TRƯỜNG ĐẠI HỌC MỞ THÀNH PHỐ HỒ CHÍ MINH

KHOA CÔNG NGHỆ THÔNG TIN

XÂY DỰNG CHƯƠNG TRÌNH GAME XẾP GẠCH

BÁO CÁO BÀI TẬP LỚN

Ngành: Đò họa máy tính

Lớp: DH61TH02

NGUYỄN ANH KHÔI

TĂNG HÁN LUÔNG

TRƯƠNG CÔNG HOÀN VŨ

THÀNH PHỐ HỒ CHÍ MINH – 06/2020

TRƯỜNG ĐẠI HỌC MỞ THÀNH PHỐ HỒ CHÍ MINH

KHOA CÔNG NGHỆ THÔNG TIN

XÂY DỰNG CHƯƠNG TRÌNH GAME XẾP GẠCH

BÁO CÁO BÀI TẬP LỚN

Ngành: Đò họa máy tính

Lớp: DH61TH02

NGUYỄN ANH KHÔI - 1651010083

TĂNG HÁN LUÔNG – 1651012108

TRƯƠNG CÔNG HOÀN VŨ – 1651010169

Giảng viên giảng dạy: **ThS. Võ Thị Hồng Tuyết**

THÀNH PHỐ HỒ CHÍ MINH – 06/2020

Contents

[**1.** **Giới thiệu** 4](#_Toc42027933)

[**1.1.** **Giới thiệu đề tài** 4](#_Toc42027934)

[**1.2.** **Mục đích đề tài** 4](#_Toc42027935)

[**1.3.** **Giới hạn đề tài** 4](#_Toc42027936)

[**2.** **Xây dựng Tetris qua graphics.h** 5](#_Toc42027937)

[**2.1.** **Tổng quan** 5](#_Toc42027938)

[**2.2.** **Các kiến thức cơ bản** 5](#_Toc42027939)

[**2.3.** **Tạo hình & vẽ hình** 6](#_Toc42027940)

[**2.4.** **Class Blocks** 7](#_Toc42027941)

[**2.5.** **TBlock, IBlock, OBlock, LBlock, JBlock, SBlock và ZBlock** 8](#_Toc42027942)

[**2.6.** **Bảng** 9](#_Toc42027943)

[**2.7.** **Các class khác:** 9](#_Toc42027944)

[**2.8.** **Engine - nơi mà trò chơi được hoàn chỉnh** 10](#_Toc42027945)

[**2.9.** **Main - nơi chạy game** 15](#_Toc42027946)

[**3.** **Kết luận** 17](#_Toc42027947)

**Danh mục hình ảnh:**

Hình dạng của một ô gạch 6

Các hình dạng của 7 nhóm gạch, với hai màu L R tượng trưng cho những gì hai mảng left và right chứa. 7

Hình mẫu của bảng game, với ô vuông màu xanh dương là khu vực xuất hiện gạch. 9

Minh họa cho logic tính khả năng. 11

Minh họa cho 3 hàm kiểm tra phục vụ cho việc di chuyển. 12

Minh họa việc kiểm tra xoay qua việc kiểm tra phiên bản cạnh bên của nhóm gạch. 14

Kết quả giao diện của trò chơi. 17

1. **Giới thiệu**
   1. **Giới thiệu đề tài**

Tetris, một trò chơi đơn giản nhưng đòi hỏi trí tuệ và khả năng người chơi, lần đầu tiên được tạo ra vào năm 1984 bởi kĩ sư phần mềm Alexey Pajitnov, và được phát hành bởi nhiều công ti vào cuối những năm 1980. Sau khi được phát hành trong một khoảng thời gian đáng kể bởi Nintendo, các quyền được trao lại cho Pajitnov vào năm 1996, người đồng sáng lập Công ty Tetris với Henk Rogers để quản lý cấp phép Tetris.

Vì tính chất đòi hỏi trí tuệ và khả năng phản xạ từ người chơi, Tetris đã trở thành một trò chơi rất nổi tiếng và hứng thú, vào tháng 12 năm 2011, nó trở thành tựa game bán chạy nhất mọi thời đại, bản cho máy Game Boy nói riêng cũng đã trở thành tựa game bán chạy nhất mọi thời đại. Trò chơi này có sẵn trên hơn 65 nền tảng, đặt kỉ lục Guiness về tựa game hỗ trợ nhiều nhất. Tetris đã bắt nguồn một nền văn hóa phổ biến và văn hóa của nó vượt ra ngoài phạm vi là một trò chơi, hình ảnh của người chơi đã ảnh hưởng đến cả kiến trúc, âm nhạc và cosplay. Trò chơi cũng đã là một chủ đè cho nhiều nghiên cứu khác nhau đã cho thấy tác động của nó lên não người sau một phiên chơi Tetris, mà sau này được đặt tên là Tetris Effect.

* 1. **Mục đích đề tài**

Sử dụng các kiến thức đã học từ môn Đồ họa máy tính và thư viện graphics.h, và các kiến thức lập trình bổ sung khác vào việc tạo game Tetris. Từ đó, kết luận sự cần thiết của đồ họa nói chung và thư viện graphics.h nói riêng trong việc lập trình trò chơi.

* 1. **Giới hạn đề tài**

Giải thích quá trình xây dựng trò chơi chi tiết trong việc làm việc với graphics.h và các kiến thức cơ bản trong lập trình trong ngôn ngữ C++ đã học. Các kiến thức bổ sung khác sẽ được giải thích sơ lược.

1. **Xây dựng Tetris qua graphics.h**
   1. **Tổng quan**

Trò chơi xếp gạch, hoặc có thể được gọi với tên gốc là Tetris, là kiểu trò chơi ở dạng bảng.

Người chơi sẽ được điều khiển một kiểu gạch ngẫu nhiên, họ có quyền đưa nó qua trái hoặc phải, xoay gạch, cho rớt xuống nhanh hơn, hoặc cho rớt thẳng xuống đáy của bảng hoặc nơi tiếp xúc. Người chơi cũng có quyền “giữ gạch”, nghĩa là gạch sẽ được tạm giữ và chuyển sang gạch khác để điều khiển, gạch tạm này có thể được chuyển lại (đổi chỗ) với gạch đang điều khiển nếu có nhu cầu, nhưng chỉ được quyền đổi 1 lần và có thể đổi tiếp sau khi nhóm gạch họ điều khiển đã chạm đáy và chuyển sang điều khiển nhóm gạch mới.

Gạch đồng thời cũng sẽ tự động di chuyển xuống 1 nấc sau 1 khoảng thời gian nhất định, và tốc độ rơi tự động của nó sẽ tùy theo màn chơi.

Khi gạch chạm đáy hoặc vật tiếp xúc, người chơi có một khoảng thời gian ngắn để ra quyết định di chuyển và xoay nó sao cho được vị trí phù hợp của họ. Sau khoảng thời gian đó, người chơi sẽ mất kiểm soát nhóm gạch đó, và sẽ được điều khiển 1 nhóm gạch mới xuất hiện ở trên đầu bảng.

Nếu các viên gạch sắp xếp được một dòng trên bảng, thì bảng sẽ xóa đi dòng đó và di chuyển các hàng gạch bên trên xuống.

* 1. **Các kiến thức cơ bản**

Hiểu rõ về các cơ bản trong lập trình C++, biết sử dụng tốt thư viện graphics.h và sự sáng tạo và tư duy lập trình.

* 1. **Tạo hình & vẽ hình**

Game Tetris thường có 7 kiểu hình dạng gạch cổ điển có tên là kiểu gạch T, I, O, L, J, S và Z.

Mỗi kiểu gạch này thật ra chỉ là một nhóm gạch chứa 4 ô gạch nhỏ ghép vào thành 7 kiểu hình dạng khác nhau.

Với mỗi ô vuông nhỏ được vẽ theo một hàm mà nó sẽ vẽ ra một ô vuông và thùy theo người lập trình, có thể thiết kế ngoại hình của nó. Trong bài tập lớn này, ô vuông sẽ có 2 gạch chéo như hình bên dưới.



*Hình dạng của một ô gạch*

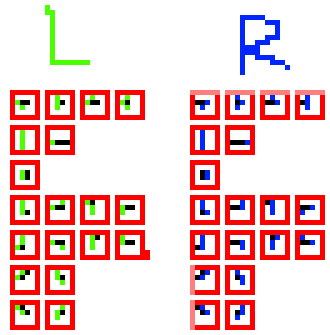
Quay lại các hình dạng gạch, nhắc lại việc một nhóm gạch chứa 4 ô gạch nhỏ ghép vào thành các hình dạng khác nhau. Cơ bản là hàm vẽ ô gạch được thực hiện lặp lại 4 lần, khi mà hình dáng gạch được hình thành.

Vậy thì làm thế nào để vẽ?

Đầu tiên, ta có các class Blocks và TBlock, IBlock, OBlock, LBlock, JBlock, SBlock và ZBlock, với Blocks là class cha của 7 class còn lại.

Sau đó, ta có một Struct có tên là Formation để chứa các mảng 4x4 phục vụ cho việc định hình cho các kiểu gạch này để cho việc vẽ ô gạch tiện lợi hơn.

Formation bao gồm 3 mảng, một mảng chính formation phục vụ cho việc tạo hình dạng nhóm gạch, trong khi 2 mảng left và right phục vụ cho xác định cạnh trái và cạnh phải của các nhóm gạch, sẽ cần thiết cho việc xử lý xoay hình sẽ được miêu tả sau.



*Các hình dạng của 7 nhóm gạch, với hai màu L R tượng trưng cho những gì hai mảng left và right chứa*

Mảng kiểu struct tự viết Formation sẽ chứa 4 phiên bản khác nhau của nhóm gạch, tương đương với 4 góc quay. Nó bao gồm sẽ chứa 1 khóa đại diện cho góc quay hiện tại và 1 khóa đại diện cho số phiên bản xoay khác nhau của các nhóm gạch (ví dụ như gạch T sẽ có 4 góc quay, trong khi gạch I chỉ có 2 góc quay, vì 2 góc quay còn lại sẽ như 2 góc quay trước).

x, y là vị trí của nhóm gạch, sẽ cần thiết cho việc định vị trí gạch trong bảng.

chance sẽ phục vụ trong việc kiểm soát khả năng xuất hiện gạch, sẽ được miêu tả sau.

* 1. **Class Blocks**

Là class cha chứa 7 kiểu nhóm gạch khác nhau, nó đồng thời cũng chứa các phương thức tương tác với nó ở tư cách là giao diện.

Khi khởi tạo, hai vòng lặp sẽ chạy, duyệt cho tất cả khung mảng hai chiều cho cả bốn góc xoay, cho mọi giá trị trong mảng là 0, thay vì là một giá trị rỗng.

Vị trí x và y khi khởi tạo sẽ được định tại vị trí hằng số, đó là vị trí xuất hiện gạch (và rồi cho gạch rớt xuống và người chơi điều khiển gạch). Sẽ có 1 khóa Chance, tương tự cho việc xác định khả năng hiện gạch, sẽ có giá trị mặc định là 2, sẽ được giải thích sau.

Hai hàm getNextRotate() và getPrevRotate() sẽ được dùng cho việc xoay gạch, nó sẽ trả về mảng kế tiếp hoặc trước đó xét từ phiên bản góc xoay hiện tại.

Hàm xoay gạch cơ bản sẽ chuyển khóa góc xoay hiện tại tiến lên phiên bản góc xoay sau hoặc lùi về bản góc xoay trước, xóa gạch ở giao diện tại vị trí hiện tại của nó, rồi vẽ lại nó ở dạng góc xoay mới.

Các hàm di chuyển cơ bản chỉ là các hàm thay đổi tiến lùi 2 giá trị x và y đã đặt trước khi khởi tạo, sau đó hàm thực hiện xóa và vẽ lại nhóm gạch ở vị trí mới.

Hàm xóa gạch chỉ đơn thuần là xóa gạch ở mặt giao diện, không ảnh hưởng đến mảng đằng sau nó (mảng của bảng), cần thiết cho việc đảm bảo gạch bị bôi trước khi thay đổi hình dạng hoặc/và vị trí của nó, tránh trường hợp giao diện bị vẽ thừa ô gạch.

* 1. **TBlock, IBlock, OBlock, LBlock, JBlock, SBlock và ZBlock**

Từ Class Blocks cha ta sinh ra được 7 class con TBlock, IBlock, OBlock, LBlock, JBlock, SBlock và Zblock.

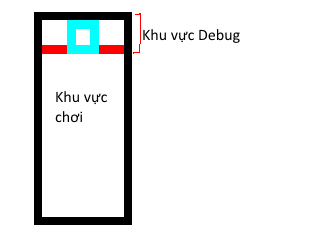
Mỗi hàm này đều chỉ có một hàm khởi tạo khác nhau.

Khi khởi tạo, các ô gạch sẽ được đặt vào mảng gạch với giá trị 1, 2 tương đương với đáy của nhóm gạch, rất cần thiết cho việc di chuyển gạch xuống và kiểm tra xem nếu có tiếp xúc với bề mặt hoặc ô gạch khác ở bên dưới.

* 1. **Bảng**

Bảng trong Tetris thật ra là một mảng hai chiều lớn, các nhóm gạch sẽ được tương tác trong bảng. Bảng này đóng vai trò lớn trong việc tạo khung chơi cho Tetris.

Trong Tetris cổ điển, một mảng có diện tích (tính theo ô vuông) là 10 x 20. Trong môi trường lập trình, người lập trình game thường cài đặt cho bảng có kích cỡ là 10 x 24, để có khoảng trống để hiển thị gạch mà người chơi điều khiển, trong khi diện tích chính vẫn là phần 10 x 20 bên dưới.



*Hình mẫu của bảng game, với ô vuông màu xanh dương là khu vực xuất hiện gạch.*

Ngoài ra thì có các hàm cơ bản khác cho bảng như vẽ khung bảng và vẽ theo giả trị hiện có trong mảng bảng (nếu phần tử mảng lớn hơn 0 thì sẽ vẽ 1 ô gạch tại vị trí của phần tử đó trên giao diện đồ họa).

* 1. **Các class khác:**

Class Timer: class chính trong việc tự động hóa trò chơi, là class chính cho việc giúp cho gạch điều khiển tự động rơi xuống theo thời gian.

Class Score: là class hỗ trợ cho việc tính điểm số của người chơi. Class này có sử dụng tính năng ghi và đọc file, phục vụ cho việc lưu điểm số cao nhất.

* 1. **Engine - nơi mà trò chơi được hoàn chỉnh**

Class Engine này là nơi kết nối, ghép mọi class thành phần đã lược qua lại với nhau, rồi tạo nên một trò chơi hoàn chỉnh có thể chơi được.

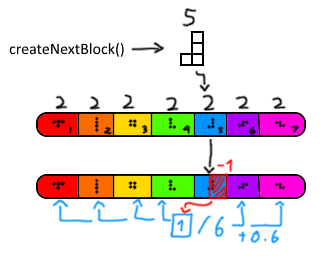
Class này đồng thời là nơi mà các class thành phần tương tác với nhau, sắp xếp với nhau thành từng bước để trò chơi được hoạt động.

Khi khởi tạo Engine, nó sẽ thiết lập trước các thiết lập của trò chơi,

class sẽ thiết lập trc giá trị mặc định cho các khóa giữ gạch, mốc thời gian 1 và khoảng delay và penaltyTick (cần thiết cho làm việc với class Timer và tự động hóa trò chơi), màn chơi, số dòng đã xóa, số dòng cần thiết để lên màn tiếp theo, chain (dùng cho việc tính điểm, nhân điểm cộng khi số dòng được xóa liên tục mỗi lần đặt gạch), khóa isPaused và isGameOver, khởi tạo hết 7 mẫu gạch, tính khả năng hiện gạch (sẽ được giải thích sau), vẽ bảng chơi và các bảng giao diện khác cần thiết, rồi tạo gạch

Hàm CreateNextBlock() là hàm khởi tạo trước nhóm gạch tiếp theo, hàm này đồng thời sẽ xử lý khả năng xuất hiện số gạch bên trong.

Hàm ChanceController() và CalculateChance() sẽ tính toán khả năng xuất hiện của từng nhóm gạch, hạn chế việc cùng một nhóm gạch xuất hiện liên tục và đảm bảo tất cả từng nhóm gạch có thể xuất hiện tuần tự. Thuật toán tính khả năng này hoạt động như sau: game sẽ nhớ số thứ tự của nhóm gạch vừa xuất hiện, lấy đi một phần khả năng của nó, và chia đều cho các phần gạch còn lại.



*Minh họa cho logic tính khả năng.*

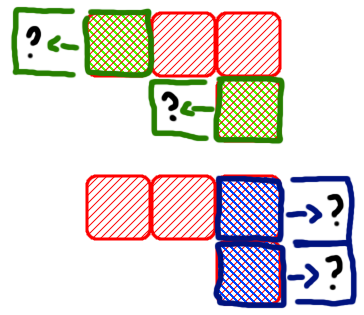
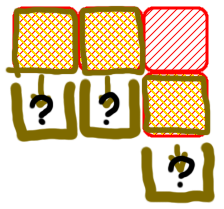
Một hàm CreateBlock() là hàm tạo nhóm gạch có thể điều khiển chính của game, nó sẽ lấy nhóm gạch tiếp theo đã tạo từ CreateNextBlock(), vẽ nó trên bảng game ở dạng gạch có thể điều khiển được, chạy lại hàm tạo nhóm gạch tiếp theo, rồi gán nhóm gạch có thể điều khiển được vào mảng bảng.

Hàm GameStart() là hàm bắt đầu trò chơi. Hàm này đơn giản chỉ khởi tạo cho viên gạch điều khiển được đầu tiên được xuất hiện, và rồi vẽ các giao diện trò chơi khác như số điểm, số dòng và số màn chơi

Hàm RemoveOldPos() và AddNewPos() là hai hàm thực hiện việc cập nhật vị trí của gạch đang được đối chiếu lên mảng bảng, hoạt động cơ bản là xóa vị trí cũ của gạch trên mảng bảng rồi đối chiếu lại nó, thêm nó vào mảng bảng.

Hàm CheckAbove(), CheckGround() và CheckBorder() là ba hàm kiểm tra 4 hướng của gạch xem có bề mặt bảng hoặc vật tiếp xúc hay không.

Hàm CheckAbove() và CheckGround() sẽ kiểm tra theo mảng formation thường của nhóm gạch, với mảng có giá trị là 2 sẽ được CheckGround() dựa theo mà kiểm tra. Hàm CheckBorder() sẽ phụ thuộc theo hai hàm phụ left và right để kiểm tra.



*Minh họa cho 3 hàm kiểm tra phục vụ cho việc di chuyển*.

Hàm InitMoveDown() là hàm xử lý việc di chuyển xuống nói chung.

Hàm này sẽ kiểm tra trước qua hàm điều kiện IsGround() nếu bên dưới nhóm gạch có bề mặt tiếp xúc hoặc bề mặt. Hàm này dựa theo giá trị số 2 trên mảng formation.

Nếu không có thì hàm sẽ cho nhóm gạch đi xuống một mức, ngược lại thì sẽ đặt gạch xuống, chạy hàm kiểm tra và xóa dòng rồi sau đó cho xuất hiện gạch mới.

Hàm InitInstantDown() cơ bản cũng như InitMoveDown(), nhưng thực hiện theo vòng lặp đệ quy cho đến khi gạch tiếp xúc bề mặt và thực hiện đặt gạch rồi ngưng.

Hàm InitTimer() là hàm khởi tạo cho việc thực hiện đếm giờ, nhằm phục vụ cho việc tự động hóa trò chơi. Nếu key tạm dừng trò chơi, có thể hiểu là trạng thái tạm dừng trò chơi, không phải là TRUE, thì hàm này được thực thi.

Hàm này sẽ bắt đầu thực hiện các hàm di chuyển xuống sau mỗi tick (được tính và xác định và điều kiện nếu mốc thời gian hiện tại - mốc thời gian đã đánh dấu > khoảng thời gian delay được định)

Nếu nhóm gạch đã ở trạng thái đã tiếp xúc với bề mặt, game sẽ cho phép người chơi 1 khoảng thời gian đợi tương đương với số tick \* màn chơi (vì màn chơi càng về sau thì gạch sẽ rơi nhanh hơn, tương đương với khoảng thời gian delay giữa các tick sẽ ngắn hơn).

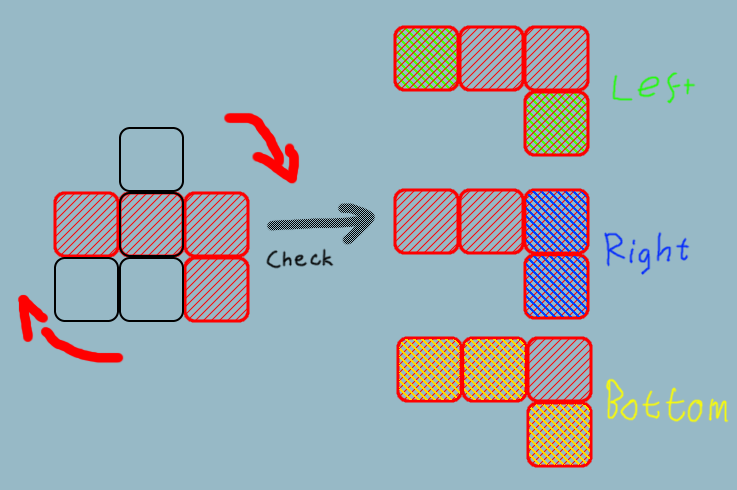
Sau khi các số tick đợi đó đã được tính qua, thì hàm này sẽ thực hiện cho gạch đi xuống một lần nữa, mà lúc đó đối với hàm MoveDown() và điều kiện của nó thì gạch lúc đó sẽ được đặt tự động.

Hàm InitMoveLeft() và InitMoveRight() xử lý cho thao tác di chuyển gạch qua trái và phải.

Như InitMoveDown(), chúng dùng chung một hàm điều kiện CheckBorder(). Hàm này sẽ kiểm tra nếu phía bên trái (hoặc phía bên phải) của nhóm gạch có tiếp xúc hoặc nằm bên cạnh biên, tùy theo hàm gọi. Hàm này sử dụng mảng left và mảng right.

Hàm InitSpinBlock() và InitRevSpinBlock() là hai hàm xử lý việc xoay gạch theo hướng thuận chiều kim đồng hồ hoặc ngược chiều kim đồng hồ. Chúng sử dụng chung một hàm kiểm tra điều kiện có thể xoay gạch hay không trên một hàm có tên là CheckSpindable(Formation formation).

Hàm kiểm tra điều kiện này là một hàm phức tạp. Nó cần đưa vào một Formation, tùy theo hàm gọi nó là xoay theo hướng nào, thì giá trị đưa vào sẽ là một Formation ở trạng thái tiếp theo hoặc trạng thái trước đó.



*Minh họa việc kiểm tra xoay qua việc kiểm tra phiên bản cạnh bên của nhóm gạch*.

Hàm đó sẽ kiểm tra xem xét nếu vị trí của Formation đã cho vào có mâu thuẫn với các giá trị của mảng bảng. Theo việc kiểm tra xoay gạch, thì chúng sẽ lấy phiên bản kế tiếp hoặc trước đó của gạch, và kiểm tra 3 hướng trái, phải và bên dưới như hình trên, qua phép đối chiếu với mảng bảng và xem việc mâu thuẫn với giá trị trong mảng bảng có xảy ra hay không. Tùy theo trường hợp, nhóm gạch đó sẽ được xử lý như sau:

* Nếu cả 4 hướng đều trống, thì cho gạch xoay tại chỗ.
* Nếu bên trái bị mâu thuẫn và bên phải trống, thì gạch sẽ được dịch qua bên phải rồi cho phép xoay gạch và ngược lại.
* Nếu bên dưới bị mâu thuẫn nhưng ở phía trên là trống, thì gạch sẽ được dịch lên trên và cho phép xoay gạch.
* Nếu cả ba hướng trái, phải và bên dưới đều mâu thuẫn thì không cho xoay gạch.
* Nếu bên trái và bên phải đều mâu thuẫn thì không cho xoay gạch (việc này xảy ra là rất ít nhưng nếu có xảy ra thì gạch vẫn sẽ không có ở tình trạng có thể xoay được qua mắt nhìn).

Hàm InitHoldBlock() và SwitchBlock() là hai hàm thực hiện việc “giữ gạch” tạm, theo quy luật của trò chơi.

Gạch điều khiển hiện tại sẽ được đem vào một biến Formation tạm để giữ cho đến khi người chơi thao tác nút để gọi lại InitHoldBlock(), từ đó thực hiện đến SwitchBlock(). Nhưng việc này chỉ được cho phép người chơi thực hiện một lần cho đến khi nhóm gạch đang được điều khiển của họ được đặt xuống.

Hàm InitPause() và InitStop() là hai hàm có khả năng cho việc tự động hóa trò chơi bị tạm dừng, ngừng hẳn hoặc tiếp tục chạy.

Nếu InitStop() được thực thi, và key isGameOver là TRUE (nghĩa là trò chơi đã kết thúc tự nhiên, không phải qua yêu cầu của người chơi), thì số điểm của người chơi nếu lớn hơn điểm cao nhất thì điểm cao nhất đó sẽ được thay thế và lưu lại trong file.

Hàm Input là hàm quản lí hết tất cả các thao tác hoạt động của game.

* 1. **Main - nơi chạy game**

Sau khi đã xây dựng được Engine, ta quay vào hàm main để chạy game.

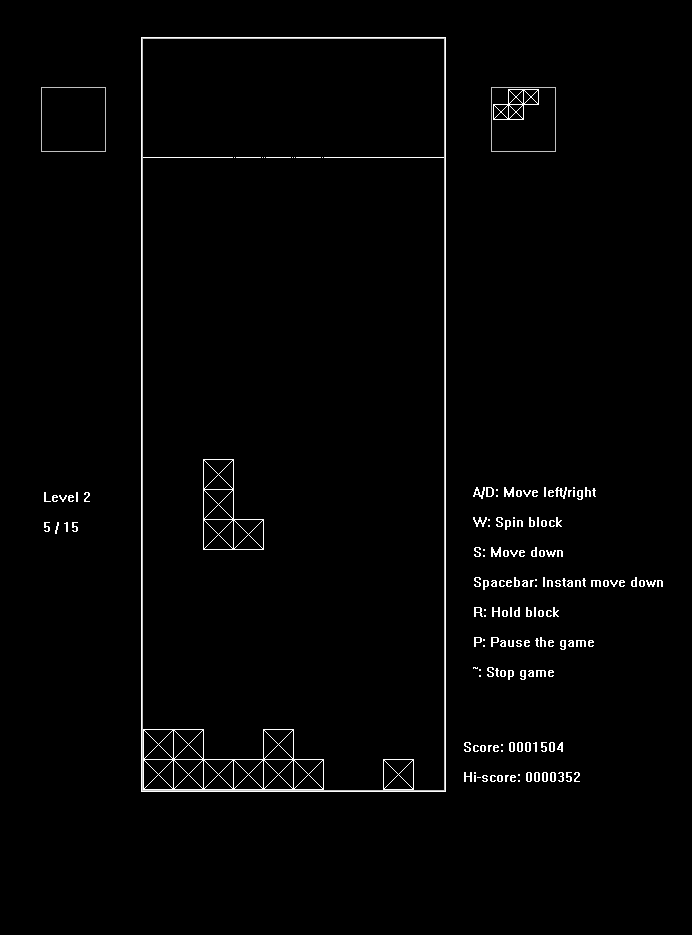
Trước khi chạy, chương trình sẽ khởi tạo ra một biến char tương tự cho biến input của thao tác bàn phím.

Sau khi hiển thị hết tất cả các nút người chơi có thể thao tác trên giao diện (các nút hiện trên console là các nút debug game), hàm sẽ gọi class Engine đó, bắt đầu trò chơi, rồi tạo vòng lặp chạy liên tục cho hai hàm tự động hóa trò chơi và đợi input từ bàn phím. Trong khi khóa k đang ở tình trạng cho phép game tiếp tục.

Khi Engine gọi InitStop(), khóa k đó sẽ thay đổi tùy thuộc vào lựa chọn của người chơi sau khi kết thúc game. Nếu người chơi muốn chơi lại, khóa k sẽ là 0, còn muốn thoát trò chơi thì khóa k sẽ là 1.

Nếu khóa k là 0 thì trò chơi sẽ được thực thi lại từ đầu qua dòng lệnh goto, còn nếu là 1 thì chương trình sẽ đóng lại, kèm theo trò chơi.

1. **Kết luận**



*Kết quả giao diện của trò chơi.*

Sau khi hoàn tất được bài tập lớn về làm game tetris, và làm game nói chung, chúng em thấy rằng giao diện đóng một vai trò rất lớn trong việc tạo hóa một trò chơi. Nhưng, graphics.h rất thiệt thòi về mảng này, vì sự hỗ trợ về tương tác giữa các vật thể gần như là không có, và thư viện này hoàn toàn chỉ giúp cho việc vẽ và tô hình, và việc sử dụng thêm các thư viện bên ngoài khác như chrono cho xử lý đếm thời gian và tự động hóa cho trò chơi, thread cho việc thực hiện các hàm của game một cách liên tục cùng một lúc. Mặc dù thế, graphics.h đã thực hiện tốt trong việc hiển thị trò chơi trước mặt người chơi qua các đoạn code. Nhưng sau này, nhiều thư viện cao cấp hơn được tạo ra để tương thích hơn với mảng tạo game này, thậm chí cả một chương trình chuyện về tạo game, thường được mọi người gọi cho chúng là các Game Engine, là giao diện kết hợp rất nhiều giữa các yếu tố cần thiết cho việc tạo hóa trò chơi. Kết luận tóm tắt lại thì, trò chơi Tetris này là một trò chơi phổ biến, nhưng lại khá đơn giản và có thể thực hiện từ graphics.h.

Tài liệu tham khảo

<https://en.wikipedia.org/wiki/Tetris> (Ngày truy cập: 2/6/2020)

http://javilop.com/gamedev/tetris-tutorial-in-c-platform-independent-focused-in-game-logic-for-beginners/ (Ngày truy cập: 2/6/2020)