TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

ĐẠI HỌC QUỐC GIA TP.HCM

Lớp 20CLC08

BÁO CÁO ĐỒ ÁN

CROSSING GAME

GV hướng dẫn: *Trương Toàn Thịnh*

Nhóm 5

20127039 Trần Đàm Gia Huy

20127043 Nguyễn Thoại Đăng Khoa

20127655 Trần Quốc Trung

20127666 Huỳnh Tấn Vinh

Thành phố Hồ Chí Minh, tháng 12 năm 2021

MỤC LỤC

THÔNG TIN NHÓM

GV hướng dẫn: *Trương Toàn Thịnh*

Lớp: *20CLC08*

Nhóm 5:

- *20127039 – Trần Đàm Gia Huy*
- *20127043 – Nguyễn Thoại Đăng Khoa*
- *20127655 – Trần Quốc Trung*
- *20127666 – Huỳnh Tấn Vinh*

Phân chia công việc:

<table>
<tr><th>Tên</th><th>Nội dung</th></tr>
<tr><td>Nguyễn Thoại Đăng Khoa</td><td rowspan="2">• Thiết kế quá trình xử lý game theo kịch bản của đối tượng (CANIMAL,CVEHICLE,CPEOPLE,CTRAFFICLIGHT,…).
• Tìm hiểu kĩ thuật đa tiểu trình.
• Xử lý tạm dừng các toa xe.
• Tổ chức và thiết kế theo đúng tinh thần hướng đối tượng.
• Đóng góp ý tưởng về giao diện, menu game.
• Tìm hiểu, hỗ trợ về di chuyển cái đối tượng và hình vẽ, save load game.
• Thực hiện báo cáo đồ án bản Word.
• Hỗ trợ PowerPoint thuyết trình.</td></tr>
<tr><td>Huỳnh Tấn Vinh</td></tr>
<tr><td>Trần Quốc Trung</td><td rowspan="2">• Xử lý giao diện (CCONSOLE , CMENU).
• Đọc file hình vẽ và di chuyển các đối tượng (CANIMAL, CVEHICLE, CPEOPLE, …).
• Save, load game.
• Xử lý va chạm và hiệu ứng thắng thua.
• Tổ chức các đối tượng theo tinh thần hướng đối tượng.
• Đóng góp ý tưởng về kịch bản game.
• Tìm hiểu, hỗ trợ về xử lý tạm dừng toa xe và kĩ thuật đa tiểu trình.
• Thực hiện PowerPoint thuyết trình.
• Hỗ trợ báo cáo bản Word.</td></tr>
<tr><td>Trần Đàm Gia Huy</td></tr>
<tr><td colspan="2">*Note: Phân chia công việc chỉ để cho biết những ai chủ đạo trong việc xử lý và lên ý tưởng chính, còn phần lập trình thì tất cả thành viên đều cùng nhau thực hiện từ đầu.
Thời gian họp nhóm: 20h-22h tối thứ 2,4,6, Chủ nhật (Zoom)</td></tr>
</table>

Các công việc đã hoàn thành:

- Giao diện game (menu game, khung game cho các lane).
- Kết nối giao diện menu và game
- Xử lý các đối tượng từng lane di chuyển và con người di chuyển.
- Xử lý va chạm giữa con người, vật thể theo kịch bản.
- Âm thanh của đối tượng và va chạm
- Xử lý tạm dừng các toa xe (đèn giao thông) + hiệu ứng đỏ xanh
- Nâng cấp độ khó sau mỗi vòng chiến thắng
- Save, load game.
- Tổ chức đối tượng theo tinh thần hướng đối tượng.
- Áp dụng kỹ thuật đa tiểu trình
- Thiết kế Power Point
- Hoàn thành báo cáo chi tiết

Đánh giá mức độ hoàn thành đồ án: 100%

Những khó khăn khi thực hiện đồ án:

- Nhiều hàm xử lý giao diện với cú pháp lạ
- Tổ chức và thiết kế đồ án theo tinh thần hướng đối tượng
- Kết nối ý tưởng của các thành viên trong nhóm
- Lựa chọn hình vẽ các đối tượng sao cho hợp kích thước console
- Tìm hiểu nhiều nguồn tài liệu Tiếng Anh

Giải pháp khắc phục:

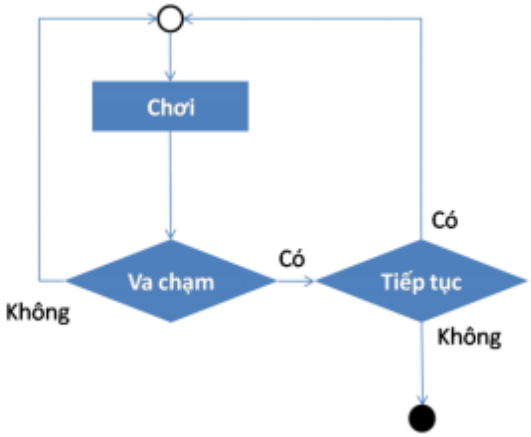
- Về các hàm xử lý giao diện thì tìm hiểu nhiều trên youtube và các diễn đàn
- Tổ chức nhiều cuộc họp để trao đổi nhiều hơn trong việc kết nối các ý tưởng
- Trau dồi Tiếng Anh
- Tìm hiểu trước những kiến thức trên lớp lý thuyết chưa học để thực hiện đồ án
- Tập chia nhỏ vấn đề trước khi thực hiện vào đồ án
- Trao đổi ý tưởng với các nhóm khác

GIỚI THIỆU TRÒ CHƠI

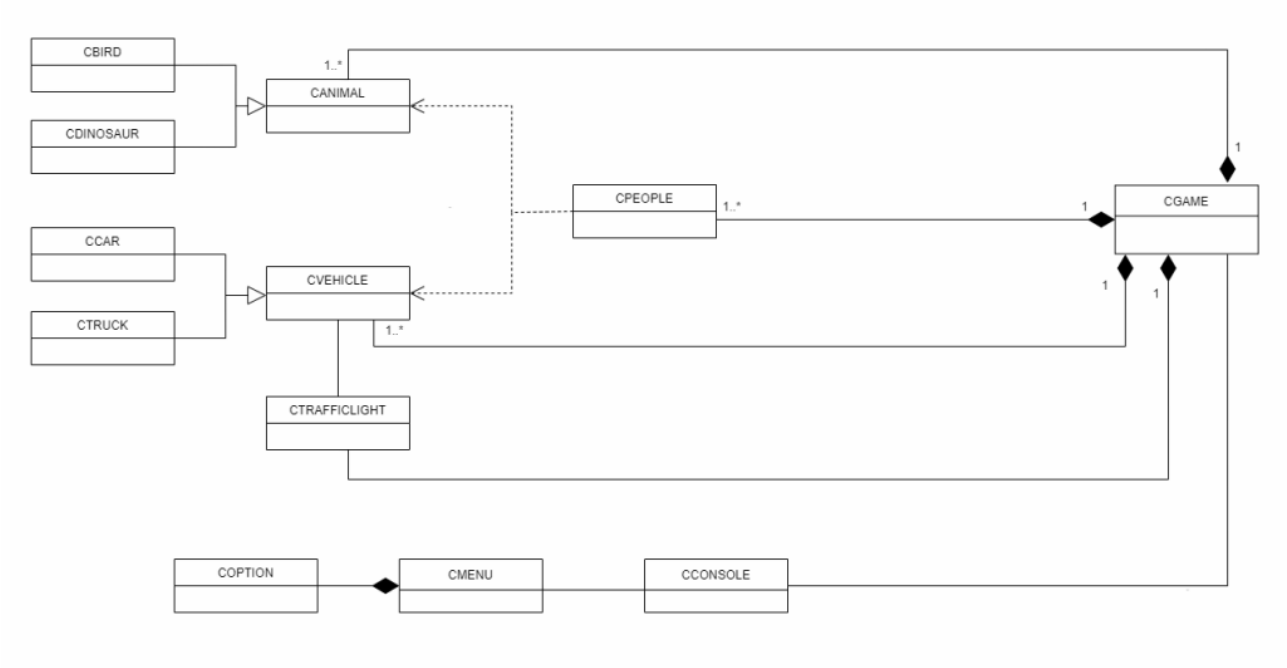
Lúc đầu khi vào game sẽ xuất hiện các xe và thú chạy qua lại, người chơi sử dụng các phím **'W'**, **'A'**, **'S'**, **'D'** để điều chỉnh hướng di chuyển của người qua đường và cố gắng tránh các xe và thú.

Khi người va chạm các xe hay thú thì kết thúc game và thông báo các option cho người chơi lựa chọn thoát hay tiếp tục chơi.

Khi người đi qua được hết các xe và thú thì sẽ lên cấp kế tiếp, độ khó của trò chơi chính là số lượng xe và thú tham gia di chuyển trên đường (Vị trí của con người mới sẽ xuất hiện trở lại khi lên cấp). Khi vượt qua 5 cấp độ thì chiến thắng trò chơi.



CÁC ĐỐI TƯỢNG TRONG TRÒ CHƠI



Đầu tiên ta cần biết sơ lược về các đối tượng trong trò chơi. Đầu tiên để xây dựng được thực đơn của trò chơi ta cần xây dựng lớp **CMENU**, và để có thể hỗ trợ cho lớp **CMENU** thực hiện từng option riêng lẻ, ta sẽ xây dựng thêm lớp **COPTION** giúp hỗ trợ đánh dấu lựa chọn của người chơi. 2 Lớp này có mối quan hệ composition với nhau.

Để có thể xử lý các kĩ thuật liên quan đến giao diện, ta cần xây dựng lớp **CCONSOLE**, lớp này đóng vai trò hỗ trợ các lớp khác xử lý các vấn đề về âm thanh, đồ họa , vì thế nó sẽ có mối quan hệ liên kết association đến lớp **CMENU** và lớp **CGAME**.

Lớp **CGAME**, lớp này đóng vai trò trung tâm trò chơi sẽ điều phối toàn bộ các đối tượng trong trò chơi này, nó bao gồm lớp trừu tượng là **CANIMAL** (2 lớp con kế thừa là **CBIRD** & **CDINOSAUR**) xử lý thú, lớp trừu tượng **CVEHICLE** (2 lớp con kế thừa là **CCAR** và **CTRUCK**) xử lý cho xe, và lớp **CPEOPLE** xử lý người.

Ngoài ra còn còn có lớp **CTRAFFICLIGHT** xử lý đèn giao thông. Lớp này sẽ liên kết với lớp **CVEHICLE** giúp xử lý tạm dừng toa xe. Tất cả những lớp con này là thành phần của **CGAME** vì thế chúng sẽ có mối quan hệ composition với lớp **CGAME**.

KĨ THUẬT HỖ TRỢ CONSOLE

```
class CCONSOLE
{
public:
	CCONSOLE();
	static SHORT GetKeyState(int nVirtKey);
	static void FixConsoleWindow();
	static void GotoXY(int x, int y);
	static void fixCur(bool CursorVisibility);
	static void disableSelection();
	static void sizeEdit(short width, short height);
	static void editFontSizeConsole(); // responsive
	static void SetBufferSize();
	static void GetSize();
	// clean area
	static void removeSpace(POINT, POINT);
	static void setColor(int color);
	static void drawGraphics(const char* filename, POINT coord, int color, int sleep=0);
	// CONSOLE HANDLING
	void ConsoleHandle();
	// loading bar
	static void initLoadingBar(int);
	// music
	static void playMusic(string);
	static void turnOffMusic(string);
	// clean key-press
	static void cleanKeyPress();
};
```

Tiếp theo, chúng ta đến với lớp **CCONSOLE** để xem các kĩ thuật hỗ trợ console. Lớp **CCONSOLE** sẽ được chia làm 2 phần, thứ nhất là các hàm hỗ trợ hiển thị console thường được sử dụng trong các trò chơi như là cố định màn hình chính FixConsoleWindow, di chuyển con trỏ đến vị trí mong muốn nhớ hàm GotoXY, xóa con trỏ FixCur, hàm GetKeyState để bắt sự kiện nhập phím từ người dùng, và các hàm còn lại sẽ hỗ trợ cho việc tùy chỉnh kích thước màn hình Console.

Nhóm thứ 2 sẽ là các hàm do Nhóm xây dựng để hỗ trợ game như là removeSpace để xóa vùng, setColor để set màu, drawGraphics để vẽ giao diện tùy thích dựa trên file hình bên ngoài đưa vào và các tính năng bật, tắt nhạc,…..

CÀI ĐẶT TRÒ CHƠI

THỰC ĐƠN

```
class COPTION
{
private:
	bool game_state;
	int level_game;
	bool load_state;
	bool play_music;
	bool exit_game;
	bool about_mn;
	bool setting_mn;
public:
	COPTION();
	~COPTION();
	// getter
	bool newGame();
	bool loadGame();
	int  levelGame();
	bool exitGame();
	bool aboutMenu();
	bool settingMenu();
	// setter
	void setGameState(bool);
	void setLevelGame(int);
	void setLoadState(bool);
	void setExitState(bool);
	void setAboutMenu(bool);
	void setSettingMenu(bool);
	// get level game
	int getLevelGameSetting();
};
```

```
class CMENU
{
private:
	COPTION *option;
public:
	CMENU();
	CMENU(COPTION*);
	~CMENU();
	// draw title "cross the road"
	void drawTitle();
	void drawMenu(); // draw menu form
	void drawOption(const char* str, POINT
point, int color); // draw elements in menu
	void checkKey(); // check key pressed
	void runMenu(); // run menu game
	void AboutHandle(); // handleoption
	void LoadHandle(bool);
	void SettingHandle();
	void soundSetting(); // setting music
	void levelSetting(); // setting level
	void resetOption();
};
```

Tiếp theo ta sẽ xem xét cách cài đặt và tổ chức trò chơi. Đầu tiên khi mới vào game, sẽ hiện ra thực đơn trò chơi cho người dùng lựa chọn. Để có thể xây dựng được menu như trên, Nhóm xây dựng 2 lớp là **COPTION** và **CMENU**, **COPTION** có các thuộc tính kiểu bool nhằm đánh dấu các option là người dùng lựa chọn. Các phương thức getter và setter để lấy và tùy chỉnh giá trị cho các thuộc tính tương ứng.

Lớp **CMENU** thì sẽ bao gồm thuộc tính là con trỏ **COPTION**, dựa vào đó **CMENU** sẽ biết được option nào đang được đánh dấu và sẽ có các phương thức xử lý cụ thể tương ứng như load game, new game, about setting, exit game…..

CGAME

```
class CGAME
{
private:
    // List of Objects
    vector<CTRUCK*> list_trucks;
    vector<CCAR*> list_cars;
    vector<CBIRD*> list_birds;
    vector<CDINOSAUR*> list_dinosaurs;
    vector<CPEOPLE> list_people;
    CPEOPLE cn;
    // Traffic light
    CTRAFFICLIGHT<CTRUCK> tl_trucks;
    CTRAFFICLIGHT<CCAR> tl_cars;
    bool IS_RUNNING; // check game is
    running or not
    int level; // level game
    int score; // score game
```

```
public:
        // Draw game
        .....
        void runGame(); // Run Game
        template<class T>
        void updatePosOfInstance(vector<T*>&);
        void updatePosOfVehicle();
        void updatePosOfAnimal();
        void updateGameScore();
        bool isRunning(); // check running state
        void resetGame(); // Reset game
        void pauseGame(); // Pause game
        void resumeGame();// Resume Game
        void loadGame(string); // Load game
        void saveGame(string); // Save game
        void exitGame(thread& t); // Exit game
        void setLevelGame(int); // Set level game
};
```

Tiếp đến ta sẽ xem xét lớp đóng vai trò trung tâm của trò chơi là lớp **CGAME**. Nó bao gồm véc tơ danh sách các con trỏ đối tượng xe tải, xe hơi, chim và khủng long, lớp **CPEOPLE** để xử lý con người và một danh sách lưu trữ những con người đã băng qua đường thành công.

Tiếp đến là lớp **CTRAFFICLIGHT** sử dụng template nhằm mục đích xử lý tín hiệu đèn giao thông cho 2 lớp con cụ thể là xe tải và xe hơi, tiếp đến là biến bool **IS_RUNNING** nhằm mục đích check xem game có đang chạy hay không, biến **level** để xử lý tăng cấp và tăng độ khó cho game.

Các phương thức như là runGame để chạy game, các hàm update vị trí cho các objects, các hàm cần thiết của 1 trò chơi như save, load, reset, exit game.

Cụ thể ta sẽ xem xét các lớp đối tượng chính của trò chơi là **CPEOPLE** và **CANIMAL** và **CVEHICLE** để xem cơ chế hoạt động của trò chơi.

CANIMAL

```
class CANIMAL
{
protected:
    int mX, mY;
    int width, height;
    bool moving_state; // 1: moving
public:
    CANIMAL(int x, int y); //
    constructor
    virtual void Move() = 0;
    virtual void Tell() = 0;
    // Get, set coordinate
    .....
    // width + height
    int getWidth();
    int getHeight();
    // Get, set moving state
    bool getState();
    void setState(bool);
};
```

```
class CBIRD : public CANIMAL
{
public:
    CBIRD(int x, int y); // Constructor
    virtual void Move(); // Move Object CBIRD
    virtual void Tell(); // Tell when impacting
};
```

```
class CDINOSAUR : public CANIMAL
{
public:
    CDINOSAUR(int x, int y); // Constructor
    virtual void Move(); // Move object CDINOSAUR
    virtual void Tell(); // Tell when impacting
};
```

Đầu tiên là lớp trừu tượng **CANIMAL** với các thuộc tính độ rộng, cao, tọa độ đứng, và trạng thái sống chết. 2 Phương thức thuần ảo là Move() sẽ thực hiện cập nhật vị trí mới của đối tượng **CANIMAL** tùy vào từng loài, phương thức Tell() sẽ phát ra tiếng kêu với ứng với từng loài, các phương thức còn lại là getter và setter cho các thuộc tính tương ứng.

Hai lớp con kế thừa lại **CANIMAL** là **CBIRD** và **CDINOSAUR** sẽ có các thuộc tính như cha, đồng thời buộc phải cài đặt lại các phương thức thuần ảo ở lớp cha ứng với từng loài cụ thể, ở đây là Move và Tell.

CVEHICLE

```
class CVEHICLE
{
protected:
    int mX, mY;
    int width, height;
    bool moving_state;
    bool tl_state;
public:
    virtual void Move() = 0;
    virtual void Tell() = 0;
    // Get, set coordinate
    .....
    // Get width + height
    .....
    //moving state
    bool getState();
    void setState(bool);
    // traffic light state
    bool getTLState();
    void setTLState(bool);
};
```

```
class CCAR : public CVEHICLE
{
public:
    CCAR(int x, int y); // Constructor
    virtual void Move(); // Move object CCAR
    virtual void Tell(); // Beep when impacting
};
```

```
class CTRUCK : public CVEHICLE
{
public:
    CTRUCK(int x, int y); // Constructor
    virtual void Move(); // Move object
    CTRUCK
    virtual void Tell(); // Beep when impacting
};
```

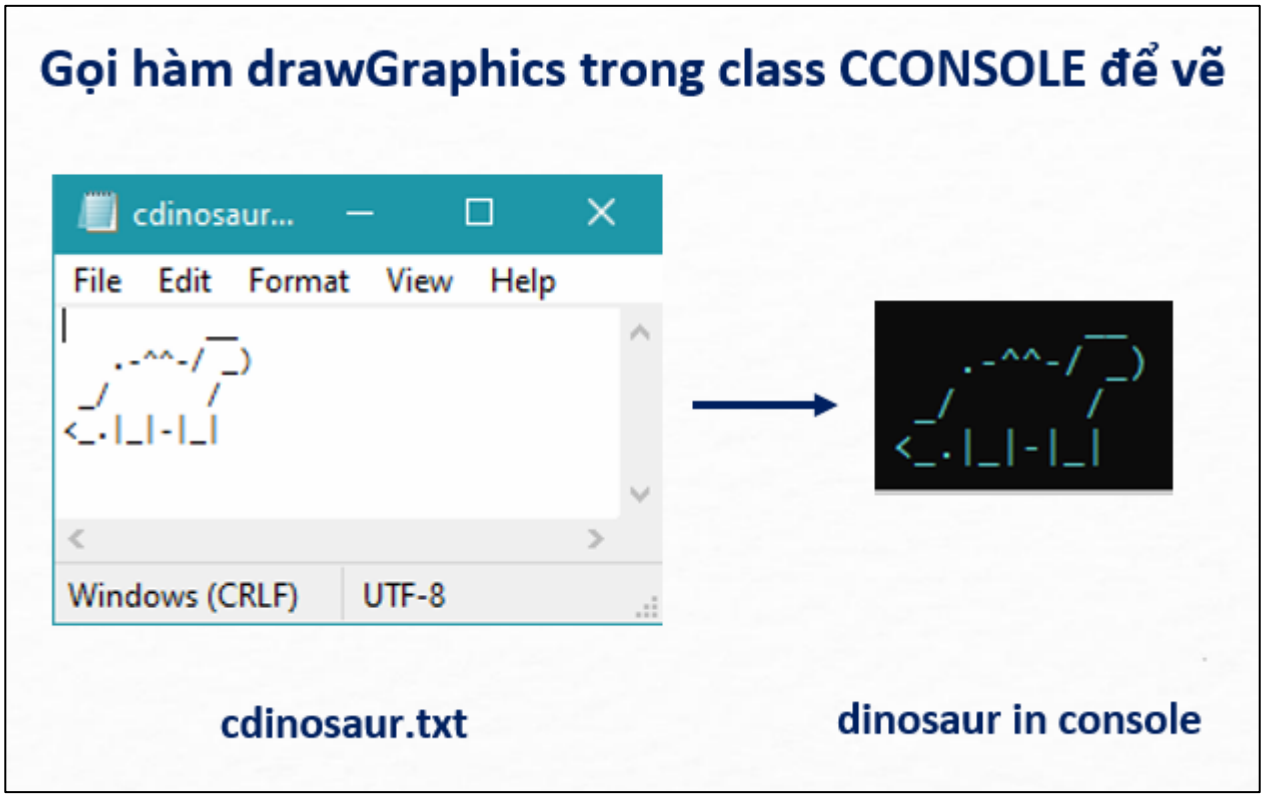
Tương tự với **CANIMAL** thì **CVEHICLE** cũng là lớp trừu tượng với các thuộc tính và phương thức đã nêu, ngoài ra còn có thêm thuộc tính và phương thức để xử lý đèn giao thông nhằm xử lý cho việc tạm dừng toa xe. 2 lớp con kế thừa **CVEHICLE** là **CCAR** và **CTRUCK** cũng buộc phải cài đặt lại 2 phương thức thuần ảo của lớp cha là Move và Tell. Hàm move của **CCAR** và C**CTRUCK** chỉ xử lý khi biến trạng thái đèn giao thông (tl_state) được bật là true.

CPEOPLE

```
class CPEOPLE
{
private:
    int width, height;
    int mX, mY;
    bool mState;
public:
    // Getter, Setter
    .....
    // Move
    void goUp();
    .....
    // checking Impact
    bool isImpact(CANIMAL*);
    bool isImpact(CVEHICLE*);
    bool isImpact(CPEOPLE);
    bool isFinish();
    bool isDead();
};
```

Lớp **CPEOPLE**, lớp sẽ có các thuộc tính là độ rộng, cao, tọa độ đứng, và trạng thái sống chết. Các phương thức getter, setter, các hàm di chuyển con người, các hàm xử lý va chạm. Ở đây có thể thấy là có 3 hàm xử lý va chạm, cụ thể là 2 hàm để xử lý va chạm cho các lớp xe và thú. Đối số truyền vào là con trỏ cha, nhờ ứng dụng tính đa hình và kế thừa nên ta chỉ cần thiết kế 2 thay vì 4 hàm. Tiếp đến là hàm xử lý va chạm với con người đã đến đích trước đó.

CÀI ĐẶT HÀM MOVE



Để có thể di chuyển ảnh liên tục

Bước 1: Di chuyển đến vị trí x,y cần vẽ. Gọi hàm drawGraphics từ class CCONSOLE để vẽ ảnh.

Bước 2: Gọi hàm removeSpace từ class CCONSOLE để in đè khoảng trắng với chiều dài, chiều rộng vật thể tương ứng.

Bước 3: Tăng tọa độ x, quay lại bước 1

Mỗi vật thể có chiều dài, rộng khác nhau, do đó tọa độ di chuyển cũng khác nhau

==> CẦN ÁP DỤNG TÍNH ĐA HÌNH

CẬP NHẬT VỊ TRÍ XE VÀ THÚ

```
template<class Obj>
void CGAME::updatePosOfInstance(vector<Obj*>& ins)
{
	for (int i = 0; i < ins.size(); i++)
	{
		if (ins[i]->getState()) // Instance is able to move
		{
			ins[i]->Move();
			if (cn.isImpact(ins[i])) // Person is impacted
			{
				ins[i]->Tell();
				cn.setState(false);
			}
		}
		if (rand() % (30 / level)) ins[i]->setState(true);
	}
}
```

```
void CGAME::updatePosOfVehicle()
{
	updatePosOfInstance(list_cars);
	updatePosOfInstance(list_trucks);
}
```

```
void CGAME::updatePosOfAnimal()
{
	updatePosOfInstance(list_birds);
	updatePosOfInstance(list_dinosaurs);
}
```

Tiếp đến ta sẽ có hàm xử lý di chuyển chính cho cả 4 lớp xe tải, xe hơi, chim và khủng long là updatePosOfInstance. Hàm được thiết kế sử dụng template để xử lý di chuyển cho cả 4 loại vector. Đồng thời trong hàm sẽ kết hợp với việc kiểm tra va chạm với người.

Chúng ta sẽ khởi tạo các đối tượng trong vector của từng loại, sau đó dùng hàm random để tạo xác suất xuất hiện của từng đối tượng (set biến trạng trái là **true**). Nó được kết hợp với biến level nhằm tăng độ khó cho game, khi tăng cấp, xác suất xuất hiện của đối tượng sẽ nhiều hơn, làm trò chơi khó chơi hơn.

Khi đối tượng được tạo ra, chúng ta sẽ gọi hàm di chuyển, sau đó tiến hành kiểm tra va chạm với đối tượng con người, nếu có va chạm chúng ta sẽ gọi hàm phát âm thanh tiếng kêu tương ứng với từng loại, đồng thời set trạng thái của con người về **false**, báo hiệu con người đã chết.

Đây là hàm xử lý chính, sau đó chúng ta sẽ có 2 hàm là updatePosOfVehicle để update vị trí của xe, chúng ta gọi hàm xử lý và truyền vào đối số là 2 danh sách xe hơi và xe tải.

Tương tự với Animal, chúng ta gọi hàm và truyền vào 2 đối số là 2 danh sách chim và danh sách khủng long.

CẬP NHẬT VỊ TRÍ CON NGƯỜI

```
void CGAME::updatePosOfPeople()
{
        if (CCONSOLE::GetKeyState(W))
            cn.goUp();
        else if (CCONSOLE::GetKeyState(A))
            cn.goLeft();
        else if (CCONSOLE::GetKeyState(S))
            cn.goDown();
        else if (CCONSOLE::GetKeyState(D))
            cn.goRight();
}
```

```
bool CPEOPLE::isDead()
{
    return (mState == false);
}
```

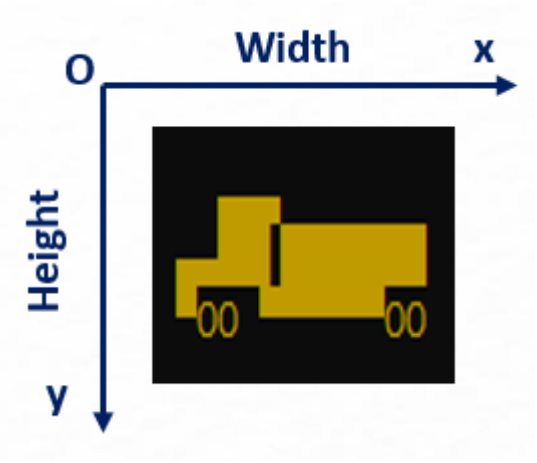
```
bool CPEOPLE::isFinish()
{
    return mY <= finishLane;
}
```

Hàm update vị trí con người khá đơn giản, ta dùng hàm getKeyState đã xây dựng ở lớp **CDINOSAUR** để bắt sự kiện nhập bàn phím của người dùng, dựa vào đó có thể di chuyển đúng theo các nút tương ứng.

Để kiểm tra con người chết, nếu trạng trái của con người là **false** tức đã chết thì ta sẽ trả về **true**, với hàm kiểm tra hoàn thành thì cũng đơn giản, ta chỉ việc kiểm tra tọa độ của con người đã đến lane cuối cùng hay chưa, nếu rồi thì trả về true.

XỬ LÝ VA CHẠM

```
bool CPEOPLE::isImpact(CVEHICLE* obj)
{
	if (mX >= obj->getX() && mX <= obj->getX()  + obj->getWidth()
	&& mY >= obj->getY() && mY <= obj->getY() + obj->getHeight())
	{
		return true;
	}
	return false;
}
```




```
bool CPEOPLE::isImpact(CANIMAL* obj)
{
	if (mX >= obj->getX() && mX <= obj->getX() + obj->getWidth()
	&& mY >= obj->getY() && mY <= obj->getY() + obj->getHeight())
	{
		return true;
	}
	return false;
}
```

```
// Impact other people
bool
CPEOPLE::isImpact(CPEOPLE cn)
{
	if (mX == cn.getX()) return
	true;
	return false;
}
```

Va chạm xe và thú:

Tiếp đến ta sẽ xem xét kĩ thuật xử lý va chạm, nhờ áp dụng tính đa hình nên ta chỉ cần cài đặt 2 hàm xử lý va chạm cho xe và thú vì con trỏ cha có thể giữ địa chỉ của những lớp con kế thừa.

Về việc kiểm tra cũng khá đơn giản, ta chỉ cần kiểm tra tọa độ con người có nằm trong vùng tọa độ của xe hay thú hay không, nếu có thì trả về true. Vùng tọa độ này có thể xem ảnh minh họa để dễ hình dung hơn, nó được tính bằng cách lấy tọa độ x + chiều rộng xe, và tọa độ y + chiều cao xe, nếu tọa độ của người trùng trong vùng tọa độ này thì ta sẽ xem đó là va chạm

Va chạm người:

Việc xử lý va chạm người cũng rất đơn giản, nếu người sau di chuyển lên trùng tới tọa độ của người đứng trước đó, thì mặc nhiên người sau đã đi đến được lane cuối cùng của trò chơi (tức tọa độ Y sẽ bằng nhau), vì thế lúc này ta chỉ cần kiểm tra tọa độ X của người sau có trùng với người trước hay không, nếu có ta sẽ trả về true.

XỬ LÝ TRÒ CHƠI

```
// Run game
void CGAME::runGame()
{
    while (IS_RUNNING)
    {
        // if it's alive, update position
        if (!cn.isDead())
            updatePosOfPeople();
        //update position of objects
        updatePosOfAnimal();
        updatePosOfVehicle();
        // Checking impact
        if (cn.isDead())
        {
            // Losing effect
            return;
        }
```

```
        //Finish
        else if (cn.isFinish())
        {
            // checking impact other people
            .....
            // update game score
            updateGameScore();
            // reset position of people
        }
        Sleep(300 - (30 * level));
    }
}
```

Sau khi đã tìm hiểu , ta sẽ xem xét cách áp dụng các hàm kĩ thuật đó vào hàm RunGame

Để hoạt động:

Đầu tiên ta cần cập nhật vị trí liên tục cho các vật thể vì thế ta cần tạo 1 vòng lặp với điều kiện là biến **IS_RUNNING** còn đang true thì thực hiện các hàm update vị trí. Nếu con người chưa chết ta sẽ cập nhật vị trí con người, cập nhật vị trí xe và thú, ngược lại nếu con người chết ta sẽ xử lý hiệu ứng thua và return. Nếu con người đã đến đích thì chúng ta sẽ kiểm tra va chạm với những con người trước đó, nếu có va chạm thì sẽ cho kết thúc trò chơi, nếu không thì cập nhật điểm số và level game, sau đó reset tọa độ con người về vị trí bắt đầu.

Để xử lý tăng mức độ khó của game:

Ở đây hàm Sleep kết hợp với biến level nhằm tăng độ khó cho game khi lên cấp cao hơn. Level tăng thì thời gian Sleep sẽ ngắn lại, việc di chuyển của những vật thể đồng thời sẽ nhanh hơn tạo độ khó cho game.

Đây là toàn bộ những hàm xử lý chính để trò chơi có thể hoạt động theo đúng kịch bản.

TẠM DỪNG CÁC TOA XE

```
template <class Obj>
class CTRAFFICLIGHT
{
private:
    bool tl_state; // trạng thái đèn giao thông
    int  cnt_time // thời gian hiện tại
    int  red_time, green_time; // thời gian đèn đỏ, đèn xanh
public:
    CTRAFFICLIGHT();
    CTRAFFICLIGHT(int green_time, int red_time);
    void updateTLState(vector<Obj*>& c_vehicle, bool flag);
```

```
// getter
    bool getTLState();
    int  getTime();
    int  getRedTime();
    int  getGreenTime();
    // setter
    void setTLState(bool);
    void setTime(int);
    void setRedTime(int);
    void setGreenTime(int);
};
```

Để có thể xử lý tạm dừng các toa xe, Nhóm xây dựng lớp **CTRAFFICLIGHT** kết hợp với lớp **CVEHICLE** để tạm dừng khi tín hiệu đèn đỏ, và chạy khi đèn xanh. Lớp này chúng em sử dụng template để có thể xây dựng cho cả kiểu **CTRUCK** và **CCAR**.

Thuộc tính của lớp sẽ gồm thuộc tính trạng thái đèn giao thông (tl_state), thời gian hiện tại (cnt_time), thời gian đèn đỏ và đèn xanh.

Các phương thức hỗ trợ như là hàm tạo. Các phương thức getter và setter cần thiết cho save & load game, kết hợp với phương thức cập nhật trạng thái đèn giao thông cho lớp **CVEHICLE**.

```
template<class Obj>
void CTRAFFICLIGHT<Obj>::updateTLState(vector<Obj*>& c_vehicle, bool
flag)
{
	cnt_time += 1;
	if (tl_state) // true (able to drive)
	{
		if (cnt_time >= green_time) tl_state = false, cnt_time = 0;
		if (flag == 0) //draw green light for CTRUCK
		else if (flag == 1)  //draw green light for CCAR
	}
	else // false (unable to drive)
	{
		if (cnt_time >= red_time) tl_state = true, cnt_time = 0;
		if (flag == 0) //draw red light for CTRUCK
		else if (flag == 1) //draw red light for CCAR
	}
	for (int i = 0; i < c_vehicle.size(); i++)
		// Set traffic light state for c_vehicle[i]
}
```

Đơn vị thời gian Nhóm tính theo vòng lặp, hàm update trạng thái đèn sẽ được đặt vào trong vòng while của hàm runGame chúng em đã đề cập. Mỗi lần vòng lặp chạy đơn vị cnt_time sẽ được tăng lên 1.

Thời gian đèn xanh và đèn đỏ Nhóm sẽ quy định cụ thể cho lớp **CCAR** và **CTRUCK** ở hàm tạo. Để phân biệt được đâu là **CCAR** và đâu là **CTRUCK** Nhóm sẽ có một biến cờ flag để đánh dấu phục vụ cho việc vẽ đèn giao thông. Nếu flag là 0 thì đó là **CTRUCK**, ngược lại là **CCAR**.

Mặc định trò chơi thì các xe sẽ được chạy tức là đèn xanh được bật, biến trạng thái đèn giao thông (tl_state) sẽ được bật là true. Trong khoảng thời gian của đèn xanh, nếu là biến cờ là 0 thì ta sẽ vẽ đèn xanh cho **CTRUCK**, ngược lại ta sẽ đèn xanh cho **CCAR**.

Vì thời gian đèn xanh và đèn đỏ đã được quy định cụ thể, do đó nếu thời gian hiện tại (cnt_time) vượt quá thời gian quy định cho đèn xanh, ta sẽ cập nhật tl_state lại là **false**, tức là tắt đèn xanh và bật đèn đỏ. Và tương tự trong khoảng thời gian đèn đỏ, nếu biến cờ là 0 ta sẽ vẽ đèn đỏ cho **CTRUCK**, ngược lại vẽ đèn đỏ cho **CCAR**.

Cuối cùng ta sẽ cập nhật trạng thái đèn cho từng đối tượng trong danh sách lớp tương ứng. Nếu đèn đỏ các toa xe sẽ dừng lại, đèn xanh các toa xe sẽ chạy.

```
CGAME::CGAME() // Constructor
{
    //.....
    // traffic light
    // green-time=20, red-time=15
    tl_trucks = CTRAFFICLIGHT<CTRUCK>(20,
    15);
    // green-time=30, red-time=15
    tl_cars = CTRAFFICLIGHT<CCAR>(30, 15);
}
```

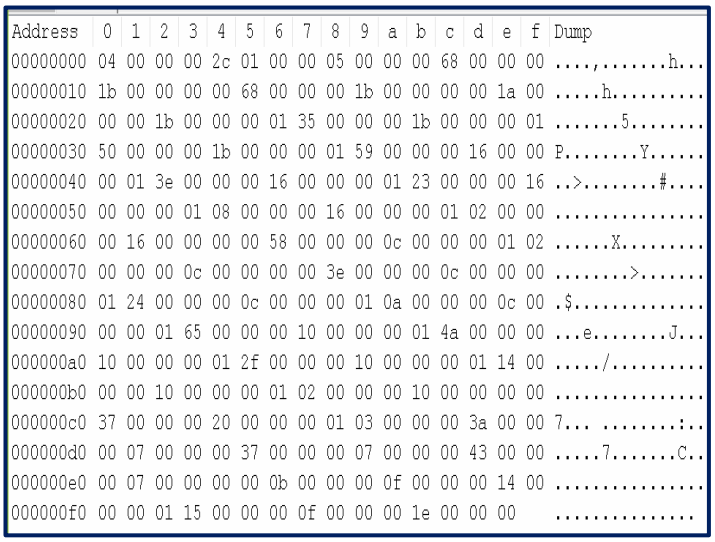
```
// Run game
void CGAME::runGame()
{
        // traffic light state
        while (IS_RUNNING)
        {
            // update traffic light state
            tl_trucks.updateTLState(list_trucks, 0);
            tl_cars.updateTLState(list_cars, 1);
            //.....
        }
}
```

Ở đây có thể thấy, ta sẽ khởi tạo các đối tượng đèn giao thông cho **CTRUCK** và **CCAR** ở hàm tạo với thời gian đèn xanh, đèn đỏ tương ứng. Tiếp theo ta sẽ dùng các đối tượng này gọi hàm cập nhật trạng thái đèn trong runGame.

Đây là toàn bộ ý tưởng về việc xử lý đèn giao thông và tạm dừng các toa xe của nhóm.

LƯU VÀ TẢI TRÒ CHƠI

Ở phần này chúng em đã có sự thay đổi so với dự tính ban đầu là lưu dữ liệu bằng **file text**. Nhưng sau đó cả nhóm đã cùng nhau thống nhất sẽ lưu trữ dữ liệu dưới dạng **file binary**. Vì nhóm thấy rằng điều này có tác dụng giúp một phần nào đó cho việc bảo mật trò chơi, người chơi sẽ khó có thể thay đổi dữ liệu của trò chơi. Khi họ không biết được cấu trúc lưu trữ trong trò chơi. Còn đối với file text thì điều đó rất dễ dàng.



Address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	Dump
00000000	04	00	00	00	2c	01	00	00	05	00	00	00	68	00	00	00	....,.......h...
00000010	1b	00	00	00	00	68	00	00	00	1b	00	00	00	00	1a	00	.....h..........
00000020	00	00	1b	00	00	00	01	35	00	00	00	1b	00	00	00	01	.......5........
00000030	50	00	00	00	1b	00	00	00	01	59	00	00	00	16	00	00	P........Y......
00000040	00	01	3e	00	00	00	16	00	00	00	01	23	00	00	00	16	..>........#....
00000050	00	00	00	01	08	00	00	00	16	00	00	00	01	02	00	00	....
00000060	00	16	00	00	00	00	58	00	00	00	0c	00	00	00	01	02	......X.........
00000070	00	00	00	0c	00	00	00	00	3e	00	00	00	0c	00	00	00	........>.......
00000080	01	24	00	00	00	0c	00	00	00	01	0a	00	00	00	0c	00	.$..............
00000090	00	00	01	65	00	00	00	10	00	00	00	01	4a	00	00	00	...e........J...
000000a0	10	00	00	00	01	2f	00	00	00	10	00	00	00	01	14	00	...../..........
000000b0	00	00	10	00	00	00	01	02	00	00	00	10	00	00	00	00	....
000000c0	37	00	00	00	20	00	00	00	01	03	00	00	00	3a	00	00	7... ....:..
000000d0	00	07	00	00	00	37	00	00	00	07	00	00	00	43	00	00	.....7.......C..
000000e0	00	07	00	00	00	00	0b	00	00	00	0f	00	00	00	14	00	....
000000f0	00	00	01	15	00	00	00	0f	00	00	00	1e	00	00	00		....

Cấu trúc file binary:

- Level – score.
- Số lượng các đối tượng.
- Vị trí, trạng thái của đối tượng.
- Vị trí, trạng thái con người đang qua đường.
- Danh sách vị trí các con người đã qua đường thành công
- Là trạng thái của đèn giao thông. Bao gồm:
    - Màu
    - Thời gian hiện tại
    - Thời gian đèn xanh
    - Thời gian đèn đỏ

Và sau đây là những đoạn code minh họa tổng quát cho phần lưu trò chơi

Level (int)

```
outfile.write((const char*)&level, sizeof(level));
```

Score (int)

```
outfile.write((const char*)&score, sizeof(score));
```

Số lượng đối tượng (int)

```
outfile.write((const char*)&n, sizeof(n));
```

Vị trí, trạng thái đối tượng

```
for (int i = 0; i < n; i++)
{
    int x = list_objects[i]->getX();
    int y = list_ objects[i]->getY();
    bool state = list_ objects[i]->getState();
    outfile.write((const char*)&x, sizeof(x));
    outfile.write((const char*)&y, sizeof(y));
    outfile.write((const char*)&state,
    sizeof(state));
}
```

Vị trí, trạng thái con người hiện tại

```
int x = cn.getX();
int y = cn.getY();
bool state = cn.getState();
outfile.write((const char*)&x, sizeof(x));
outfile.write((const char*)&y, sizeof(y));
outfile.write((const char*)&state, sizeof(state));
```

Vị trí danh sách con người qua đường

```
int n = list_people.size();
outfile.write((const char*)&n, sizeof(n));
if (n > 0)
{
    for (int i = 0; i < n; i++)
    {
        x = list_people[i].getX();
        y = list_people[i].getY();
        outfile.write((const char*)&x, sizeof(x));
        outfile.write((const char*)&y, sizeof(y));
    }
}
```

Trạng thái đèn giao thông

```
bool tl_state = tl_trucks.getTLState();  //Trạng thái đèn giao thông
int cnt_time = tl_trucks.getTime();  //Thời gian hiện tại
int red_time = tl_trucks.getRedTime(); //Thời gian đèn đỏ
int green_time = tl_trucks.getGreenTime(); //Thời gian đèn xanh
outfile.write((const char*)&tl_state, sizeof(tl_state));
outfile.write((const char*)&cnt_time, sizeof(cnt_time));
outfile.write((const char*)&red_time, sizeof(red_time));
outfile.write((const char*)&green_time, sizeof(green_time));
```

Đến với phần tải trò chơi chúng ta chỉ việc làm tương tự với lưu, chỉ đổi write thành read và get thành set

Level (int)

```
outfile.read((char*)&level, sizeof(level));
```

Score (int)

```
outfile.read((char*)&score, sizeof(score));
```

Số lượng đối tượng (int)

```
outfile.read((char*)&n, sizeof(n));
```

Vị trí, trạng thái đối tượng

```
for (int i = 0; i < n; i++)
{
	infile.read((char*)&x, sizeof(x));
	infile.read((char*)&y, sizeof(y));
	infile.read((char*)&state, sizeof(state));
	list_objects[i]->setX(x);
	list_ objects[i]->setY(y);
	list_ objects[i]->setState(state);
}
```

Vị trí, trạng thái con người hiện tại

```
infile.read((char*)&x, sizeof(x));
infile.read((char*)&y, sizeof(y));
infile.read((char*)&state, sizeof(state));
cn.setX(x);
cn.setY(y);
cn.setState(state);
```

Vị trí danh sách con người qua đường

```
infile.read((char*)&n, sizeof(n)); // size of list people
list_people.resize(n);
if (n > 0)
{
	for (int i = 0; i < n; i++)
	{
		infile.read((char*)&x, sizeof(x));
		infile.read((char*)&y, sizeof(y));
		list_people[i].setX(x);
		list_people[i].setY(y);
	}
}
```

Trạng thái đèn giao thông

```
infile.read((char*)&tl_state, sizeof(tl_state));
infile. read((char*)&cnt_time, sizeof(cnt_time));
infile. read((char*)&red_time, sizeof(red_time));
infile. read((char*)&green_time, sizeof(green_time));
tl_trucks.setTLState(tl_state);
tl_trucks.setTime(cnt_time);
tl_trucks.setRedTime(red_time);
tl_trucks.setGreenTime(green_time);
```

KỸ THUẬT ĐA TIỂU TRÌNH

```
int temp;
thread t1(&CGAME::runGame, game); // Run Thread Game
while (1)
{
	temp = toupper(_getch());
	if (!game->getPeople().isDead()) // If game is running
	{
		if (temp == 27) // ESC
		{
			game->exitGame(t1); // Exit game
			delete game;
			game=nullptr;
			return 0;
		}
		else if (temp == 'P') // Pause game
		{
			 game->pauseGame();
		}
		else if (temp == 'E') // Resume game
		{
			 if (!game->isRunning())
			 {
				game->resumeGame();
				t1.join();
				t1 = thread(&CGAME::runGame, game);
			 }
		}
		else if (temp == 'L') // Save game
		{
			 game->pauseGame(); // pause game
			 t1.join(); // Join thread
			 // Handle Save Game
			 // .....
			 // Resume Game & Re-create thread
			 game->resumeGame();
			 t1 = thread(&CGAME::runGame, game);
		}
```

```
            else if (temp == 'T') // Load game
            {
                    game->pauseGame(); // Pause game
                    t1.join(); // Join thread
                    // Handle Load game
                    // ....
                    // Resume game & Re-create thread
                    game->resumeGame();
                    t1 = thread(&CGAME::runGame, game);
            }
            else if (temp == 'B') // Back to menu
            {
                    game->exitGame(t1); // Exit game
                    // Delete memory
                    delete game;
                    game = nullptr;
                    break;
            }
            else if (temp == 'M')
            {
                    // Turn on/off game music
            }
        }
        if (game->getPeople().isDead()) // if game is over
        {
             if (temp == 'Y' ) // Renew game
             {
                  game->resetGame(); // Reset game
                  t1.join(); // Join thread
                  // Re-create thread
                  t1 = thread(&CGAME::runGame, game);
             }
        // Handle 'B', 'ESC'
        //.....
        }
}
```

Ở đây, Nhóm sẽ tách phần chạy game ra một tiểu trình con chạy song song với tiểu trình main. Nói dễ hiểu hơn là sẽ có 1 luồng để chơi game và luồng còn lại để xử lý sự kiện nhấn phím từ người dùng. Đầu tiên ta sẽ khởi tạo biến thread t1 để tạo tiểu trình chạy hàm runGame song song.

Nếu con người chưa chết thì:

Trong quá trình 'main' chạy nếu người dùng nhấn phím '**ESC**' thì sẽ gọi hàm 'exitGame'. Trong hàm 'exitGame' ta sẽ gán giá trị false cho biến **IS_RUNNING** đã đề cập để hàm runGame dừng lại, đồng thời ta cho tiểu trình 't1' join với hàm 'main' (Vì theo quy định tiểu trình 'main' phải kết thúc sau các tiểu trình con).

P: Tiếp đến, nếu người dùng ấn phím '**P**' thì sẽ gọi hàm pauseGame, ta sẽ gán giá trị **false** cho biến **IS_RUNNING** để hàm runGame dừng lại, nếu muốn chơi tiếp người dùng cần nhấn phím '**E**' lúc này sẽ gọi hàm resumeGame, ta sẽ gán giá trị true cho biến **IS_RUNNING**, sau đó cho 't1' join vào main và tạo mới lại luồng runGame.

L: Nếu người dùng ấn phím 'L' để SaveGame ta sẽ gọi hàm pauseGame, cho 't1' join vào main, đồng thời xử lý Save Game, sau đó sẽ gọi hàm resumeGame và tái tạo lại luồng runGame.

T: Tương tự với với LoadGame khi người dùng ấn 'T', ta sẽ xử lý như SaveGame, thay phần xử lý save thành load game.

B: Nếu người dùng ấn phím '**B**' để back về menu, ta sẽ gọi hàm exit game, đồng thời giải phóng bộ nhớ cho đối tượng game của **CGAME**, tránh rò rỉ vùng nhớ.

M: Ngoài ra để bật tắt nhạc trong lúc chơi người dùng có thể nhấn phím 'M'.

Y: Nếu con người đã chết, tức game kết thúc, ta sẽ hiện các option cho người dùng. Trong đó, để có thể chơi lại người dùng cần nhấn phím '**Y**', lúc này sẽ gọi hàm resetGame để reset lại toàn bộ dữ liệu trò chơi về ban đầu, join tiểu trình 't1' vào main, sau đó tái tạo lại thread runGame. Các option còn lại như '**B**' để back to menu, '**ESC**' để exit game ta sẽ xử lý như cũ.

HÌNH VẼ, HIỆU ỨNG

HÌNH VẼ ĐỐI TƯỢNG

```
// draw graphics
void CCONSOLE::drawGraphics(const char* filename, POINT coord, int color, int sleep)
{
	setColor(color);
	ifstream infile;
	string str;
	infile.open(filename);
	while (!infile.eof())
	{
		Sleep(sleep);
		getline(infile, str);
		GotoXY(coord.x, coord.y);
		cout << str;
		coord.y++;
	}
	infile.close();
}
```

Để có thể vẽ ảnh, ta cần có một file ảnh sẵn dưới dạng txt. Sau đó truyền tên file vào như 1 đối số trong hàm drawGraphics, sau đó truyền vào tọa độ muốn vẽ, màu sắc, nếu muốn tạo hiệu ứng delay cần thiết ta cần thêm tham số mặc định sleep (ban đầu set là 0).

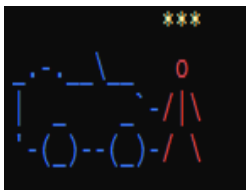
Đầu tiên ta set màu, sau đó gọi hàm đọc file, di chuyển tới tọa độ cần vẽ, đọc kí tự trong file và ghi ra ngay tọa độ trong console với màu tương ứng. Sau đó chỉ cần cố định tọa độ x và tăng tọa độ y trong vòng lặp. Kết thúc vòng lặp ta sẽ đóng file

HÌNH VẼ GIAO DIỆN GAME

Giao diện chơi game sẽ chiếm giữa, bên phải sẽ là hướng dẫn trò chơi và các phím tắt cần thiết của 1 game.

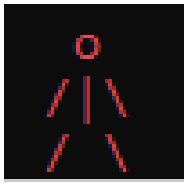
HIỆU ỨNG CPEOPLE

Hiệu ứng va chạm



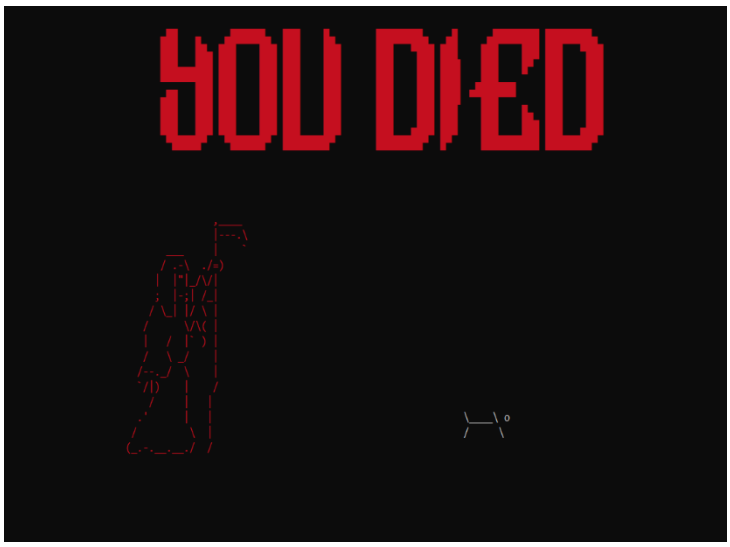
Khi trò chơi kết thúc, game sẽ hiện ra các option cho người dùng chọn lựa như là 'Y' để có thể reset lại trò chơi, 'B' để trở về menu game, 'ESC' để exit game.

Hiệu ứng thắng



Tương tự khi người chơi chiến thắng , game cũng hiện ra các option cần thiết để người chơi lựa chọn.

HIỆU ỨNG THUA



HIỆU ỨNG THẮNG

TÀI LIỆU THAM KHẢO

- **Tài liệu hướng dẫn của thầy Trương Toàn Thịnh**
- **Youtube**:
  - https://www.youtube.com/watch?v=TPVH_coGAQs&list=PLk6CEY9XxSIAeK-EAh3hB4fgNvYkYmghp&t=0s
- **Github:**
  - https://github.com/baopdh/Road-Crossing
  - https://github.com/demdecuong/Project-CS202