

ĐẠI HỌC QUỐC GIA THÀNH PHỐ
HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC
KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO
PHƯƠNG PHÁP LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

ĐỒ ÁN THỰC HÀNH CUỐI KÌ
LẬP TRÌNH TETRIS



Nguyễn Thiên Bảo - 22127032
Lương Quốc Dũng - 22127078
Nguyễn Bảo Long - 22127243
Bùi Minh Quân - 22127342

GIÁO VIÊN HƯỚNG DẪN
Thầy Bùi Tiến Lên
Thầy Trương Tấn Khoa
Thầy Lê Thanh Phong

LỜI CẢM ƠN

Kính gửi Thầy Bùi Tiên Lê, Thầy Trương Tân Khoa và Thầy Lê Thanh Phong,

Trước tiên, chúng em xin gửi lời cảm ơn chân thành đến Thầy - những người đã dành thời gian và tâm huyết hướng dẫn chúng em trong quá trình thực hiện đồ án này.

Nhờ sự giúp đỡ tận tâm của Thầy, chúng em đã có cơ hội hiểu rõ hơn về cách thiết kế, vận hành và bảo trì một chương trình bằng những phương pháp và kỹ thuật của môn Lập trình hướng đối tượng. Một chương trình không chỉ hoạt động tốt về cấu trúc, thuật toán mà còn phải mang những đặc trưng về giao diện, tối ưu hóa mã nguồn, dễ bảo trì và dễ theo dõi. Với nguyên tắc lập trình SOLID và phương pháp cấu trúc, hành vi của Design Pattern, nhóm đã vận dụng các phương pháp và kỹ thuật trên để thiết kế, xây dựng và vận hành lập trình Tetris.

Cảm ơn Thầy vì những kiến thức, kinh nghiệm quý báu mà Thầy đã truyền đạt cho chúng em. Bài luận này không thể hoàn thành tốt đẹp mà không có sự hướng dẫn và sự hỗ trợ nhiệt tình của Thầy.

Chúng em xin gửi lời tri ân và biết ơn sâu sắc đến Thầy - những người đã đồng hành và giúp đỡ, tận tâm trong suốt quá trình thực hiện đồ án này.

Kính chúc Thầy sức khỏe, hạnh phúc và thành công trong công việc và cuộc sống.

Trân trọng,

TÓM TẮT

Bài cáo cáo này gồm ba phần chính. Trong phần 1, nhóm sẽ giới thiệu cách triển khai chương trình Tetris bằng các công cụ IDE, thư viện sử dụng và các hàm đặc trưng. Ngoài ra, cũng ở phần 1, nhóm sẽ cung cấp một số hình ảnh đặc trưng thể hiện các chức năng, các khối Tetromino và giao diện người dùng.

Ở phần 2, nhóm sẽ đi sâu phân tích chi tiết thuật toán xoay của các khối Tetromino, thuật toán va chạm hay thuật toán xóa hàng, thuật toán rơi sử dụng thư viện Raylib.h. Kết hợp hình ảnh, mã giả và trình bày để thể hiện các chức năng quan trọng của game, giúp chương trình trực quan và dễ sử dụng, bảo trì.

Cuối cùng, là phần thực thi chương trình, bao gồm các yếu tố logic, luồng chạy và các hướng dẫn hoàn thiện chương trình.

Báo cáo này có ý nghĩa thực tiễn cao, giúp nhóm nâng cao kỹ năng thiết kế, cài đặt và quản lý một chương trình - Game Tetris. Nhóm đã áp dụng các kiến thức lý thuyết về Design Pattern và thực hành về các nguyên tắc SOLID. Nhóm hy vọng qua báo cáo này sẽ nhận được những nhận xét và góp ý từ Thầy để hoàn thiện hơn trong các đồ án sau.

Keywords: Tetris, SOLID, Tetromino, raylib.h

1 GIỚI THIỆU

1.1 THÔNG TIN

Tên, ID, E-mail thành viên trong nhóm

ID	Name	Email
22127032	Nguyễn Thiên Bảo	22127032@student.hcmus.edu.vn
22127078	Lương Quốc Dũng	22127078@student.hcmus.edu.vn
22127243	Nguyễn Bảo Long	22127243@student.hcmus.edu.vn
22127342	Bùi Minh Quân	22127342@student.hcmus.edu.vn

PROJECT LINK: <https://github.com/thienbao/tetris>

BẢNG PHÂN CÔNG CÔNG VIỆC:

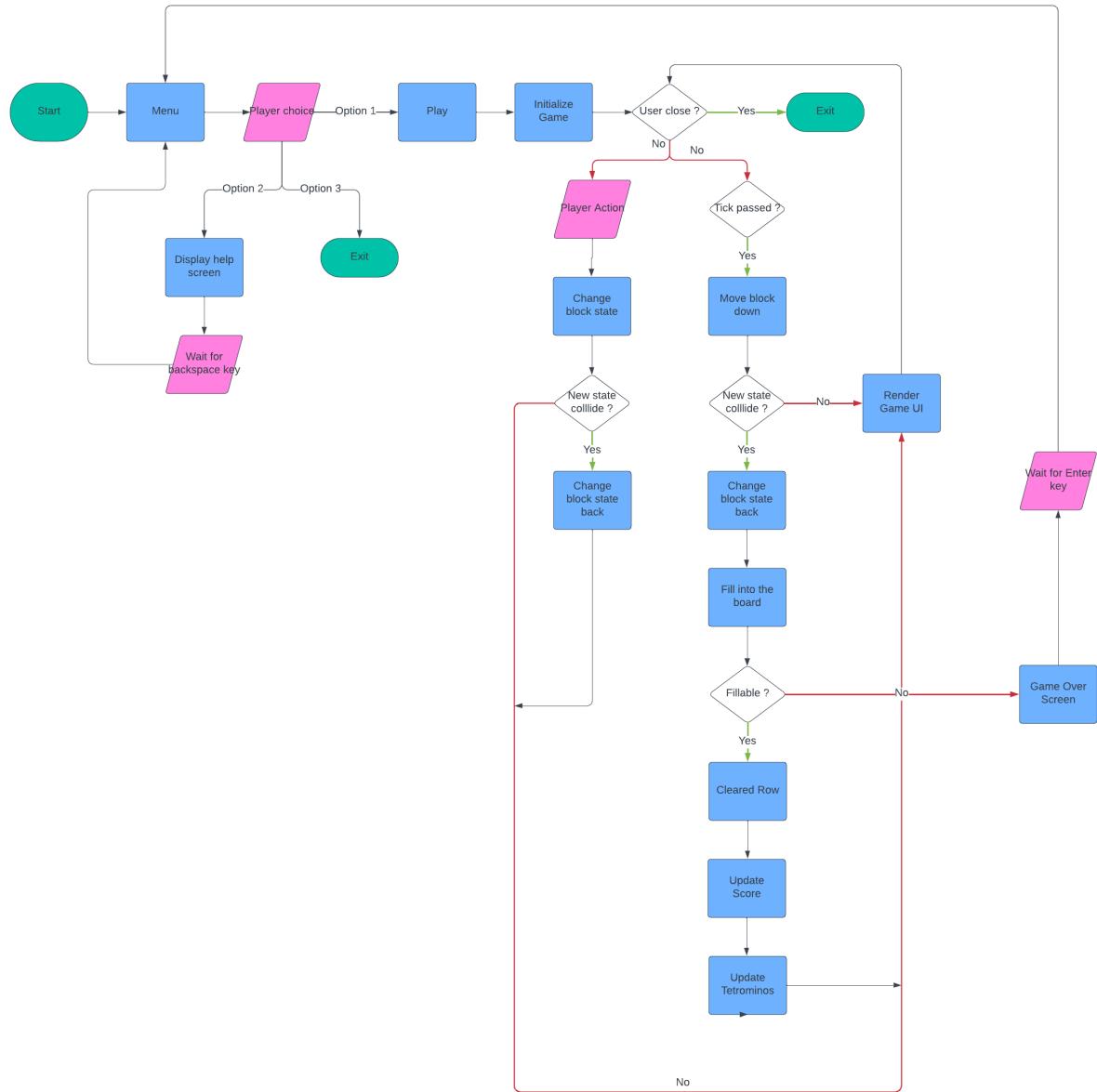
TÊN	CÔNG VIỆC	PHẦN TRĂM
22127032 - Nguyễn Thiên Bảo	<ul style="list-style-type: none">Lập trình Thuật toán va chạm, xoay và xóa Tetromino, logic GGame.Render Bảng (Board), Tetromino, màu sắc (Constant).Tổng hợp và quản lý Code.	100%
22127078 - Lương Quốc Dũng	<ul style="list-style-type: none">Viết Báo cáo phần Game TutorialPhân tích thuật toán Xoay và thiết kế các mẫu ví dụCung cấp hình ảnh, âm thanh, hỗ trợ về thuật toán và logic Game.	100%
22127243 - Nguyễn Bảo Long	<ul style="list-style-type: none">Lập trình các chức năng liên quan đến User Interface, sounds - giao diện Menu.Viết mã giả, các ví dụ cho mỗi ClassViết báo cáo phần Project Format - Tổng hợp thực hiện báo cáo	100%
22127342 - Bùi Minh Quân	<ul style="list-style-type: none">Lập trình các chức năng liên quan đến User Interface, sounds - giao diện Game.Tổng hợp các hàm sử dụng trong chương trình, của thư viện Raylib.hViết báo cáo phần Program Executing Instructions.	100%

Mục lục

1 GIỚI THIỆU	3
1.1 THÔNG TIN	3
2 Game tutorial	5
2.1 Pre-Starting	5
2.2 Game Starting	7
2.3 In-Game	10
2.4 Game Finishing	14
3 Project Format	16
3.1 Header Files Used	16
3.2 Classes Used	17
3.2.1 Class Tetromino	18
3.2.2 Class Board	24
3.2.3 Class Game	27
3.3 Game Mechanics	30
3.4 User Interface	30
4 Program Executing Instruction	33
5 Conclusion	34
A Reference	35
REFERENCES	35

2 Game tutorial

Trước khi vào Hướng dẫn chi tiết khởi động Game, đây là Flow chart thể hiện toàn bộ quá trình hoạt động của Game. Sơ đồ Flowchart thể hiện cách các hàm, các chức năng vận hành, từng chi tiết và mạch logic của game:



Hình 1: Flow chart của Game

2.1 Pre-Starting

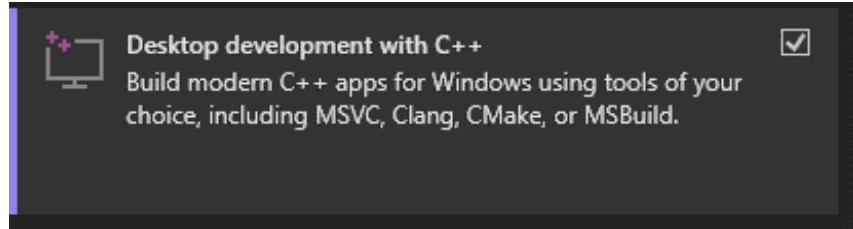
Để tiến hành khởi chạy chương trình Tetris bằng IDE, nhóm đã sử dụng Microsoft Visual Studio Community 2022 (64-bit), và -std=c++11 tối thiểu.

Để khởi động Game, người dùng có thể chạy main.exe hoặc sử dụng Visual Studio để chạy tetris-cpp.sln (đuôi .sln tạo project trong IDE).

Cài đặt Thư viện Raylib.h trên Visual Studio

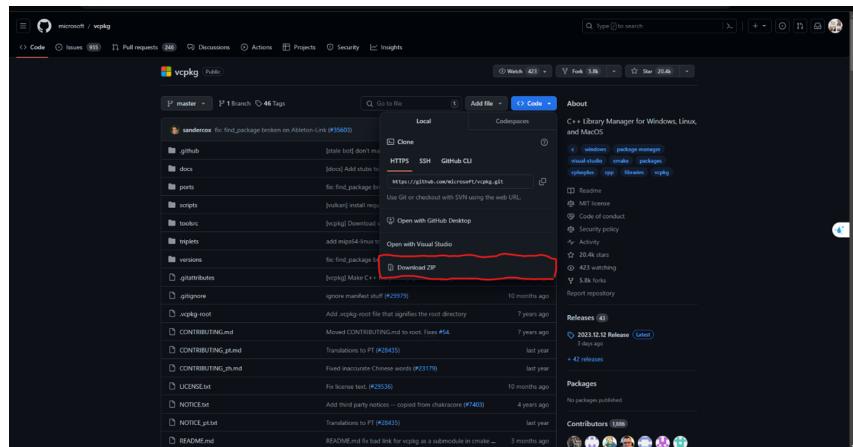
1. Bước 1: Cài đặt Visual Studio (nếu bạn chưa có)

- Cài đặt Visual Studio phiên bản Community tại đây
- Trong trang thiết lập hãy cài đặt Desktop development with C++.



2. Bước 2: Cài đặt vcpkg

- Tải về vcpkg tại đây.
- Tải file zip về bằng cách nhấp vào "Code" ở bên trên danh sách các file và chọn tải về file zip.
- Giải nén file và chọn nơi giải nén là ổ C.



- Sau đó đổi tên folder vừa giải nén vcpkg-master thành vcpkg.
- Sau đó nhấp chuột phải vào folder vcpkg và chọn
- Sau đó ta gõ "./bootstrap-vcpkg.bat" trong terminal.
- Sau khi chạy xong ta tiếp tục nhập lệnh: "./vcpkg integrate install"
- Sau khi chạy xong ta chạy đoạn lệnh sau để cài đặt thư viện raylib: "./vcpkg install raylib:x64-windows"
- Dợi lệnh chạy xong và bây giờ chúng ta có thể sử dụng thư viện raylib trong chương trình trên Visual Studio

3. Bước 3: Thêm thư viện raylib vào chương trình

- Để sử dụng thư viện raylib trong chương trình trên Visual Studio thì sau khi cài đặt thành công thư viện raylib từ bước 2 thì bây giờ bạn chỉ cần vào chương trình trên Visual Studio và thêm lệnh: "include <raylib.h>"
- Vào đầu file thì bạn đã có thể sử dụng được thư viện raylib trong C++

Khởi chạy Game Trình biên dịch hỗ trợ chuẩn C++11 trở lên. Nếu chưa được hỗ trợ, có thể chạy bằng dòng lệnh sau

```
1 $ g++ -std=c++11 your_file.cpp -o your_program
```

Bảng điều khiển Tetris:

Di chuyển:

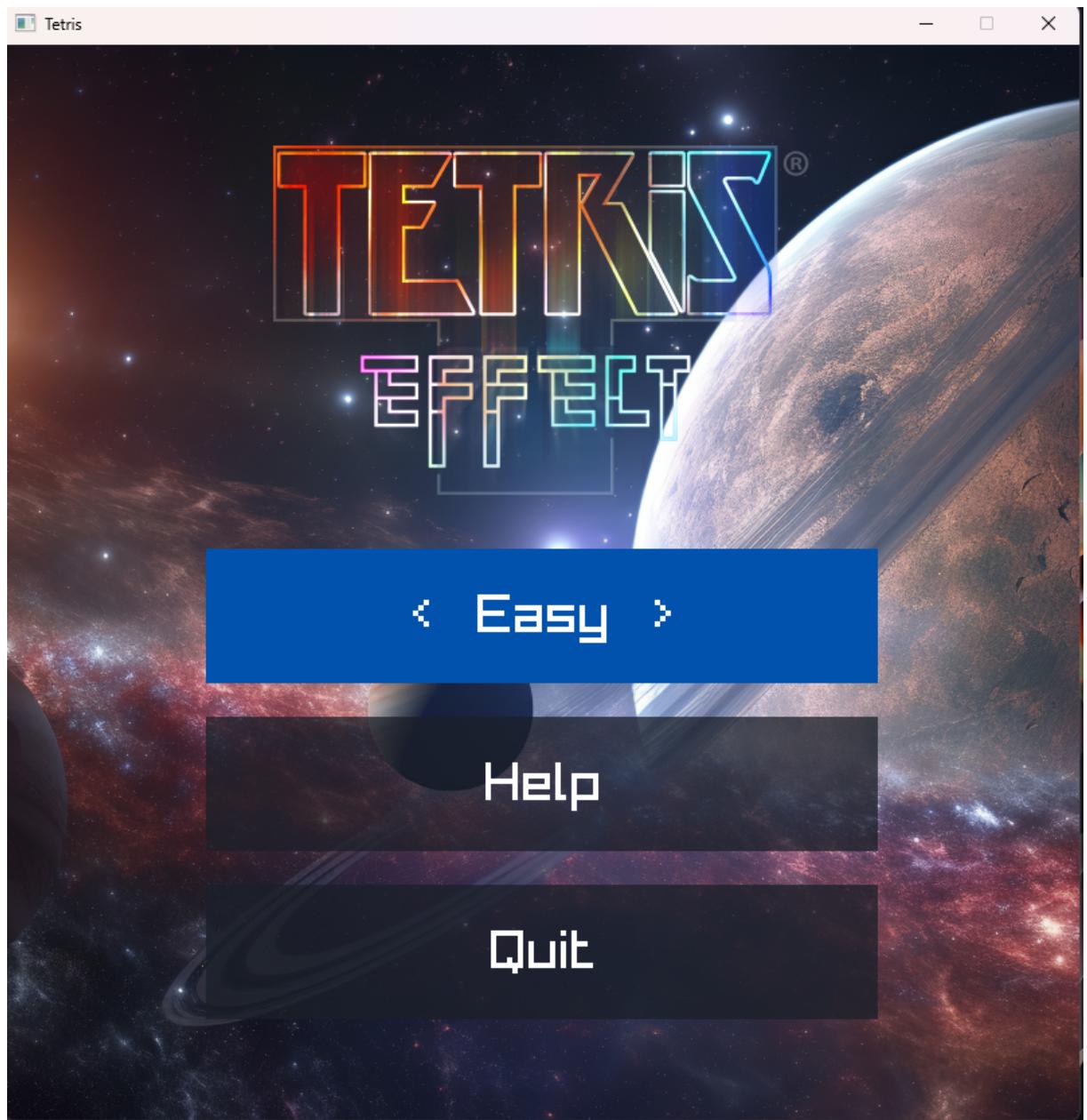
- Phím mũi tên lên: Di chuyển khối Tetris theo chiều kim đồng hồ
- Phím mũi tên xuống: Di chuyển khối Tetris xuống
- Phím mũi tên trái: Di chuyển khối Tetris sang trái
- D hoặc phím mũi tên phải: Di chuyển khối Tetris sang phải
- ESC: Thoát khỏi trò chơi

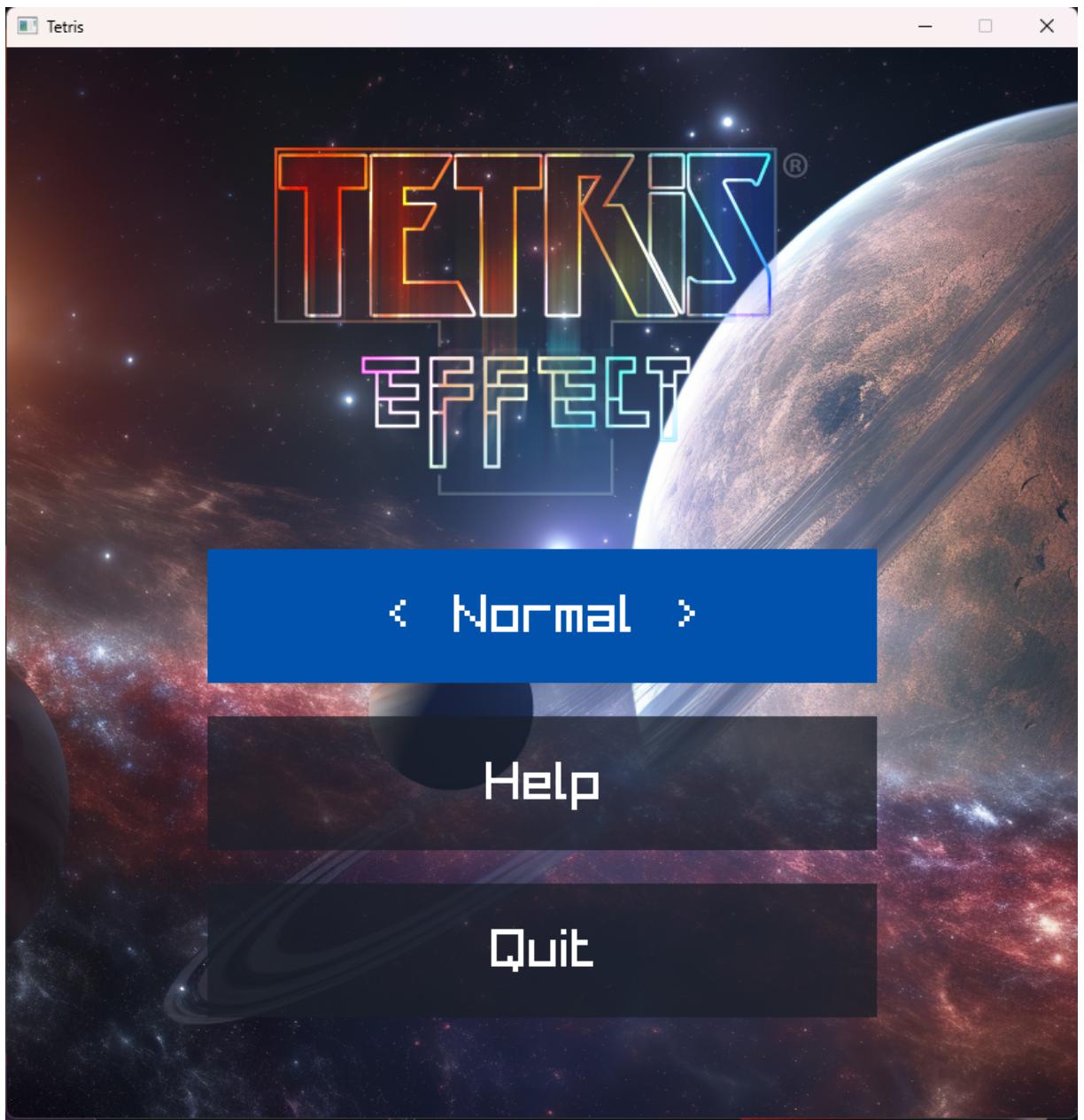
Thao tác:

Phím Space: Thả khối Tetris xuống Phím Enter: Chọn các mục và chức năng cần sử dụng

2.2 Game Starting

Sau khi khởi chạy chương trình, một cửa sổ hiện ra với Menu - các chức năng chính của chương trình như "Easy - Normal - Hard - Asian" tương ứng với mức độ khó tăng dần, Help - Màn hình hướng dẫn các chức năng sử dụng điều khiển, cách điều khiển và dừng/thoát chương trình.





Hình 2: Giao diện Menu, người dùng có thể sử dụng Left - Right Arrow để chọn chế độ khó

Khi người dùng chọn Help - cửa sổ sẽ hiển thị cách sử dụng phím để di chuyển, xoay và tăng tốc Tetromino, cách tính điểm và các hiệu ứng Gameplay:



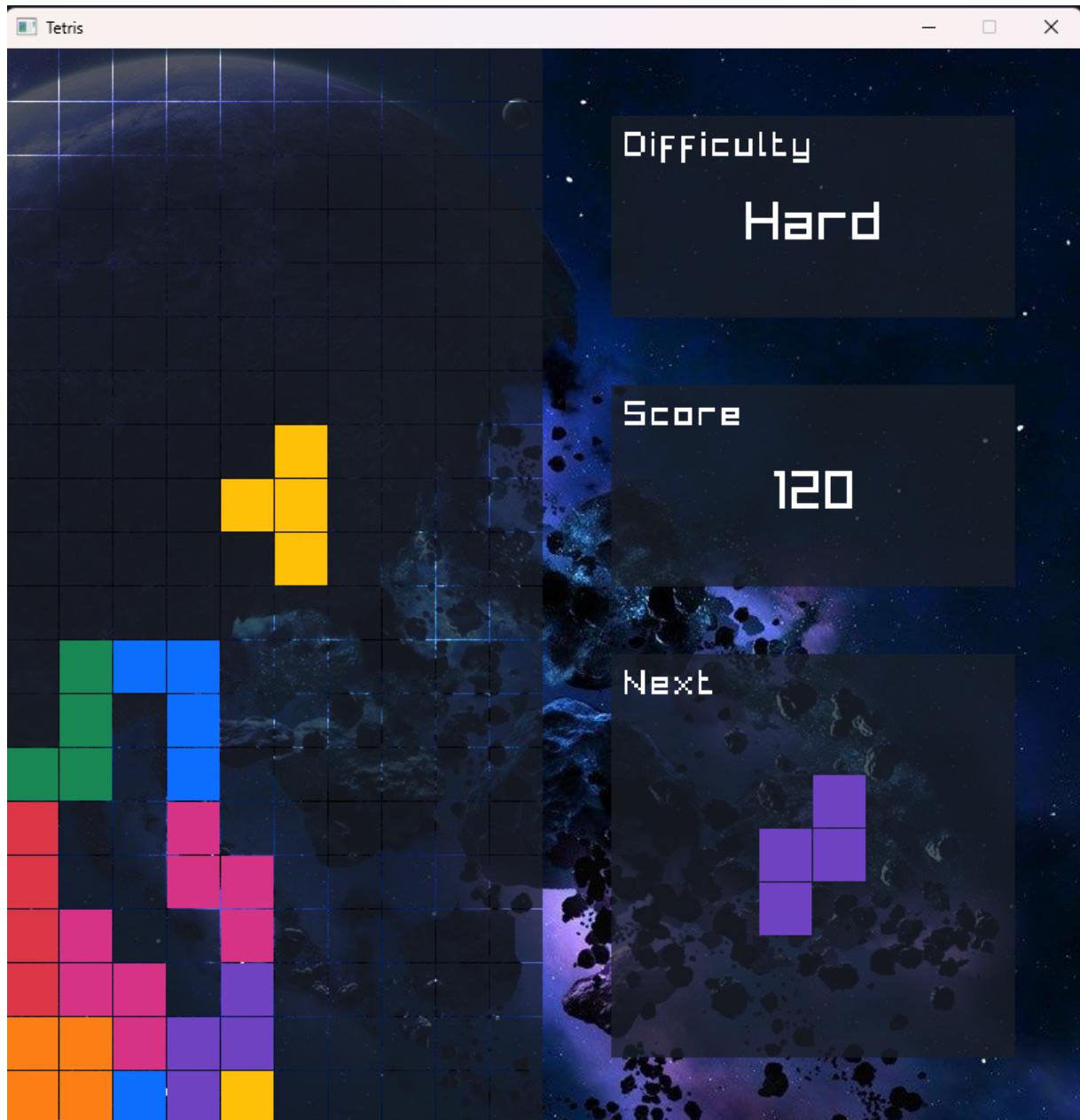
Hình 3: Instruction Board

2.3 In-Game

Khi người chơi nhấn Enter để chọn chế độ chơi, tiến hành vào Game, chương trình sẽ tiến hành các bước:

(a) **Khởi tạo:**

- i. Tạo đối tượng Game với mức độ difficulty.
- ii. Khởi tạo bảng Tetris và Tetromino hiện tại.
- iii. Chọn ngẫu nhiên Tetromino tiếp theo.
- iv. Căn chỉnh vị trí Tetromino hiện tại tùy theo loại.



Mỗi chế độ chơi, chương trình sẽ có những User Interface, background khác nhau.

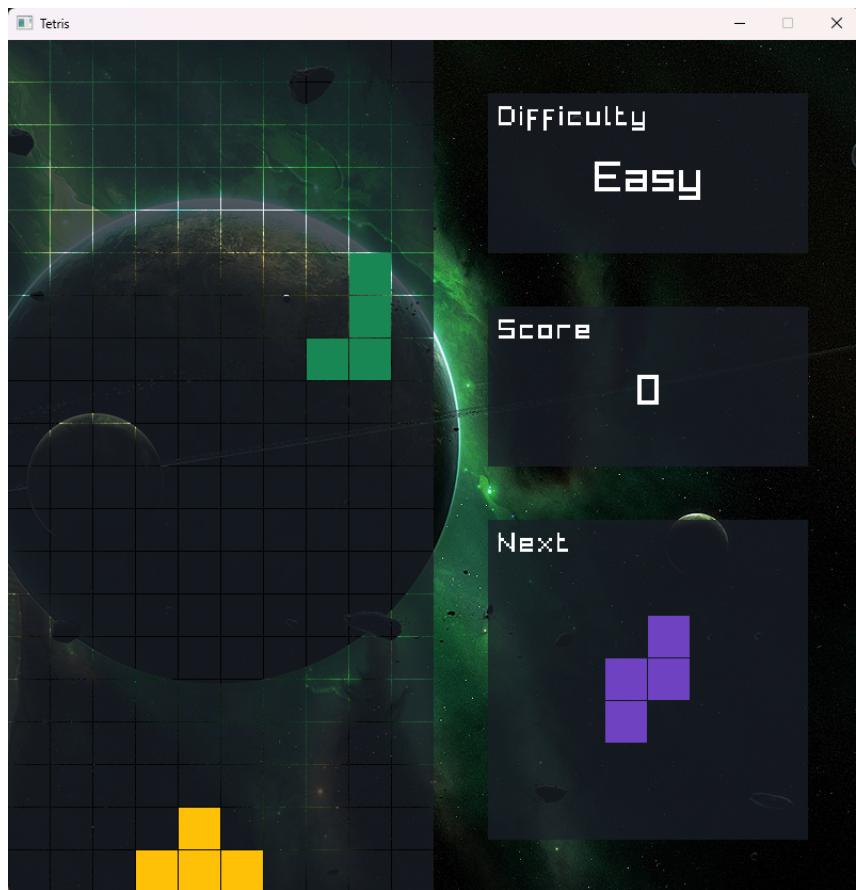
Hình 4: Hình ảnh ở chế độ chơi HARD

(b) Vòng lặp chính:

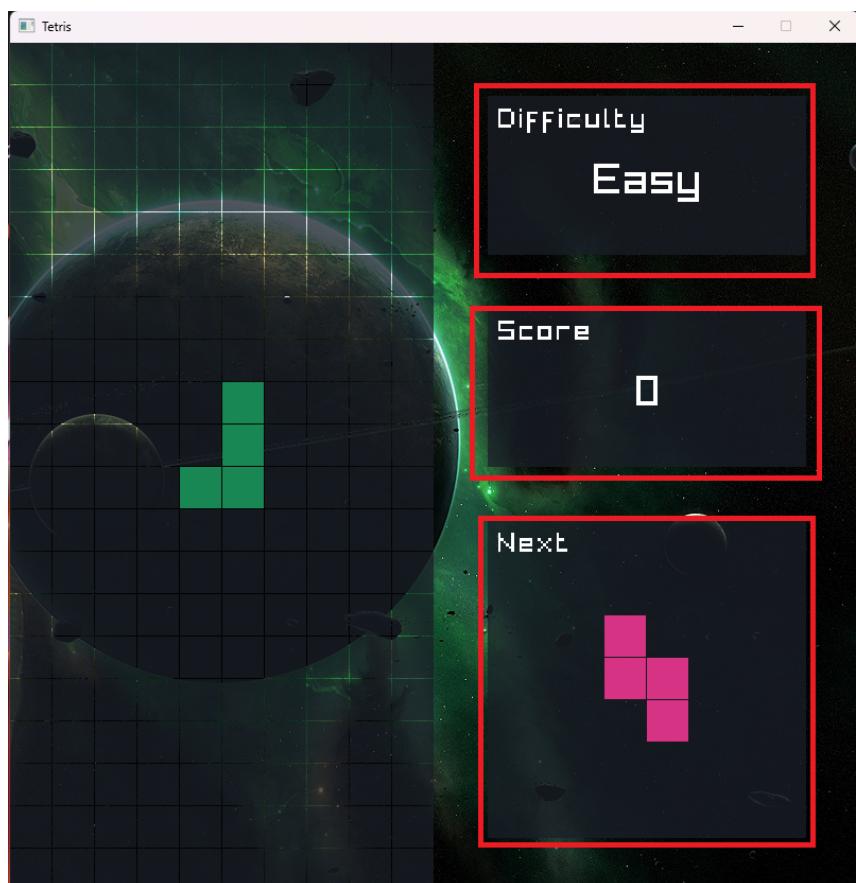
- i. Kiểm tra điều kiện cửa sổ đóng: Nếu đóng thì thoát chương trình.
- ii. Nhận thao tác người chơi: Phím bấm để điều khiển Tetromino.

(c) Xử lý thao tác:

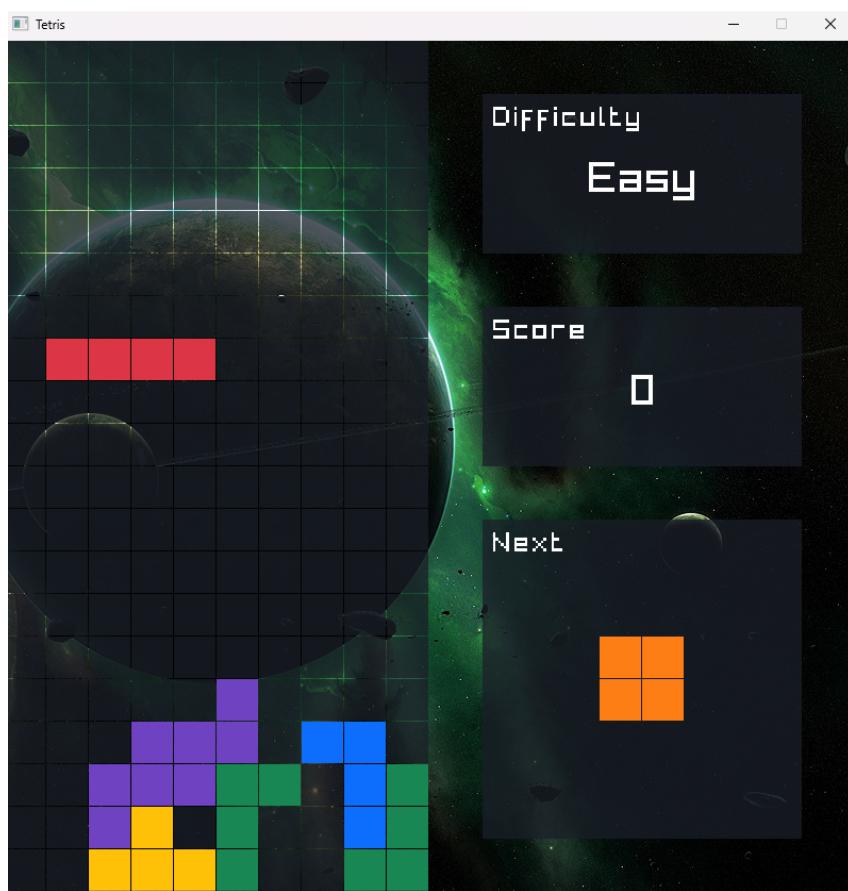
- i. Xoay Tetromino CW.
- ii. Di chuyển Tetromino sang trái, phải.
- iii. Di chuyển Tetromino xuống.
- iv. Di chuyển Tetromino xuống nhanh nếu nhấn Space.



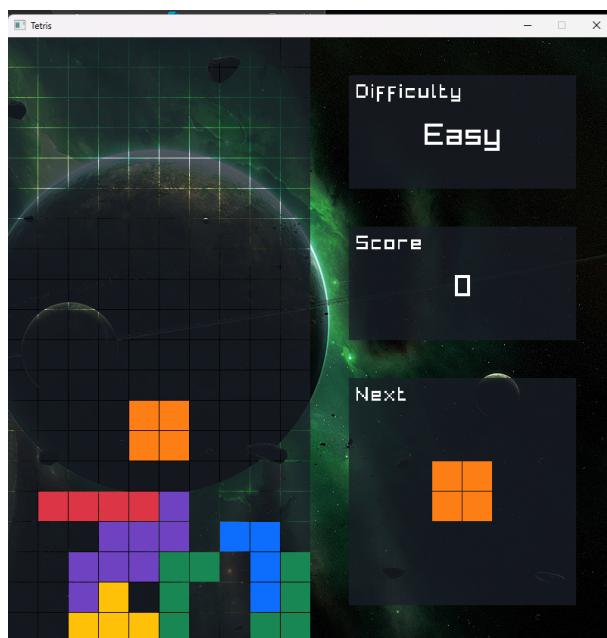
Hình 5: Ingame T-block outlook



Hình 6: Ingame L-block outlook



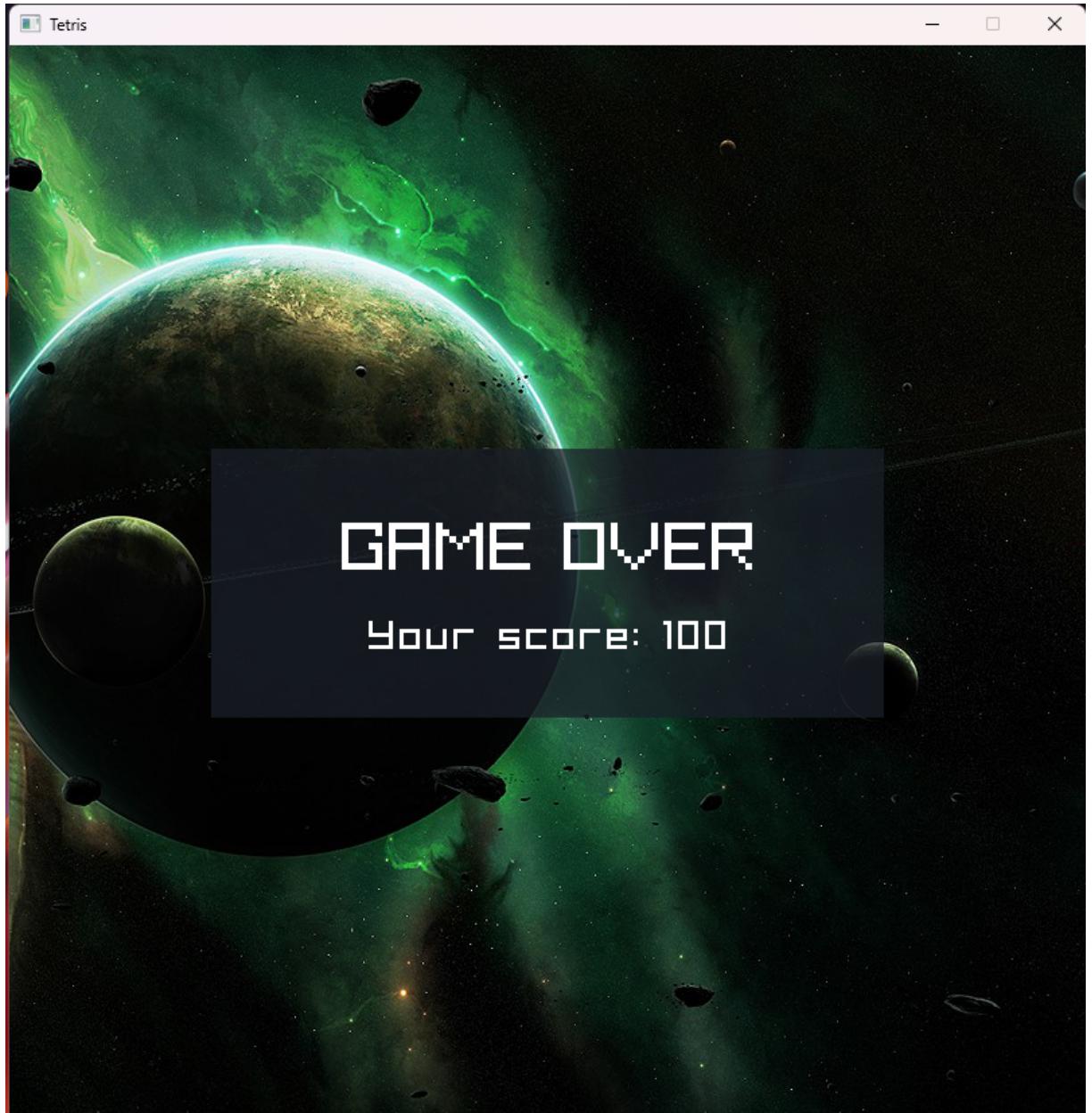
Hình 7: Ingame I-block outlook



Hình 8: Ingame O-block outlook

2.4 Game Finishing

Khi kết thúc game, người chơi sẽ được chuyển đến cửa sổ thể hiện điểm cuối cùng với thông báo như sau:



Hình 9: Z matching example

- **if (isOver):** Kiểm tra biến isOver. Biến này được thiết lập thành true trong hàm fall() khi Tetromino va chạm với phần trên cùng của bảng hoặc không thể đặt xuống được nữa.
- Bắt đầu vẽ (BeginDrawing()): Khởi đầu quá trình vẽ một frame mới.
- ClearBackground(RAYWHITE): Xóa màn hình, chuẩn bị cho việc vẽ các nội dung mới.
- **DrawTexture(background, ...):** Vẽ background của màn hình Game Over.

- DrawRectangle(150, 300, 500, 200, COLORS.at(0)): Vẽ một khung hình chữ nhật màu xanh đậm (COLORS.at(0)) ở giữa màn hình.
- **Vector2 gameOverTextSize = MeasureTextEx(font, ...):** Tính toán kích thước của text "GAME OVER" theo font đã được load.
- **DrawTextEx(font, ...):** Vẽ text "GAME OVER" màu trắng, căn giữa khung hình chữ nhật màu xanh đậm.
- **Vector2 scoreTextSize = MeasureTextEx(font, ...):** Tính toán kích thước của text "Your score: " + score.
- **DrawTextEx(font, ...):** Vẽ text "Your score: " + score màu trắng, căn giữa khung hình chữ nhật màu xanh đậm, thấp hơn text "GAME OVER".
- EndDrawing(): Kết thúc quá trình vẽ frame.

Trong vòng lặp while:

- Kiểm tra điều kiện thoát khỏi vòng lặp:
 - !WindowShouldClose(): Kiểm tra cửa sổ chưa được yêu cầu đóng.
 - GetKeyPressed() != KEY_ENTER : KimtrangichichanhnphmEnter.Nuchathotkhivnglp, tiptcvframevinidung
- Kết thúc vòng lặp while:**
- Thoát khỏi vòng lặp if (isOver).
 - Tiếp tục thực thi các dòng lệnh bên dưới trong hàm run().

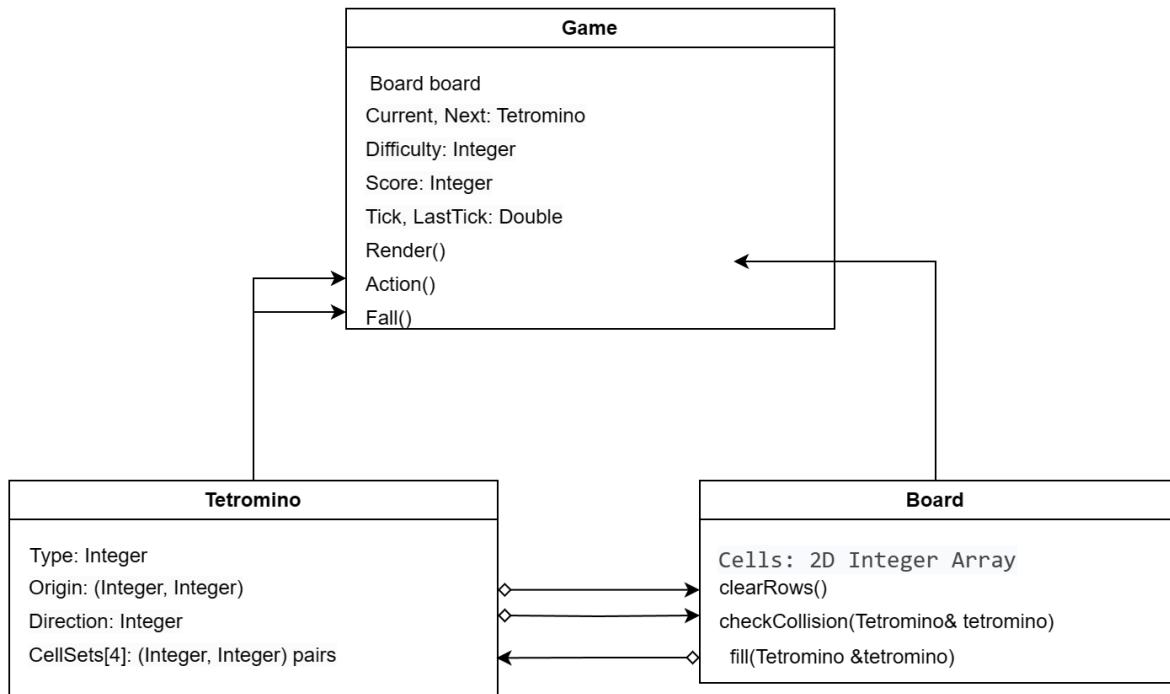
```
if (isOver) {
    // Game over screen
    while (!WindowShouldClose() && GetKeyPressed() != KEY_ENTER) {
        BeginDrawing();
        ClearBackground(RAYWHITE);
        DrawTexture(background, 800 / 2 - background.width / 2, 800 / 2 - background.height / 2, WHITE);
        DrawRectangle(150, 300, 500, 200, COLORS.at(0));
        Vector2 gameOverTextSize = MeasureTextEx(font, string("GAME OVER").c_str(), 50, 5);
        DrawTextEx(font, string("GAME OVER ").c_str(), { 150 + (250 - gameOverTextSize.x / 2), 350 }, 50, 5, WHITE);
        Vector2 scoreTextSize = MeasureTextEx(font, ("Your score: " + to_string(score)).c_str(), 30, 5);
        DrawTextEx(font, ("Your score: " + to_string(score)).c_str(), { 150 + (250 - scoreTextSize.x / 2), 425 }, 30, 5, WHITE);
        EndDrawing();
    }
    break;
}
```

Hình 10: Mã chương trình thông báo GameOver

3 Project Format

3.1 Header Files Used

Trước khi đi chi tiết vào các Hàm và lớp sử dụng, chúng ta sẽ cùng tìm hiểu cấu trúc hoạt động của 3 lớp chính sẽ được trình bày sau bao gồm BOARD, GAME, VÀ TETROMINO:



Board.h:

(a) **Trách nhiệm:** Khai báo lớp Board.

(b) **Nội dung:**

- i. `#ifndef BOARD_H` và `#define BOARD_H`: Bảo vệ khỏi việc bao gồm trùng lặp.
- ii. `#include` cho các thư viện cần thiết (`<vector>`, `<raylib.h>`), các tệp header khác (`constants.h`, `tetromino.h`) và khai báo `using namespace std`.
- iii. Khai báo lớp Board với phần private và public.
- iv. Biến thành viên (`content`).
- v. Hàm thành viên (`Board()`, `render()`, `fill()`, `checkCollision()`, `clearRows()`).
- vi. `#endif`: Kết thúc tệp header.

Constants.h:

(a) **Trách nhiệm:** Khai báo các hằng số và biến toàn cục.

(b) **Nội dung:**

- i. `#ifndef CONSTANTS_H` và `#define CONSTANTS_H`: Bảo vệ khỏi việc bao gồm trùng lặp.
- ii. `#include` cho các thư viện cần thiết (`<vector>`, `<raylib.h>`).
- iii. Khai báo hằng số `CELL_SIZE` và vector toàn cục `COLORS`.

iv. `#endif`: Kết thúc tệp header.

Game.h:

(a) **Trách nhiệm:** Khai báo lớp Game.

(b) **Nội dung:**

- i. `#ifndef GAME_H` và `#define GAME_H`: Bảo vệ khỏi việc bao gồm trùng lặp.
- ii. `#include` cho các thư viện cần thiết (`<string>`), và các tệp header khác (`board.h`, `tetromino.h`, `menu.h`).
- iii. Khai báo lớp Game với phần private và public.
- iv. Biến thành viên (`board`, `current`, `next`, `difficulty`, `score`, `tick`, `lastTick`, `gameMusic`, `gameSound`).
- v. Hàm thành viên (`Game()`, `run()`, `action()`, `fall()`).
- vi. `#endif`: Kết thúc tệp header.

Menu.h:

(a) **Trách nhiệm:** Khai báo lớp Menu.

(b) **Nội dung:**

- i. `#ifndef MENU_H` và `#define MENU_H`: Bảo vệ khỏi việc bao gồm trùng lặp.
- ii. `#include` cho các thư viện cần thiết (`<raylib.h>`), và các tệp header khác (`constants.h`).
- iii. Khai báo lớp Menu với phần private và public.
- iv. Biến thành viên (`option`, `level`, `musicSound`, `moveSound`, `lastColorChangeTime`, `currentTextColor`).
- v. Hàm thành viên (`Menu()`, `getRandomColor()`, `getOption()`, `getLevel()`, `drawHelpScreen()`, `render()`).
- vi. `#endif`: Kết thúc tệp header.

Tetromino.h:

(a) **Trách nhiệm:** Khai báo lớp Tetromino.

(b) **Nội dung:**

- i. `#ifndef TETROMINO_H` và `#define TETROMINO_H`: Bảo vệ khỏi việc bao gồm trùng lặp.
- ii. `#include` cho các thư viện cần thiết (`<iostream>`, `<vector>`, `<raylib.h>`), và các tệp header khác (`constants.h`).
- iii. Khai báo lớp Tetromino với phần private và public.
- iv. Biến thành viên (`type`, `origin`, `direction`, `cellSet`).
- v. Hàm thành viên (`Tetromino()`, `setOrigin()`, `getType()`, `getCurrentCells()`, `render()`, `move()`, `rotateCW()`, `rotateCCW()`).
- vi. `#endif`: Kết thúc tệp header.

3.2 Classes Used

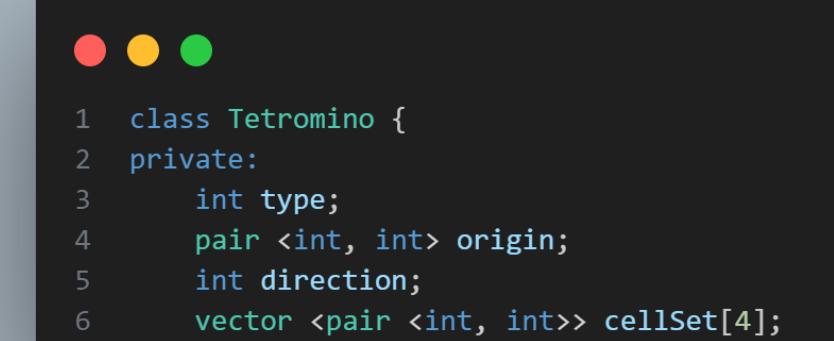
Trong chương trình, nhóm sử dụng 3 class chính: Board, Game, Tetromino. Sau đây, nhóm sẽ trình bày chi tiết thuật toán và cấu trúc của từng loại class, hàm có trong class.

3.2.1 Class Tetromino

Có chức năng tổ chức các loại Tetromino khác nhau và cung cấp các phương thức để di chuyển, quay và vẽ chúng trên bảng trò chơi.

Định nghĩa Lớp – Tetromino là một lớp đóng gói chức năng khối hình và đồ họa cho trò chơi

<i>Tetromino</i>	
type: int	
pair<int, int> origin	
direction: int	
vector <pair <int, int>> cellSet[4]	
	Tetromino(int type = 0);
	getType(): int
	vector <pair <int, int>> getCurrentCells();
	render(): void
	move(int x, int y): void
	rotateCW(): void
	rotateCCW(): void



Các biến thành viên được khai báo riêng tư:

- **int type:** Để lưu trữ loại của Tetromino. (ví dụ: 0 cho I-BLOCK, 1 cho O-BLOCK, v.v.).
- **pair<int, int> origin:** Để lưu trữ một cặp số nguyên biểu thị điểm trung tâm của Tetromino.
- **int direction:** Một số nguyên biểu thị trạng thái xoay hiện tại (0-3)
- **vector<pair<int, int>> cellSet[4]:** Một mảng 4 phần tử để lưu trữ tập hợp các ô (cell) của Tetromino cho từng hướng.

Constructor:

- Nhận một số nguyên **type** và khởi tạo một Tetromino với loại này.
- Tăng giá trị **type** lên 1 để điều chỉnh.
- Thiết lập tọa độ gốc (**origin**) là {0, 0} và hướng (**direction**) là 0.
- Mỗi loại Tetromino được định nghĩa thông qua các tập hợp các ô (**cell**) trong biến **cellSet** tương ứng với hướng (**direction**) của Tetromino.
- Ví dụ, đối với Tetromino I-Blocks:
 - **cellSet[0] = {{0, 1}, {1, 1}, {2, 1}, {3, 1}};**
 - **cellSet[1] = {{2, 0}, {2, 1}, {2, 2}, {2, 3}};**
 - **cellSet[2] = {{0, 2}, {1, 2}, {2, 2}, {3, 2}};**
 - **cellSet[3] = {{1, 0}, {1, 1}, {1, 2}, {1, 3}};**
- Trong trạng thái xoay 0, các ô của Tetromino được đặt thành một hàng ngang với điểm trung tâm của nó ở vị trí (2, 1). Khi xoay theo chiều kim đồng hồ, điểm trung tâm được giữ nguyên là (2,1), tiếp tục xoay ngang theo trục Ox ta được các điểm lần lượt là (2,0); (2,1); (2,2) và (2,3).

Các Hàm Thành vién:

- **Hàm getType:** Trả về loại của Tetromino.
- **Hàm getCurrentCells:**

```

● ● ●

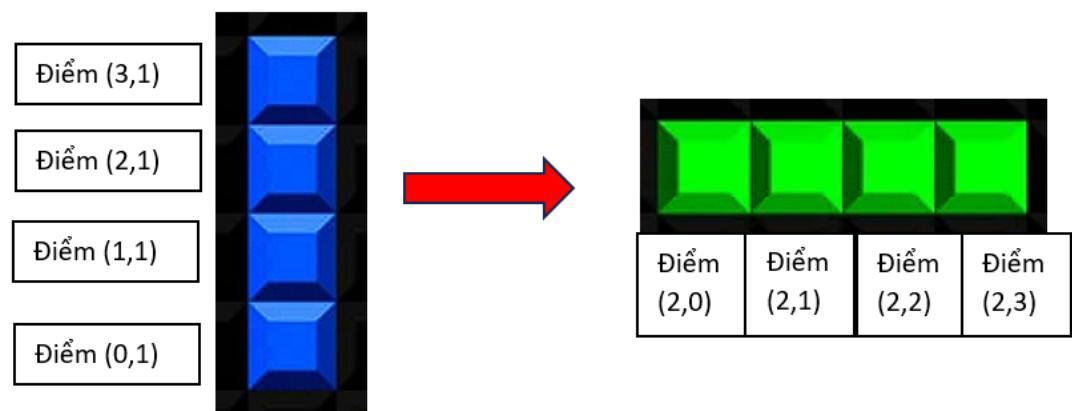
1  vector <pair <int, int>> Tetromino::getCurrentCells() {
2      vector <pair <int, int>> cells;
3      for (auto cell : cellSet[direction]) {
4          cells.push_back({ cell.first + origin.first, cell.second + origin.second });
5      }
6      return cells;
7  }

```

Trả về tập hợp các ô hiện tại của Tetromino dựa trên hướng và tọa độ gốc (`origin`).
Tọa độ mới được tính toán bằng cách thêm giá trị tọa độ của ô từ `cellSet` vào tọa độ gốc.

- **Hàm render:** Vẽ Tetromino lên màn hình bằng cách sử dụng `DrawRectangle` cho mỗi ô của Tetromino với màu sắc tương ứng.
- **Hàm move:** Di chuyển Tetromino theo hướng x và y bằng cách cập nhật tọa độ gốc.
- **Hàm rotateCW và rotateCCW:**
 - Quay Tetromino theo chiều kim đồng hồ (CW) hoặc ngược chiều kim đồng hồ (CCW) bằng cách tăng giá trị hướng lên 1 và chia lấy dư cho 4.
 - Xoay các cặp số nguyên trong mỗi vector của mảng `cellSet`.
 - Cập nhật gốc của Tetromino dựa trên trạng thái xoay mới.
 - Ví dụ, để xoay Tetromino I-piece từ trạng thái xoay 0 sang trạng thái xoay 1, các phương thức `rotateCW()` sẽ thực hiện các bước sau:
 - * Xoay các cặp số nguyên trong mỗi vector của mảng `cellSet` I-blocks như sau:

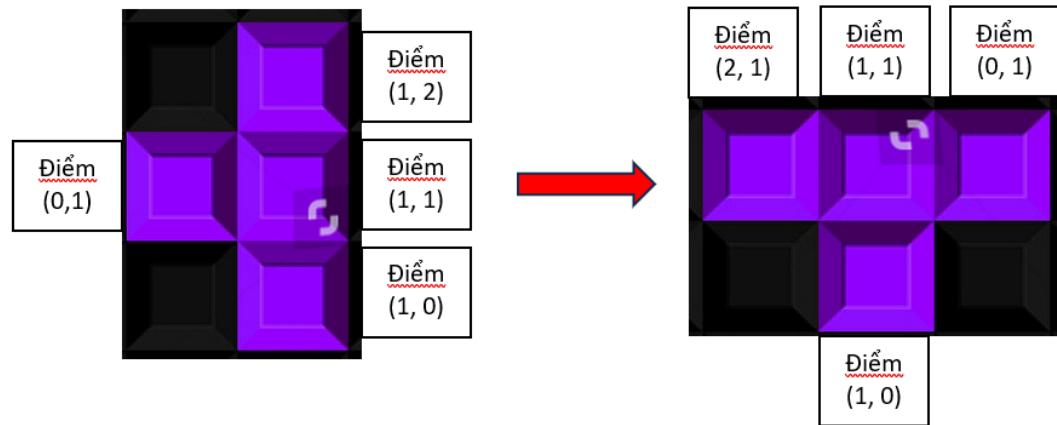
- `cellSet[0] = {0, 1}, {1, 1}, {2, 1}, {3, 1};`
- `cellSet[1] = {2, 0}, {2, 1}, {2, 2}, {2, 3};`
- `cellSet[2] = {0, 2}, {1, 2}, {2, 2}, {3, 2};`
- `cellSet[3] = {1, 0}, {1, 1}, {1, 2}, {1, 3};`



- Cập nhật gốc của Tetromino thành (2, 1).

Tương tự, ta có thuật toán xoay với khối T-blocks:

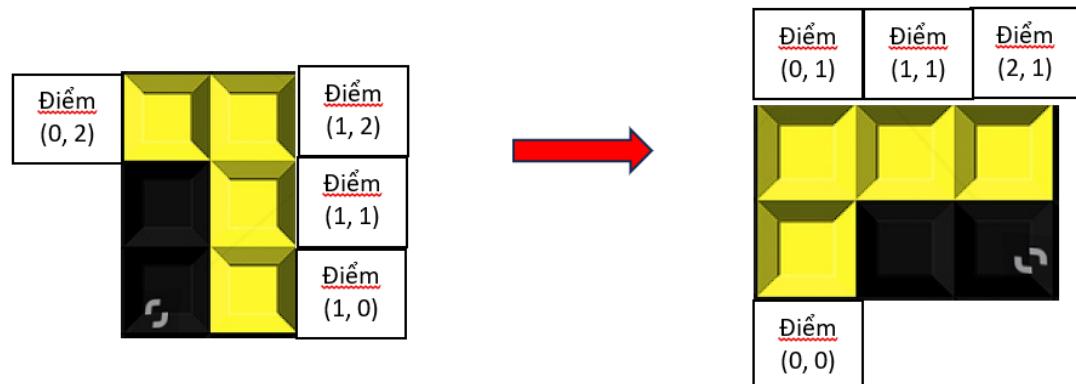
- $\text{cellSet}[0] = 0, 1, 1, 0, 1, 1, 1, 1, 2 ;$
- $\text{cellSet}[1] = 0, 1, 1, 0, 1, 1, 2, 1 ;$
- $\text{cellSet}[2] = 1, 0, 1, 1, 1, 2, 2, 1 ;$
- $\text{cellSet}[3] = 0, 1, 1, 1, 1, 2, 2, 1 ;$



Sau đó khối T cập nhật gốc của Tetromino thành (1, 0) khi xoay theo chiều kim đồng hồ.

Thuật toán xoay của khối L-blocks:

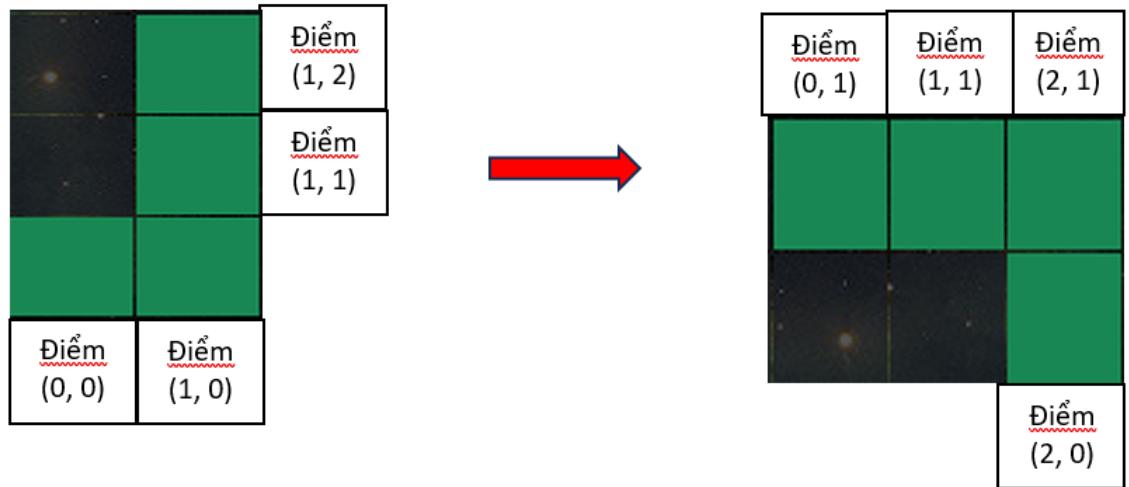
- $\text{cellSet}[0] = 0, 2, 1, 0, 1, 1, 1, 1, 2 ;$
- $\text{cellSet}[1] = 0, 0, 0, 1, 1, 1, 2, 1 ;$
- $\text{cellSet}[2] = 1, 0, 1, 1, 1, 2, 2, 0 ;$
- $\text{cellSet}[3] = 0, 1, 1, 1, 2, 1, 2, 2 ;$



Cập nhật gốc của Tetromino thành (0, 0).

Thuật toán xoay của khối J-blocks:

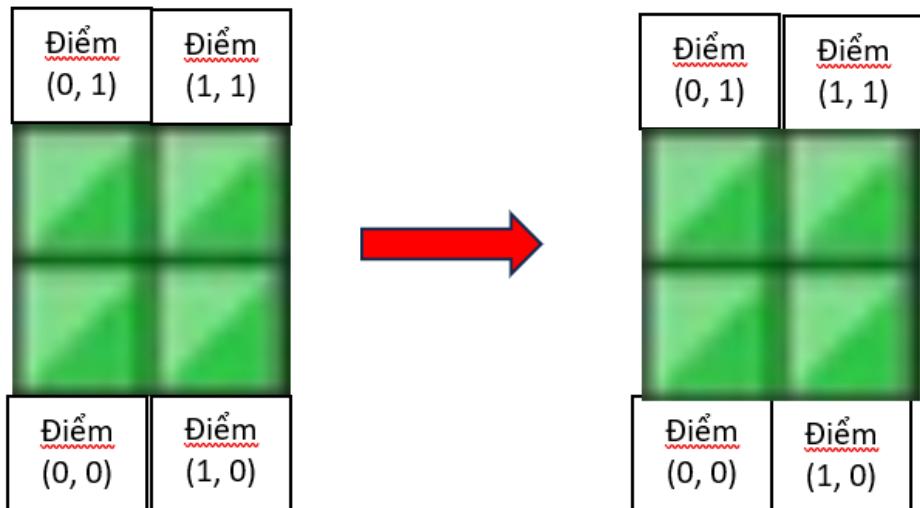
- $\text{cellSet}[0] = 0, 0, 1, 0, 1, 1, 1, 1, 2 ;$
- $\text{cellSet}[1] = 0, 1, 1, 1, 2, 0, 2, 1 ;$
- $\text{cellSet}[2] = 1, 0, 1, 1, 1, 2, 2, 2 ;$
- $\text{cellSet}[3] = 0, 1, 0, 2, 1, 1, 2, 1 ;$



Cập nhật gốc của Tetromino thành (2, 0).

Thuật toán xoay của khối O-blocks:

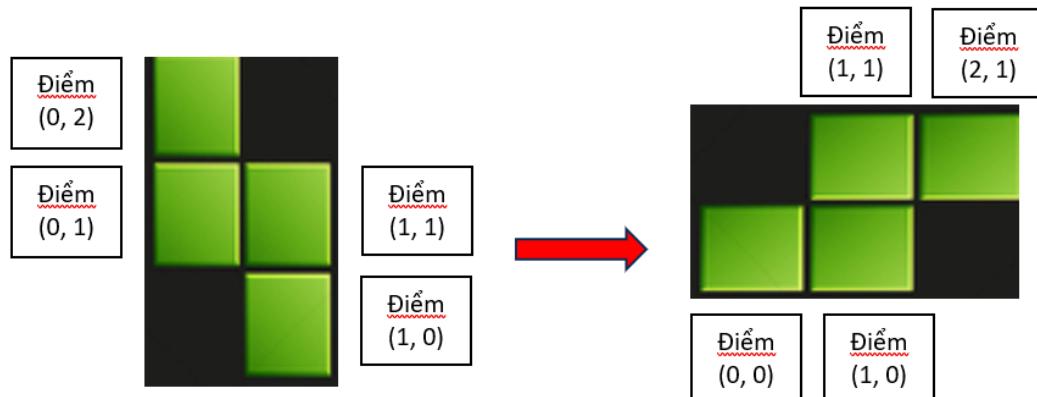
- $\text{cellSet}[0] = 0, 0, 0, 1, 1, 0, 1, 1 ;$
xoay thành khối $\text{cellSet}[1] = 0, 0, 0, 1, 1, 0, 1, 1 ;$



Cập nhật gốc của Tetromino thành (0, 0) nên gốc Tettromio không đổi.

Thuật toán xoay của khối S-blocks:

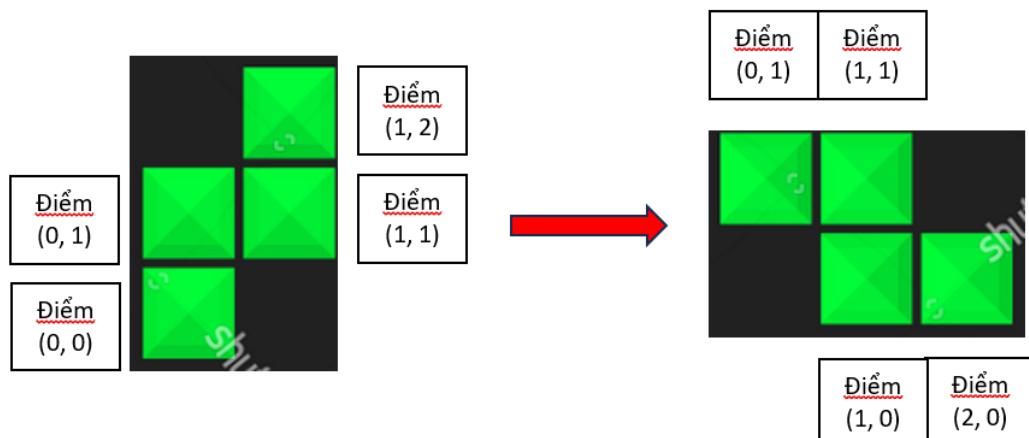
- $\text{cellSet}[0] = 0, 1, 0, 2, 1, 0, 1, 1 ;$
- $\text{cellSet}[1] = 0, 0, 1, 0, 1, 1, 2, 1 ;$
- $\text{cellSet}[2] = 1, 1, 1, 2, 2, 0, 2, 1 ;$
- $\text{cellSet}[3] = 0, 1, 1, 1, 1, 2, 2, 2 ;$



Cập nhật gốc của Tetromino thành (0, 0).

Thuật toán xoay của khối Z-blocks:

- $\text{cellSet}[0] = 0, 0, 0, 1, 1, 1, 1, 2 ;$
- $\text{cellSet}[1] = 0, 1, 1, 0, 1, 1, 2, 0 ;$
- $\text{cellSet}[2] = 1, 0, 1, 1, 2, 1, 2, 2 ;$
- $\text{cellSet}[3] = 0, 2, 1, 1, 1, 2, 2, 1 ;$



Cập nhật gốc của Tetromino thành (0, 1)

Lưu ý rằng trong hàm `rotateCCW`, sử dụng phép chia lấy dư `direction = (direction - 1) % 4` có thể tạo ra giá trị âm. Điều này có thể gây ra lỗi trong một số trường hợp. Thay vào đó, bạn có thể sử dụng biểu thức `(direction + 3) % 4` để đảm bảo giá trị hướng luôn là dương.

3.2.2 Class Board

Định nghĩa lớp Class Board: là một lớp đóng gói chức năng và quản lý trạng thái của bảng trò chơi Tetris.



Board có biến thành viên string được khai báo riêng tư là content, là một vector 2 chiều của các số nguyên, trong đó mỗi số nguyên đại diện cho loại khối Tetromino chiếm ô đó (0 cho rỗng).

Constructor:



- Khởi tạo một vector 2D có tên là **content** đại diện cho bảng trò chơi với chiều rộng (**width**) và chiều cao (**height**) được chỉ định.
- Mỗi ô ban đầu được đặt là 0.

Các Hàm Thành vien:

- render()**: Duyệt qua vector **content** để vẽ bảng trò chơi. Hàm này hiển thị bảng trên màn hình bằng hàm **DrawRectangle** của Raylib. Nó lặp qua từng ô và vẽ một hình chữ nhật với màu tương ứng dựa trên vector **content**.

Dưới đây là Pseudocode thể hiện chức năng render để vẽ bảng cho Tetris:

Algorithm 1 Render Board

```

1: for  $y \leftarrow 0$  content.size() - 1 do
2:   for  $x \leftarrow 0$  content[y].size() - 1 do
3:     DrawRectangle( $x \times \text{CELL\_SIZE} + 1, y \times \text{CELL\_SIZE} + 1, \text{CELL\_SIZE} - 1, \text{CELL\_SIZE} - 1, \text{COLORS}[\text{content}[y][x]]$ )

```

- Fill(Tetromino &tetromino)**: Có gắng đặt một Tetromino trên bảng dựa trên các ô hiện tại của nó. Nếu thành công, đặt các ô tương ứng trên bảng thành loại của Tetromino. Trả về true nếu diễn thành công và false nếu bất kỳ ô nào nằm ngoài ranh giới của bảng hoặc đã bị chiếm dụng.

Algorithm 2 Fill Tetromino on Board

```

1: function FILLTETROMINO(Board content, Tetromino tetromino)
2:   for all cell in tetromino.getCurrentCells() do
3:     content[cell.second][cell.first]  $\leftarrow$  tetromino.getType()
4:   // Fill all cells successfully
5:   return true out_of_range
6:   // Cell out of the board
7:   return false

```

- checkCollision(Tetromino &tetromino)**: Kiểm tra xem Tetromino đã cho có va chạm với bất kỳ khối hiện có nào trên bảng hay không. Nếu tìm thấy bất kỳ va chạm nào, trả về true.

Sau đây là Pseudocode thể hiện việc kiểm tra va chạm:

Algorithm 3 Check Collision with Tetromino

```

1: function CHECKCOLLISION(Tetromino tetromino)
2:   for all cell in tetromino.getCurrentCells() do
3:     if content[cell.second][cell.first]  $\neq$  0 then
4:       return true
5:     return false out_of_range
6:     // Cell out of the board
7:     return true

```

- ClearRow()**: Xác định và thu thập chỉ số của các dòng đã đầy đủ ô. Xóa các hàng đầy và chèn các hàng mới với số 0 ở đầu bảng. Trả về số hàng đã xóa.

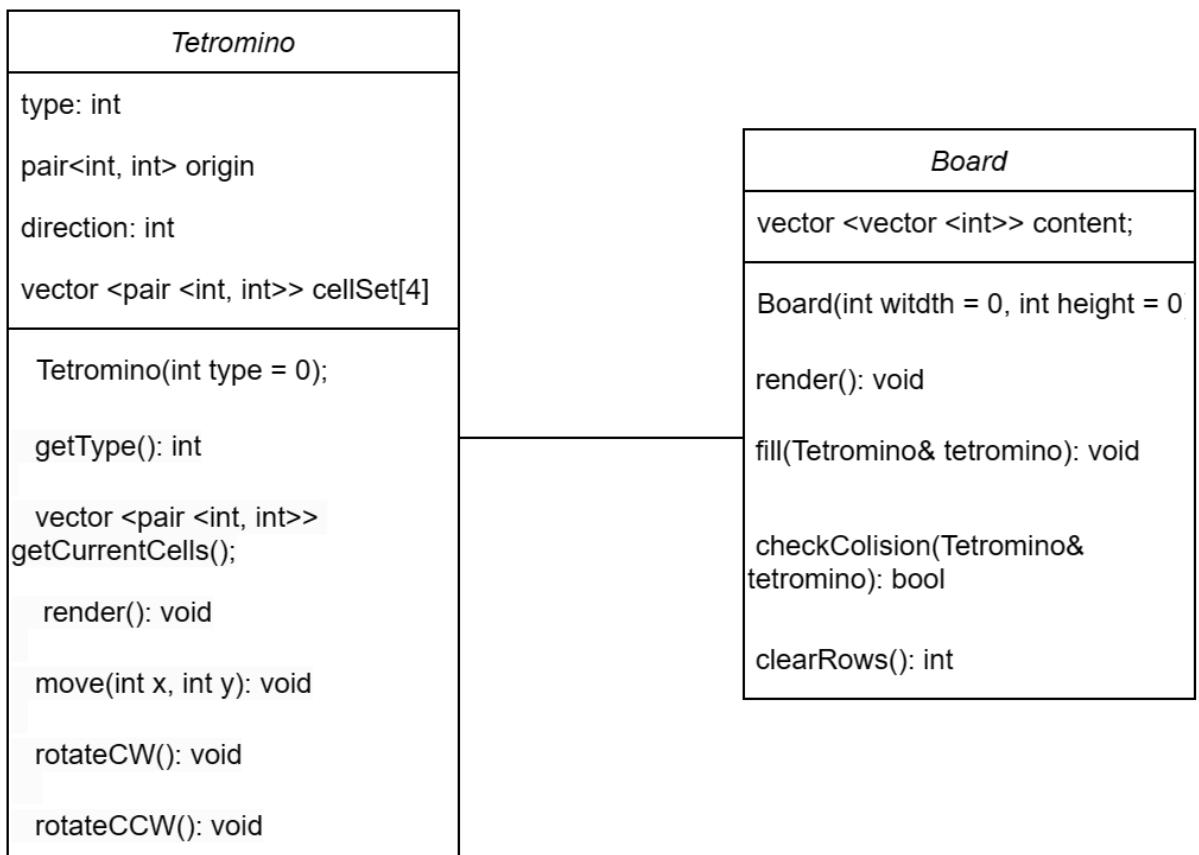
Algorithm 4 Clear Full Rows

```
1: function CLEARROWS
2:   vector<int> fullRows                                ▷ Vector to store indices of full rows
3:   for  $i \leftarrow 0$  content.size() - 1 do
4:     if find(content[i].begin(), content[i].end(), 0) == content[i].end() then ▷ Row is full of blocks
5:       fullRows.push_back(i)
6:   for all  $i \in$  fullRows do
7:     content.erase(content.begin() + i)
8:     content.insert(content.begin(), vector<int>(content[0].size(), 0))    ▷ Insert a new row at the
beginning
9:   return fullRows.size()
```

Tóm lại Class Board quản lý trạng thái của bảng trò chơi Tetris. Nó cung cấp các hàm để hiển thị bảng, đặt Tetrominos, phát hiện va chạm và xóa các hàng đã hoàn thành. Các thuật toán được sử dụng tương đối đơn giản:

- Vị trí: Để lắp đầy một Tetromino, bạn cần lặp qua các ô của nó và cập nhật các ô bảng tương ứng. Nếu bất kỳ ô nào nằm ngoài giới hạn hoặc đã bị chiếm dụng, việc đặt sẽ thất bại.
- Phát hiện va chạm: Kiểm tra va chạm chỉ đơn giản là kiểm tra xem bất kỳ ô Tetromino nào có trùng lặp với ô bảng không trùng hay không.
- Xóa hàng: Xác định các hàng đầy chỉ đơn giản là kiểm tra xem bất kỳ hàng nào có chứa chỉ các giá trị không bằng 0 (khối) hay không. Xóa các hàng đầy liên quan đến việc thao tác vector content để xóa và chèn hàng.

Cả 2 class **Board** và class **Tetromino** liên kết chặt chẽ với nhau để tạo nên các interface bảng biểu và thuật toán xoay khối, phát hiện va chạm và xóa hàng.



Class Board tương tác với class Tetromino theo nhiều cách:

- Nó nhận các đối tượng Tetromino và gọi các phương thức của chúng để truy cập thông tin như ô hiện tại và loại.
- Nó sử dụng các ô hiện tại của Tetromino để xác định các kích bản đặt và va chạm tiềm ẩn.
- Sau khi đặt một Tetromino, nó gọi hàm clearRows() để xóa bất kỳ hàng đã hoàn thành nào do việc đặt gây ra.

3.2.3 Class Game

Định nghĩa Lớp - Game:

Class Game: quản lý logic chơi Tetris, bao gồm xử lý điều khiển người chơi, rơi Tetromino, va chạm, đặt Tetromino, tính điểm, và hiển thị. **Thuộc tính:**

- board: Một đối tượng Board đại diện cho bảng Tetris với kích thước 10x20.
- current: Một đối tượng Tetromino biểu thị Tetromino hiện tại đang rơi.
- next: Một đối tượng Tetromino biểu thị Tetromino tiếp theo sẽ xuất hiện.
- score: Điểm của người chơi.
- tick: Thời gian giữa các lần rơi của Tetromino (tính bằng giây).
- lastTick: Thời gian thực tế của lần rơi cuối cùng.

Hàm Constructor:

Constructor khởi tạo một trò chơi mới. Nó tạo một bảng với kích thước 10x20, chọn một tetromino ngẫu nhiên cho trạng thái hiện tại (current) và tiếp theo (next). score được khởi tạo là 0, tick là tốc độ rơi của khối, và lastTick là thời điểm cuối cùng mà khối đã rơi.

Các Hàm Thành viên:

Hàm action() - Xử lý điều khiển người chơi:

- Kiểm tra phím bấm: Hàm lấy phím được bấm và lưu vào biến key.
- Lưu trạng thái Tetromino: Hàm lưu trữ trạng thái hiện tại của current trước khi thực hiện điều khiển.
- Xử lý điều khiển:
 - Phím UP: Xoay Tetromino theo chiều kim đồng hồ.
 - Phím LEFT: Di chuyển Tetromino 1 ô sang trái.
 - Phím RIGHT: Di chuyển Tetromino 1 ô sang phải.
 - Phím DOWN: Di chuyển Tetromino 1 ô xuống dưới.
 - Phím SPACE: Thả rơi Tetromino tự do cho đến khi va chạm.
- Kiểm tra va chạm: Sau khi thực hiện điều khiển, hàm kiểm tra xem Tetromino có va chạm với các khối khác trên bảng hay không.
- Xử lý va chạm: Nếu va chạm, hàm khôi phục Tetromino về trạng thái trước đó.

Hàm fall() - Xử lý rơi Tetromino:

- Kiểm tra thời gian: Hàm lấy thời gian hiện tại và so sánh với thời điểm rơi cuối cùng (lastTick).

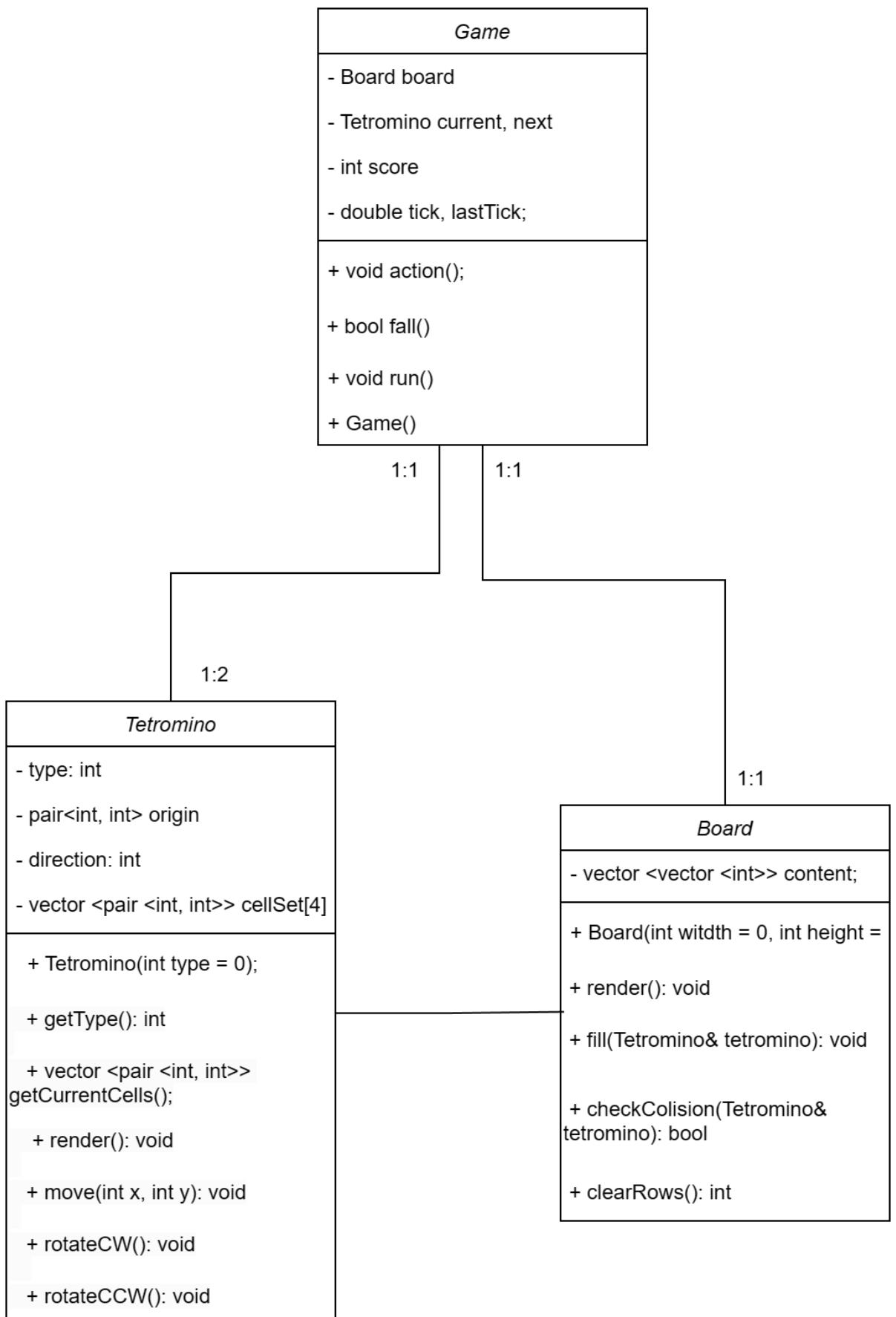
- (b) Rơi Tetromino: Nếu đủ thời gian giữa các lần rơi, hàm di chuyển current xuống 1 ô.
- (c) Kiểm tra va chạm khi rơi: Hàm kiểm tra xem Tetromino có va chạm với các khối khác trên bảng sau khi rơi hay không.
- (d) Xử lý va chạm khi rơi:
 - i. Nếu va chạm, hàm di chuyển Tetromino lên 1 ô để trở lại trạng thái trước khi rơi.
 - ii. Hàm kiểm tra khả năng đặt Tetromino lên bảng.
 - iii. Nếu không đặt được, hàm trả về true báo hiệu game over.
 - iv. Nếu đặt được, hàm cộng điểm dựa trên số hàng xóa, thay thế current bằng next, tạo Tetromino mới cho next, và cập nhật thời điểm rơi cuối cùng (lastTick).
- (e) Trả về false nếu không game over.

Hàm run() - Vòng lặp chính của game: Vòng lặp cho đến khi cửa sổ đóng:

- Gọi hàm action() để xử lý điều khiển người chơi.
- Gọi hàm fall() để cập nhật vị trí và trạng thái của Tetromino.
- Kiểm tra kết quả của fall(): Nếu game over, thoát khỏi vòng lặp.
- Bắt đầu vẽ:
 - Xóa màn hình.
 - Vẽ bảng.
 - Vẽ Tetromino hiện tại.
 - Kết thúc vẽ.

Class Game có mối liên hệ chặt chẽ với các class Board và Tetromino. Class Game sử dụng các class Board và Tetromino để quản lý logic chơi Tetris.

Sau đây là sơ đồ thể hiện mối quan hệ giữa 3 class chính Game, Board và Tetromino:



Cụ thể, class Game có các mối liên hệ sau với các class Board và Tetromino:

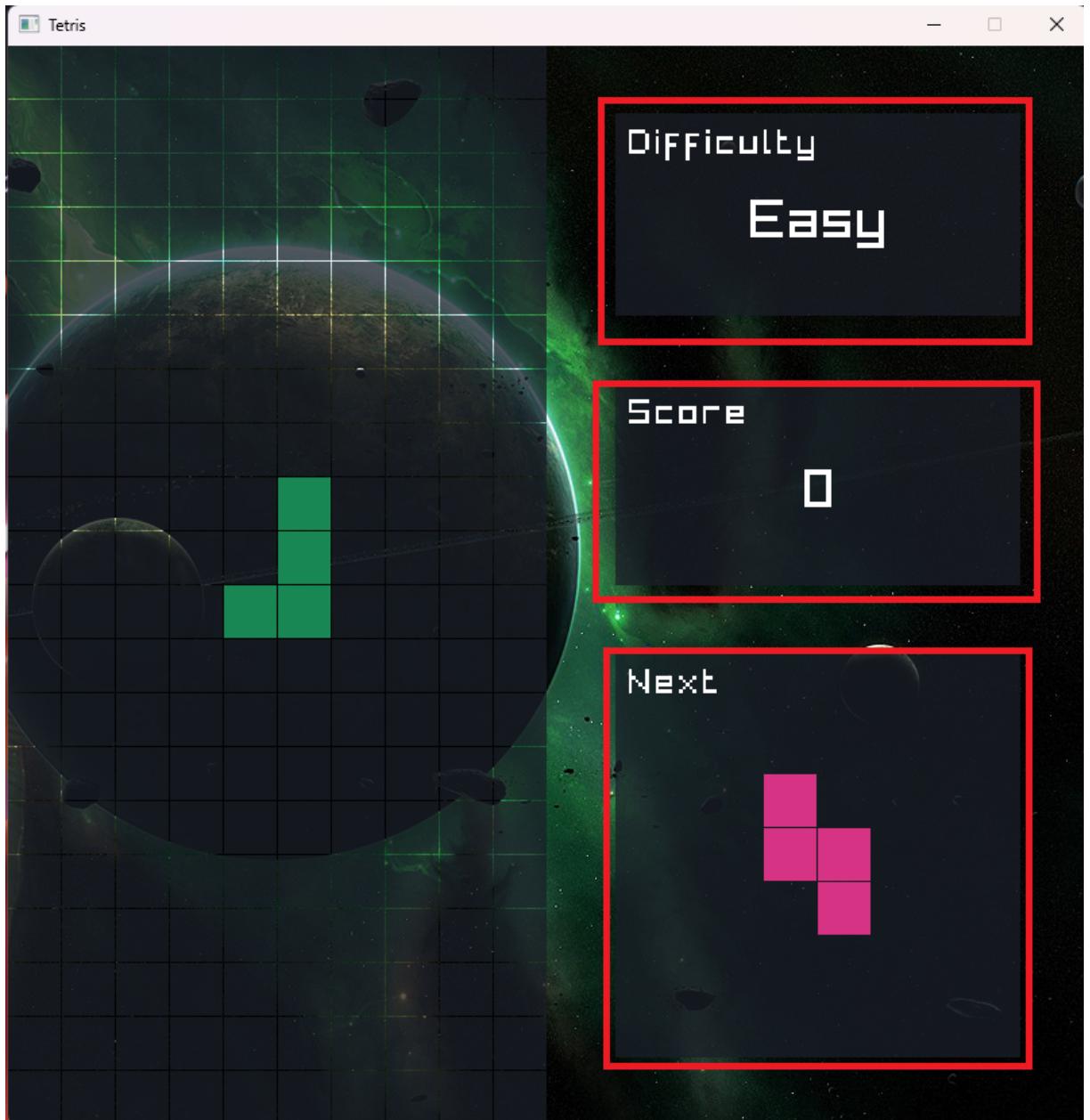
- Thuộc tính:
 - Class Game có các thuộc tính board, current, và next là các đối tượng Board, Tetromino, và Tetromino.
- Hàm:
 - *Class Game sử dụng các hàm của class Board để:*
 - * Xóa các hàng đã hoàn thành (board.clearRows()).
 - * Kiểm tra va chạm (board.checkCollision()).
 - * Đặt Tetromino (board.fill()).
 - *Class Game sử dụng các hàm của class Tetromino để:*
 - * Xoay Tetromino (tetromino.rotateCW()).
 - * Di chuyển Tetromino (tetromino.move()).
- Vòng lặp chính:
 - Trong vòng lặp chính của game, class Game gọi các hàm của class Board và Tetromino để cập nhật trạng thái của game.

3.3 Game Mechanics

Game Tetris là một game giải đố xếp khối, trong đó người chơi cố gắng xếp các khối hình chữ nhật (Tetromino) thành hàng hoàn chỉnh để chúng biến mất. Game có 7 loại Tetromino khác nhau, mỗi loại có hình dạng và kích thước khác nhau. Tetromino mới xuất hiện ở đỉnh màn hình và rơi xuống màn hình theo từng ô. Người chơi có thể di chuyển, xoay, hoặc thả rơi Tetromino để cố gắng xếp chúng thành hàng hoàn chỉnh. Nếu một hàng hoàn chỉnh được tạo ra, hàng đó sẽ biến mất và người chơi sẽ nhận được điểm. Nếu Tetromino chạm đến đỉnh màn hình mà không tạo được hàng hoàn chỉnh, game sẽ kết thúc.

3.4 User Interface

Cấu trúc cơ bản:



- **Bảng Tetris:** Hiển thị chính chiếm phần lớn màn hình, với các ô lưới có kích thước tùy chỉnh theo $CELLSIZE$ (mcOnh40px). *Tetromino hinti : Hinht Tetromino Ôngриенхң, chongichibithnhdngspØt.*
- **Tetromino tiếp theo:** Hiển thị hình dạng Tetromino tiếp theo sẽ xuất hiện, cho phép người chơi lập kế hoạch trước, đồng thời kết hợp với sử dụng phím Space để tăng tốc độ hoàn thành.
- **Điểm số và mức độ:** Hiển thị điểm số hiện tại và mức độ chơi, cung cấp phản hồi về tiến trình.
- **Nút điều khiển:** Nút ảo hoặc phím tắt để xoay Tetromino, di chuyển sang trái/phải, và thả Space nhanh xuống.
- **Tùy chọn âm thanh:** Cho phép người chơi bật/tắt nhạc nền và hiệu ứng âm thanh.

Nâng cao giao diện:

- **Background:** Background đem lại cảm giác mới mẻ cho người dùng, sử dụng hàm `DrawTexture()` và `UnloadTexture()`; của thư viện Raylib.h để vẽ.

- Màn hình game over: Hiển thị điểm số cuối cùng và nút "Chơi lại" hoặc "Thoát" khi game over. Bằng việc sử dụng hàm MeasureTextEx() và DrawTextEx()

Giao diện mạnh mẽ không chỉ về mặt thẩm mỹ mà còn về mặt chức năng:

- Diều khiển mượt mà: Phản ứng nhanh chóng và chính xác với thao tác của người chơi.
- Cân bằng độ khó: Tăng dần độ khó theo mức chơi, tạo thách thức vừa phải và thú vị.
- Hướng dẫn và trợ giúp: Cung cấp hướng dẫn cơ bản về cách chơi và các phím điều khiển.
- Hỗ trợ đa ngôn ngữ: Bản địa hóa giao diện và hướng dẫn để phục vụ người chơi quốc tế.
- Hiệu suất: Giao diện, âm thanh không ảnh hưởng đến tốc độ và độ mượt của trò chơi.
- Tính đơn giản: Giao diện trực quan và dễ hiểu, không gây rối cho người chơi.
- Khả năng tùy chỉnh: Cho phép người chơi tùy chỉnh một số yếu tố giao diện theo sở thích.

4 Program Executing Instruction

Game Tetris được thực thi theo các bước sau:

(a) **Khởi tạo game**

Ở bước này, game sẽ khởi tạo các đối tượng cần thiết, bao gồm:

- i. Một đối tượng Board đại diện cho bảng Tetris
- ii. Một đối tượng Tetromino đại diện cho Tetromino hiện tại đang rơi
- iii. Một đối tượng Tetromino đại diện cho Tetromino tiếp theo sẽ xuất hiện
- iv. Điểm của người chơi
- v. Thời gian giữa các lần rơi của Tetromino
- vi. Thời điểm rơi cuối cùng của Tetromino

(b) **Vòng lặp chính của game** Vòng lặp chính của game sẽ lặp lại cho đến khi game kết thúc.

Trong vòng lặp này, game sẽ thực hiện các nhiệm vụ sau:

- i. Xử lý điều khiển người chơi
- ii. Rơi Tetromino
- iii. Va chạm
- iv. Đặt Tetromino
- v. Tính điểm
- vi. Hiển thị

(c) **Xử lý điều khiển người chơi** Người chơi có thể điều khiển Tetromino bằng các phím sau:

- i. Phím UP: Xoay Tetromino theo chiều kim đồng hồ
- ii. Phím LEFT: Di chuyển Tetromino 1 ô sang trái
- iii. Phím RIGHT: Di chuyển Tetromino 1 ô sang phải
- iv. Phím DOWN: Di chuyển Tetromino 1 ô xuống dưới
- v. Phím SPACE: Thả rơi Tetromino tự do cho đến khi va chạm

(d) **Rơi Tetromino**

Tetromino sẽ rơi xuống màn hình theo từng ô. Thời gian giữa các lần rơi của Tetromino có thể được điều chỉnh bằng tham số tick.

(e) **Va chạm**

Nếu Tetromino va chạm với các khối khác trên bảng, Tetromino sẽ dừng lại và game sẽ kiểm tra xem Tetromino có thể được đặt lên bảng hay không. Nếu không thể đặt, game sẽ kết thúc.

(f) **Đặt Tetromino**

Nếu Tetromino không va chạm với các khối khác trên bảng, Tetromino sẽ được đặt lên bảng. Các hàng hoàn chỉnh sẽ biến mất và người chơi sẽ nhận được điểm.

(g) **Tính điểm**

Số điểm mà người chơi nhận được sẽ được tính dựa trên số hàng hoàn chỉnh. Mỗi hàng hoàn chỉnh sẽ mang lại cho người chơi 100 điểm.

(h) **Hiển thị**

Game sẽ hiển thị bảng Tetris và Tetromino lên màn hình.

(i) **Kết thúc**

Game sẽ hiển thị điểm số cuối cùng sau khi trò chơi kết thúc (khi block cuối cùng chạm trần cao nhất)

5 Conclusion

Trong bài báo cáo này, nhóm đã giới thiệu cách triển khai chương trình Tetris bằng ngôn ngữ lập trình C++ và thư viện Raylib. Chương trình có đầy đủ các chức năng cơ bản của Tetris, bao gồm:

- Chọn độ khó
- Xoay khối Tetromino
- Di chuyển khối Tetromino
- Thả khối Tetromino
- Xóa hàng
- Tính điểm
- Hiển thị giao diện người dùng

Để triển khai chương trình, nhóm đã sử dụng IDE Visual Studio 2022 và thư viện Raylib.h. Nhóm đã cung cấp một số hình ảnh đặc trưng thể hiện các chức năng, các khối Tetromino và giao diện người dùng.

Ở phần 2, nhóm đã đi sâu phân tích chi tiết thuật toán xoay của các khối Tetromino, thuật toán va chạm hay thuật toán xóa hàng, thuật toán rơi sử dụng thư viện Raylib.h. Nhóm đã sử dụng hình ảnh, mã giả và trình bày để thể hiện các chức năng quan trọng của game, giúp chương trình trực quan và dễ sử dụng, bảo trì.

Cuối cùng, là phần thực thi chương trình, bao gồm các yếu tố logic, luồng chạy và các hướng dẫn hoàn thiện chương trình.

Nhìn chung, chương trình Tetris được triển khai trong bài báo cáo này có đầy đủ các chức năng cơ bản và đáp ứng các yêu cầu đề bài. Chương trình có giao diện trực quan, dễ sử dụng và có thể được mở rộng thêm các chức năng khác.

Một số hướng dẫn hoàn thiện chương trình có thể được thực hiện như sau:

- Thêm các chế độ chơi khác nhau, chẳng hạn như chế độ nhiều người chơi, chế độ thời gian, chế độ thách thức,...
- Thêm các khối Tetromino mới với các hình dạng và khả năng khác nhau.
- Thêm các hiệu ứng âm thanh và hình ảnh để tăng tính hấp dẫn của game.

Nhóm hy vọng bài báo cáo này sẽ là một tài liệu tham khảo hữu ích cho các bạn sinh viên.

A Reference

- YouTube Video Reference <https://github.com/Louis2602/Pikachu-Game>
- ChatGPT
- InGame Assets:
 - raylib::Sound Class Reference https://robloach.github.io/raylib-cpp/classraylib_1_1_sound.html