**VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY**

**UNIVERSIRY OF SCIENCE**

**FACULTY OF INFORMATION TECHNOLOGY**

**Logo

Description automatically generated**

**PROJECT REPORT**

**TOPIC: CROSS THE ROAD GAME**

**COURSE: Object-oriented Programming**

**GROUP MEMBERS (GROUP 3 – 20CLC03):**

* Nguyễn Lê Hoàng Thông – 20127078
* Đào Đại Hải – 20127016
* Lại Minh Thông – 20127635
* Trần Nguyễn Anh Tài – 20127319

**UNDER THE GUIDANCE OF**

Mr. Trương Toàn Thịnh

Mr.Nguyễn Thành An

Mr. Đỗ Trọng Lễ

Mr. Nguyễn Hải Đăng

**TABLE OF CONTENTS**

[CHAPTER 1: INTRODUCTION 3](#_Toc90241972)

[1) Group’s members 3](#_Toc90241973)

[2) Topic description 3](#_Toc90241974)

[3) Final product 3](#_Toc90241975)

[4) Description of game’s interface 4](#_Toc90241976)

[CHAPTER 2: CODE IMPLEMENTATION 6](#_Toc90241977)

[1) UML diagram 6](#_Toc90241978)

[2) Algorithms and structures of classes’ properties and methods in the program 7](#_Toc90241979)

[3) Combination of threads (Main thread + Sub thread) 31](#_Toc90241980)

[CHAPTER 3: CONCLUSIONS 34](#_Toc90241981)

[REFERENCES 35](#_Toc90241982)

# 

# CHAPTER 1: INTRODUCTION

## Group’s members

|  |  |
| --- | --- |
| **Full name** | **Student ID** |
| Nguyễn Lê Hoàng Thông | 20127078 |
| Đào Đại Hải | 20127016 |
| Lại Minh Thông | 20127635 |
| Trần Nguyễn Anh Tài | 20127319 |

## Topic description

A **Cross the Road game**, written in C/C++, runs directly on console window. Use the knowledge learned in OOP course and some extended knowledge about C/C++ to apply to product implementation

## Final product

- Video demonstration link: <https://bit.ly/Team3Project>

- Game scenario: At the start of the game, the player chooses one of the options in the main Menu. The player can choose ***Start*** new game or **Load** previously saved levels. The player’s goal is to reach the Finish line as fast as possible to get high score and come to the next level. The player will use the W-A-S-D keys to control the player’s character. The player can also use some other keys which are instructed in the game to make some choice such as: Save, Pause, Continue, Exit, and Reset.

## Description of game’s interface

*At the Menu interface, use W-S and Enter keys to select these options:*

1. **Play**: Start a new game.
2. **Load Game**: Load previously saved levels and start game.
3. **Leader Board**: Check the highest scores.
4. **Instruction**: Instruction for game control keys.
5. **Settings**: Player can set sound volume and window size.
6. **Credits**: Check information about development team and instructors.
7. **Quit**: Exit the game.

Text

Description automatically generated

*Pic 1.1: Menu interface*

*At the in-game interface:*

* On the right side, player can see some information about passed time, level, current scores and some interactive keys.
* The left side is the game’s main interface. It consists of four lanes with four obstacles (Cars, Vans, Birds and Aliens). Be careful of the traffic lights on the right side of the Car and Vans lanes. The player will start at the sidewalk named START and try to react the sidewalk named FINISH to go to next level with more obstacles. If player loses (collision with an obstacle occurs), the GAME OVER interface will appear and ask player for some information.

A screenshot of a computer screen

Description automatically generated with medium confidence

*Pic 1.2: In-game interface*

# CHAPTER 2: CODE IMPLEMENTATION

## UML diagram

Diagram

Description automatically generated (Because of large number of methods and properties, in this section we only analyze the relationship of all classes in the program, not draw all the methods and properties into the UML diagram)

*Pic 2.1: UML diagram*

+ In the program, class CGame is the parent class of CVehicle, CAnimal, Player and Trafficlight classes because it’s a class that runs the whole game, so its properties include these other classes.

+ CANNIMAL is the parent class of Alien and Bird (they are animal obstacles), CVEHICLE is the parent class of Car and Van (they are vehicle obstacles).

+ Class BOX and MENU are used to draw user interface, combine with Main thread to navigate the game flow.

+ Class Data is used to import and export player’s information.

## Algorithms and structures of classes’ properties and methods in the program

In this section, we will analyze some important methods of classes in the program.

Firstly, in this program we use many Window-handle function to built a console UI.





Secondly, we draw the menu by many Window functions and use some one-dimensional strings to draw some beautiful text in the interface.





Now, we will analyze one of the most important classes in the program: **Game objects classes** => **CVehicle** (Car, Van), **CAnimal** (Alien, Bird) and **Player**.

Because the properties and methods of base class **CVehicle** and **CAnimal** are almost the same, we just consider **CVehicle** for short:



*Some important* ***CVehicle’s*** *methods:*

**Draw** method is used to draw object’s shape at its position (x, y). The idea of algorithm is print each line of object shape in one dimensional array string.



**Remove** method is used to remove all characters of object’s shape. So, we come up with an idea for better moving animation is that just remove the first character of each column of object shape => **RemoveMoving** method appears to better UX (but we can’t delete Remove method in case we need to delete all characters of object’s shape).







Next, we consider the ‘child’ of base class **CVehicle** and **CAnimal**: Car, Van, Bird, Alien. We will analyze Car and Bird classes (because Car and Van classes are nearly similar, Bird and Alien also).

We use one dimensional string array to keep the shape of each class that have a shape.

With classes **Car** and **Van**, there are some important methods like **isImpact**, **Move** and **Tell**, other support methods are inherited from **CVehicle**.

Note: “**Move**, **isImpact**, **Tell**” methods are similar in meaning between classes (Bird, Alien, Car, Van). So we just analyze these method in Car.



With **Move** method, the idea is: remove the shape at the old position, update position and draw a new shape.



With **Tell** method, make a sound when Tell method is called.



**isImpact** method is used to check if collision between Car and Player occurs. The idea is check position of Player with each character different from ‘ ‘.



Continually, we consider about **Bird** class



There are some methods (constructor, getter, setter, **isImpact**, **Tell**, **Move**) similar to Car’s, the difference is just about the different moving animation. Because Bird can go left or right, there is a “direct” properties to check Bird’s direction. And we need to make sure the Bird doesn’t go too far in one direction, use **setCount** to count how many step the Bird moved to change direction by **setDirect**.

About the **Trafficlight** class, it will work with Vehicles in **CGame** to check if Traffic light is red, the vehicle can not move.



The **getState** method is used to get the current signal (red or green). The **setXY** and **Draw** method will help to place Trafficlight at the right line. **changeLight** method help to change current state of the Trafficlight.

Now we continue to Player class, this class manages player’s character throughout the game.





There are some methods that print Player moving animation: **UP**, **DOWN**, **LEFT** **RIGHT**. **isDead** method is used for check if Player dead.





There is also a template T for function **isImpact**, this function will combine with all others **isImpact** method from the obstacle’s classes to handle collision between obstacles and Player.



If collision between player and one of obstacle objects (vector<T> v) happen, Player’s state comes to dead, the sound of that object will be called (**Tell** function). Return true for impact.

To handle with Player information for load game, save game and leaderboard (high score), we need a class called Data to keep some information and deal with some methods.



Some information: Player’s name, current level and score, time passed in this level, current position of player. And input, output method to export and import data.

Lastly, we will analyze the most important class in the program: **CGame**. This class runs the game by combining all other game objects classes: **CVehicle** (Car, Van), **CAnimal** (Alien, Bird) and **Player** and create some actions for the game such as: save, load, reset, pause, continue, exit.





We will clarify some important methods:

All objects of the program will be initialized and called with **Init** function:



**resetGame**, **exitGame**, **pauseGame**, **resumeThread**, **pauseThread**, **resumeThread** functions are used to handle with Sub thread, **loadGame** and **saveGame** functions will write/get player’s data to provide information for the next initializing time.

**updatePosPlayer** function will receive buf variable which is stand for player’s request, and it will run event base on that request.



**updatePosVehicle** and **updatePosAnimal** functions are use for update obstacles’ position to print moving animation.



**updateGameStatus**, **updateTime** and **calcScore** function handle with game’s information.





**checkImpact** function will call player’s impact method at position which the matching obstacle.



## Combination of threads (Main thread + Sub thread)

The whole game runs on two threads, they’re main thread and sub thread. There are two global variables called **game** (datatype: CGame) and **buf** (datatype: integer) which work with two threads. **game** variable runs the game while **buf** is used to get user commands through keyboard.

The mission of sub thread is to handle in-game events (when player is playing his/her character).



In Sub thread, we can see that some animation such as: **updatePosAnimal**, **updatePosVehicle**, **updatePosPeople** and **pause** command. It will check **game** and **buf** variables which sent by Main thread to make sure that in-game animation and interfaces work correctly.

In Main thread, we run the Menu in loop to receive and execute commands for the user. We use toupper(\_getch()) to receive keyboard’s commands and store it to **buf** variable, so Sub thread can know the commands and do its jobs.

While in-game interface showing, Main thread goes to another loop and continues to get user’s command by **buf** variable.





# CHAPTER 3: CONCLUSIONS

In this project, we focus on object-oriented programming design approach. We try to apply as much of OOP learned knowledge as possible such as:

+ Inheritance: an important knowledge of this project, many classes which have inheritance relationship typical as CVEHICLE – Car, CVEHICLE – Van, CANIMAL – Bird, CANIMAL – Alien, … They make the program semantically clearer and easily to reuse code.

+ Encapsulation: many classes are created to make sure that all methods and properties can be used flexibly. It makes code easily to reuse and protect private/protected properties from external influences.

+ Polymorphism: there are many “child” classes of a base class (CVEHICLE – Car, CVEHICLE – Van) but their methods not always have the same processing logic (Car and Van have Move method, but the Car moves right, and the Van moves left). We need to use polymorphism for many methods so that their “child” classes can have their own way of working.

+ Abstract: some pure virtual methods are built such as (ex: CANIMAL doesn’t have **Move** method in meaning). And, when we are working with main thread or sub thread, we don’t need to remember implementation of some functions, we just need to know what they can do and use their result.

+ There is also other OOP knowledge that implement in the code such as: static (to keep direction of all Birds, Aliens, …), friend (to overload some operators, …), template (used in check impact of Player), working with files (to store data), ....

The project also needs some knowledge about multithreading in C/C++, handle with Window console and font functions, get keyboard’s buffer, sound effects, …

All these pieces of knowledge are combined to create a project that fully meets the requirements from the topic with optimized UX-UI.

This project is also a good chance to practice some skills that IT student need as teamwork, Git/GitHub, huge time for research new knowledge, ….

In conclusion, this project helps us to practice much OOP knowledge that are taught in the class and know the basic structure of a project design.

# REFERENCES

* Sticky Bits, Glennan Carnie – Template and Polymorphism

<https://blog.feabhas.com/2014/07/templates-and-polymorphism/>

* Cplusplus forum

<http://www.cplusplus.com/forum/beginner/76522/>

<http://www.cplusplus.com/forum/beginner/1481/>

* Codelearn – J.Delta – Windows.h và hàm định dạng nội dung console

<https://codelearn.io/sharing/windowsh-ham-dinh-dang-noi-dung-console>

* Stackoverflow forum questions:

<https://stackoverflow.com/questions/9965710/how-to-change-text-and-background-color>

<https://stackoverflow.com/questions/8578909/how-to-get-current-console-background-and-text-colors>

<https://stackoverflow.com/questions/21238806/how-to-set-output-console-width-in-visual-studio/21259867>

<https://stackoverflow.com/questions/23369503/get-size-of-terminal-window-rows-columns>

<https://stackoverflow.com/questions/4053554/running-a-c-console-program-in-full-screen>

* Microsoft documents:

<https://docs.microsoft.com/en-us/windows/console/getconsolescreenbufferinfo?redirectedfrom=MSDN>

<https://support.microsoft.com/en-us/office/create-or-edit-a-hyperlink-5d8c0804-f998-4143-86b1-1199735e07bf>

* Codeguru forum:

<https://forums.codeguru.com/showthread.php?539241-RESOLVED-can-anyone-tell-me-all-colors-const-for-SetConsoleTextAttribute()-function>

* Geekforseek article: <https://www.geeksforgeeks.org/multithreading-in-cpp/>