ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO

ĐỒ ÁN CROSSING ROAD (NHÓM 7)

Giáo viên hướng dẫn : Thầy Trương Toàn Thịnh

PHƯƠNG PHÁP LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG
Thành phố Hồ Chí Minh , ngày 10 tháng 12 năm 2

1. Introduction

The Crossing Road project is the second game that we, the K22 class, have had the opportunity to work on. This is the result of the collective effort of the four of us, although there may still be room for improvement.

Throughout the game development process, we have learned a great deal and enhanced our teamwork skills. We would like to express our gratitude to our teacher for providing us with this opportunity to experience game development, and we hope that you will be pleased with the final product.

2. Job Assignments

Group 7:

- Hồ Đăng Phúc - 22127492
- Võ Thị Hồng Minh - 22127277
- Nguyễn Văn Xuân Lộc - 22127237
- Hồ Trương Viết Long - 22127241

Person	Tasks
Hồ Đăng Phúc	Render image algorithm, sprites design, general algorithm, UI design
Võ Thị Hồng Minh	Check Collisions algorithm, Save/Load game, general algorithm
Nguyễn Văn Xuân Lộc	Collision algorithm, Game setup, Handling algorithm, Menu handling, General algorithm
Hồ Trương Viết Long	Sound Setting, Draw Art

3. Detailed Description

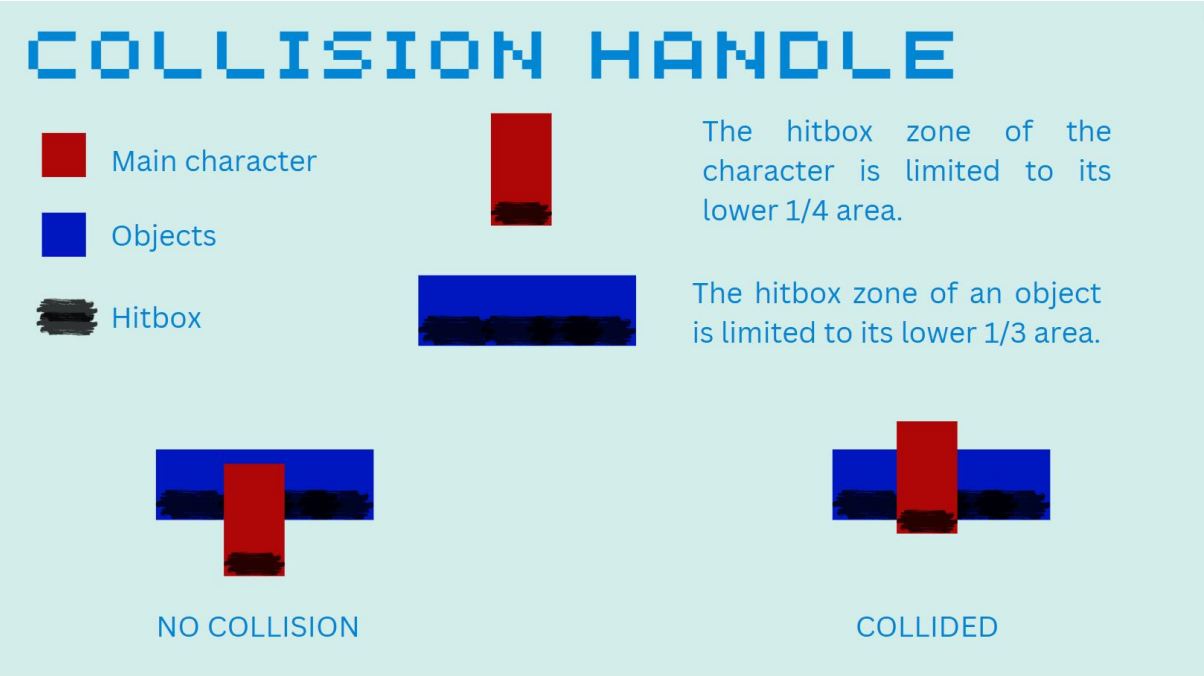
3.1 Overview

The primary goal of the game is to attain the highest score by safely navigating through the road without colliding with obstacles.

3.2 Tutorial

Movement: Use UP, DOWN, LEFT, RIGHT arrow button to control selected option and main character when in game.

Collision:



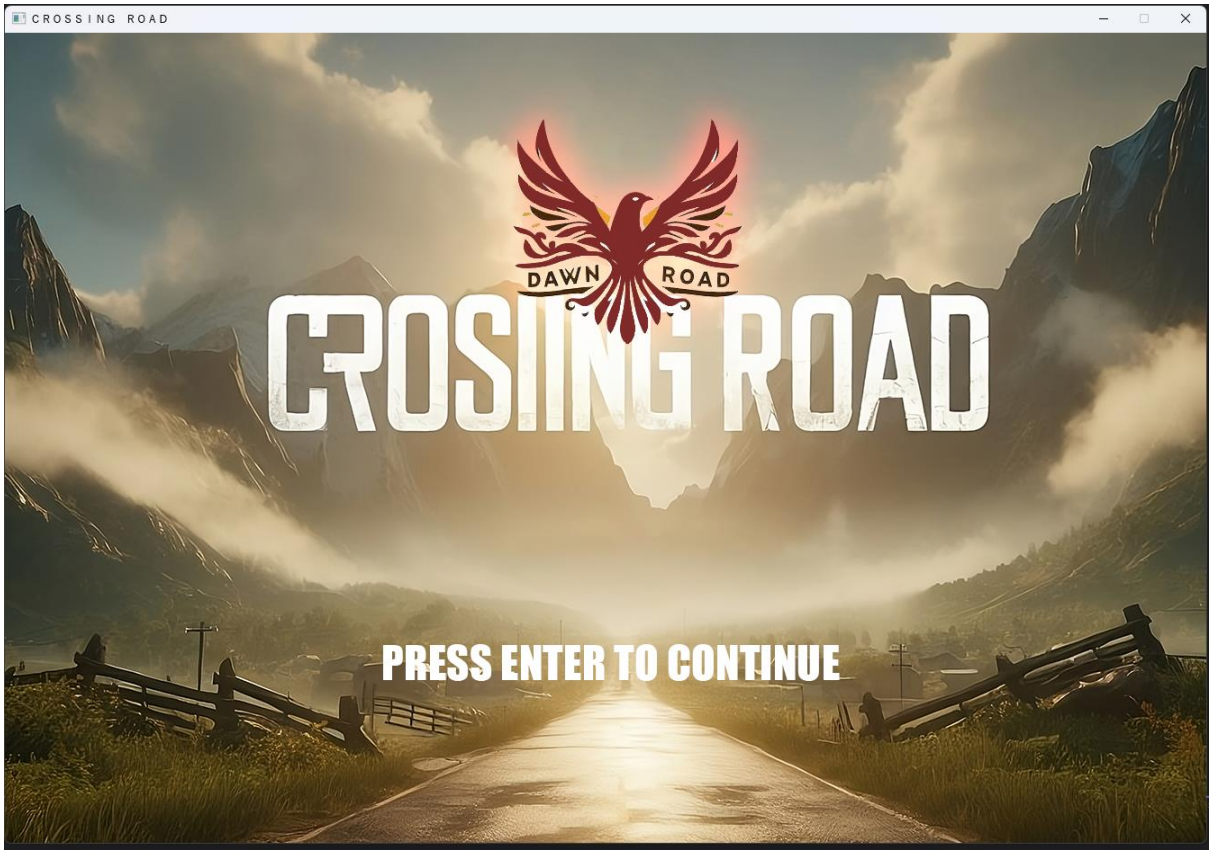
3.3 Game design

3.3.1 Menu and ingame

Upon launching the game, a brief video introduction will play, showcasing a car driving down a road. Once the introduction concludes, a title screen will appear, presenting the game's name. To proceed, the player is prompted to press the ENTER key.

*Some images:


CROSSING ROAD


CROSSING ROAD
DAWN
ROAD
CROSSING ROAD
PRESS ENTER TO CONTINUE

After that, main menu will show up, our main menu is in list type which contains :

New Game: Start a new game.

Load Game: Continue previous saved game.

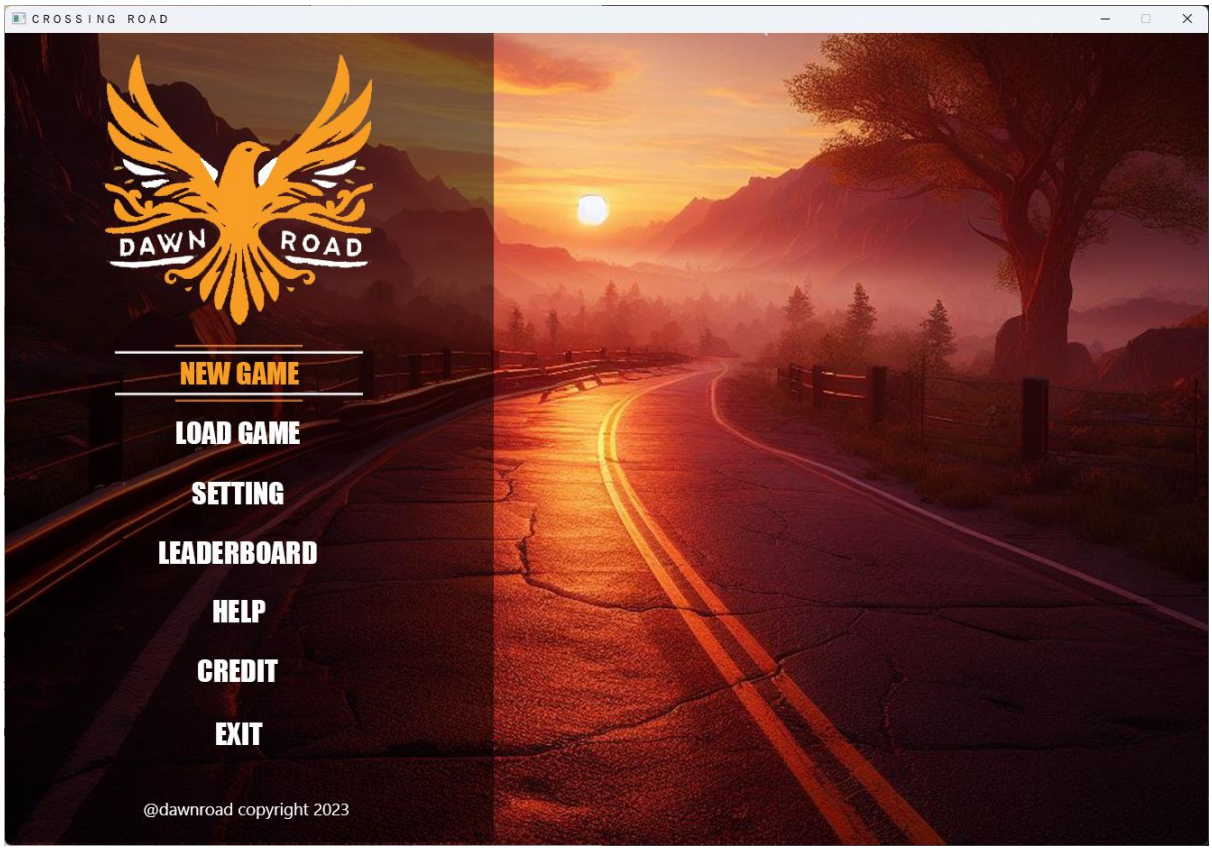
Setting*:* Modify volume of background and sound effect.

Leaderboard*:* Show top 3 players have the highest score.

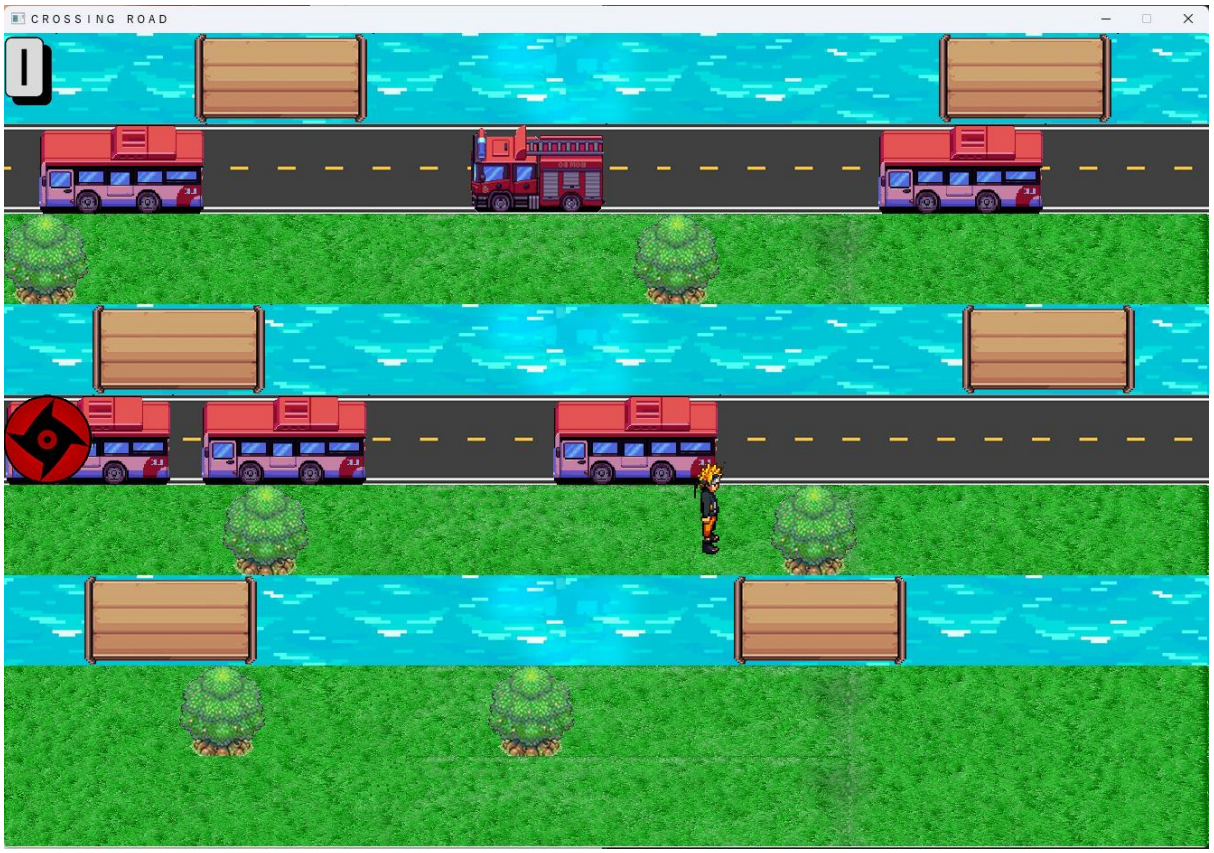
Help*:*A comprehensive tutorial for the game, which includes instructions on movement.

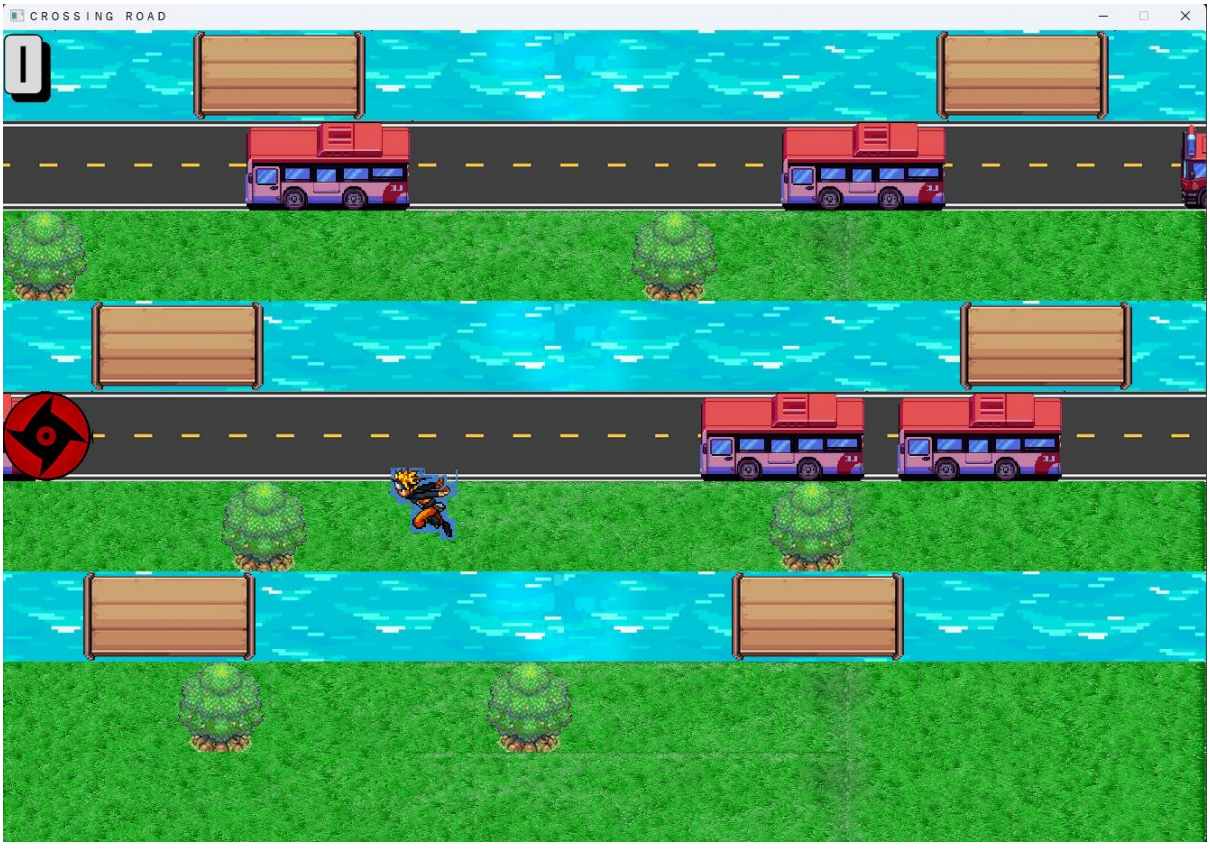
Credit*:* Display Information of team' members and our instructor.

Exit Game: Stop the game.



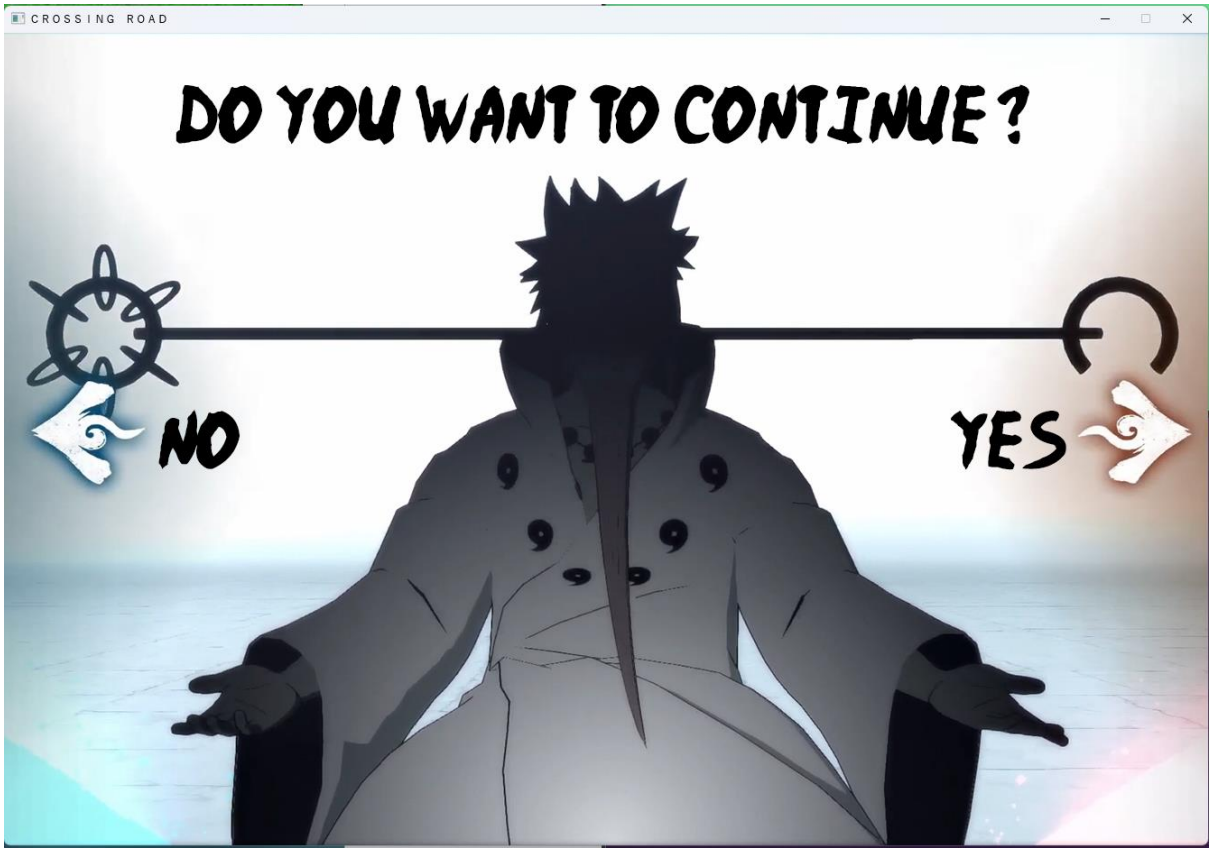
After clicking the Start Game button, the screen will switch to the game map. The player can move around the game by pressing the arrow keys: right arrow: left arrow, up arrow, down arrow to avoid obstacles (cars, rivers).


CROSSING ROAD
0


CROSSING ROAD
0

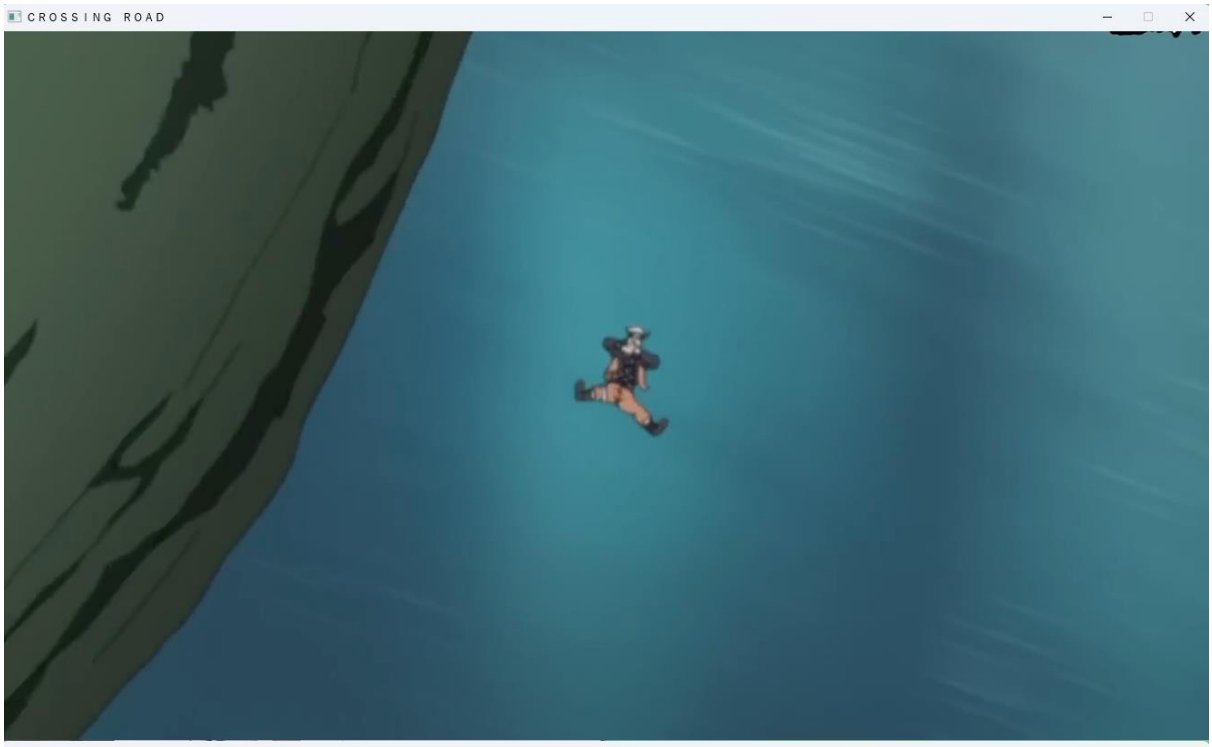
During gameplay, if the player wants to pause, they can press the ESC button, and the game will display the Pause Game screen with functions similar to the Menu. The player can select Save Game to save their progress, and they will need to enter their name. The player can find their saved file by clicking on the Load Game button in the main menu.

The game only has an Endless mode, players will play until they collide with objects in the game, leading to death. Then, a scene will appear asking if they want to continue. If they choose Yes, a new level will be created, and if they choose No, they will return to the main menu.

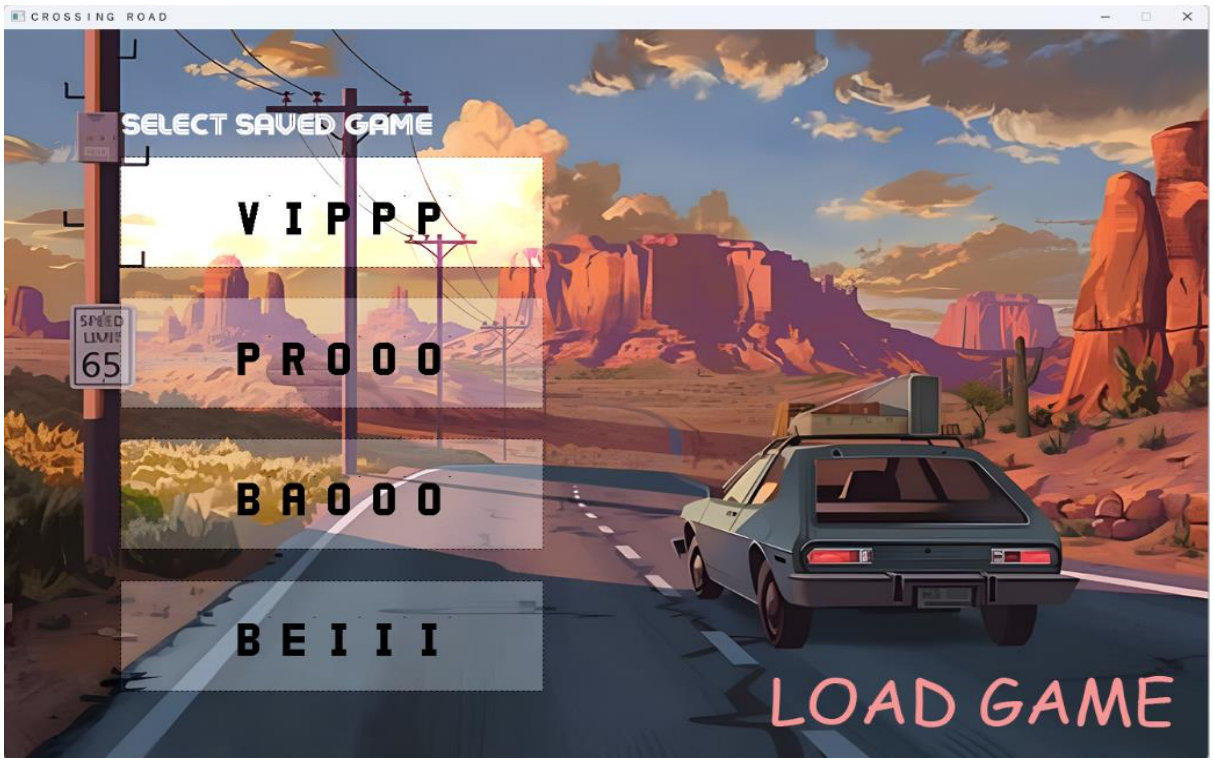


Dead effect:

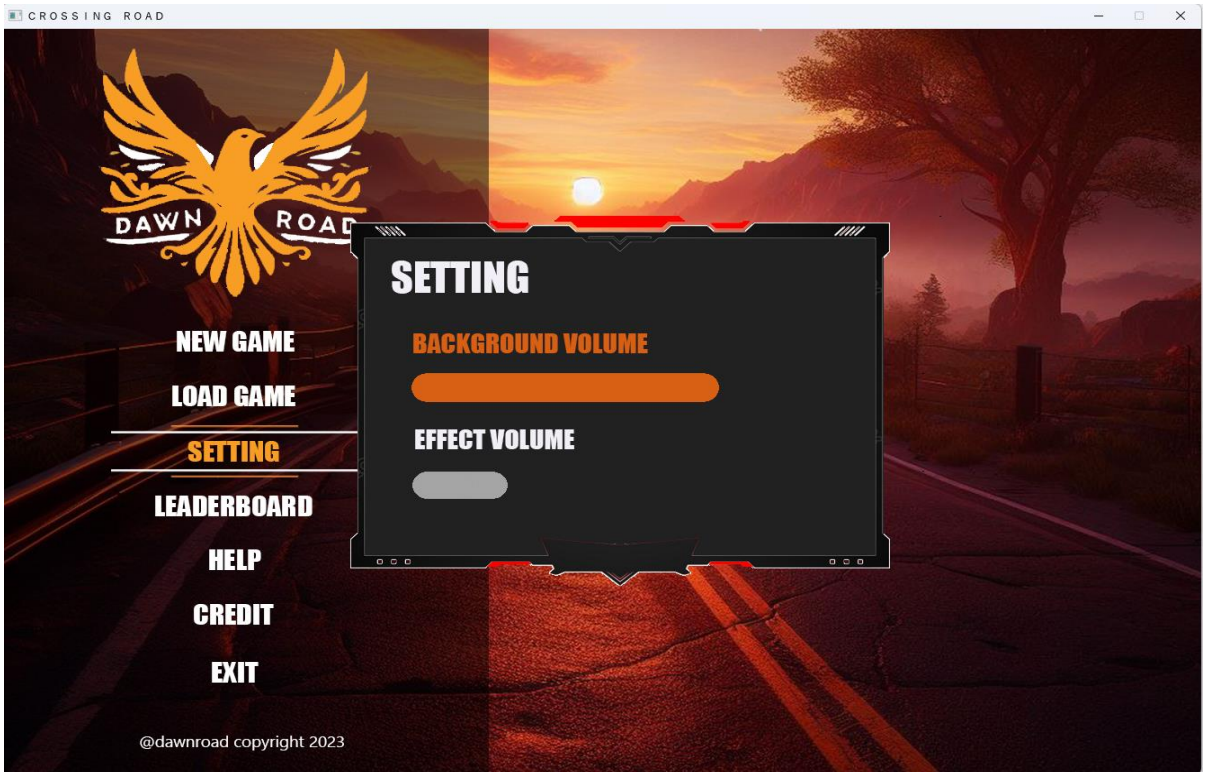
There are two types, death caused by obstacles and by falling into water. Each type has its own scene.



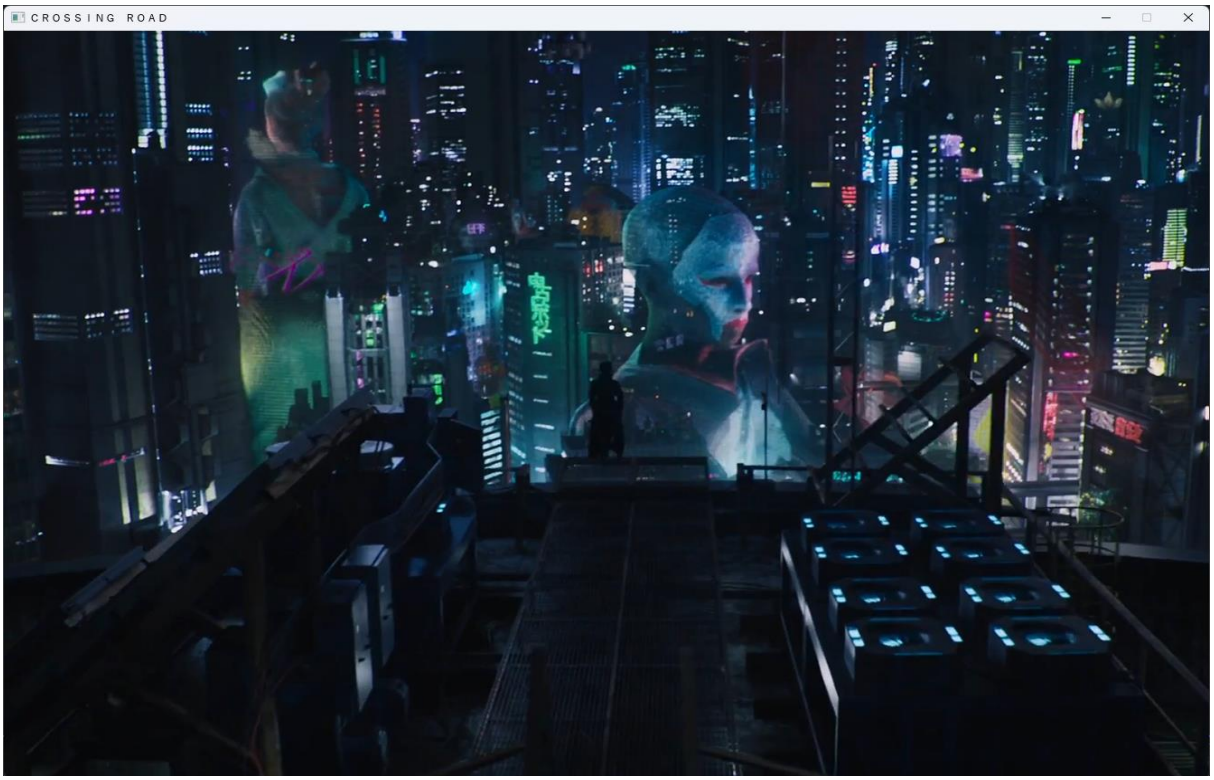
Upon selecting the "Load Game" button, a menu will appear, presenting the player with the option to choose from one of their previously saved games. The methodology behind the save and load file algorithm will be discussed in greater detail in the upcoming sections.

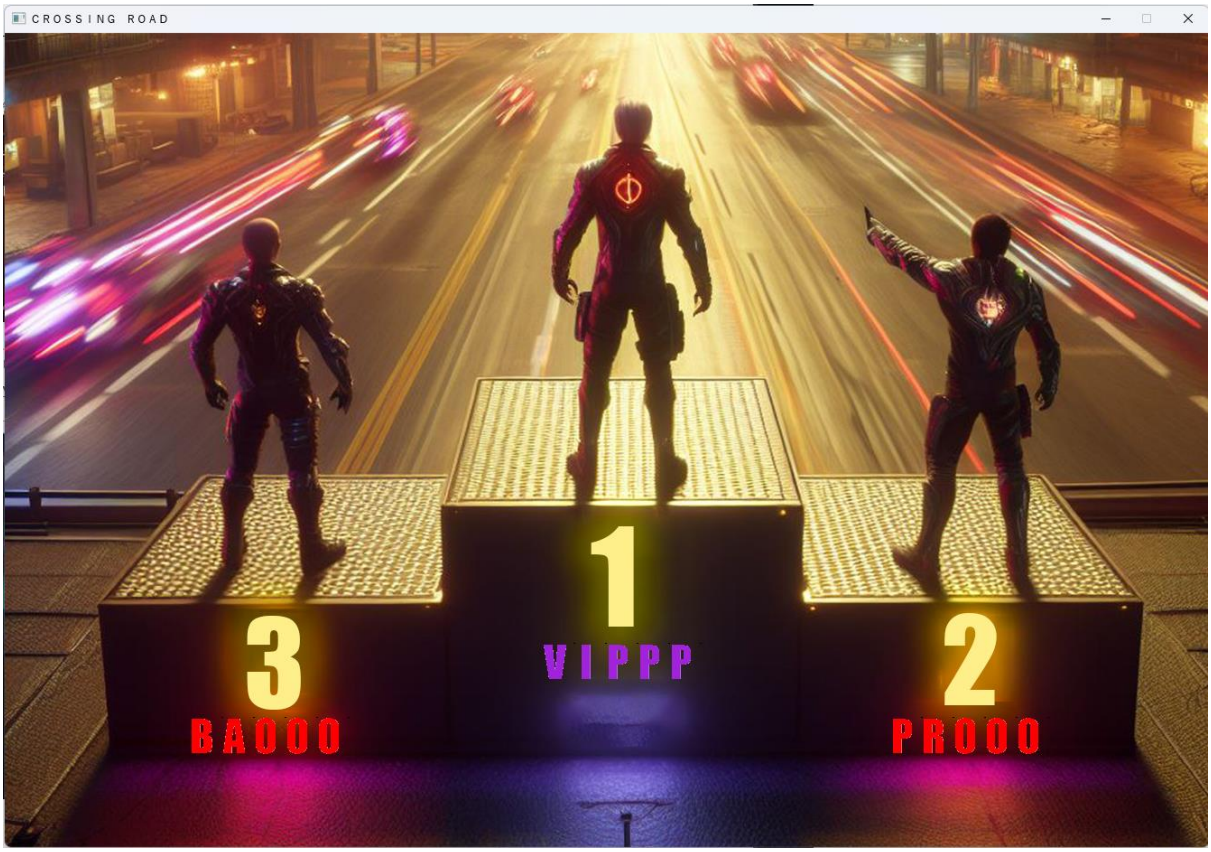


In the settings menu, users are provided with the option to modify the sound volume .This feature encompasses two distinct types of sound, background and effect sounds. The sound volume can be adjusted on a scale ranging from 0 to 5, allowing players to tailor their audio experience to their preferences



After entering Leaderboard, a short intro will be presented before being presented with the top three players who have achieved the highest scores.



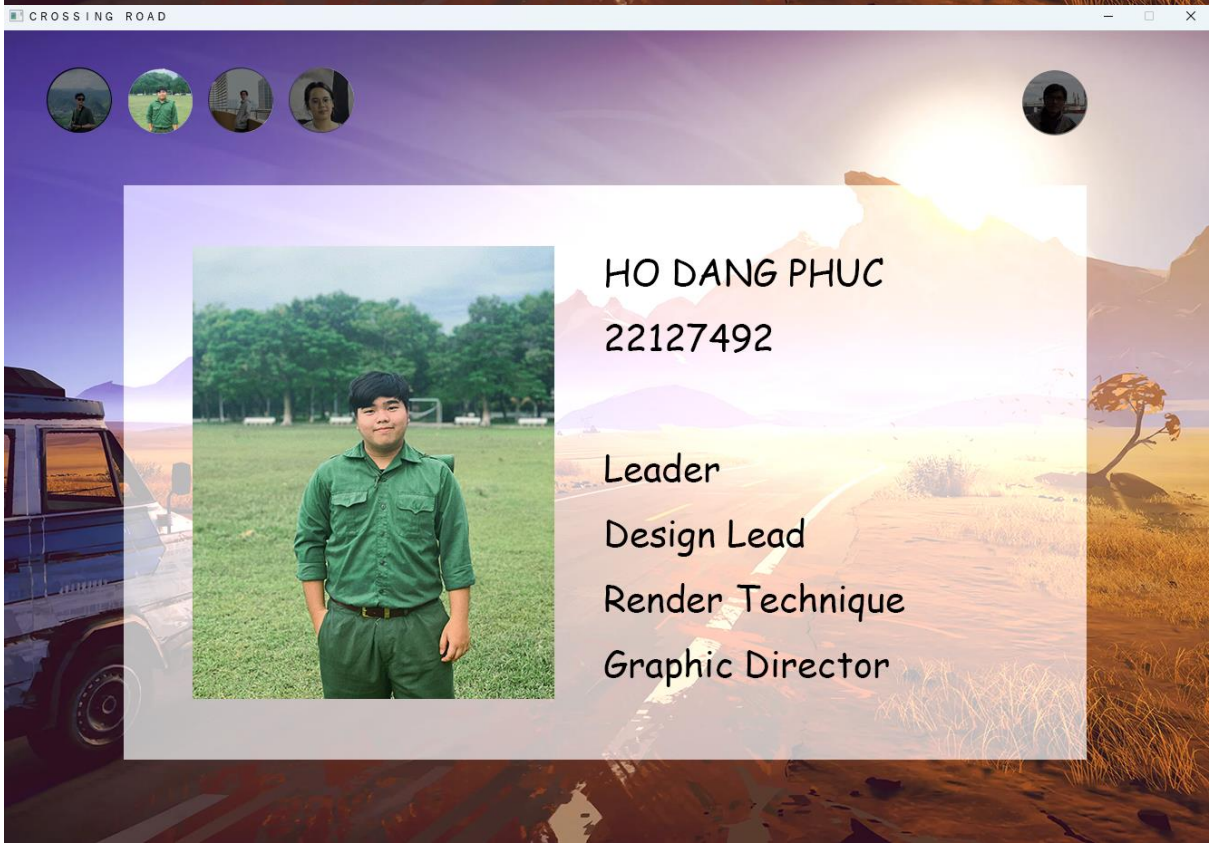


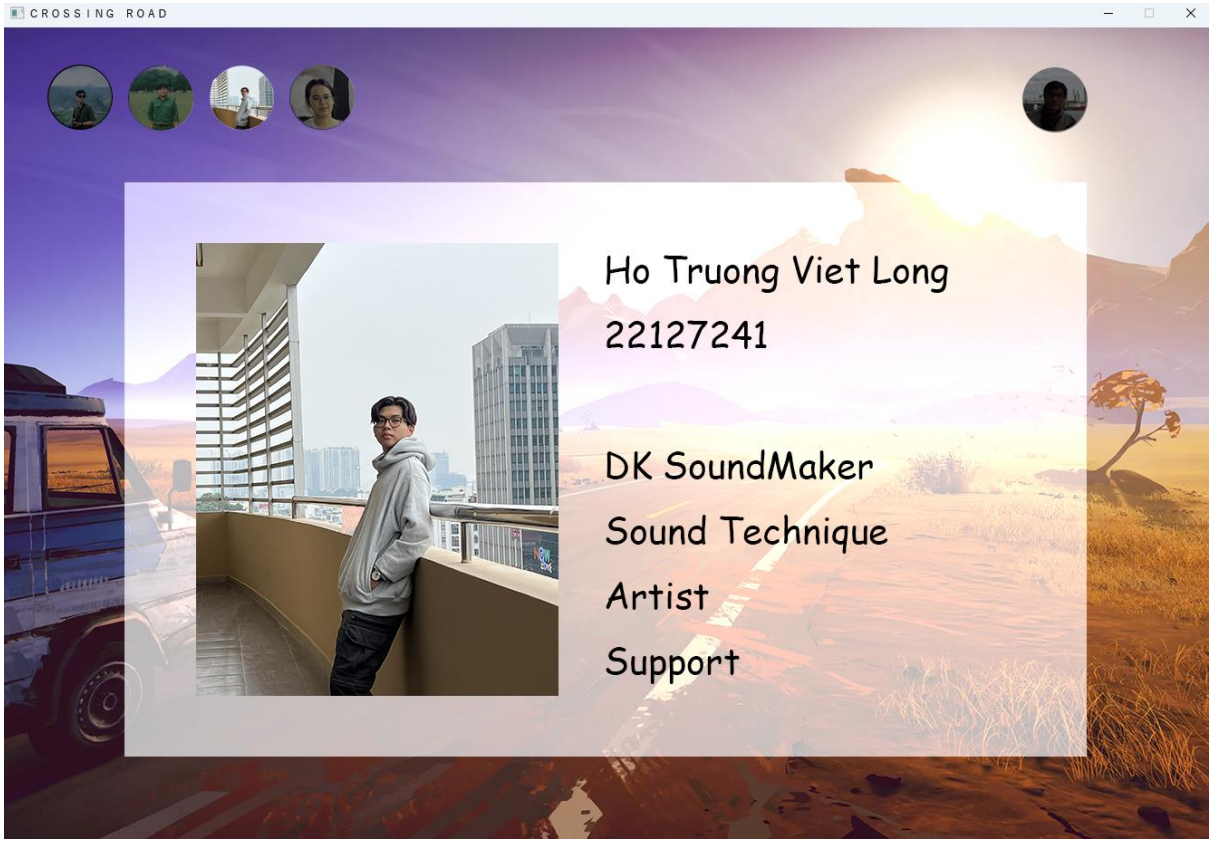
For helping players to enhance their understanding of the game's rules we created a help menu. This menu includes information on how to control the main character, what the stop signs are, as well as instructions for navigating the application in cases of collision.

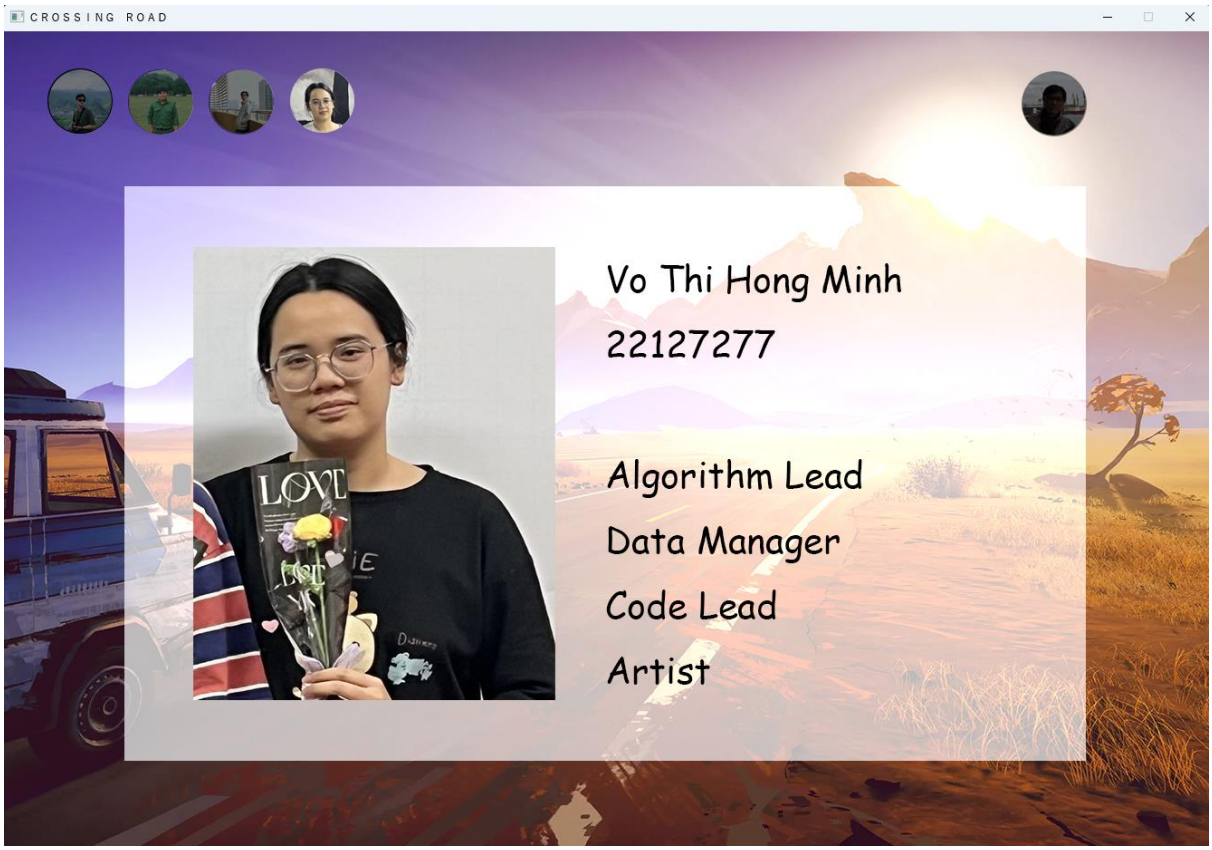

CROSSING ROAD
MOVE
UP
LEFT
DOWN
RIGHT
Use to move
main character
STOP SIGN
Every object will stop when
this sign showed up


CROSSING ROAD
COLLISION
SOME CASES THAT CAUSE DEATH

In order to honor the individuals who have created this game as well as the instructor who have provided guidance, we have constructed a credit menu as a token of appreciation.




CROSSING ROAD
Ho Truong Viet Long
22127241
DK SoundMaker
Sound Technique
Artist
Support


CROSSING ROAD
Vo Thi Hong Minh
22127277
Algorithm Lead
Data Manager
Code Lead
Artist

3.3.2 Sound Background and Effect

Using library <Windows.h> to play sound.

These are some sounds that we have used in our game, including:

- ❖ Background:
  - ➢ Game 0
  - ➢ Game 1
  - ➢ Game 2
  - ➢ Game 3 (4 random backgrounds)
  - ➢ Intro
  - ➢ Intro Leaderboard

❖ Effect:

➢ Menu Move

➢ Select

➢ Cancel

➢ Quit

➢ Continue

➢ Continue: Yes

➢ Continue: No

➢ Hit

➢ Water

➢ Run

3.4 Input

Mechanism of input:

We have two variables: is_down and changed which are bool type. These two variables will indicate 3 status of the user: release the keyboard, press, press and hold. By this, we will optimize the user's experience. Users don’t need to constantly press and release the keyboard to move toward, they just have to press and hold the button.

```
for (int i = 0; i < BUTTON_COUNT; i++) {
    (*input).buttons[i].changed = false;
}
```

So in our input function, there will be a loop to set all the button changed variables to false, which means every button is released.

When a user presses a keyboard, data will be stored in a MSG type variable. But before storing data, we will use PM_REMOVE to remove all data in this MSG style variable to avoid conflict.

```
while (PeekMessage(&message, *window, 0, 0, PM_REMOVE))
```

```
switch (vk_code) {
    process_button(BUTTON_UP, VK_UP);
    process_button(BUTTON_DOWN, VK_DOWN);
    process_button(BUTTON_W, 'W');
    process_button(BUTTON_S, 'S');
    process_button(BUTTON_LEFT, VK_LEFT);
    process_button(BUTTON_RIGHT, VK_RIGHT);
    process_button(BUTTON_ENTER, VK_RETURN);
}
```

Because all the input from the keyboard is a virtual key, we have to convert it to real key (which is already defined in enum) for our further purpose.

```
default: {
    TranslateMessage(&message);
    DispatchMessage(&message);
}
```

There'll be a default case, so if the user does not press any key, our program will skip the input from the user and continue to operate normally. So we do not have to use multi-thread (because we do not have to wait for the user input data for continuing running).

4. Classes

4.1 cLoadSave

Class cLoadSave including 9 operation, 0 attribute.Function: save user's gameplay to a txt file, then read the txt file to load the user's file.

```
class cLoadSave//save, load game, and leaderboard
{

public:
    //with every player, we'll have a file txt to save player's map
    void InputNameFile(string& a, Render_State* rs, HDC* hdc, Input* input, HWND* window);
    void SaveGame(cMap& a, Render_State* rs, HDC* hdc, Input* input, HWND* window);
    void SaveDataToFile(const string filename, cMap& a);
    char FromButtonToCharacter(Input* input);
    void OutputText(char a, Render_State* rs, int x, int y, int sign);
    int  ReturnChoice(HDC* hdc, Render_State* a, Input* input, HWND* window, vector<string>filename);
    void LoadingGame(cMap& map, HDC* hdc, Render_State* rs, Input* input, HWND* window);
    void leaderBoard(HDC* hdc, Render_State* rs, Input* input, HWND* window);
    vector<string> ProcessName(vector<string>filename);

};
```

❖ **InputNameFile** operation:

➢ Receive a reference variable of type string. HDC *hdc, Input *input, HWND *window are responsible for reading

images and printing out the image of the letter the user just entered.

➢ Function: help user enter name of file to save, then from the data user enters from the keyboard, convert the newly typed keys and add letters to the string variable. Function will stop when the user presses the enter key.

❖ **SaveGame** operation:

➢ Function: save the player's progress.

➢ Receive a reference object of class cMap.

➢ When the user selects SaveGame while playing the game, the program will immediately jump to the SaveGame function of the cLoadSave class, the transmitted cMap contains the information of the map that user is playing (including coordinates, player's score; parameters of lane: location, list of obstacles, type of road lane ,....).

➢ Then SaveGame calls SaveDataToFile operation, information of map is saved according to a certain structure in the txt file to facilitate the loading of the game later.

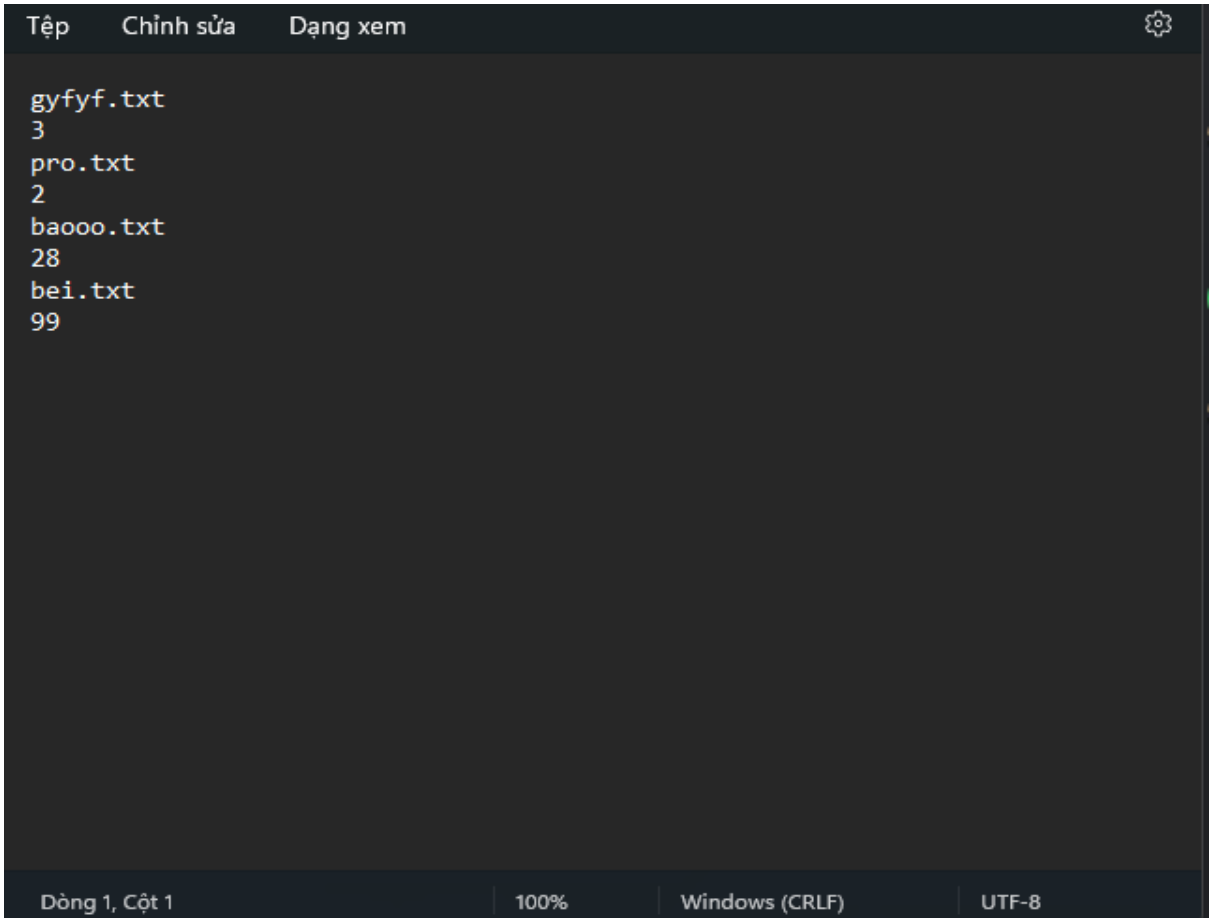
❖ **LoadingGame** operation:

➢ Load Game from file that user has selected.

➢ Read file according to the structure that have been saved before

❖ **leaderBoard** operation:

➢ Read data from “LOADGAME.txt” which include name of all files have been saved and score of these users



From information that has been read from the file, output 3 players with highest score.

4.2 cMenu

```
class cMenu
{
    //handle the Menu
public:
    void firstScene(HDC* hdc, Render_State* a, Input* input, HWND* window);
    void runIntro(HDC* hdc, Render_State* a, Input* input, HWND* window);
    int runMenu(HDC* hdc, Render_State* a, Input* input, HWND* window);
    void runSetting(HDC* hdc, Render_State* a, Input* input, HWND* window);
    void runLeaderBoard(HDC* hdc, Render_State* a, Input* input, HWND* window);
    void runHelp(HDC* hdc, Render_State* a, Input* input, HWND* window);
    void runCredit(HDC* hdc, Render_State* a, Input* input, HWND* window);
    void runIntroLead(HDC* hdc, Render_State* a, Input* input, HWND* window);
};
```

❖ cMenu is used to handle the Main Menu, include:

- ➢ **runIntro()**: play the intro video
- ➢ **firstScene()**: load the Screen that waiting for user press the enter button
- ➢ **runMenu()**: control the options of the menu, whichever option is selected will return the corresponding value to perform the next task
- ➢ **runSetting()**: pop-up the setting box, receive input from player to control the sound
- ➢ **runLeaderBoard()**: run the LeaderBoard operations (include a cut scene and a leader board screen that displays the 3 players with the highest score.
- ➢ **runHelp()**: load the screen shows help to play
- ➢ **runCredit()**: load the credit screen
- ➢ **runIntroLead()**: run the cut scene before display the leader board

4.3 cObject

cObject: an abstract class that controls all the objects of every lane. It will be inherited by 5 classes: cTree, cBridge, cBus, cFireFighter

```
class cObject
{
protected:
    float x, y; //top-left point
    int width, height;
    string filename;
    //an object image file name contain: object+direction but direction is control by the lane class
public:
    cObject() {
        x = y = 0;
        width = height = 1;
        filename = "";
    };

    void assignX(int);
    void assignY(int);
    void assignHeight(int);
    void assignWidth(int);
    float getX();
    float getY();
    int getWidth();
    int getHeight();
    virtual string getObjectFileName() = 0;
};
```

❖ Some description about cObject:

- ➢ **float x, y**: coordinate of the top-left point, use in control the display of the object and collision
- ➢ **width, height**: depending on which class is inherited, used to handle the collision checking operation.
- ➢ **filename**: name of the image corresponding to the object, we will print the image at the coord of the object
- ➢ needed getters, setters

4.4 cTree, cBridge, cBus, cFireFighter

Classes that inherit the cObject, have the same structure

```
cFireFighter::cFireFighter()
{
    this->filename = "firefighter";
    this->width = 154;
    this->height = 100;
}
string cFireFighter::getObjectFileName()
{
    return filename;
}
cFireFighter::cFireFighter(int x, int y)
{
    this->filename = "firefighter";
    this->width = 154;
    this->height = 100;
    this->x = x;
    this->y = y;
}
```

4.5 cLane

```
class cLane
{
    int numObj; //default is three
    vector<cObject*> objs;
    bool direction; //1 is to left, 0 is to right
    int coordY; //coordinate y
    int type; //1 is grass, 2 is road, 3 is river
    string filename; //depends on type
    int time;//duration between red light and green light
public:
    cLane(int numObj, int direct, int y, int type, string filename);
    cLane(int numObject, int y);
    cLane();
    ~cLane();
    void createObject();
    void setNumObj(int n);
    void SetLaneFilename(string filename);
    void setY(int y);
    void setTime(int time);
    void setType(int type);
    void setListObject(vector<cObject*>list);
    void setDirection(bool a);
    int getTime();
    int getType();
    int getDirection();
    vector<cObject*> getList();
    int getNumObj();
    int getY();
    string GetLaneFileName();
};
```

createObject operation: create list of objects according to the type of lane. For instance, lane grass, list of objects is tree, but with lane that has more than 1 type of object, objects will be randomly created. More detail, distance between these objects will also be random.

The rest operations are constructor, destructor, setter and getter.

4.6 cListLane

```
class cListLane
{
    //this class is only use to
    //easily initialize and
    //destroy the lanes
private:
    static vector<cLane*> lanes;
    static int numLanes;
public:
    cListLane();
    ~cListLane();
    void Create_Lane(int NumberObj);
    vector<cLane*> Get_ListLane();
};
```

❖ This class includes 2 attributes and 4 operations.

➢ **vector<cLane*> lanes**: contain list of pointer to object of class cLane.

➢ **int numLanes**: indicate number of lanes in a map.

➢ **Create_Lane** operation: this is the initialization step for the map. Eight lanes will be created randomly, so with every time the user chooses a new game, there'll be a different map.

➢ The rest of the operations are getter, constructor, destructor.

4.7 cPeople

```
class cPeople
{
    float x, y; // up-left point
    int height, width; //
    int frame; //every movement has 8 frames
    string mode; //get mode (up, down, left, right) to read the right image
    //an people image file name contain: mode+frame
public:

    cPeople();
    void assignX(int);
    void assignY(int);
    int getX();
    int getY();
    int getHeight();
    int getWidth();
    int getFrame();
    void newFrame();
    void setFrame(int x);
    string getMode();
    void setMode(string a);

};
```

❖ It has the following member variables:

➢ **x** and **y**: the coordinates of the upper-left point of the people's image.

➢ **height** and **width**: the height and width of the people's image.

➢ **frame**: an integer that represents the current frame of the people's image. Every movement has 8 frames.

➢ **mode**: a string that represents the direction of the people's image (up, down, left, right). The people's image file name contains the mode and frame number

➢ and needed getters, setter

4.8 cMap

```
class cMap
{
private:
    cPeople people;
    static cListLane temp1;//a static member of cMap, only use to initialize and destroy the cLane
    vector<cLane*> lanes;//this vector will be a pointer to a vector<cLane*> in cListLane temp1
                         //we will handle the game by this member
    int score;
    static float speed;
    string playerName;
    clock_t start, end;
    double timeUse = 0;//Clock start at begin of a match
                       //use to handle the traffic light


    static bool flagLoad;//handle load, save, and pause (press Esc button)
    static bool flagSave;//
    static int flagStop; //
public:
    ~cMap() { ... }
    void setScore(int);
    int getScore();
    vector<cLane*>& getLanes();//use in save load
    cPeople& getPeople();//use in save load
```

❖ Firstly, we talk briefly about member attributes and some basic function:

➢ **people**: an object of class cPeople.

➢ **temp1**: a static member of cMap that is only used to initialize and destroy the cLane.

➢ **lanes**: a vector of pointers to cLane objects. This vector is a pointer to a vector of cLane objects in cListLane temp1.

➢ **score**: an integer that represents the score of the game.

➢ **speed:** a static float that represents the speed of the game.

➢ **playerName**: a string that represents the name of the player.

➢ **start and end**: clock_t variables that represent the start and end time of the game.

➢ **timeUse**: a double that represents the time elapsed since the start of the game.

➢ **flagLoad, flagSave, and flagStop**: static boolean and integer variables that are used to handle load, save, and pause (press Esc button) operations.
Basic functions:

➢ **cMap()**: the destructor of the class.

➢ **setScore(int)**: a function that sets the value of score.

➢ **getScore()**: a function that returns the value of score.

➢ **getLanes()**: a function that returns a reference to the lanes vector. This function is used in save and load operations.

➢ **getPeople()**: a function that returns a reference to the people object. This function is used in save and load operations.

```
public:
    //-------------Game Process Function-----------
    void gameProcess(HDC* hdc, Render_State* a, Input* input, HWND* window);
    //this function will be called in main and handle all the match process

private:
    //-------------These function will be call in gameProcess function
    void simulatePeopleAndDrawObject(Input* input, float dt, Render_State* a);  //Receive input, update people, draw people and object every frame
    void simulateObject(int timeuse, float speed);                              //update object coordinate every frame
    void drawTrafficLight(int timeuse, Render_State* a);                        //check and redraw traffic light
    void Endless(int timeUse);                                                  //Update the lanes continuously
    void OutputScore(int score, Render_State* rs, int x, int y);                //draw Score

    int checkCollison();                                                        //check Collision
    int checkCollison2(int lane1, int lane2);                                   //

    int runEscape(HDC* hdc, Render_State* a, Input* input, HWND* window);       //pause game and show the escape options
    void runSetting(HDC* hdc, Render_State* a, Input* input, HWND* window);     //
    void runHelp(HDC* hdc, Render_State* a, Input* input, HWND* window);        //
    bool AskContinue(HDC* hdc, Render_State* a, Input* input, HWND* window);    //ask continue if player dead

    //water dead scene
    void runWaterDead(HDC* hdc, Render_State* a, Input* input, HWND* window) { ... }
    //road dead scene
    void runRoadDead(HDC* hdc, Render_State* a, Input* input, HWND* window) { ... }
    void runAskContinueScene(HDC* hdc, Render_State* a, Input* input, HWND* window) { ... }
};
```

❖ It has several private functions that are called in the gameProcess function. These functions include:

➢ **simulatePeopleAndDrawObject:** This function receives input, updates people, and draws people and objects every frame.

- ➢ **simulateObject**: This function updates the object coordinate every frame.
- ➢ **drawTrafficLight:** This function checks and redraws the traffic light.
- ➢ **Endless**: This function updates the lanes continuously.
- ➢ **OutputScore**: This function draws the score.
- ➢ **checkCollision**: This function checks for collision.
- ➢ **checkCollision2**: This function checks for collision between two lanes.
- ➢ **runEscape**: This function pauses the game and shows the escape options.
- ➢ **runSetting**: This function runs the game settings.
- ➢ **runHelp**: This function runs the game help.
- ➢ **AskContinue**: This function asks the player if they want to continue if they die.

Let's talk about these function carefully:

```
void cMap::gameProcess( HDC *hdc, Render_State * a, Input * input, HWND * window) {                //
flag:                                                                                              //flag to return if player dead and play again
    bool flagExit = false;                                                                         //Some declareation of a new match
    flagLoad = false; // Gán biến nhận biết có thực hiện chức năng load là không                  //
    flagSave = false;// Gán biến nhận biết có thực hiện chức năng save là không                   //
    flagStop = false;                                                                              //
    start = clock();// Bắt đầu đếm thời gian                                                       //
                                                                                                   //
    speed = 3;
    while (true)
    {
        MSG message;
        for (int i = 0; i < BUTTON_COUNT; i++) { ... }
        while (PeekMessage(&message, *window, 0, 0, PM_REMOVE)) { ... }
        if (pressed(BUTTON_ESCAPE)) { ... }
        if (flagStop==0) {
            speed += (float)score/2000;
            clear_screen(0xffffff, *&a);
            simulateObject(timeUse, speed);
            simulatePeopleAndDrawObject(input, 10, *&a);
            drawTrafficLight(timeUse, *&a);
            Endless(timeUse);
            //draw Score
            OutputScore(score, *&a, 0, 0);
            if (checkCollison() == 1) { ... }
            else if (checkCollison() == 2) { ... }
            StretchDIBits(*hdc, 0, 0, a->width, a->height, 0, 0, a->width, a->height, a->memory, &a->bitmap_info, DIB_RGB_COLORS, SRCCOPY);
            Sleep(10);
            end = clock();// Bấm thời gian kết thúc
            timeUse = (double)(end - start) / CLOCKS_PER_SEC; // Đo thời gian thực
        }
    }
}
```

➔ **gameProcess():**

Initialize the necessary things and start a loop. In the loop we check stopping conditions. If the player presses escape, the game will be

paused and a popped-up escape box. If player is not collision yet, the loop repeat these functions:

```
void cMap::simulateObject(int timeuse, float speed)//thay doi thang ListObject trong cMap
{
    //assign lai toa do x
    for (int i = 0; i < lanes.size(); i++) {                                    //update object coordinate
        if (lanes[i]->getType() == 2)                                   //
        {                                                                       //
            for (int j = 0; j < lanes[i]->getList().size(); j++) {              //
                if (!((timeuse / lanes[i]->getTime()) % 2 == 1))                //check condition to detect
                {                                                               //traffic light status
                    if (lanes[i]->getDirection() == 1) { ... }
                    else { ... }
                }                                                               //
            }                                                                   //
        }                                                                       //
    }                                                                           //
}
```

➔ simulateObject():

Update the coordinate x of every object that is on the road and the light check is green

```
void cMap::simulatePeopleAndDrawObject(Input* input, float speed, Render_State* a)
{
    int count = 0;                                                              //redraw object
    for (int i = 0; i < lanes.size(); i++) { ... }

    if (is_down(BUTTON_UP) && people.getY() > 0 + speed ) { ... }
    else if (is_down(BUTTON_DOWN) && people.getY() + people.getHeight() + speed < 901) {
        cSound::playEffectSound(9);
        people.setMode("Movement\\D");
        people.newFrame();
        this->people.assignY(this->people.getY() + speed);
        loadImage(people.getMode() + to_string(people.getFrame()) + (string)".bmp", *&a, this->people.getX(), this->people.getY(), 64, 127, 206);
    }
    else if (is_down(BUTTON_RIGHT) && people.getX() + people.getWidth() + speed < 1334) { ... }
    else if (is_down(BUTTON_LEFT) && people.getX() > 0) { ... }
    else { ... }

    count = 0;                                                                  //Update Object graphic
    for (int i = 0; i < lanes.size(); i++) { ... }
}
```

➔ simulatePeopleAndDrawObject():

First, go through all the objects and then draw them.

Next we have commands to receive input from the player. For each appropriate button, the player will update the mode (direction) and the frames will also change continuously.

Finally, we will go through the entire object again to check if that object according to the vision must be behind, then we will draw again

➔ drawTrafficLight():

Go through every lane and redraw the light if it is red

```
void cMap::Endless(int timeUse)                                                    //
{                                                                                  //
    if (people.getY() < SIZELANE*3)                                                //if people stand up the designated
    {                                                                              //coordinate y, the map move up 1 lane
        if (lanes.empty()) return;                                                 //and score + 1
        cLane* temp2 = lanes.back();                                               //
        lanes.pop_back();                                                          //
        temp2->~cLane();                                                           //
                                                                                   //
                                                                                   //
        cLane* temp = new cLane(3, 0);                                             //2 newest lane cannot be water
        if (temp->getType() == 3) {                                                //so we have a check here
            if (lanes[0]->getType() == 3) {                                        //
                while (temp->getType() == 3)                                       //
                {                                                                  //
                    temp->~cLane();                                                //
                    delete temp;                                                   //
                    temp = new cLane(3, 0);                                        //
                }                                                                  //
            }                                                                      //
                                                                                   //
        }                                                                          //
        temp->createObject();                                                      //
        lanes.insert(lanes.begin(), (temp));                                       //
        for (int i = 1; i < lanes.size(); i++)                                     //reset coordinate y for remaining lanes
        {                                                                          //(2-9)
            lanes[i]->setY(lanes[i]->getY() + SIZELANE);                           //
            //reset y of every object                                              //
            for (int j = 0; j < lanes[i]->getList().size(); j++)                   //
            {                                                                      //
                lanes[i]->getList()[j]->assignY(lanes[i]->getList()[j]->getY() + SIZELANE); //
            }                                                                      //
        }                                                                          //
        people.assignY(people.getY() + SIZELANE);                                 //people coord y update too
        score += 1;                                                                //
    }
}
```

➔ Endless():

The function checks if the y coordinate of the people object is less than SIZELANE*3 (a default coordinate y). If it is, the function moves the map up one lane and increments the score by 1. The function does this by:

+ Removing the last lane from the lane vector.
+ Creating a new lane object and adding it to the beginning of the lanes vector.
+ Resetting the y coordinate of each remaining lane and every object in the lane.
+ Updating the y coordinate of the people object.

The function also checks if the two newest lanes are both water lanes. If they are, the function creates a new lane object until it is not a water

lane. This is to ensure that there is always a safe lane for the player to move on.

➔ OutputScore():

Just to print the score on the top-left corner.

```
int cMap::checkCollison()//1: chet duoi nuoc, 2: chet vat the, 0: false
{
    //y
    int lane1 = (people.getY() + people.getHeight()*3/4) / SIZELANE;
    int lane2 = (people.getY() + people.getHeight()) / SIZELANE;
    if (lane2 > 8) lane2 = lane1;
    int x = people.getX();                          //
    int xMax = people.getX() + people.getWidth();   //we use 3d graphic so that
    int y = people.getY() + people.getHeight() * 3/4;   //hitbox is a rectangle with left corner (x, y) and right corner(xMax, yMax)
    int yMax = people.getY() + people.getHeight();      //

    if(lane1>=lanes.size())
    {
        return 0;
    }

    for (int i = lane1; i <= lane2; i++)
    {
        if (lanes[i]->getType() == 3)                   //check on water, if true return a sub-function
        {
            return checkCollison2(lane1, lane2);
        }

        if (lanes[i]->getType() == 2) {                 //check on Road,
            for (int j = 0; j < lanes[i]->getList().size(); j++)
            {
                cObject* temp = lanes[i]->getList()[j];
                if (((x >= temp->getX() && x <= (temp->getX() + temp->getWidth())) ||                                   //check if the rectangle hitbox
                    (xMax >= temp->getX() && xMax <= (temp->getX() + temp->getWidth())) ||                              //of player is overlap with
                    (x <= temp->getX() && xMax >= (temp->getX() + temp->getWidth())) &&                                 //the rectangle hitbox of every
                    ((y >= (temp->getY() + temp->getHeight() * 2 / 3) && y <= (temp->getY() + temp->getHeight())) ||    //object on the road
                        (yMax >= (temp->getY() + temp->getHeight() * 2 / 3) && yMax <= (temp->getY() + temp->getHeight())) ||   //
                        (y <= (temp->getY() + temp->getHeight() * 2 / 3) && yMax >= (temp->getY() + temp->getHeight()))))       //
                {
                    return 2;
                }
            }
        }
    }
    return false;
}
```

➔ checkCollision():

The function checks if the player object collides with any object on the road or is on the water. The function does this by:

+ Calculating the lane1 and lane2 values based on the y coordinate of the player object.
+ Checking if the current lane is a water lane. If it is, the function returns the result of calling the checkCollison2 function.
+ Checking if the current lane is a road lane. If it is, the function checks if the player object's hitbox overlaps with any object on the road. If it does, the function returns 2. Otherwise, the function returns false.

➔ checkCollision2():

Similar to checkCollision(), if the player's hitbox is on the safe area (bridge), it returns true.

```
int cMap::runEscape(HDC* hdc, Render_State* a, Input* input, HWND* window) {
int choice = 1;
while (true) {
    MSG message;
    for (int i = 0; i < BUTTON_COUNT; i++) { ... }
    while (PeekMessage(&message, *window, 0, 0, PM_REMOVE)) { ... }
    if (pressed(BUTTON_UP)) { ... }
    if (pressed(BUTTON_DOWN)) { ... }
    if (pressed(BUTTON_ENTER)) { ... }

    switch (choice) { ... }
    StretchDIBits(*hdc, 0, 0, a->width, a->height, 0, 0, a->width, a->height, a->memory, &a->bitmap_info, DIB_RGB_COLORS, SRCCOPY);
}
```

➔ runEscape()

Similar to runMenu() function, receive player input and do the correspond options (which includes runSetting and runHelp function)

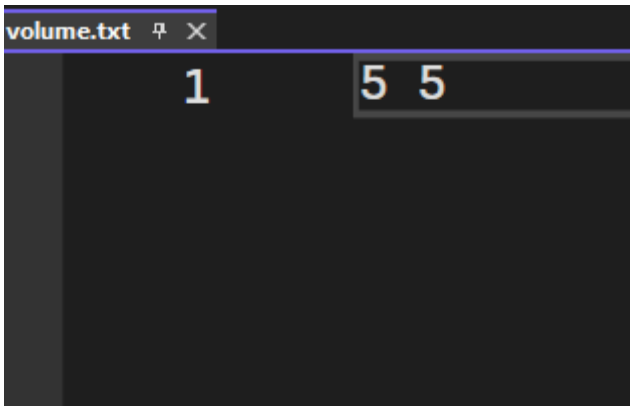
In addition, there are some sub-functions like askContinue to ask if the player continues, and 3 functions to run cutscene: runWaterDead, runRoadDead and runAskContinueScene.

4.9 cSound

```
class cSound {
public:
    static void playBackgroundSound(int i) {...
    }
    static void playBackground(wstring command);
    static void playGame0(int counterBackground);//Background has 4 random sounds from Game0 to Game 3
    static void playGame1(int counterBackground);
    static void playGame2(int counterBackground);
    static void playGame3(int counterBackground);
    static void playIntro(int counterBackground);
    static void playIntroLeader(int counterBackground);

    static void playEffectSound(int i) {...
    }
    static void playEffect(wstring command);
    static void playMenuMove(int counterVolume);
    static void playSelect(int counterVolume);
    static void playCancel(int counterVolume);
    static void playQuit(int counterVolume);
    static void playContinue(int counterVolume);
    static void playContinueYes(int counterVolume);
    static void playContinueNo(int counterVolume);
    static void playHit(int counterVolume);
    static void playWater(int counterVolume);
    static void playRun(int counterVolume);
};
```

Function: Read text file “volume.txt” and get 2 numbers for *counterBackground* and *counterEffect*. Then use it for playing a sound at that level. There are 5 levels from 1 to 5 and 0 is mute.



➔ **playBackgroundSound** and **playEffectSound** operator:

Two main operators for playing music that are contained in the game, using a switch for a variable to choose sound.

➔ **playBackground** and **playEffect** operator:

Read the .wav file from the folder to run and use the function mciSendString() in Winmm.lib to play sounds.

➔ **playGame0, playGame1, playGame2, playGame3**:

These are 4 functions that read the file name and get the 2 counters for suitable sound levels. We use 4 operators for random 4 background sounds.

➔ **playIntro and playIntroLead**:

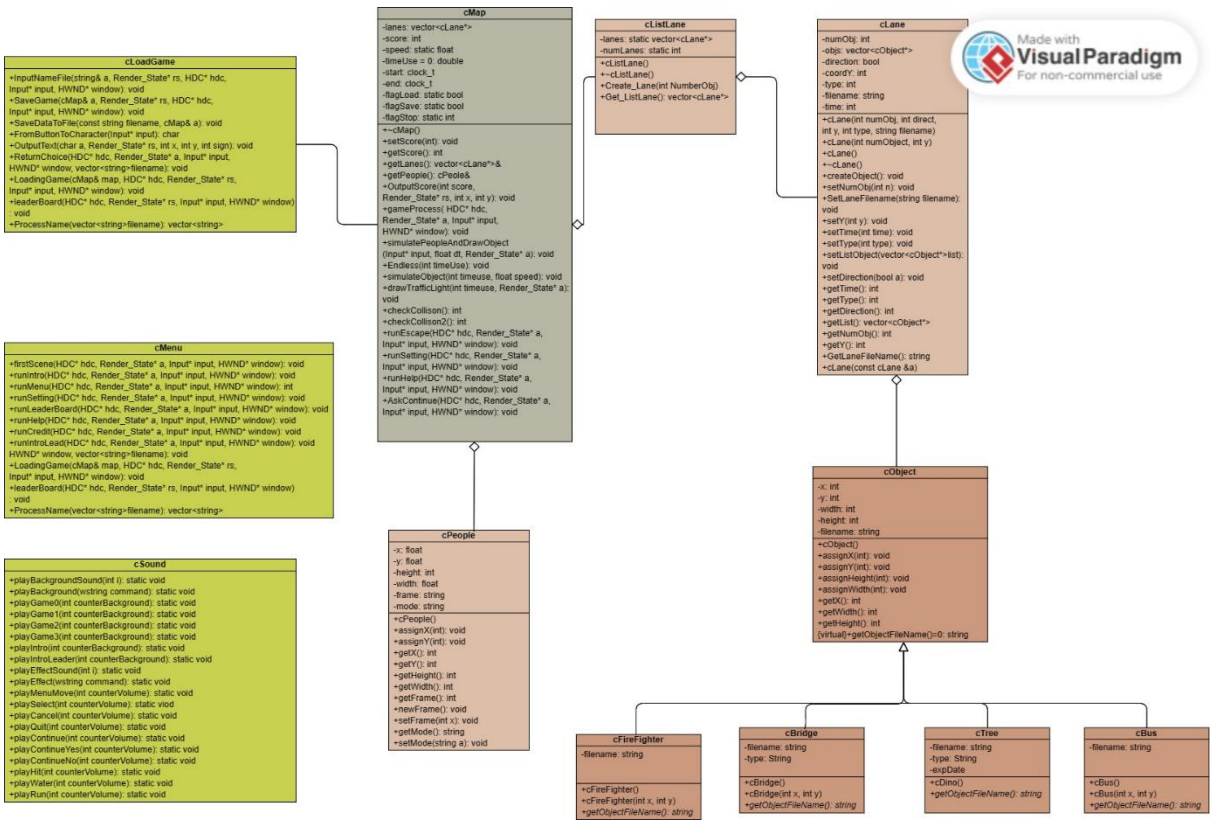
Play sound for the scene at first and before LeaderBoard.

➔ **playMenuMove, playSelect, playCancel…:**

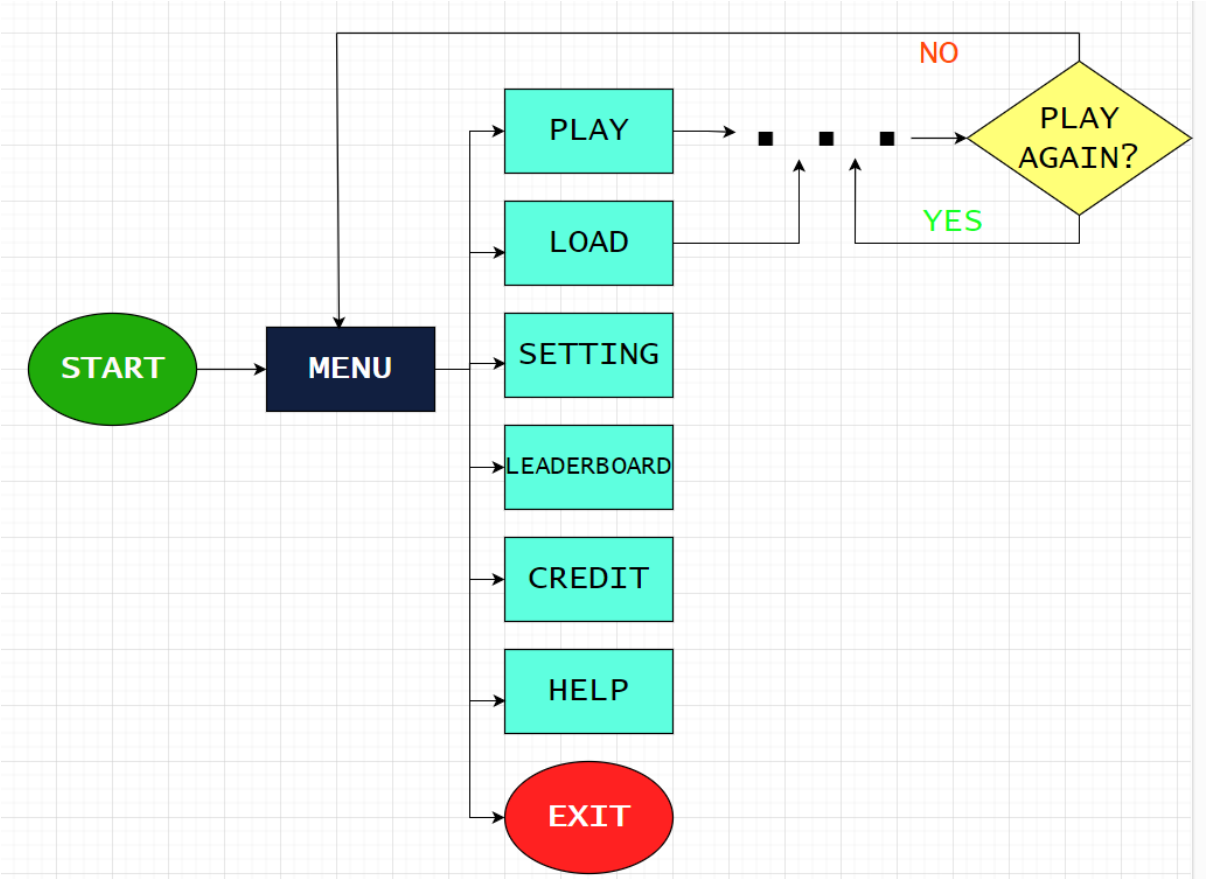
Play sound corresponding to its name.

5. UML Graph / Diagram

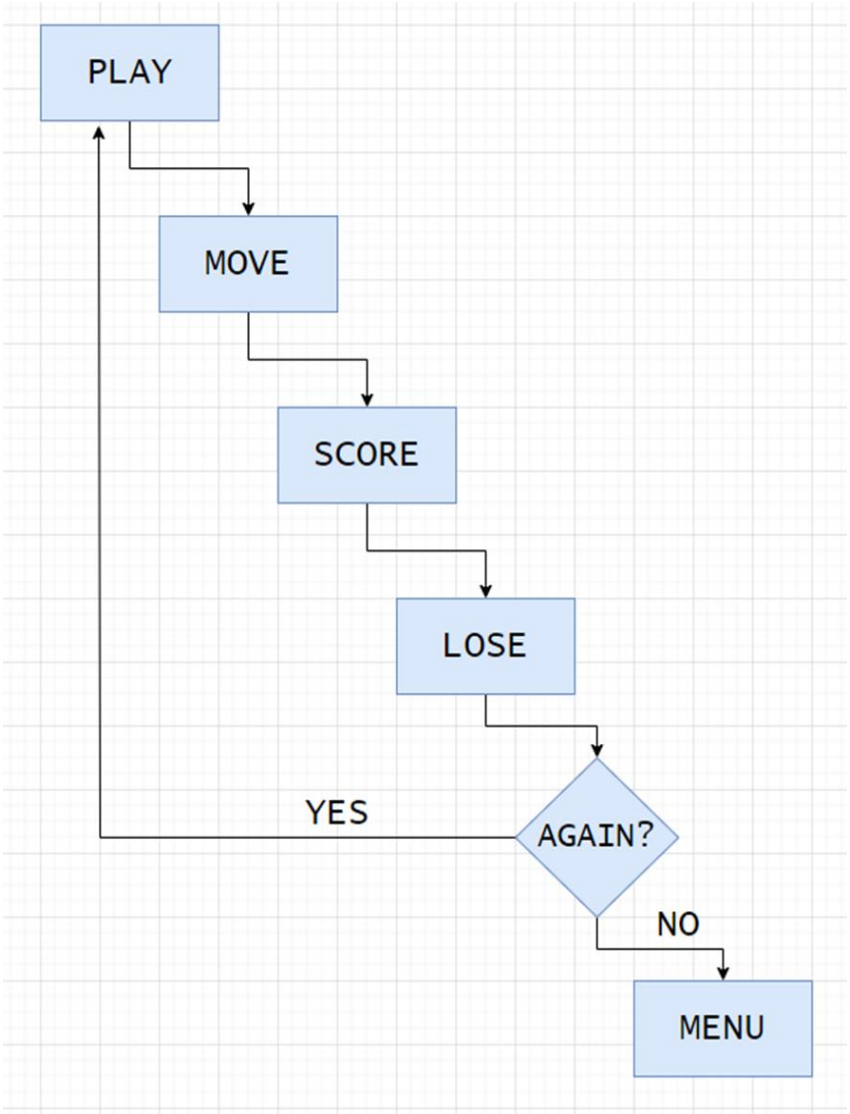
5.1 UML



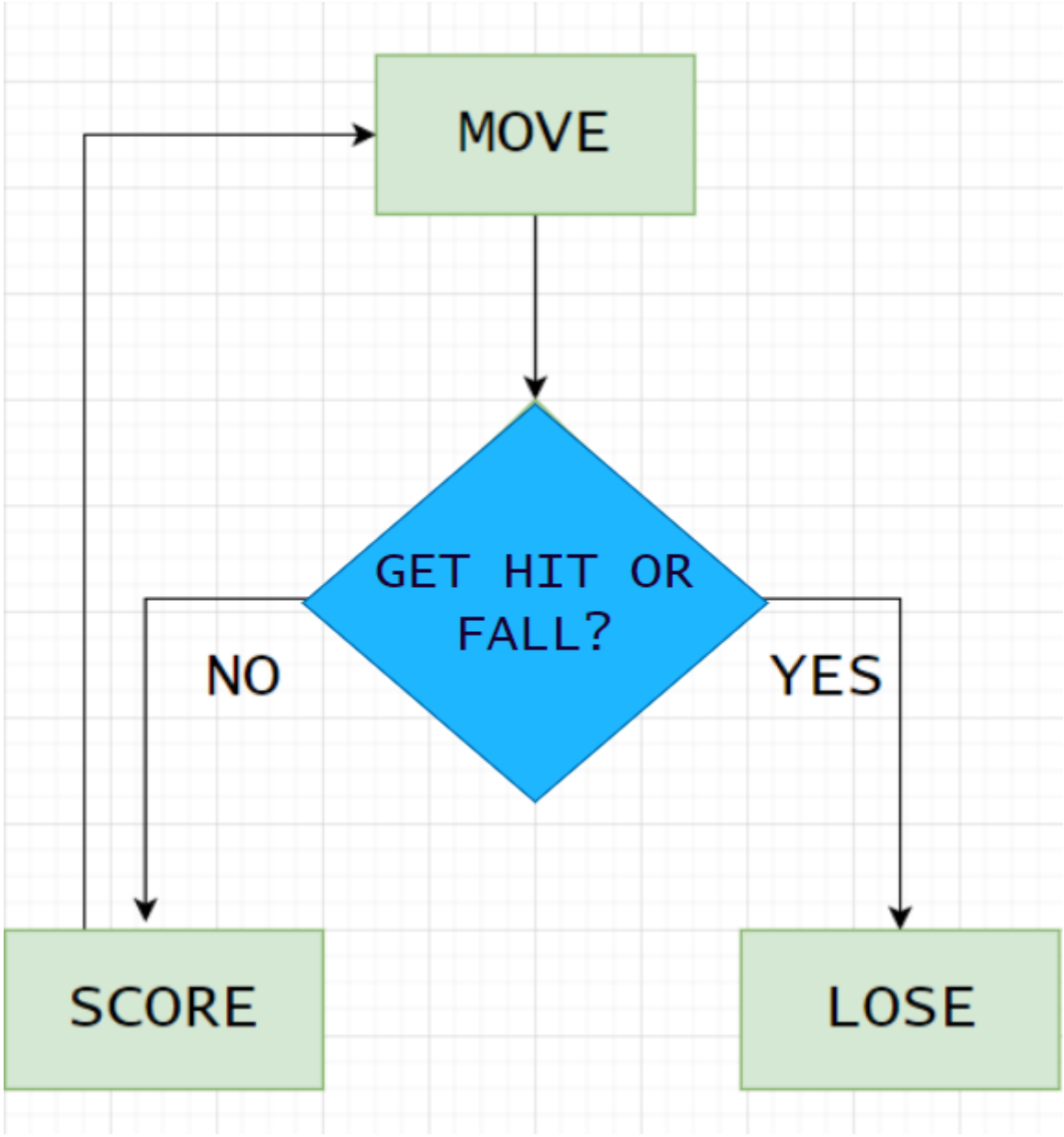
5.2 Menu diagram



5.3 Game diagram



5.4 Process Score diagram



6. Result

The 'Crossing Road' project has been completed, during the completion process, we have received extremely specific and detailed instructions from Mr. Truong Toan Thinh. We would like to give sincere thanks for enabling us to complete the project as best as we can. In the project progress, we've a learned many lessons and drawn a lot of experience which will be useful for our study and work

This is the second game that we implement, though we have try our best, there're still some mistakes. But we have put a lot of effort on this project. So we hope you will like it.

A sincere thanks from group 7.