VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY

UNIVERSIRY OF SCIENCE

FACULTY OF INFORMATION TECHNOLOGY

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

1) 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

2) 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

3) 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.

4) 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.

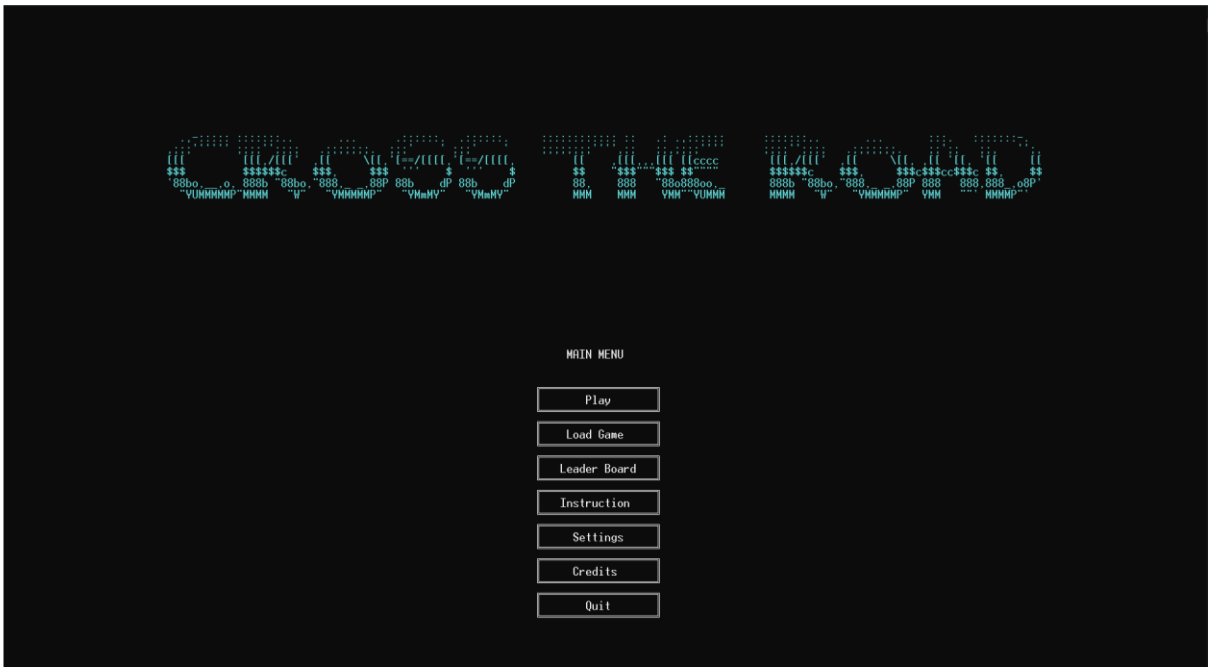
[3] Leader Board: Check the highest scores.

[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.

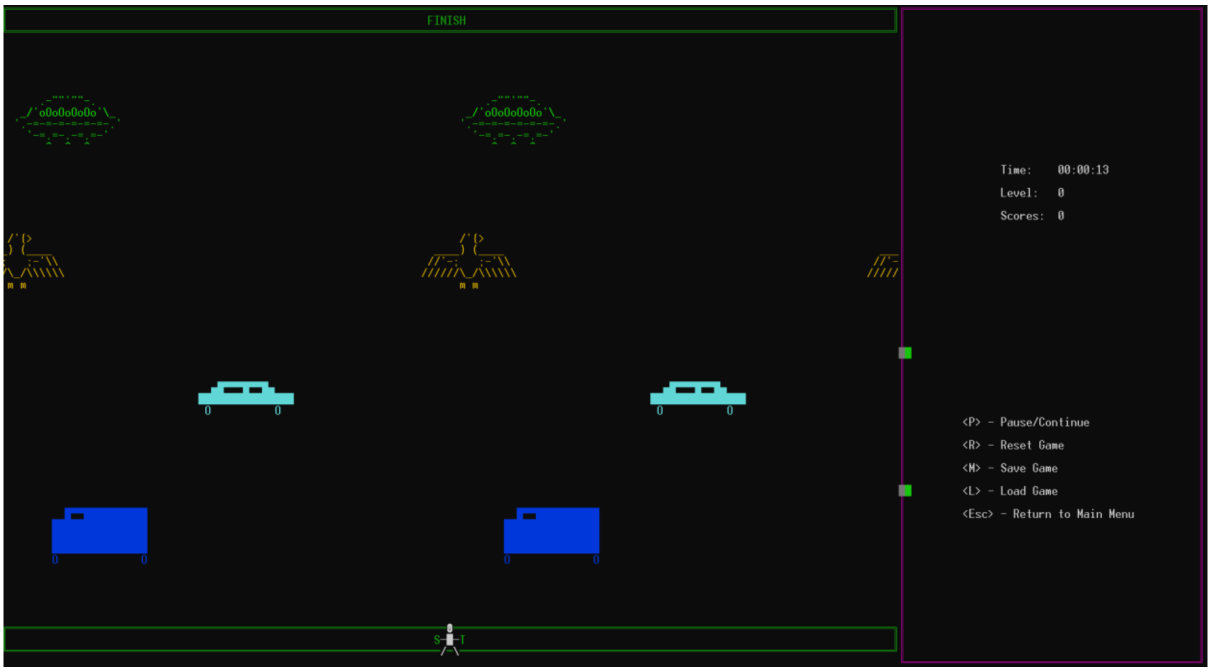
[7] Quit: Exit the game.



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.



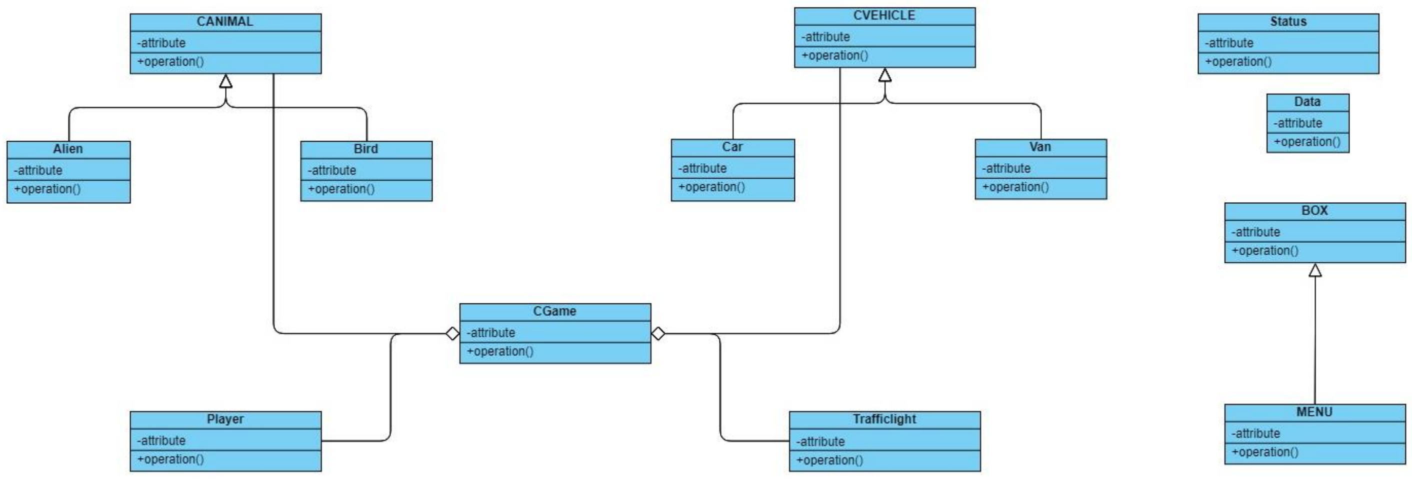
```
Time:     00:00:13
Level:    0
Scores:   0

<P> - Pause/Continue
<R> - Reset Game
<M> - Save Game
<L> - Load Game
<Esc> - Return to Main Menu
```

Pic 1.2: In-game interface

CHAPTER 2: CODE IMPLEMENTATION

1) UML diagram

(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.

2) 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.

```
#define BLACK			0
#define BLUE			1
#define GREEN			2
#define CYAN			3
#define RED				4
#define MAGENTA			5
#define BROWN			6
#define LIGHTGRAY		7
#define DARKGRAY		8
#define LIGHTBLUE		9
#define LIGHTGREEN		10
#define LIGHTCYAN		11
#define LIGHTRED		12
#define LIGHTMAGENTA 13
#define YELLOW			14
#define WHITE			15

#define KEY_UP 72
#define KEY_DOWN 80
#define KEY_LEFT 75
#define KEY_RIGHT 77
#define ENTER 13
#define ESC 27

extern int SCREEN_WIDTH_PXL;
extern int SCREEN_HEIGHT_PXL;
extern int SCREEN_WIDTH;
extern int SCREEN_HEIGHT;
extern int GAMEPLAY_W;
extern int STATUS_W;
extern int ROAD_H;
extern int SIDEWALK_H;
extern int LANE[4];
extern int SIDEWALK[2];
extern int MAX_DISTANCE;
extern int LEADERBOARD_SIZE;
extern string CCODE;
extern string SoundPath;
extern string SavePath;
extern vector <Data> SavedPlayers;
extern vector <Data> LeaderBoard;
void FixConsoleWindow();
void HideCursor();
void GotoXY(int x, int y);
void SetTextColor(WORD wColor);
WORD DefineColor(int t_color, int t_background);
void GetWindowSize();
void SetWindowSize(int width, int height);
void SetUpParameters();
void FullScreenMode();
void WindowedMode();
void setRasterFonts();
bool GetXY(int& x, int& y);
bool GetColor(int& color);
void OpenSoundFiles();
void SetAllVolumes(int volume);
void SavePlayer(const Data& playerData, int index);
void AddPlayer(const Data& Player);
void RemovePlayer(int index);
void AddDataToLeaderBoard(const Data& playerData);
void RemoveDataToLeaderBoard(int index);
string* ExtractPlayerName();
void LoadPlayerSaves();
void SavePlayerSaves();
void LoadLeaderBoard();
void SaveLeaderBoard();

int artWidth(string *art, int height);
void printMessCenter(string message);
void printMessCenter(string message, int text_color, int bg_color);
int midWidth(int width, string content);
int midWidth(int width, int content_width);
int midHeight(int height, int content_height);
int Distance(int objWidth, int objNum);

void StartUp();

//Save status of coordinates and text/background color
class Status
{
private:
	int _X, _Y;
	int _COLOR;
public:
	Status()
	{
		GetXY(_X, _Y);
		GetColor(_COLOR);
	}
	~Status()
	{
		ResetToCurrent();
	}
	int getX();
	int getY();
	int getColor();

	void ResetToCurrent()
	{
		GotoXY(_X, _Y);
		SetTextColor(_COLOR);
	}
};
```

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

```
extern string title[];
extern int title_height;
extern int title_width;
extern string angel[];
extern int angel_height;
extern int angel_width;
extern string game_over[];
extern int game_over_height;
extern int game_over_width;

extern string MAINMENU[];
extern int MAINMENU_SIZE;
extern string INSTRUCTION[];
extern int INSTRUCTION_SIZE;
extern string SETTINGS[];
extern int SETTINGS_SIZE;
extern string CREDITS[];
extern int CREDITS_SIZE;
extern string YES_NO_SELECTION[];
extern int YES_NO_SELECTION_SIZE;
extern string GUIDEBUTTONS[];
extern int GUIDEBUTTONS_SIZE;
extern string STATUSVAR[];
extern int STATUSVAR_SIZE;


class BOX
{
private:
	string content;
protected:
	int x, y;
	int width, height;
	int text_color, bg_color;
public:
	BOX();
	BOX(int _x, int _y, int _width, int _height, int _text_color, int _bg_color,
string _content);
	BOX(int _x, int _y, int _width, int _height, int _text_color, int _bg_color);

	void setBox(int _x, int _y, int _width, int _height, int _text_color, int
_bg_color, string _content);
	void setPosition(int _x, int _y);
	void setFormat(int _w, int _h, int _text_color, int _bg_color);
	void setContent(string _content);

	int getX() const;
	int getY() const;
	int getWidth() const;
	int getHeight() const;
	string getContent() const;

	void printBox();
	void printBorder();
	void printContent();
	void printContent(int _text_color, int _bg_color);
	void removeBox();
};
```

```
class MENU : public BOX
{
private:
	BOX* nBox;
	int n;
	string title;
public:
	MENU();
	MENU(string title, string* content, int n, int x, int y, int width, int height, 
int text_color, int bg_color);
	~MENU();


	void setMenu(string title, string* content, int n, int x, int y, int width, int 
height, int text_color, int bg_color);
	void setBox(int _x, int _y, int _width, int _height, int _text_color, int 
_bg_color);
	void setBoxPosition(int box_index, int _x, int _y);
	void setBoxFormat(int box_index, int _width, int _height, int _text_color, int 
_bg_color);
	void setBoxContent(string* content);
	void setTitle(string _title);

	void printMenu();
	void printTitle();
	void selectedBox(int index);

	int inputMenu();
};

void drawStartFinishLine();
void drawStatusBox();
void drawScoreBoard(const Data&);
void drawLeaderBoard();
void drawInstruction();
void drawCredits();
void drawArt(string* art, int height, int x, int y, int text_color = WHITE, int bg_color 
= BLACK);

void Settings_Menu();
int Save_Menu();
int Load_Menu();
int Remove_Menu();

int Ask_ChangeWindowMode();
int Ask_NumberVolumes();
int Ask_SaveGame();
int Ask_PlayAgain();
int Ask_PlayerName();
```

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:

```
class CVEHICLE
{
protected:
	//Current position of objects (x,y) <-> (mX, mY)
	int mX, mY;

	//Height and width of objects
	int height, width;

	//Color of objects
	int text_color, bg_color;

	//An one dimensional array that keeps the shape of
	//the vehicle objects (Car or Van).
	string* vehicle_table;

public:
	//Constructor and Destructor
	CVEHICLE();
	~CVEHICLE();

	//Draw method is used for draw Vehicle shape
	//at its position (mX, mY).
	virtual void Draw();

	//Remove method is used for remove Vehicle shape
	//at its position (mX, mY).
	virtual void Remove();

	//Move method is used for draw moving animation for Vehicle.
	virtual void Move() = 0;

	//Tell method is used for play a sound when Player – Vehicle impact occurs
	virtual void Tell() = 0;

	//Remove object animation (go to the end of the lane) and
	//set direction for objects, better UX for user
	virtual void RemoveMoving(bool);

	//Getter of some properties
	int X();
	int Y();
	int getHeight();
	int getWidth();
	string* getFigure();
	//Setter of some properties
	void setX(int);
	void setY(int);
	void setXY(int, int);
	void setColor(int, int);
};
```

*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.

```
void CVEHICLE::Draw()
{
	//Get color of object
	Status SavedStatus;
	SetTextColor(DefineColor(text_color, bg_color));

	// If the full object is in the lane
	if (mX + width <= GAMEPLAY_W)
	{
		for (int i = 0; i < height; i++)
		{
			for (int j = 0; j < width; ++j)
			{
				if (vehicle_table[i][j] != ' ')
				{
					GotoXY(mX + j, mY + i);
					cout << vehicle_table[i][j];
				}
			}
		}
	}// If haft of the object is at the end of the line
	 // and the other is at the beginning of the line
	else if (mX + width > GAMEPLAY_W)
	{
		int part2_length = (mX + width) - GAMEPLAY_W;
		int part1_length = width - part2_length;
		// Print each char of object' s first haft shape
		for (int i = 0; i < height; ++i)
		{
			for (int j = 0; j < part1_length; ++j)
			{
				if (vehicle_table[i].substr(0, part1_length)[j] != ' ')
				{
					GotoXY(mX + j, mY + i);
					cout << vehicle_table[i].substr(0, part1_length)[j];
				}
			}
		}
		// Print each char of object' s second haft shape
		for (int i = 0; i < height; ++i)
		{
			for (int j = 0; j < part2_length; ++j)
			{
				if (vehicle_table[i].substr(part1_length, part2_length)[j] !=
' ')
				{
					GotoXY(0 + j, mY + i);
					cout << vehicle_table[i].substr(part1_length,
part2_length)[j];
				}
			}
		}
	}
}
```

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).

```
void CVEHICLE::RemoveMoving(bool direct)
{
    Status SavedStatus;
    SetTextColor(SavedStatus.getColor());

    // If direct is the right
    if (direct)
    {
        //In case that object is still at the middle of the line
        if (mX + width <= GAMEPLAY_W)
        {
            for (int i = 0; i < height; ++i)
            {
                for (int j = 1; j < width; ++j)
                {
                    //Kiem tra xem ky tu ben trai co phai la ky tu ' '
khong, neu phai thi xoa
                    if (vehicle_table[i][j] != ' ' && vehicle_table[i][j -
1] == ' ')
                    {
                        GotoXY(mX + j, mY + i);
                        cout << " ";
                    }
                }
            }
        }

        //In case that object come to the end of the line
        else if (mX + width > GAMEPLAY_W)
        {
            // We divide the shape into two halves, the first haft of the object
is at the
            // end of the line and the other is at the beginning of the line
            int part2_length = (mX + width) - GAMEPLAY_W;
            int part1_length = width - part2_length;

            // Part1
            for (int i = 0; i < height; ++i)
            {
                for (int j = 1; j < part1_length; ++j)
                {
                    if (vehicle_table[i][j] != ' ' && vehicle_table[i][j -
1] == ' ')
                    {
                        GotoXY(mX + j, mY + i);
                        cout << " ";
                        }
                    }
            }
```

```
                        // Part2
                        for (int i = 0; i < height; ++i)
                               {
                                      for (int j = 0; j < part2_length; ++j)
                                      {
                                             if (vehicle_table[i][part1_length + j] != ' ' && 
vehicle_table[i][part1_length + j - 1] == ' ')
                                             {
                                                    GotoXY(0 + j, mY + i);
                                                    cout << " ";
                                             }
                                      }
                               }
       }

              //Delete the last char of object when moving
              for (int i = 0; i < height; ++i)
              {
                     if (vehicle_table[i][0] != ' ')
                     {
                            GotoXY(mX, mY + i);
                            cout << " ";
                     }
              }
       }
       // If direct is the left
       else
       {
              //In case that object is still at the middle of the line
              if (mX + width <= GAMEPLAY_W)
              {
                     for (int i = 0; i < height; ++i)
                     {
                            for (int j = 0; j < width - 1; ++j)
                            {
                                   if (vehicle_table[i][j] != ' ' && vehicle_table[i][j + 
1] == ' ')

                                   {
                                          GotoXY(mX + j, mY + i);
                                          cout << " ";
                                   }
                            }
                     }
                     //Delete the last char of object when moving
                     for (int i = 0; i < height; ++i)
                     {
                            if (vehicle_table[i][width - 1] != ' ')
                            {
                                   GotoXY(mX + width - 1, mY + i);
                                   cout << " ";
                            }
                     }
              }
              //In case that object come to the end of the line
              else if (mX + width > GAMEPLAY_W)
              {
                     // We divide the shape into two halves, the first haft of the object 
is at the end of the line and the other is at the beginning of the line
```

```
	int part2_length = (mX + width) - GAMEPLAY_W;
			int part1_length = width - part2_length;

			//Part 1
			for (int i = 0; i < height; ++i)
			{
				for (int j = 0; j < part1_length; ++j)
				{
					if (vehicle_table[i][j] != ' ' && vehicle_table[i][j + 
1] == ' ')
					{
						GotoXY(mX + j, mY + i);
						cout << " ";
					}
				}
			}

			//Part 2
			for (int i = 0; i < height; ++i)
			{
				for (int j = 0; j < part2_length - 1; ++j)
				{
					if (vehicle_table[i][part1_length + j] != ' ' && 
vehicle_table[i][part1_length + j + 1] == ' ')
					{
						GotoXY(0 + j, mY + i);
						cout << " ";
					}
				}
			}

			//Delete the last char of object when moving
			for (int i = 0; i < height; ++i)
			{
				if (vehicle_table[i][width - 1] != ' ')
				{
					GotoXY(0 + part2_length - 1, mY + i);
					cout << " ";
				}
			}
		}
	}
}
```

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.

```
class Car : public CVEHICLE
{
	//A dimensional array that keeps the shape of Car
	string table[3] =
	{
		"  ÜÛßßßÛßßÛÜ   ",
		"ÛÛÛÛÛÛÛÛÛÛÛÛÛÛÛ",
		" O          O  ",
	};

public:
	//Constructor and destructor
	Car();
	Car(int, int, int text_color = LIGHTCYAN,int bg_color = BLACK);
	Car(const Car&);
	~Car();

	//Car's check impact occur method (this method will work with Player class)
	bool isImpact(int, int);

	//Car's Move method
	void Move();

	//Car's Tell method (sound when impact occurs)
	void Tell();
};
```

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

```
void Car::Move()
{
	//RemoveMoving will set direction for object and delete the shape at the old 
position
	RemoveMoving(true);

	//Increase x position by 1 unit
	mX++;

	//If the object come to the end of the line => start again at 0
	if (mX == GAMEPLAY_W)
		mX = 0;

	//Draw the object after RemoveMoving
	Draw();
}
```

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

```
void Car::Tell()
{
	// Play impact sound when Car impact with Player
	mciSendString(TEXT("play Car_Crash from 0"), NULL, 0, NULL);
}
```

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 ‘ ‘.

```
bool Car::isImpact(int x, int y)
{
	//	x, y will be the position of Player
	//	So use x,y to check each character in the shape of the object
	//	If x,y is in the shape of the object => impact occurs
	if ((y == Y() + 1) && (x + 1 == X() - 1)) {
		return false;
	}
	if ((y == Y() + 1) && (x + 1 == X() + 15)) {
		return false;
	}
	if (x + 2 <= X() - 1 || x >= X() + 15 || y + 3 <= Y() || y >= Y() + 3) return 
false;
	else
	{
		if ((y + 2 == Y()) && (x + 2 < X() + 2)) return false;
		else if ((y + 2 == Y()) && (x >= X() + 12)) {
			return false;
		}
		else if ((y == Y() + 2) && (x + 1 <= X())) {
			return false;
		}
		else if ((y == Y() + 2) && (x + 1  != X() + 1 && x + 1 != X() + 12)) {
			return false;
		}
		else if ((y == Y() + 2) && (x >= X() + 13)) {
			return false;
		}
	}
	return true;
}
```

Continually, we consider about **Bird** class

```cpp
class Bird : public CANIMAL
{
private:
	//A dimensional array that keeps the shape of Bird
	string table[5] =
	{
		"      /'{>     ",
		"  ____) (____  ",
		" //'-;   ;-'\\\\ ",
		"//////\\_/\\\\\\\\\\\\",
		"      m m      ",
	};

	//All Bird objects can only move in one direction so the static variable "direct" 
is every object direction
	// true -> move right, false -> move left
	static bool direct;

	//The static variable "count" will count the number of bird's steps since the last 
change direction
	static int count;

public:
	//Constructor and destructor
	Bird();
	Bird(int, int, int text_color = BROWN, int bg_color = BLACK);
	Bird(const Bird&);
	~Bird();

	//Getter of some properties
	bool getDirect();
	int getCount();

	//Setter of some properties
	void setDirect(bool);
	void setCount(int);

	//Turn function will negate "direct" variable to change direction
	void Turn();

	//Bird's Move method
	void Move();

	// Bird's check impact occur method (this method will work with Player class)
	bool isImpact(int, int);

	//Bird's Tell method (sound when impact occurs)
	void Tell();
};
```

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.

```
class Trafficlight
{
	//Position of the Traffic Light
	int x, y;

	//State of the Traffic Light: 1: green ; 0: red
	bool state;

public:
	//Constructor
	Trafficlight();
	Trafficlight(int, int, bool);

	//Getter state
	bool getState();

	//Setter x and y
	void setXY(int, int);

	//Draw the traffic light at position x, y
	void Draw();

	//Change state of the traffic light (0 -> 1), (1 -> 0) and Draw the new traffic
light state at position x, y
	void changeLight();
};
```

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.

```
class Player
{
private:
	//Current position of player (x,y) <-> (mX, mY)
	int mX, mY;

	//Height and width of objects
	int height, width;

	//Color of objects
	int text_color, bg_color;

	//State of player => 1 is alive - 0 is dead
	bool mState;

	//A dimensional array that keeps the shape of Player
	string table[3]= {
	 " _ ",
	 "ÄÛÄ",
	 "/ \\"
	};

public:
	//Constructor and Destructor
	Player();
	Player(int x, int y, int text_color = WHITE, int bg_color = BLACK);
	~Player();

	//Getter of some properties
	int X();
	int Y();
	int getHeight();
	int getWidth();

	//Setter of some properties
	void setX(int);
	void setY(int);
	void setXY(int x, int y);
	void setColor(int _text_color, int _bg_color);

	//Draw method is used for draw Player picture at its position (mX, mY)
	void Draw();

	//Draw method is used for remove Player picture at its position (mX, mY)
	void Remove();

	//Draw animation when Player go up
	void UP();

	//Draw animation when Player go down
	void DOWN();

	//Draw animation when Player go left
	void LEFT();

	//Draw animation when Player go right
	void RIGHT();
//Check if Player dead
	bool isDead();

	//This Player's isImpact function will call the vector of obstacles isImpact
	//function to check if the collision between Player and obstacles occurs
	template<class T>
	bool isImpact(vector<T> v)
	{
		for (int i = 0; i < v.size(); ++i)
			// We use template so this will call isImpact function of its
appropriate v[i]
			if (v[i].isImpact(mX, mY))
			{
				mState = 0;
				v[i].Tell();
				// If impact, call Tell function of its appropriate v[i]
				// to make sound
				return true;
			}
		return false;
	}
};
```

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

```
void Player::UP()
{
	// If Player is at finish line => Can't go up
	if (mY == SIDEWALK[1])
		return;
	else
	{
		// Print Player going up animation
		Remove();
		setY(mY - 1);
		Draw();
	}

}

void Player::DOWN()
{
	// If Player is at start line => Can't go down
	if (mY == SIDEWALK[0])
		return;
	else
	{
		Remove();
		setY(mY + 1);
		Draw();
	}
}
```

```
void Player::LEFT()
{
	// If Player is at the left side of screen => Can't go left
	if (mX == 0)
		return;
	else
	{
		Remove();
		setX(mX - 1);
		Draw();
	}
}

void Player::RIGHT()
{
	// If Player is at the right side of screen => Can't go right
	if (mX + width == GAMEPLAY_W)
		return;
	else
	{
		Remove();
		setX(mX + 1);
		Draw();
	}
}

bool Player::isDead()
{
	return !mState;
}
```

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.

```
//This Player's isImpact function will call the vector of obstacles isImpact
	//function to check if the collision between Player and obstacles occurs
	template<class T>
	bool isImpact(vector<T> v)
	{
		for (int i = 0; i < v.size(); ++i)
			// We use template so this will call isImpact function of its
appropriate v[i]
			if (v[i].isImpact(mX, mY))
			{
				mState = 0;
				v[i].Tell();
				// If impact, call Tell function of its appropriate v[i]
				// to make sound
				return true;
			}
		return false;
	}
```

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.

```
class Data
{
private:
	//player's name
	string name;

	//current level, score
	int level, score;

	//time passed in this level
	clock_t TIME;

	//current position of player
	int x, y;
public:
	//Constructor
	Data(int level = 0, int score = 0, clock_t TIME = 0, int x = 0, int y = 0, string 
name = "<Empty>");

	//Setter and getter
	void setName(string);
	string getName() const;
	int getLevel() const;
	int getScore() const;
	clock_t getTime() const;
	int getX() const;
	int getY() const;

	//Overload some operator
	Data& operator=(const Data&);
	bool operator==(const string&);
	bool operator==(const Data&);
	bool operator>(const Data&);
	bool operator<(const Data&);

	//Input and output
	void input(istream&);
	void output(ostream&) const;
	friend istream& operator>>(istream&, Data&);
	friend ostream& operator<<(ostream&, const Data&);
};
```

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.

```
class CGame
{
private:
	//Obstacles
	vector <Van> vans;
	vector <Car> cars;
	vector <Alien> aliens;
	vector <Bird> birds;

	//Player Character
	Player player;

	//Trafficlight for Van and Car
	Trafficlight vanLight;
	Trafficlight carLight;

	//Current level
	int level;

	//The number of objects each road
	int objNum;

	//Current score
	int score;

	//Check point array to prevent count again the score of a road
	bool checkPoint[4];

	//Timer indices
	clock_t START_TIME;
	clock_t TIME;
	clock_t PAUSE_TIME;

	//Catch other buttons from the keyboard to check whether it is a command or not
	string buf;

	//Indicate the UNDEAD mode
	bool UnDeadCMD;


	//Pause the thread
	bool pause;

	//Indicate whether it is in game or not
	bool running;

public:
	CGame(); //Contructor
	~CGame(); //Destructor
	void Init(); //Initialize important parameters in the game
	void drawGame(); //Draw the gameplay screen
	void drawPauseScreen(); //Draw pause game screen
	void displayCommand(); //Display command that has been invoked
	void Remove(); //Remove all the current objects on the screen

	Player getPlayer();//Get the player object
	vector <CVEHICLE*>& getVehicle();//Get the vehicles objects
	vector <CANIMAL*>& getAnimal(); //Get the animal objects
	int getPoint(); //Return the [score] attribute

	void resetGame(); //Operate the reset game process
	void exitGame(); //Operate the exit game process
	void loadGame(); //Operate the load game process
	void saveGame(); //Operate the save game process
	void pauseThread(); //Change the [pause] attribute to 'true'
	void pauseGame(); //Operate the pause game process by saving the current [TIME]
then calling [pauseThread] method
	void resumeThread(); //Change the [pause] attribute to 'false'
	void resumeGame(); //Operate the resume game process by load the current [TIME]
then calling [resumeThread] method

	bool updatePosPlayer(char); //Update the position of the player object
	void updatePosVehicle(int); //Update the position of the vehicle objects
	void updatePosAnimal(); //Update the position of the aniaml objects
	void updateTime(); //Update the [TIME] attribute
	void updateGameStatus(); //Udate status: [level] and [score]
	void calcScore(); //Check player's current position and calculate the
corresponding score
	void resetData(); //Reset values all of the attributes

	void nextLevel(); //Setup new attributes for the following level
	void processAfterGame(); //Process after an impact

	bool checkImpact(); //Check whether the Player impact with an obstacle
	void checkDrawLines(); //Draw back the START and FINISH line if the player step on
it
	bool isFinish(); //Check if the player crosses the FINISH line
	bool isPause(); //Return the [pause] attribute
	bool isPlaying(); //Return the [running] attribute

	void addBuf(char key); //Catch a key from keyboard
	void CheckUnDeadCMD(); //Check whether the player types a command or not
};
```

We will clarify some important methods:

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

```
void CGame::Init()
{
	//Reset checkPoint
	for (int i = 0; i < 4; i++)
		checkPoint[i] = false;

	//Set the init position of the player
	player.setXY(midWidth(GAMEPLAY_W, 3), SIDEWALK[0]);

	//Set the init position of 2 traffic lights
	vanLight.setXY(GAMEPLAY_W, LANE[0]);
	carLight.setXY(GAMEPLAY_W, LANE[1]);

	//Check if the Distance reaches the limit or not to prevent the overwhelming of
the number of obstacles
	if (Distance(Van().getWidth(), objNum + 1) < MIN_DISTANCE)
		return;

	//Increase the number of objects in each road
	objNum = level / 2 + 2;

	vans.resize(objNum);
	cars.resize(objNum);
	birds.resize(objNum);
	aliens.resize(objNum);

	//Set the init positions of the obstacle objects
	for (int i = 0; i < objNum; ++i)
	{
		vans[i].setXY(i * (Distance(vans[i].getWidth(), objNum) +
vans[i].getWidth()), midHeight(ROAD_H, vans[i].getHeight()) + LANE[0]);
		cars[i].setXY(i * (Distance(cars[i].getWidth(), objNum) +
cars[i].getWidth()), midHeight(ROAD_H, cars[i].getHeight()) + LANE[1]);
		birds[i].setXY(i * (Distance(birds[i].getWidth(), objNum) +
birds[i].getWidth()), midHeight(ROAD_H, birds[i].getHeight()) + LANE[2]);
		aliens[i].setXY(i * (Distance(aliens[i].getWidth(), objNum) +
aliens[i].getWidth()), midHeight(ROAD_H, aliens[i].getHeight()) + LANE[3]);
	}
}
```

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.

```
bool CGame::updatePosPlayer(char MOVING)
{
	switch (MOVING)
	{
	case 'W':
		player.UP();
		checkDrawLines();
		return true;

	case 'A':
		player.LEFT();
		checkDrawLines();
		return true;

	case 'D':
		player.RIGHT();
		checkDrawLines();
		return true;

	case 'S':
		player.DOWN();
		checkDrawLines();
		return true;
	}

	return false;
}
```

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

```
void CGame::updatePosVehicle(int time)
{
	//Change the light
	if (time % 30 == 0) vanLight.changeLight();
	if (time % 50 == 0) carLight.changeLight();

	//Move Van and Car base on the state of Traffic Light
	if (vanLight.getState())
		for (int i = 0; i < objNum; ++i)
			vans[i].Move();

	if (carLight.getState())
		for (int i = 0; i < objNum; ++i)
			cars[i].Move();
}

void CGame::updatePosAnimal()
{
	//Move Bird
	//Randomly change Bird's direction
	if (rand() % 20 == 0)
	{
		//Only change direction when [count] >= 10 to prevent the abnormal movement
		if (birds[0].getCount() >= 10)
		{
			birds[0].setCount(0);
			birds[0].Turn();
		}
	}

	for (int i = 0; i < objNum; ++i)
		birds[i].Move();

	//Move Alien
	//Only change direction when [count] >= distance
	if (aliens[0].getCount() >= (objNum * Distance(aliens[0].getWidth(), objNum)))
	{
		aliens[0].setCount(0);
		aliens[0].Turn();
	}

	for (int i = 0; i < objNum; ++i)
		aliens[i].Move();
}
```

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

```
void CGame::updateTime()
{
	//Only update when the [TIME] fluctuates more than 1 second
	if (clock() / CLOCKS_PER_SEC - START_TIME - TIME < 1)
		return;

	TIME = clock() / CLOCKS_PER_SEC - START_TIME;
	score -= score > 0 ? 1 : 0;

	GotoXY(GAMEPLAY_W + midWidth(STATUS_W, STATUSVAR[0].size() + 10) + 3
+STATUSVAR[0].size(), midHeight(SCREEN_HEIGHT, STATUSVAR_SIZE + GUIDEBUTTONS_SIZE + 1) *
3 / 5);
	cout << setfill('0') << setw(2) << TIME / 3600 << ":" << setfill('0') << setw(2)
<< (TIME / 60) % 60 << ":" << setfill('0') << setw(2) << TIME % 60 << endl;
}

void CGame::updateGameStatus()
{
	GotoXY(GAMEPLAY_W + midWidth(STATUS_W, STATUSVAR[0].size() + 10) + 3 +
STATUSVAR[0].size(), midHeight(SCREEN_HEIGHT, STATUSVAR_SIZE + GUIDEBUTTONS_SIZE + 1) * 3
/ 5 + 2);
	cout << level;

	GotoXY(GAMEPLAY_W + midWidth(STATUS_W, STATUSVAR[0].size() + 10) + 3 +
STATUSVAR[0].size(), midHeight(SCREEN_HEIGHT, STATUSVAR_SIZE + GUIDEBUTTONS_SIZE + 1) * 3
/ 5 + 4);
	cout << string(STATUS_W - (midWidth(STATUS_W, STATUSVAR[0].size() + 10) + 3 +
STATUSVAR[0].size() + 1), ' ');

	GotoXY(GAMEPLAY_W + midWidth(STATUS_W, STATUSVAR[0].size() + 10) + 3 +
STATUSVAR[0].size(), midHeight(SCREEN_HEIGHT, STATUSVAR_SIZE + GUIDEBUTTONS_SIZE + 1) * 3
/ 5 + 4);
	cout << score;

	displayCommand();
}

void CGame::calcScore()
{
	//Add a score corresponding to a particular road
	if (!checkPoint[0] && player.Y() == LANE[0])
	{
		mciSendString(TEXT("play Plus_Point from 0"), NULL, 0, NULL);
		score += 100;
		checkPoint[0] = true;
	}
	else if (!checkPoint[1] && player.Y() == LANE[1])
	{
		mciSendString(TEXT("play Plus_Point from 0"), NULL, 0, NULL);
		score += 200;
		checkPoint[1] = true;
	}
else if (!checkPoint[2] && player.Y() == LANE[2])
       {
              mciSendString(TEXT("play Plus_Point from 0"), NULL, 0, NULL);
              score += 300;
              checkPoint[2] = true;
       }
       else if (!checkPoint[3] && player.Y() == LANE[3])
       {
              mciSendString(TEXT("play Plus_Point from 0"), NULL, 0, NULL);
              score += 400;
              checkPoint[3] = true;
       }
}
```

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

```
bool CGame::checkImpact()
{
       //If the UNDEAD mode is actived, the player will become invisible
       if (UnDeadCMD)
              return false;

       //Check the impact corresponding to each road
       if (player.Y() >= LANE[0])
              return player.isImpact<Van>(vans);
       if (player.Y() >= LANE[1])
              return player.isImpact<Car>(cars);
       if (player.Y() >= LANE[2])
              return player.isImpact<Bird>(birds);
       if (player.Y() >= LANE[3])
              return player.isImpact<Alien>(aliens);

       return false;
}
```

3) 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).

```
void SubThread() {

	int time = 0;
    while (true) {
		if (game.checkImpact() || game.isFinish() || buf == 'P' || buf == 'R' ||
buf == 'M' || buf == 'L' || buf == ESC)
			game.pauseGame();

		while (game.isPause()) {}

		game.drawCommand();

		if (game.updatePosPeople(buf))
			buf = 0;

		game.updatePosAnimal();
		game.updatePosVehicle((++time)%=51);


		game.updateTime();
		game.updateGameStatus();

        Sleep(100);
    }
}
```

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.

```
while (game.isPlaying())
		{
			game.calcPoint();

			if (_kbhit())
			{
				buf = toupper(_getch());

				if (buf == 'P')
				{
					while (!game.isPause()) {}

					mciSendString(TEXT("pause Gameplay_Theme"), NULL, 0, 
NULL);

					game.drawPauseScreen();

					do {
						buf = toupper(_getch());
					} while (buf != 'P');

					mciSendString(TEXT("resume Gameplay_Theme"), NULL, 0, 
NULL);

					buf = 0;
					game.resumeGame();
				}
				else if (buf == 'R')
				{
					while (!game.isPause()) {}

					game.resetGame();

					mciSendString(TEXT("play Gameplay_Theme from 0"), NULL, 
0, NULL);

					buf = 0;
					game.resumeThread();
				}
				else if (buf == 'M')
				{
					while (!game.isPause()) {}

					mciSendString(TEXT("pause Gameplay_Theme"), NULL, 0, 
NULL);

					game.saveGame();

					mciSendString(TEXT("resume Gameplay_Theme"), NULL, 0, 
NULL);

					buf = 0;
					game.resumeGame();
				}
				else if (buf == 'L')
				{
					while (!game.isPause()) {}
mciSendString(TEXT("pause Gameplay_Theme"), NULL, 0, NULL);

                                        game.loadGame();

                                        mciSendString(TEXT("resume Gameplay_Theme"), NULL, 0, 
NULL);

                                        buf = 0;
                                        game.resumeThread();
                                }
                                else if (buf == ESC)
                                {
                                        while (!game.isPause()) {}

                                        mciSendString(TEXT("stop Gameplay_Theme"), NULL, 0, 
NULL);

                                        game.exitGame();

                                        mciSendString(TEXT("play Menu_Theme from 0 repeat"), 
NULL, 0, NULL);

                                        break;
                                }
                                else
                                        game.addBuf(buf);
                        }
                        else if (game.checkImpact())
                        {
                                while (!game.isPause()) {}

                                mciSendString(TEXT("stop Gameplay_Theme"), NULL, 0, NULL);
                                system("cls");

                                game.processAfterGame();

                                if (!game.isPlaying())
                                {
                                        mciSendString(TEXT("play Menu_Theme from 0 repeat"), 
NULL, 0, NULL);
                                        break;
                                }
                        }
                        else if (game.isFinish())
                        {
                                while (!game.isPause()) {}

                                game.nextLevel();
                                game.resumeThread();
                        }
                }
```

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/