail +=1;

}

// Snake hits one of the random walls, game ends

if (Map[snakex][snakey] == -3)

{

life = 0;

}

// Snake will move one position up

head++;

Map[snakex][snakey] = head;

}

// Logic for left movement

if (dir == 'a')

{

// Snake hits left border, game ends

snakey--;

if (snakey == 0)

{

life = 0;

}

// Snake eats food, adds length to snake and updates food state

if (Map[snakex][snakey] == -1)

{

food = 0;

tail -= 1;

}

// Snake eats poison, removes length from snake and updates poison state

if (Map[snakex][snakey] == -2)

{

poison = 0;

tail +=1;

}

// Snake hits one of the random walls, game ends

if (Map[snakex][snakey] == -3)

{

life = 0;

}

// Snake will move one position left

head++;

Map[snakex][snakey] = head;

}

// Logic for down movement

if (dir == 's')

{

// Snake hits bottom border, game ends

snakex++;

if (snakex == rows - 1)

{

life = 0;

}

// Snake eats food, adds length to snake and updates food state

if (Map[snakex][snakey] == -1)

{

food = 0;

tail -= 1;

}

// Snake eats poison, removes length from snake and updates poison state

if (Map[snakex][snakey] == -2)

{

poison = 0;

tail +=1;

}

// Snake hits one of the random walls, game ends

if (Map[snakex][snakey] == -3)

{

life = 0;

}

// Snake will move down position up

head++;

Map[snakex][snakey] = head;

}

// Logic for right movement

if (dir == 'd')

{

// Snake hits right border, game ends

snakey++;

if (snakey == columns - 1)

{

life = 0;

}

// Snake eats food, adds length to snake and updates food state

if (Map[snakex][snakey] == -1)

{

food = 0;

tail -= 1;

}

// Snake eats poison, removes length from snake and updates poison state

if (Map[snakex][snakey] == -2)

{

poison = 0;

tail +=1;

}

// Snake hits one of the random walls, game ends

if (Map[snakex][snakey] == -3)

{

life = 0;

}

// Snake will move right position up

head++;

Map[snakex][snakey] = head;

}

}

// Logic to update snake's tail

void Remove()

{

// Looks for tail and assigns it a new value of 0

for (i = 0; i < rows; i++)

{

for (j = 0; j < columns; j++)

{

if (Map[i][j] == tail)

{

Map[i][j] = 0;

}

}

}

// Updates tail value back to original value of 1

tail++;

}

int main()

{

// Initializes snake to begin movement right

dir = 'd';

// Snake Initialization

Initialize();

// Creates the loop to update and print game

while (life == 1)

{

Logic();

Reset();

Frandom();

Prandom();

Wrandom();

Movement();

Remove();

Sleep(66);

}

return 0;

}

Tests I’ve Performed

I performed many tests where my snake eats a food item and a new food item, and a new wall randomly appear as intended. The key movement works fine, and the bitmap display works fine along with the keyboard emulator.

A screenshot of a computer

Description automatically generated with medium confidence

Tasks completed

I did not make any big changes to my MIPS project other than implementing a use for the stack to store the $ra register. However, tasks completed since the beginning of my project include C implementation and MIPS translation to most of my features.

Tasks not completed

So, there are some minor bugs. To begin with, sometimes my random items appear on the borders of the bitmap display where the borders of my game are supposed to be which makes it weird for the player to go over the borders to eat a food item. This brings me to the lack of collision I had to implement on the border walls for this same reason. However, the snake does interact with the randomly placed “white” walls. The biggest bug however happens when I press inputs too fast, my whole MIPS program crashes.

Conclusion

If I could do this project all over again, I would have not enrolled to Professor Arnold’s Compe 470 course along with his 271. I though I could manage both but by the end of the semester I’ve grown tired and burnt out. Now I’m presenting something I am not entirely proud since I know I could’ve written much cleaner code.

Implementation in MIPS

.data

.eqv Blue, 0x0000ffff # snake color

.eqv Green, 0x00FF00 # food color

.eqv White, 0xFFFFFF # wall color

.eqv UP, 119

.eqv RIGHT, 100

.eqv DOWN, 115

.eqv LEFT, 97

.eqv gameSize, 64

.text

############### draw border function #################

border:

li $t2, 0 # i = 0

li $t3, gameSize

li $t0, 0x10008000

li $t1,Blue

topBorder:

beq $t2, $t3, sideBorder # for ( i < topRight )

sw $t1, ($t0) # print(blue)

addi $t0, $t0, 4 # else

addi $t2, $t2, 1 # branch to sideBorder

j topBorder

sideBorder:

li $t2, 0 # reset i = 0

li $t3, 62

sideBegin:

beq $t2, $t3, botBorder # for (i < bottom)

sw $t1, ($t0) # printleft(blue)

addi $t0, $t0, 252 # printright(blue)

sw $t1, ($t0) # else

addi $t0, $t0, 4 # branch to botBorder

addi $t2, $t2, 1 # i++

j sideBegin

botBorder:

li $t2, 0 # reset

li $t3, gameSize # new $t3 for bottom

botBegin:

beq $t2, $t3, reset

sw $t1, ($t0)

addi $t2, $t2, 1 # i++

addi $t0, $t0, 4

j botBegin

############## reset registers ########################

reset:

li $t0, 0

li $t1, 0

li $t2, 0

li $t3, 0

li $t4, 0

li $t5, 0

li $t6, 0

li $t7, 0

li $t8, 0

li $t9, 0

li $s0, 0

li $s1, 0

li $s2, 0

li $s3, 0

li $s4, 0

li $s5, 0

li $s6, 0

li $s7, 0

###############################################

begin:

li $t0, 0x10008000 # bitmap start address

li $t1, Blue # $t1 = snake

li $t4, 0 # $t4 = background

li $t5, Green # $t5 = food

li $t7, 0 # holds value of input

addi $s1, $zero, 10 # speed

li $s3, 0 # used to check if food

li $s5, White # wall

li $t8, 0xFF0000 # red for speed item

li $t2, 0x1000A080 # center of bitmap

sw $t1, ($t2) # snake spawns here

li $t3, 0 # reserved to check snake position

li $s2, 5 # initial snake length

################## Random Speed Item ####################

li $a1, 1024 # possible values

li $v0, 42 # speed location = rand();

syscall

move $t6, $a0 # $t6 hold speed location

mul $t6, $t6, 16 # multiplier to stay in range of bitmap

add $t6, $t6, $t0 # from base address

sw $t8, ($t6) # places random speed item

################## Random food ########################

randomFood:

jal randomWall

li $a1, 1024 # possible values

li $v0, 42 # food location = rand();

syscall

move $s4, $a0 # $t6 hold fruit location

mul $s4, $s4, 16 # multiplier to stay in range of bitmap

add $s4, $s4, $t0 # from base address

sw $t5, ($s4) # places random location for food

j check

################## Random wall ########################

stackreturn:

lw $ra, ($sp)

addi $sp, $sp, 4

jr $ra

randomWall:

li $a1, 1024 # possible values

li $v0, 42 # wall location = rand();

syscall

move $t6, $a0 # $t6 hold wall location

mul $t6, $t6, 16 # multiplier to stay in range of bitmap

add $t6, $t6, $t0 # from base address

sw $s5, ($t6) # places random wall

addi $sp, $sp, -4

sw $ra, ($sp)

jal stackreturn

################## Checks for key input ###############

check:

lw $t9, 0xffff0004 # key input

li $v0, 32

li $a0, 50

syscall

decision:

beq $t9, UP, snakeUp

beq $t9, RIGHT, snakeRight # case (condition)

beq $t9, DOWN, snakeDown

beq $t9, LEFT, snakeLeft

beq $t7, UP, snakeUp

beq $t7, RIGHT, snakeRight

beq $t7, DOWN, snakeDown

beq $t7, LEFT, snakeLeft

beq $t9, $zero, check

beq $t7, $zero, check

snakeUp:

li $s3, 0

li $t7, UP # hold condition for check loop

add $t3, $t2, $zero # save current location

lw $s3, -256($t2) # get next location

beq $s3, $s5, end # if wall - do

addi $t2, $t2, -256

sw $t1, ($t2) # move

sw $t4, ($t3)

beq $s3, $t5, eatFood # if foodood - do

beq $s3, $t8, upSpeed # if speed - do

li $v0, 32

add $a0, $zero, $s1 # speed

syscall

j check

snakeRight:

li $s3, 0

li $t7, RIGHT # hold condition for check loop

add $t3, $t2, $zero # save current location

lw $s3, 4($t2) # get next location

beq $s3, $s5, end # if wall - do

addi $t2, $t2, 4

sw $t1, ($t2) # move

sw $t4, ($t3)

beq $s3, $t5, eatFood # if food - do

beq $s3, $t8, upSpeed # if speed - do

li $v0, 32

add $a0, $zero, $s1 # speed

syscall

j check

snakeDown:

li $s3, 0

li $t7, DOWN # hold condition for check loop

add $t3, $t2, $zero # save current location

lw $s3, 256($t2) # get next location

beq $s3, $s5, end # if wall - do

addi $t2, $t2, 256

sw $t1, ($t2) # move

sw $t4, ($t3)

beq $s3, $t5, eatFood # if food - do

beq $s3, $t8, upSpeed # if speed - do

li $v0, 32

add $a0, $zero, $s1 # speed

syscall

j check

snakeLeft:

li $s3, 0

li $t7, LEFT # hold condition for check loop

add $t3, $t2, $zero # save current location

lw $s3, -4($t2) # get next location

beq $s3, $s5, end # if wall - do

addi $t2, $t2, -4

sw $t1, ($t2) # move

sw $t4, ($t3)

beq $s3, $t5, eatFood # if food - do

beq $s3, $t8, upSpeed # if speed - do

li $v0, 32

add $a0, $zero, $s1 # speed

syscall

j check

eatFood:

j randomFood

################# Random Speed Item ######################

upSpeed:

addi $s1, $s1, -5

li $a1, 1024 # possible values

li $v0, 42

syscall

move $t6, $a0 # $t6 hold speed location

mul $t6, $t6, 16 # multiplier to stay in range of bitmap

add $t6, $t6, $t0 # from base address

sw $t8, ($t6) # places random speed item

j check

end:

li $v0, 10

syscall

Credits:

For snake array logic in C

<https://www.quora.com/What-is-the-logic-behind-a-snake-game-from-a-programming-point-of-view>

For reset screen in C

<https://stackoverflow.com/questions/10401724/move-text-cursor-to-particular-screen-coordinate>

For check any input in MIPS

<http://www.cim.mcgill.ca/~langer/273/20-slides.pdf>