CS210 MAJOR PROJECT REPORT

AVOID-THE-ENEMY GAME

Submitted by:- Kashika Aggarwal, 2201CS35

Project Description:

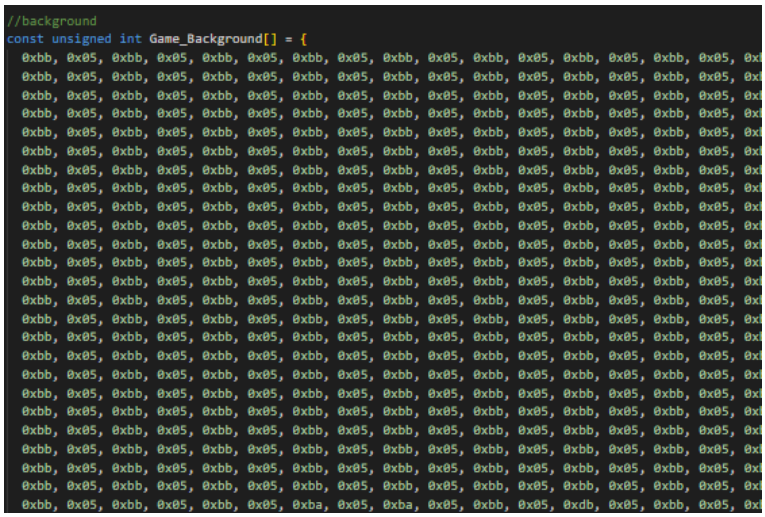
In the vast ocean, Finley, a small but determined fish, encountered a menacing school of sharks known for their tyranny. He darted through the water with waves crashing, evading the circling predators. Despite the odds, Finley persisted, relying on agility and quick reflexes. As the battle intensified, his resolve strengthened. Dodging and weaving, he escaped the sharks, emerging triumphant. With a final flourish, he dispersed the predators, becoming the sea's hero. Admired for his bravery, Finley swam into the sunset, knowing no challenge was insurmountable with courage. His tale echoed through the ocean, inspiring all who heard it.

This game is an infinite runner type game. The sharks will randomly spawn until Finley dies by hitting the shark.

Controls:

Used W-A-S-D keys on the keyboard to move your character. Simply tapping is enough to make your character to move in a direction (Holding down overloads CPUlator PS/2 port). Make sure to type/enter W-A-S-D into the PS/2 Port with address ff200100.

Explanation:

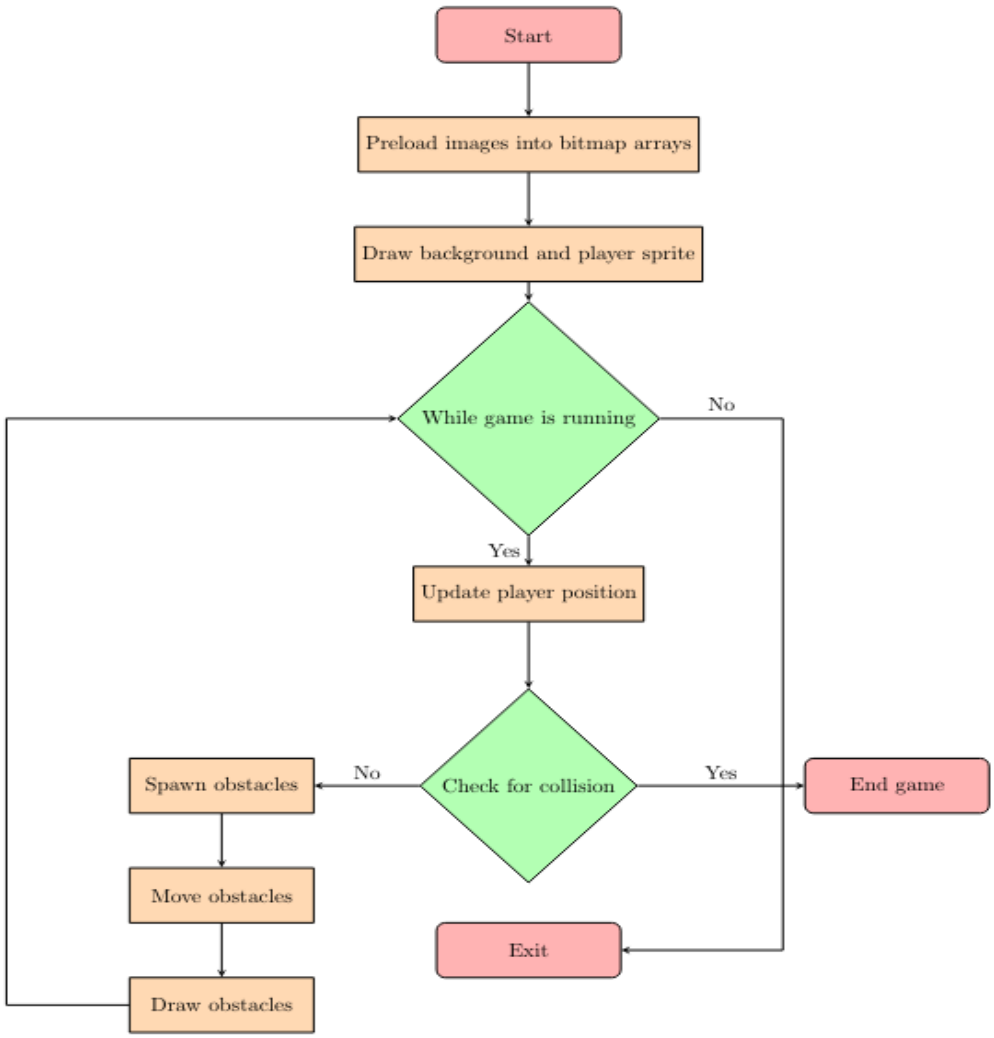


Images are preloaded into a bitmap to draw the sprites(shown). This can be seen by the large arrays at the top of the file. The draw functions only iterate the required elements of the bitmap array by using player/enemy positions as boundaries to the iteration. To re-draw sprites after
movement, the clear functions only draw the background over the previous sprite position. Doing the techniques mentioned enables for faster animation times.

The enemy spawning algorithm is based on a randomized spawning interval and position. Obstacles are stored in a linked list, enabling easy deletion once the obstacle is offscreen.

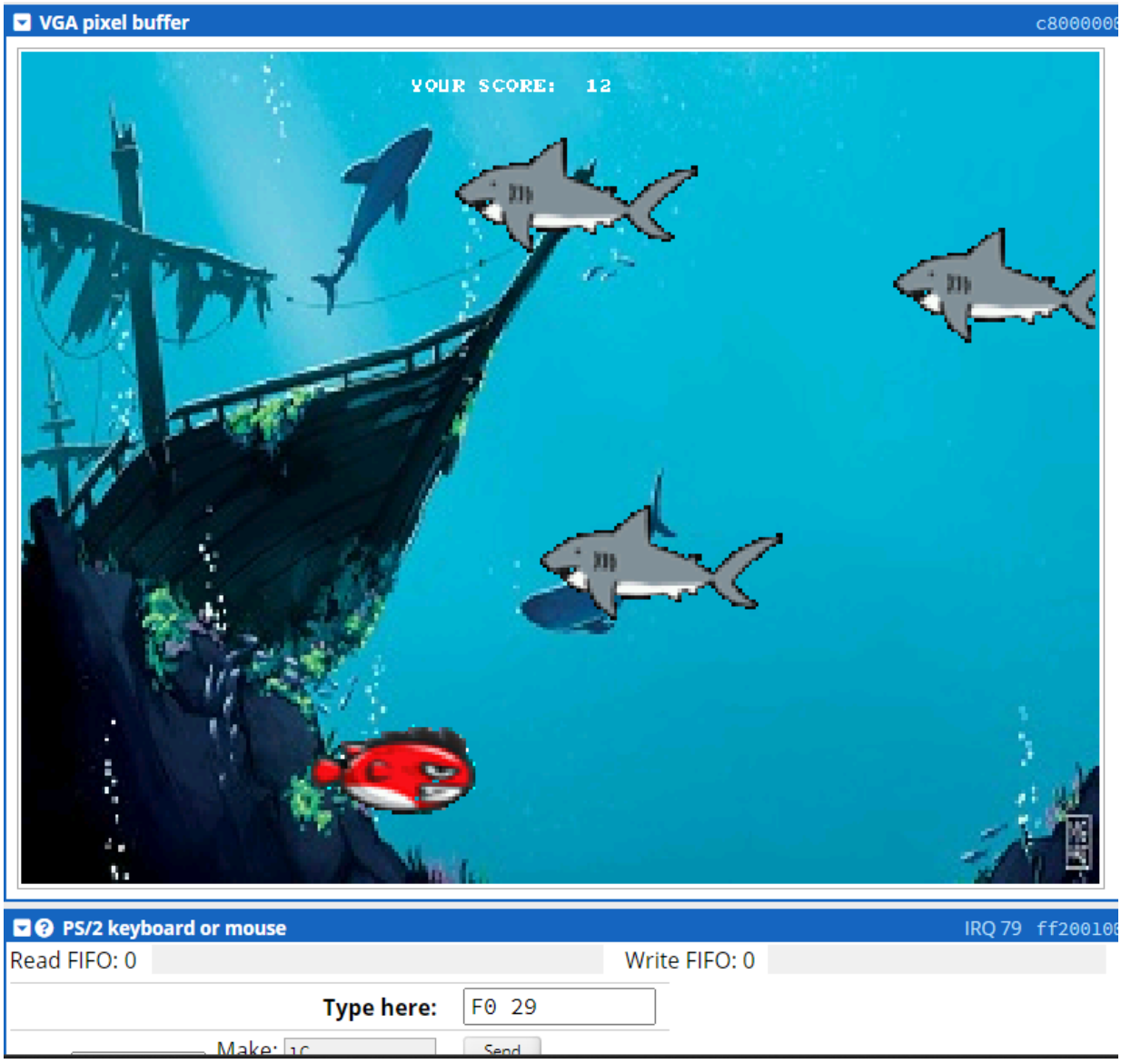
The algorithm used for collision is by position comparison. The algorithm compares the player's position with respect to each shark's position to determine if they collide.

Below is a brief flowchart of the working:

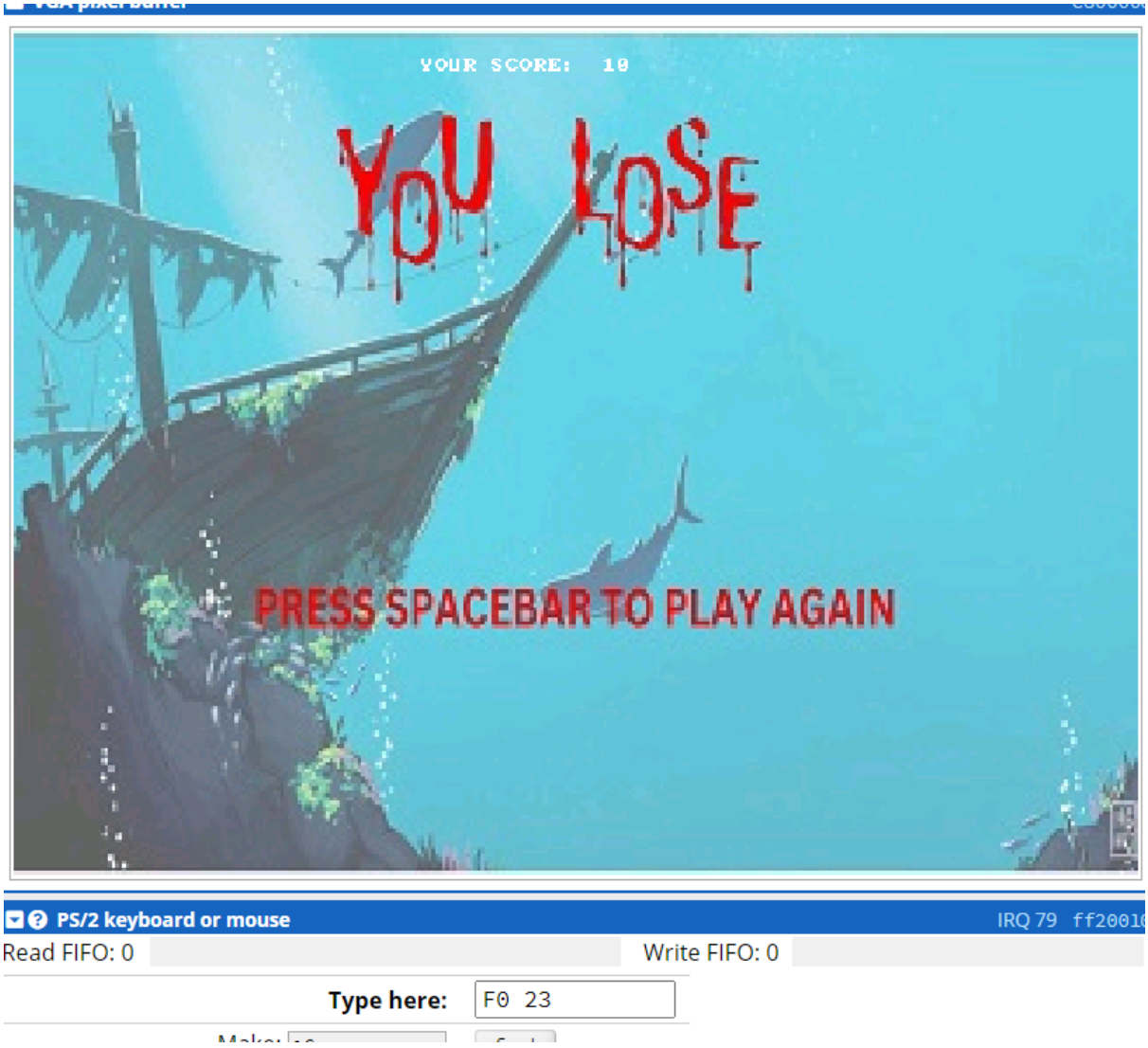


Game Display:

Below is how the game looks on CPUlator:
(both the game lose and game run screens shown)



Gameplay as it's in action



Sources:

1) To convert images into an RGB coded C-array, I used an online source:
https://lvgl.io/tools/imageconverter

2) General Information regarding the project and its working:

Google + ChatGPT

3) Images and editing:

Google + Canva

(Gameplay Video is attached in the submission)

Game Code:

```
#include <stdlib.h>
#include <stdbool.h>
#include <stdio.h>

//Global VGA pointers
volatile int pixel_buffer_start; //Pixel buffer
volatile char *character_buffer = (char *) 0xC9000000;// VGA character buffer
volatile int *LEDR_ptr = (int *) 0xFF200000;  //Led Ptr
//background
const unsigned int Game_Background[] = {};
const unsigned int Enemy[] = {};
const unsigned int Player[] = {};
const unsigned int You_Lose[] = {};
/* Structure definition */
//enemy_fish
struct enemy_fish{
    int x;
    int y;
    int x_speed;
    int width;
    int height;
};

//Player
struct Player{
    int x;
    int y;
    int x_size;
    int y_size;
    int y_dir;
    int x_dir;
};

//Linked list of enemy
struct node{
    struct enemy_fish data;
    struct node *next;
};

/* ////////////////////////////////////////////////// Functions
//////////////////////////////////////////// */
//Waits for buffers to sync before switching
void sync_wait_time()
{
    volatile int * pixel_ctrl_ptr = (int *)0xFF203020;
```

```
    volatile int * status =(int *)0xFF20302C;

    *pixel_ctrl_ptr = 1;

    while((*status &0x1) != 0)
    {
        //do nothing
    }
    return;
}
// int compute(unsigned int arr[],int k,int c){
//     if(c==1) return ((arr[k + 1] & 0xF8) >> 3) << 11;
//     if(c==2) return (((arr[k] & 0xE0) >> 5)) | ((arr[k+1] & 0x7) << 3);
//     if(c==3) return (arr[k] & 0x1f);
// }
//plots pixels on VGA
void write_pixel(int x, int y, short int line_color)
{
    *(short int *)(pixel_buffer_start + (y << 10) + (x << 1)) = line_color;
}
//clears screen to background
void clear_to_bg()
{
    int i = 0, j = 0;
    //make the bg bitmap of size (320x240)
    for (int k = 0 ; k < 320 * 240 * 2 - 1; k+= 2) //320*24*2 to access 2 bytes at a time
    {
        int red = ((Game_Background[k + 1] & 0xF8) >> 3) << 11;
        int green  = (((Game_Background[k] & 0xE0) >> 5)) | ((Game_Background[k+1] & 0x7) <<
3) ;
        int blue = (Game_Background[k] & 0x1f);
        short int p = red | ( (green << 5) | blue);
        write_pixel(i, j, p);
        i++;
        if (i == 320)
        {
            i = 0;
            j++;
        }
    }
}

void draw_player(int x, int y, int x_size, int y_size, int offset)
{
    int i = x - offset;
    int j = y;

    //Iterates through character bitmap
```

```
  for (int k = 0 ; k < x_size * y_size * 2 - 1; k+= 2)
  {
     int red = ((Player[k + 1] & 0xF8) >> 3) << 11;
     int green  = (((Player[k] & 0xE0) >> 5)) | ((Player[k+1] & 0x7) << 3) ;
     int blue = (Player[k] & 0x1f);

     short int p = red | ( (green << 5) | blue);

     //Checks if player is onscreen
     if(i <= 318 && i >= 1 && j >= 1  && j <= 238 && Player[k] != 0x00)
     {
        write_pixel(i, j, p);
     }
     i++;
     if (i == x + x_size + offset)
     {
        i = x - offset;
        j++;
     }
     if (j == y + y_size)
     {
        return;
     }
  }
}
short int return_color(int x, int y);
//erases n draws new
void clear_player(int x, int y, int x_size, int y_size, int offset){
  int i = x - offset;
  int j = y;
  for (int k = ((320 * j + i) * 2); k < (320 * 240 * 2 - 1); k+= 2)
  {
     //graphics and colours
     int red = ((Game_Background[k + 1] & 0xF8) >> 3) << 11;
     int green  = (((Game_Background[k] & 0xE0) >> 5)) | ((Game_Background[k+1] & 0x7) <<
3) ;
     int blue = (Game_Background[k] & 0x1f);

     short int p = red | ( (green << 5) | blue);

     if(i <= 318 && i >= 1 && j >= 1  && j <= 238)
     {
        short int color = return_color(i,j);
        //If color is not what is expected, update the pixel
        if(color != p)
        {
           write_pixel( 0 + i, j, p);
        }
```

```
      }

      i++;
      if (i == x + x_size + offset)
      {
         i = x - offset;
         j++;
         k = ((320*j + i)*2 - 2);
      }

      if (j == y + y_size)
      {
         return;
      }
   }
}
//draws youlose screen
void draw_game_over_screen()
{
   int i = 0, j = 0;
   for (int k = 0 ; k < 320 * 240 * 2 - 1; k+= 2)
   {
      int R = ((You_Lose[k + 1] & 0xF8) >> 3) << 11;
      int G  = (((You_Lose[k] & 0xE0) >> 5)) | ((You_Lose[k+1] & 0x7) << 3) ;
      int B = (You_Lose[k] & 0x1f);

      short int p = R | ( (G << 5) | B);

      write_pixel( 0 + i, j, p);

      i+=1;

      if (i == 320)
      {
         i = 0;
         j+=1;
      }
   }
}

//Spawning enemy fish algorithm
struct node* spawn_enemy_fish(struct node* head);
//Draws enemy fish
struct node* draw_enemy_fish(struct node* head);
//Creates enemy fish (constructor)
struct enemy_fish create_enemy_fish();
//Helper to draw enemy_fish
void draw_enemy_fish_helper(struct node* curr, int offset)
```

```
{
   //Gets current enemy_fish
   struct enemy_fish obs = curr->data;
   int i = obs.x - offset;
   int j = obs.y;

   //Iterates through arr
   for (int k = 0 ; k < obs.width * obs.height * 2 - 1; k += 2)
   {
      int red = ((Enemy[k + 1] & 0xF8) >> 3) << 11;
      int green  = (((Enemy[k] & 0xE0) >> 5)) | ((Enemy[k+1] & 0x7) << 3) ;
      int blue = (Enemy[k] & 0x1f);

      //Converts arr to RGB
      short int p = red | ( (green << 5) | blue);

      //Checks if onscreen
      if(i <= 318 && i >= 1 && j >= 1  && j <= 238)
      {
         //Checks if the color isnt already the same
         if (!(Enemy[k] == 0x00 && Enemy[k+1] == 0x00))
         {
            write_pixel(i, j, p);
         }
      }
      //Iterates through enemy_fish size
      i++;
      if (i == obs.x + obs.width + offset)
      {
         i = obs.x - offset;
         j++;
      }

      if (j >= obs.y + obs.height)
      {
         //Done
         return;
      }
   }
}

//Helper to clear enemy_fish by erasing old enemy_fish position
void clear_enemy_fish_helper(struct node* curr, int offset)
{
   //Gets enemy_fish
   struct enemy_fish obs = curr->data;
   int i = obs.x;
   int j = obs.y;
```

```
   //Iterate through backgound
   for (int k = ((320 * j + i) * 2); k < (320 * 240 * 2 - 1); k+= 2)
   {
      int red = ((Game_Background[k + 1] & 0xF8) >> 3) << 11;
      int green  = (((Game_Background[k] & 0xE0) >> 5)) | ((Game_Background[k+1] & 0x7) <<
3) ;
      int blue = (Game_Background[k] & 0x1f);

      short int p = red | ( (green << 5) | blue);

      if(i <= 318 && i >= 1 && j >= 1  && j <= 238)
      {
         short int color = return_color(i,j);
         if(color != p)
         {
            write_pixel(i, j, p);
         }
      }
      i++;
      if (i == obs.x + obs.width + offset + 1)
      {
         i = obs.x;
         j++;
         k = ((320*j + i)*2 - 2);
      }

      if (j >= obs.y + obs.height){
         return;
      }
   }
}

//Checks if player collided with enemy_fish
bool collision(struct Player player, struct node* head);

//runs game over loop waiting for restart
void game_over();

void print_score(int x, int y, char *scorePtr){
   /* assume that the text string fits on one line */
   int offset = (y << 7) + x;

   while (*(scorePtr)){ // while it hasn't reach the null-terminating char in the string
      // write to the character buffer
      *(character_buffer + offset) = *(scorePtr);
      scorePtr++;
      offset++;
```

```
    }
}
/* ////////////////////////////////////////////////// Main
function ////////////////////////////////////////////// */
int main(void)
{
    while(true)
    {
        //runs main game
        //Initialize the score to 0
    int totalScore = 0;
    int scoreOnes = 0;
    int scoreTens = 0;
    int scoreHundreds = 0;
    *LEDR_ptr = 0;
    int timeCount = 0;

    //Initialize FPGA
    volatile int * pixel_ctrl_ptr = (int *)0xFF203020; //Vga buffer
    volatile int* PS2_ptr = (int *) 0xFF200100; // ps2 keyboard

    *(pixel_ctrl_ptr + 1) = 0xC8000000;
    sync_wait_time();
    pixel_buffer_start = *pixel_ctrl_ptr;
    clear_to_bg();

    *(pixel_ctrl_ptr + 1) = 0xC0000000;
    pixel_buffer_start = *(pixel_ctrl_ptr + 1); // we draw on the back buffer
    clear_to_bg();

    //Player attributes
    //PLAYER_X_SIZE = 75;
    //PLAYER_Y_SIZE = 45;
    //PLAYER_START_X = 75;
    //PLAYER_START_Y = 185;

    //initialize player
    struct Player player;
    player.x = 75;
    player.y = 185;
    player.y_dir = 0;
    player.x_dir = 0;
    player.x_size = 75;
    player.y_size = 45;


    //Initialize enemy_fishs
    //Make linked list of enemy_fishs
    //new enemy_fishs always are at the end of the screen (end of list)
```

```
//when enemy_fishs get off screen: -> delete head of list
struct node *head = NULL;

//Initialize Ps2 Keyboard data
int PS2_data;
unsigned char data_in = 0;

bool gameOver = false;

//Main game loop
while(!gameOver){
   timeCount++;

   // Plot score
   if (timeCount == 30) {
      timeCount = 0;
      totalScore++;
      scoreHundreds = totalScore / 100;
      scoreTens = (totalScore - scoreHundreds * 100) / 10;
      scoreOnes = totalScore - scoreHundreds * 100 - scoreTens * 10;
      *LEDR_ptr = totalScore;
   }

   char myScoreString[17];
   myScoreString[0] = 'Y';
   myScoreString[1] = 'O';
   myScoreString[2] = 'U';
   myScoreString[3] = 'R';
   myScoreString[4] = ' ';
   myScoreString[5] = 'S';
   myScoreString[6] = 'C';
   myScoreString[7] = 'O';
   myScoreString[8] = 'R';
   myScoreString[9] = 'E';
   myScoreString[10] = ':';
   myScoreString[11] = ' ';

   if (scoreHundreds != 0) {
      myScoreString[12] = scoreHundreds + '0';
   }

   else {
      myScoreString[12] = ' ';
   }

   if (scoreHundreds == 0 && scoreTens == 0) {
      myScoreString[13] = ' ';
   }
```

```
    else {
        myScoreString[13] = scoreTens + '0';
    }

    myScoreString[14] = scoreOnes + '0';
    myScoreString[15] = '\0';

    print_score(285, 0, myScoreString);

    PS2_data = *PS2_ptr;
    data_in = PS2_data & 0xFF;

    //Gets direction code
    switch(data_in) {
        case 0x1B:
    if (player.y + player.y_size < 238) {
        player.x_dir = 0;
        player.y_dir = 3;
    }
    break;
        case 0x1D:
    if (player.y > 1) {
        player.x_dir = 0;
        player.y_dir = -3;
    }
    break;
        case 0x1C:
    if (player.x > 1) {
        player.y_dir = 0;
        player.x_dir = -3;
    }
    break;
        case 0x23:
    if (player.x + player.x_size < 238) {
        player.y_dir = 0;
        player.x_dir = 3;
    }
    break;
        default:
    player.y_dir = 0;
    player.x_dir = 0;
    break;
}



    // Erases player
```

```
        clear_player(player.x, player.y, player.x_size, player.y_size, 4);

        //updates new player direction
        player.y += player.y_dir;
        player.x += player.x_dir;
        // Draw player
        draw_player(player.x, player.y, player.x_size, player.y_size, 0);

        head = spawn_enemy_fish(head);
        head = draw_enemy_fish(head);
        gameOver = collision(player, head);

        sync_wait_time();
        pixel_buffer_start = *(pixel_ctrl_ptr + 1);
    }

        game_over();
    }
    return 0;
}

void game_over()
{
    volatile int * pixel_ctrl_ptr = (int *)0xFF203020;
    draw_game_over_screen();
    sync_wait_time();
    pixel_buffer_start = *(pixel_ctrl_ptr + 1);

    volatile int* PS2_ptr = (int *) 0xFF200100;
    int PS2_data;
    unsigned char data_in = 0;

    //Obtains spacebar input from user to restart
    while(true)
    {
        PS2_data = *PS2_ptr;
        data_in = PS2_data & 0xFF;
        if(data_in == 0x29)
        {
            break;
        }

    }
}

bool collision(struct Player player, struct node* head)
{
    //Pixel offset, required because enemy_fish hitbox is not perfect to the shape of enemy_fish
```

```
  if(head == NULL)
  {
    //No enemy_fish nothing to collide
    return false;
  }

  struct node* curr = head;
  while(curr->next != NULL)
  {
    struct enemy_fish obs = curr->data;
    if(obs.x <= 0)
    {
      curr = curr->next;
      //off screen but not deleted, skip
      continue;
    }
    else if(obs.x + obs.width < player.x - 5)
    {
      curr = curr->next;
      //player already passed this enemy_fish, skip
      continue;
    }
    else if(player.x + player.x_size >= obs.x && player.y >= obs.y && player.y <= obs.y +
obs.height - 25)
    {
      return true;
    }
    else if(player.x + player.x_size >= obs.x && obs.y >= player.y && obs.y <= player.y +
player.y_size - 25)
    {
      return true;
    }
    curr = curr->next;
  }
  return false;
}

//enemy fish spawning
struct enemy_fish create_enemy_fish()
{
  struct enemy_fish data;
  data.x = 318;
  data.y = rand () % 238;
  data.x_speed = -5;
  data.height = 45;
  data.width = 75;

  return data;
```

```
}
struct node* spawn_enemy_fish(struct node* head){
   //spawn rate
   int num = rand() % 50;

   if(num == 3)
   {
      //Initialize head of linked list
      if(head == NULL)
      {
         struct node* newNode = (struct node*)malloc(sizeof(struct node));
         struct enemy_fish data = create_enemy_fish();
         newNode->data = data;
         newNode->next = NULL;

         //Assigns to head
         head = newNode;
      }
      else
      {
         //Add to end of linked list
         struct node* curr = head;
         while(curr->next != NULL)
         {
            curr = curr->next;
         }
         struct node* newNode = (struct node*)malloc(sizeof(struct node));

         struct enemy_fish data = create_enemy_fish();
         newNode->data = data;
         newNode->next = NULL;

         curr->next = newNode;
      }
   }
   return head;
}

struct node* draw_enemy_fish(struct node* head) {

   //Empty list
   if(head == NULL)
   {
      return NULL;
   }

   struct node* curr = head;
```

```
  //Draws enemy_fishs
  while(curr->next != NULL)
  {
    //off screen, delete from list
    if(curr->data.x + curr->data.width <= 0)
    {
      struct node* temp = curr;
      curr = curr->next;
      free(temp);
      head = curr;
    }
    else
    {
      //clears old enemy_fishs
      clear_enemy_fish_helper(curr, curr->data.x_speed + 7);//curr->data.x_speed);
      curr->data.x +=  curr->data.x_speed;
      //Draws new enemy_fish positions
      draw_enemy_fish_helper(curr, 0);
      curr = curr->next;
    }
  }
  return head;
}
  //returns the color at a VGA pixel position
short int return_color(int x, int y)
{
  short int color = 0;
  return color = *(short int *)(pixel_buffer_start + (y << 10) + (x << 1));
}
//VGA_Y_MIN,VGA_Y_MAX,VGA_X_MIN, VGA_X_MIN defined
```