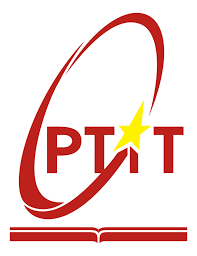
HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN

**---------------------- 🙠🕮🙢 ----------------------**



**BÁO CÁO BÀI TẬP LỚN**

**MÔN HỌC: LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG**

**Giảng viên:** Nguyễn Mạnh Sơn

**Nhóm môn học:** 03

**Nhóm BTL:** 13

**Thành viên:** B20DCCN297 – Nguyễn Tiến Hùng

B20DCCN569 – Nguyễn Như Quỳnh

B20DCCN330 – Trương Quang Huy

B20DCCN510 – Lê Quang Phúc

B20DCCN258 – Phạm Trung Hiếu

*Hà Nội, 22 tháng 11 năm 2022*

Hà Nội năm 2022

**Hà Nội, Tháng 12/2016**

1. **Giới thiệu**

* Sản phẩm: Escape Together
* Thể loại: Game Puzzle (giải đố)
* Lối chơi:
  + Game có nhiều level, trong mỗi level sẽ là một bản đồ với những nhân vật nhất định (các nhân vật chỉ có thể di chuyển trên mặt đất)

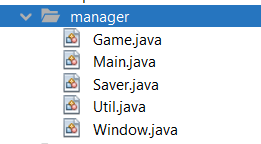


* + Mỗi nhân vật thì lại có 1 khả năng di chuyển riêng
    - Con ĐỎ chỉ có thể di chuyển sang ngang (trái phải)
    - Con XANH DƯƠNG chỉ có thể di chuyển thẳng (lên xuống)
    - Con XANH LỤC thì không thể di chuyển một mình, nhưng có thể liên kết với những con khác và di chuyển cùng con đó.
  + Việc của người chơi là thao tác các nút điều hướng 1 cách chính xác để đưa được tất cả các con nhân vật tới đích và qua màn.
  + Game sẽ có 20 level đi từ dễ tới khó, và những con nhân vật hoặc đối tượng mới (hộp gỗ, chìa khoá, cửa khoá, công tắc, ...) sẽ được giới thiệu dần dần qua từng màn chơi.

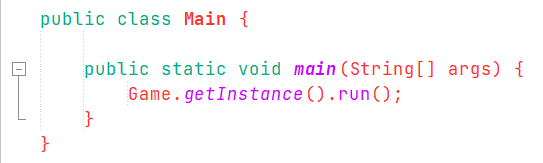
1. **Khảo sát**

* Qua tìm hiểu ban đầu, nhận thấy game là một sản phẩm không quá khó, và nhóm bọn em có thể tìm hiểu làm được.
* Về thể loại, game puzzle rất dễ dàng tiếp cận với nhiều người, tập trung chủ yếu về tư duy giải đố, có khả năng kích thích sự hứng thú của người chơi.
* “**One Action Heroes**” là một sản phẩm từ cuộc thi GMTK Game Jam 2019, ấn tượng về sản phẩm và nhận thấy nó đáp ứng được những điều trên, chúng em đã làm lại game này và có phát triển thêm một vài level + sáng tạo hình ảnh thêm một chút.
* Khảo sát trải nghiệm người dùng:
  + Gameplay đơn giản, chỉ cần thao tác với một vài nút mũi tên trên bàn phím.
  + Màu sắc hợp lý (game với tông màu trầm tối, nổi bật lên là màu sắc của các con nhân vật R-G-B).
  + Hình ảnh đơn giản và dễ thương (các vật thể trong game là những tile hình vuông có bao góc, dễ nhìn)
  + Âm thanh nhẹ nhàng, vui.
  + Game đi từ dễ tới khó, chơi qua những màn khó thấy rất vui.

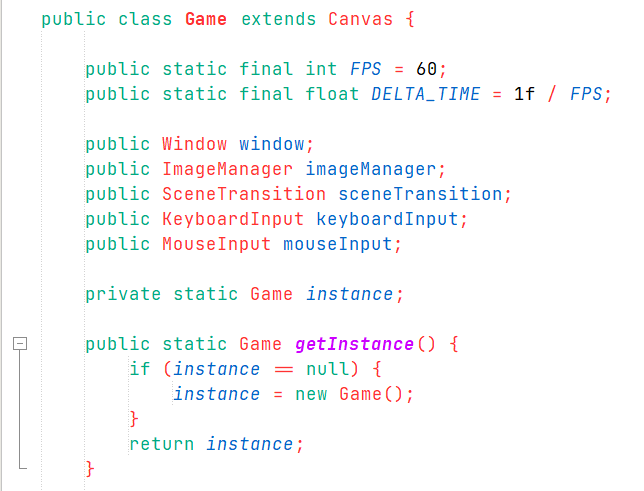
1. **Các module chính**
2. manager



* + **Main**: Chỉ có chức năng là chạy chương trình



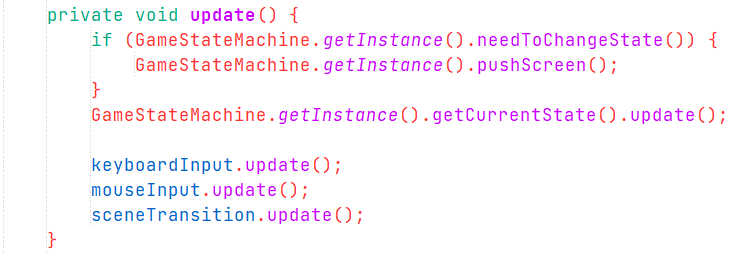
* + **Game**: Thiết kế theo Singleton
* Game được kế thừa từ Canvas, cho phép đặt các State game lên trên nó và vẽ chúng.
* Class Game thì quản lý và thực hiện việc update những đối tượng chính của game (phần lớn là những đối tượng chỉ có 1 instance, xuất hiện hầu hết mọi nơi trong game như Quản lý chuột, Quản lý bàn phím, Quản lý ảnh, nên tập trung về class Game).

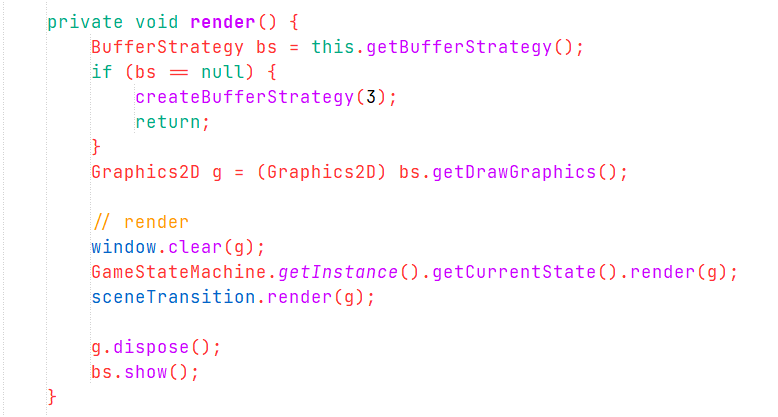


* Hàm run() của class Game sẽ là gameloop, nó tính toán deltaTime theo FPS mình target, update() và render() mọi thứ theo FPS đó.

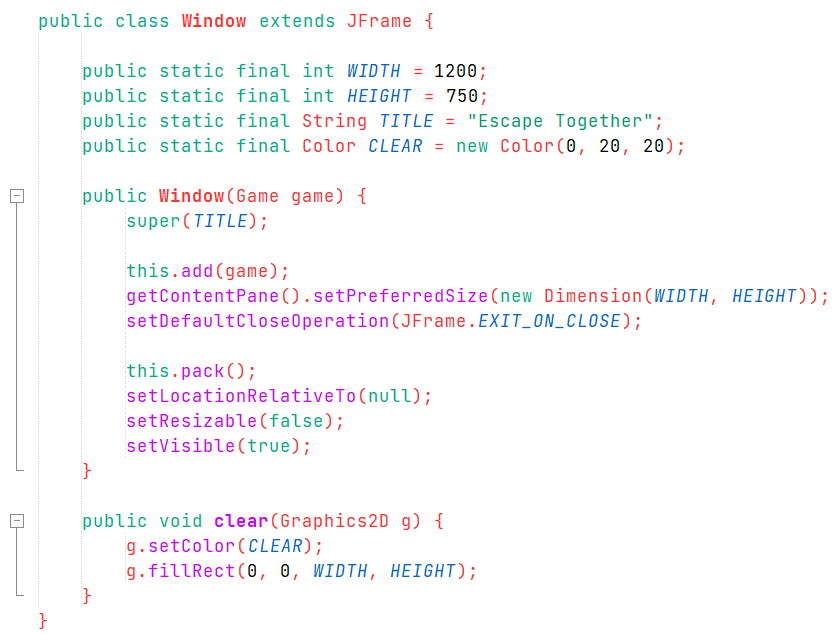


* Update() và render() của Game sẽ là nơi đầu tiên được xử lý (nguồn), sau đó đi update và render tất cả mọi thứ nhỏ nhỏ hơn nó.

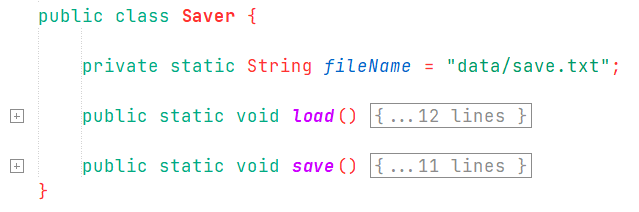




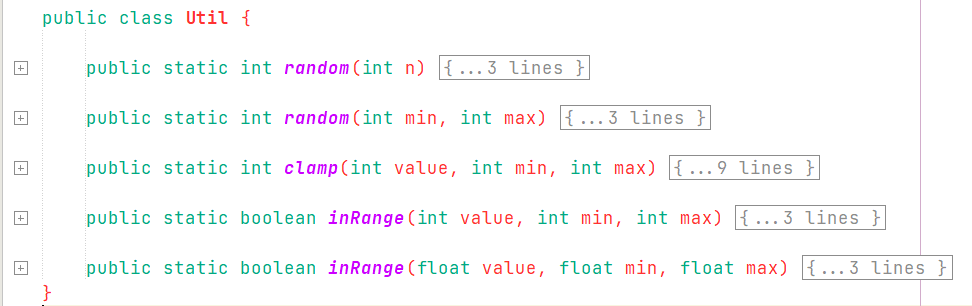
* + **Window**: là cửa sổ chính của app
* Window là 1 JFrame, nó add vào 1 Canvas là Game, Game lại add vào chính nó là các JPanel (Gọi là GameState, được trình bày phía dưới).
* Class Window thiết lập những thông số cơ bản của nó.



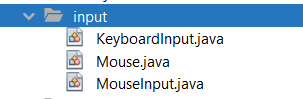
* + **Saver**: Làm 2 công việc chính là load dữ liệu (khi mới vào game) và lưu game khi có thay đổi gì đó.
* Có 2 thông số sẽ lưu lại để khi khởi động lại app sẽ có là: Level hiện tại (để tiếp tục chơi) và Setting âm thanh (tắt hay mở).



* + **Util**: class chức năng, cung cấp 1 vài static func do mình tự định nghĩa



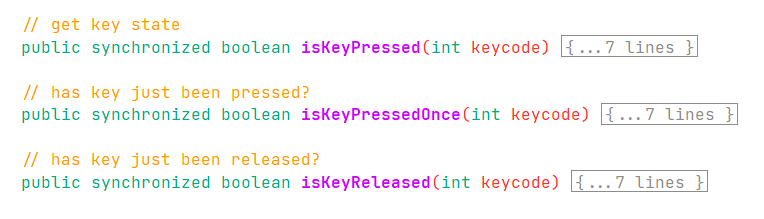
1. **input**



* + **KeyboardInput**: implements KeyListener và override lại 1 số phương thức cần dùng.



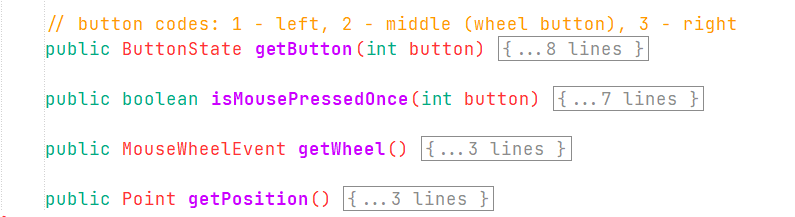
* Hỗ trợ 1 số chức năng: kiểm tra xem có phải vừa mới ấn nút, vừa mới thả nút, hay đang nhấn giữ nút nào.



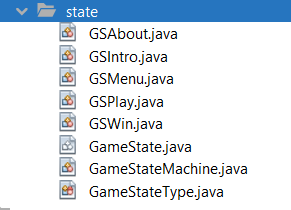
* + **MouseInput**: kế thừa từ MouseInputAdapter



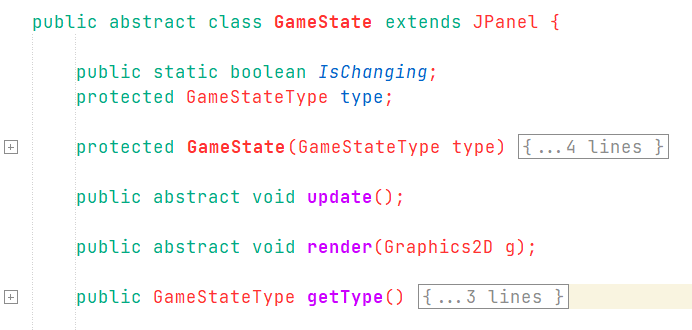
* Hỗ trợ 1 số chức năng như: lấy vị trí chuột trên màn hình, kiểm tra xem vừa có click chuột, vừa thả chuột, có đang nhấn giữ chuột hay kéo thả không, ...



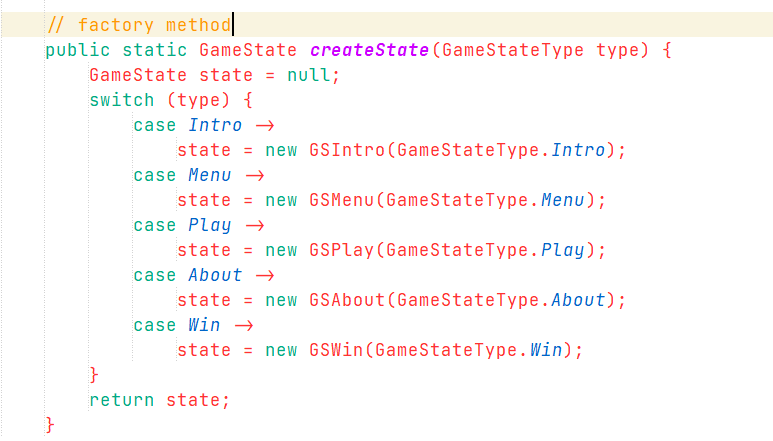
1. **State**



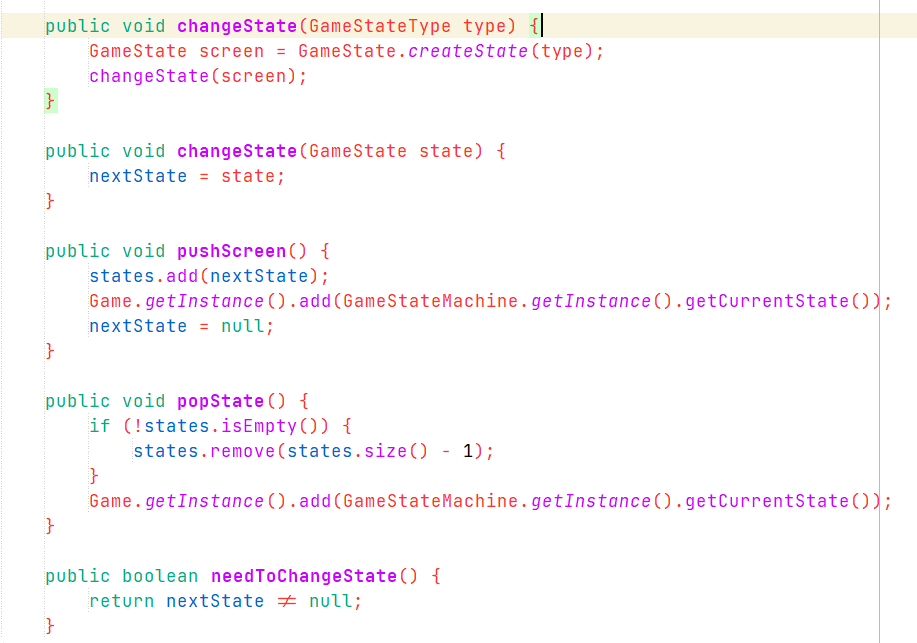
* + **GameState**: Kế thừa từ JPanel, được add lên Game (Canvas)
* Là 1 abstract class với 2 chức năng chính mà các GameState kế thừa nó cần triển khai là update() và render()



* Factory method: là nơi để khởi tạo ra các GameState khi cần



* + **GameStateMachine**: Singleton, quản lý việc chuyển qua lại giữa các state game theo cơ chế stack.



* Khi app đang mở, ta sẽ có nhiều GameState như Intro, Menu, About, Play, ...
* GameStateMachine như 1 stack, lưu các state game và chỉ update và render GameState hiện tại (tức GameState ở đỉnh stack).

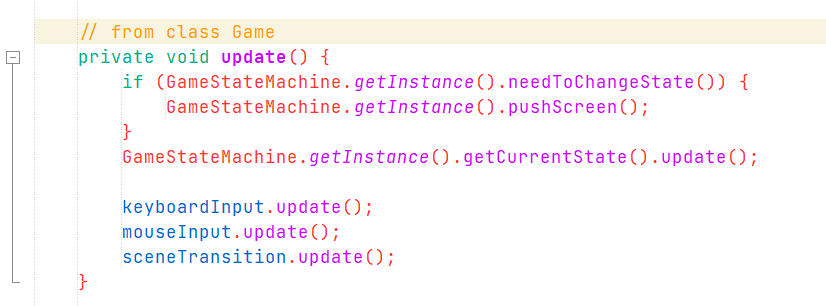
VD:

1. run game -> Intro (Được tạo vào đưa vào stack), mỗi frame sẽ chỉ cập nhật và update cái GameState Intro này.
2. Sau 2s hết Intro -> Game tự động chuyển sang GameState Menu (Lại được tạo ra và push vào stack), như vậy trong stack đang có 2 GS, nhưng GS Intro không còn cập nhật và render nữa, chỉ có GS Menu được.

Và đây là cách để chuyển từ Intro sang Menu:



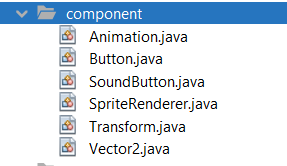
Trong lần update tiếp theo tại class Game, ta sẽ thực hiện việc push GS mới vào, và update GS mới đó, GS cũ không còn được update nữa.



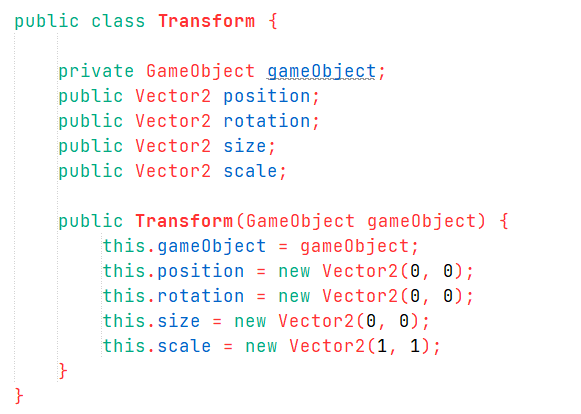
1. Tương tự khi từ GS Menu rồi, khi ta bấm Play thì nó sẽ tạo và push vào GameStateMachine 1 cái GS Play (lúc này GS Play ở đỉnh stack) thì nó được cập nhật và kết xuất lên màn hình. GS Menu vẫn còn trong stack nhưng không phải ở đỉnh nên không được cập nhật và vẽ (coi như bị tắt đi).
2. Khi đang ở GS Play và muốn trở về GS Menu, ta chỉ cần popState (Xoá GS Play khỏi stack), lúc này GS Menu đã ở đỉnh Stack, nó lại được tiếp tục update và render



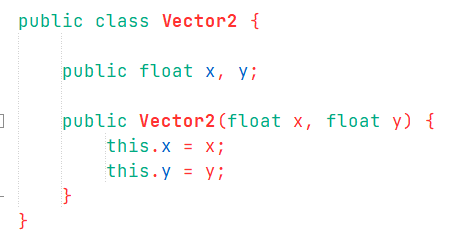
1. **Component**



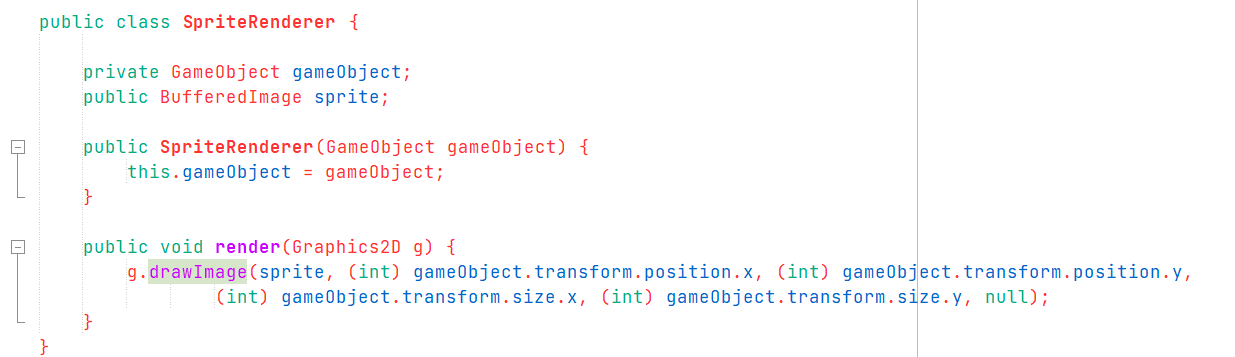
* + **Transform**: lưu các thông số như vị trí, kích thước, độ scale của 1 GameObject (hầu hết các vật thể trong game đều kế thừa từ GameObject)



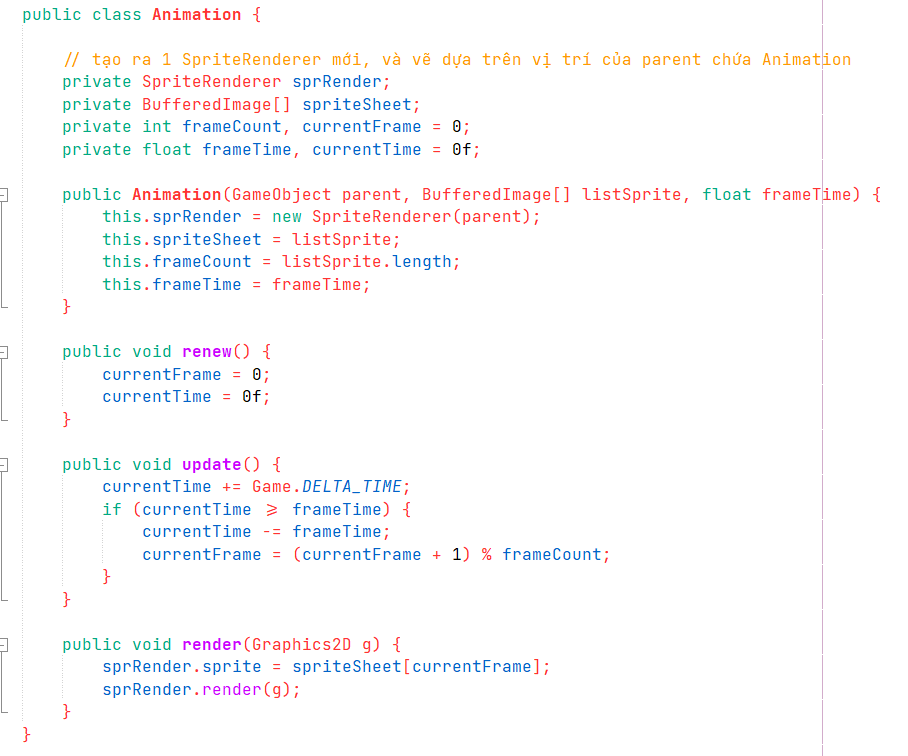
* + **Vector2**: tương tự như Point trong java, nhưng dùng kiểu dữ liệu là float -> giúp tính toán chính xác hơn trong việc di chuyển trong từng frame.



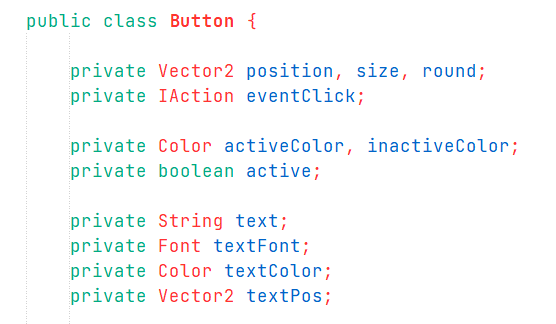
* + **SpriteRenderer**: là thành phần của 1 gameObject, cho phép tự do set texture (BufferedImage) và vẽ nó theo transform của gameObject.



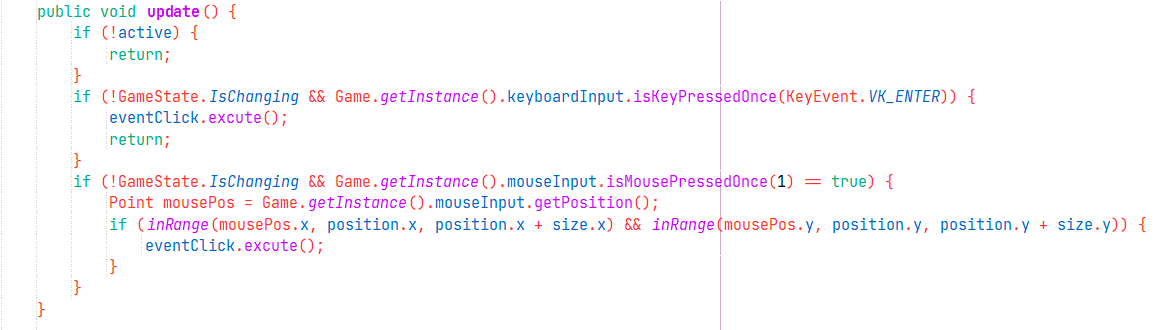
* + **Animation**: sẽ gồm 1 danh sách các frame, mỗi frame cho hiển thị trong khoảng thời gian nhỏ nhỏ (vd tầm 0.1s) , như vậy sau mỗi 0.1s nó sẽ chuyển sang vẽ frame tiếp theo, cho tới khi vẽ xong frame cuối cùng thì chuyển về vẽ frame đầu tiên.



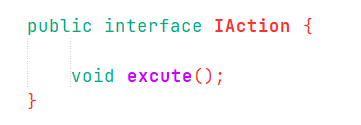
* + **Button**: đây là class nhóm em tự code lại (do khi dùng JButton của java thì không thấy hiển thị trên game và chưa khắc phục được)



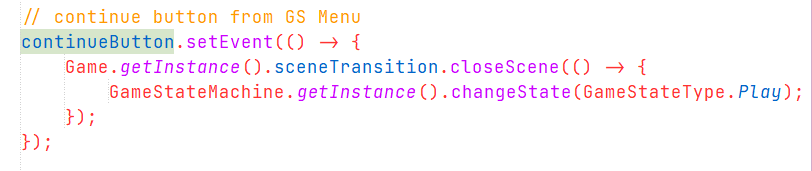
* Trong hàm update() của button: nó nhận và kiểm tra các sự kiện như nhấn phím Enter, click chuột sau đó đi vào thực thi eventClick.



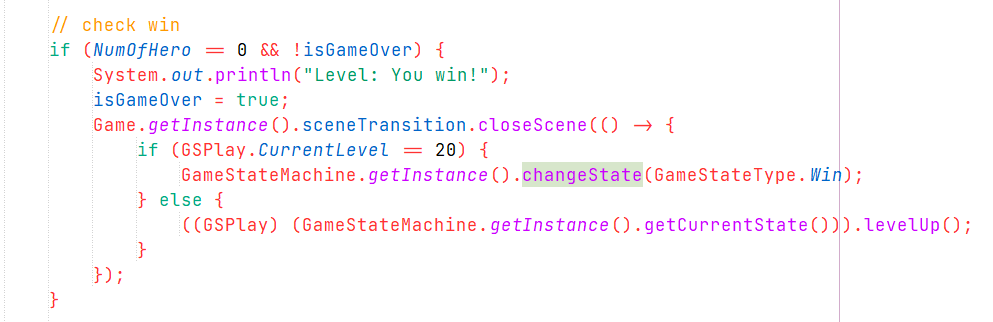
* + IAction: là một interface dùng để thực hiện 1 công việc gì đó khi có sự kiện xảy ra (bấm button, sau khi hoàn thành sceneTransition)



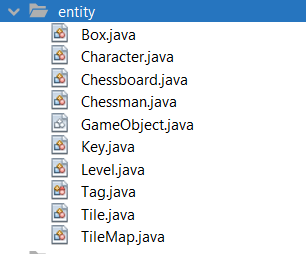
* Button:



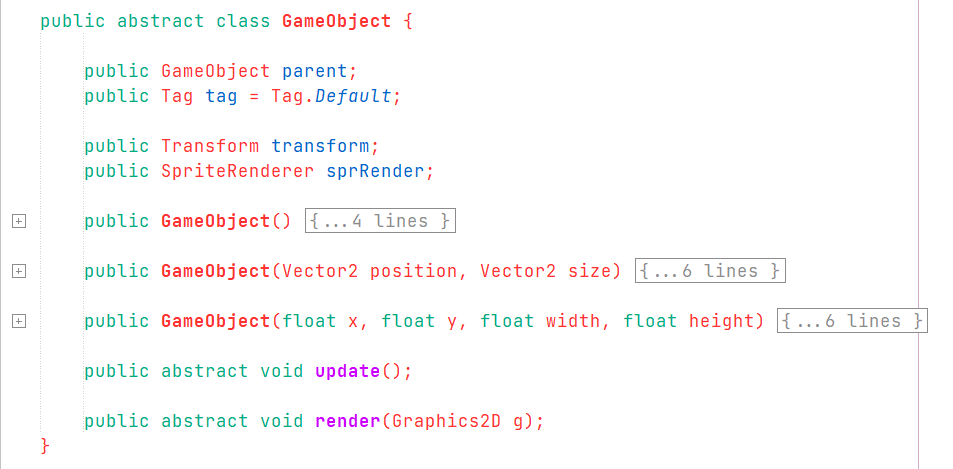
* SceneTransition: làm tối dần màn hình, khi đã tối đen hoàn toàn thì sẽ thực hiện 1 Action gì đó



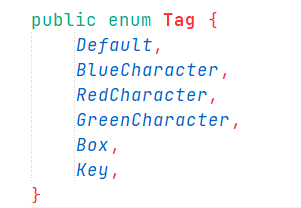
1. **Entity**



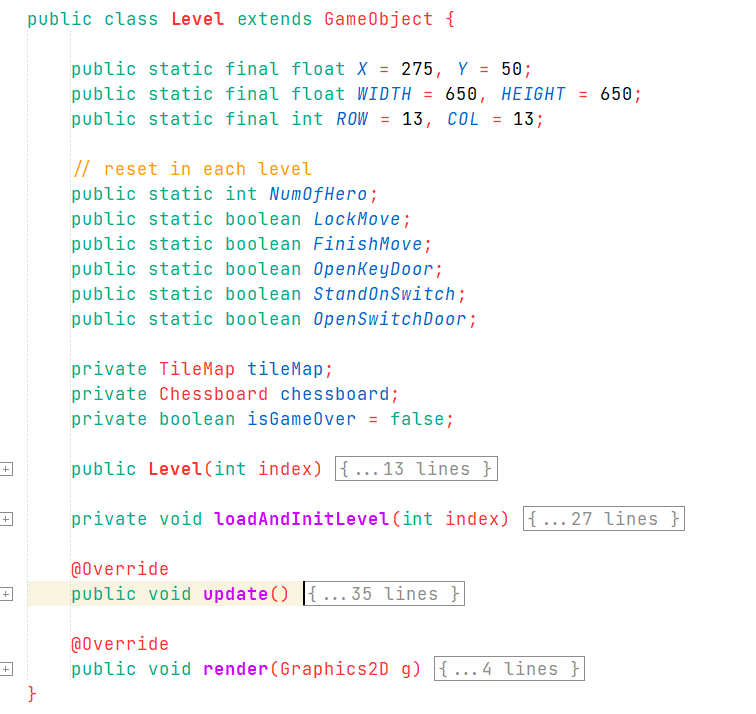
* + **GameObject**: abstract class, là class để cho hầu hết các đối tượng trong gameplay kế thừa và triển khai 2 phương thức chính là update() và render().



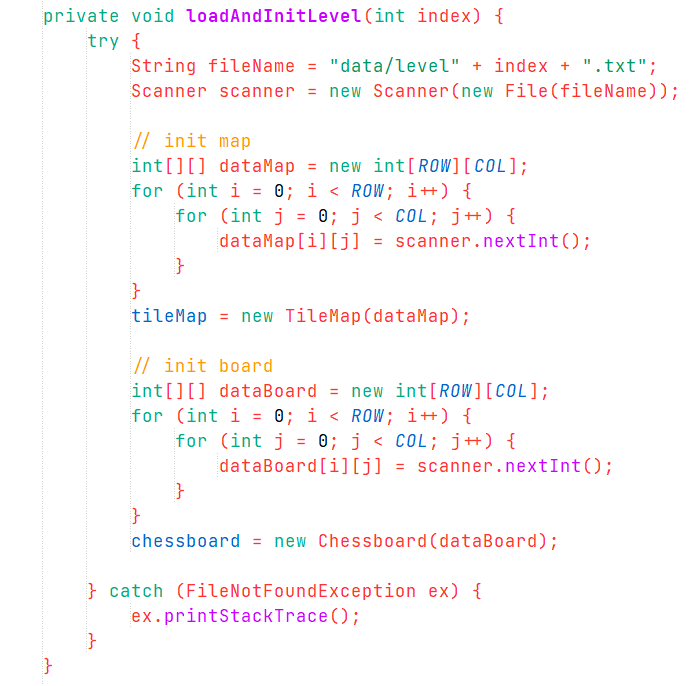
* Object Tag: Default là tag dành cho những gameObject hoặc là không tồn tại trong level đó, hoặc là không thể di chuyển, các object có tag còn lại thì có thể di chuyển được trên ma trận.



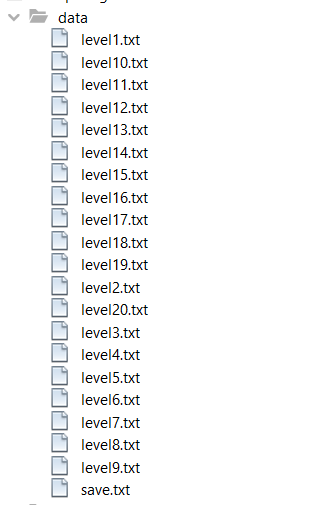
* + **Level**: GamePlay thì sẽ bao gồm level, nó sẽ nhận vào 1 số int là tên level và tiến hành đọc ghi file để lấy dữ liệu, rồi khởi tạo level đó

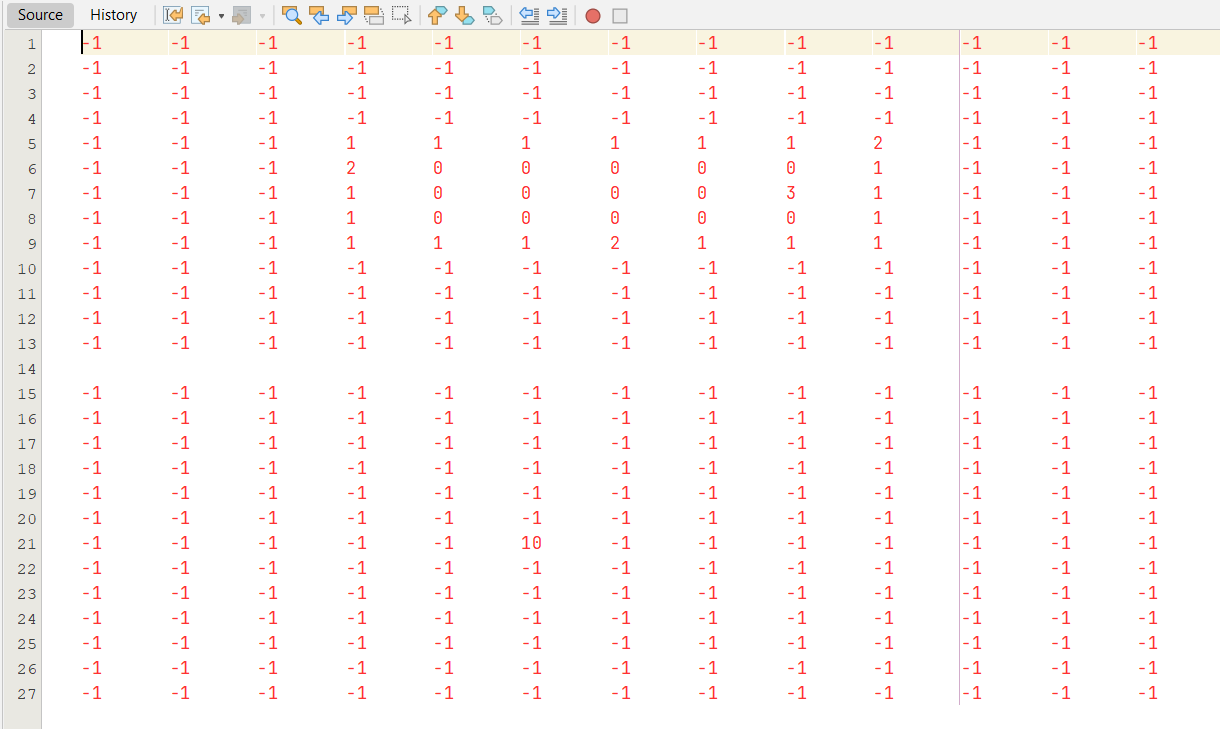


* Load data form file:

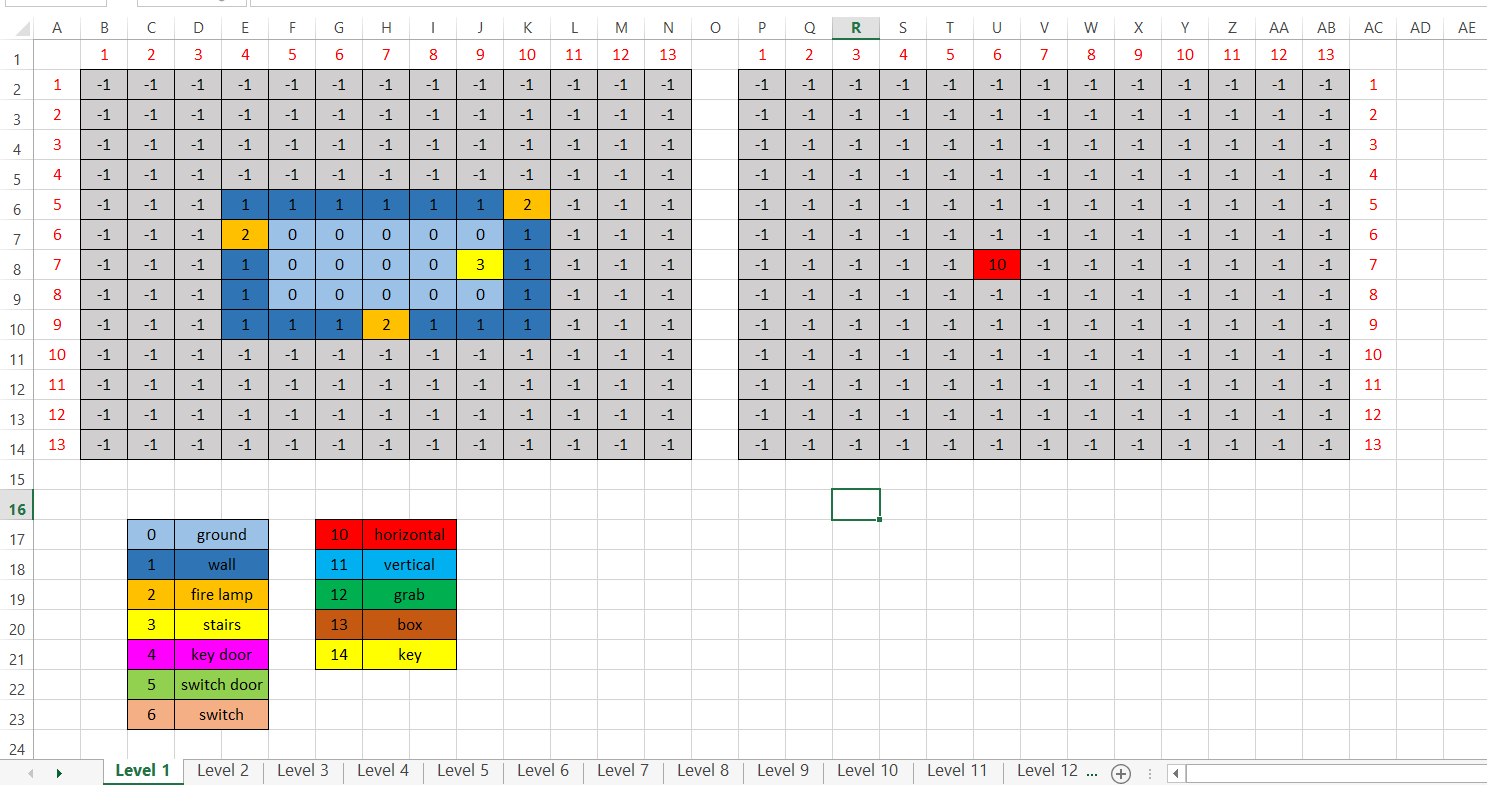


* Data được lưu file text như này:





* Giá trị của các phần tử trong ma trận đã được thiết kế và định nghĩa ở file Level.xlsx như này:

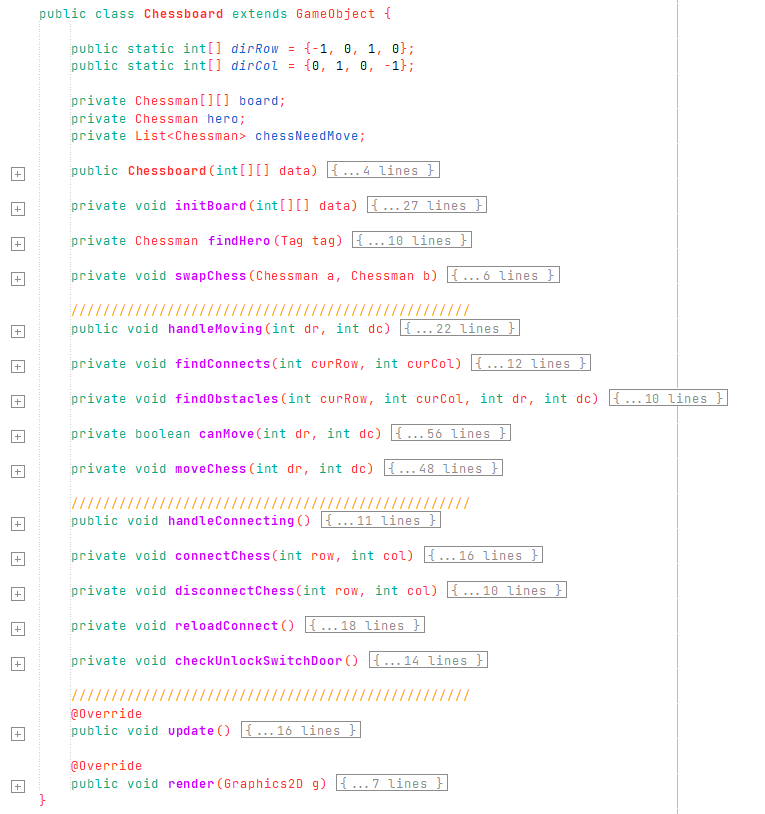


Ma trận bên trái (trong excel) cũng là ma trận thứ nhất (trong Level.txt) thực hiện việc vẽ map: vd -1 là không vẽ gì, 1 là khởi tạo 1 đối tượng wall (tường), 0 là khởi tạo đối tượng ground (đất).

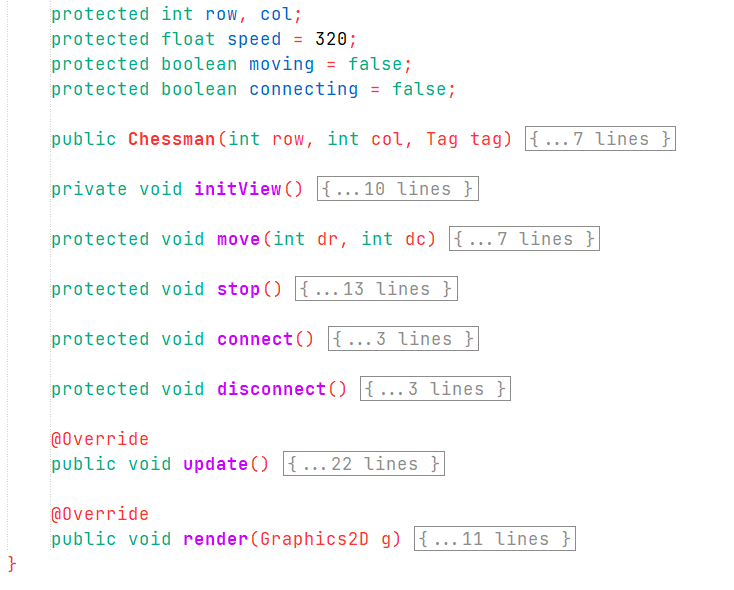
Ma trận bên phải (trong excel) và là ma trận thứ 2 (trong file.txt) thực hiện việc khởi tạo Các Object.

Level sẽ quản lý TileMap (ma trận trái) và Chessboard (ma trận phải), khi render() level, ta sẽ tiến hành vẽ tilemap trước, xong vẽ chessboard trùng lên trên như vậy thì gameobject có thể di chuyển trên mặt đất rồi.

* + **Chessboard**: quản lý vị trí (tính theo ma trận: hàng, cột) của các gameobject



* Khi update(): nó nhận vào các sự kiện bấm bàn phím, sau đó kiểm tra xem có thể di chuyển được không (không bị chặn bởi tường), thực hiện việc di chuyển, liên kết các nhân vật, ...
  + **Chessman**: đơn giản là 1 object nằm trong sự quản lý của chessboard (gameobject có thể di chuyển trong ma trận: nhân vật, hộp gỗ, chìa khoá, ..)



* Nó là class cha để các class con kế thừa, ta sẽ có:





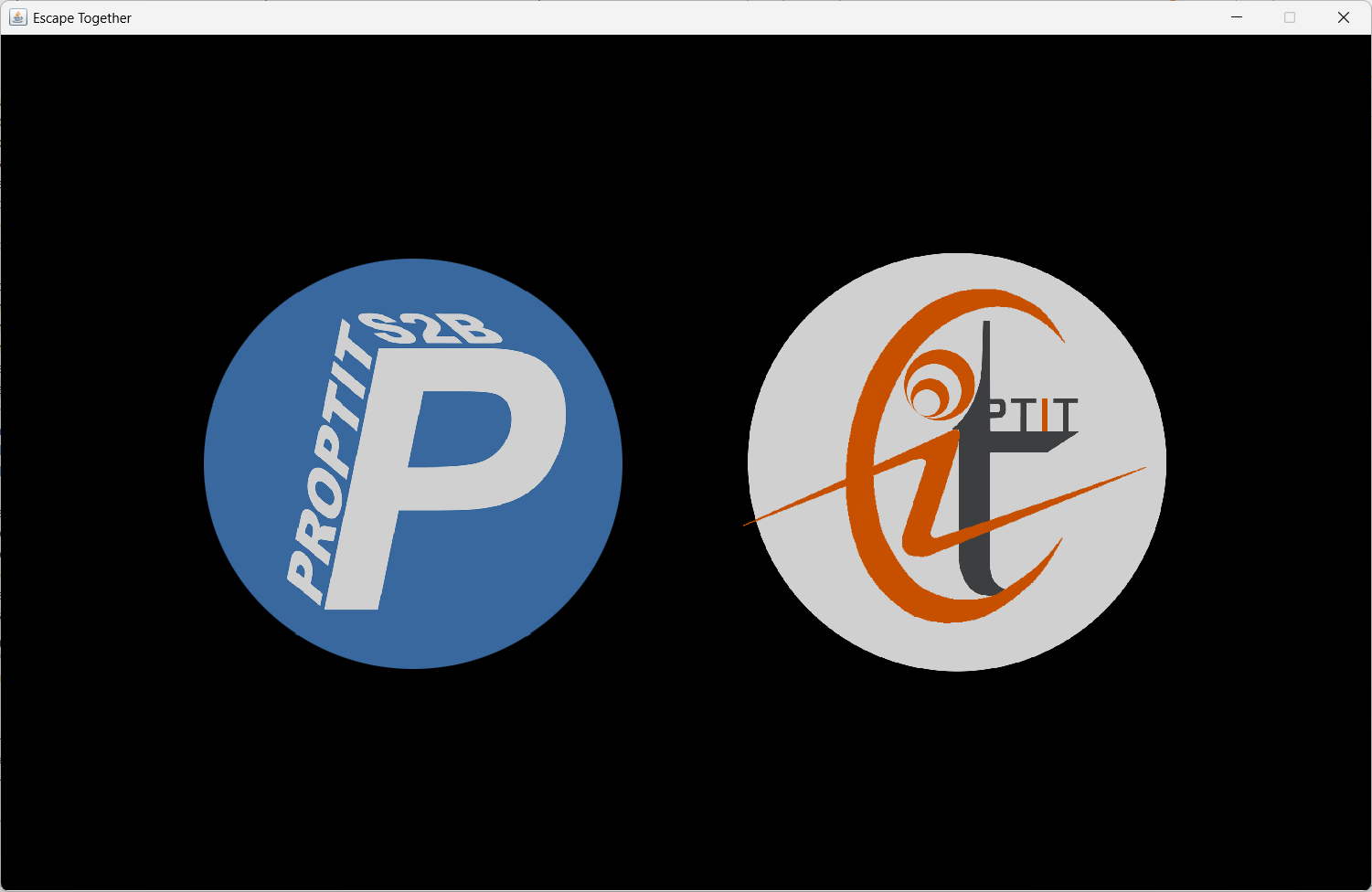


1. **Phương pháp – Ngôn ngữ - Công cụ**

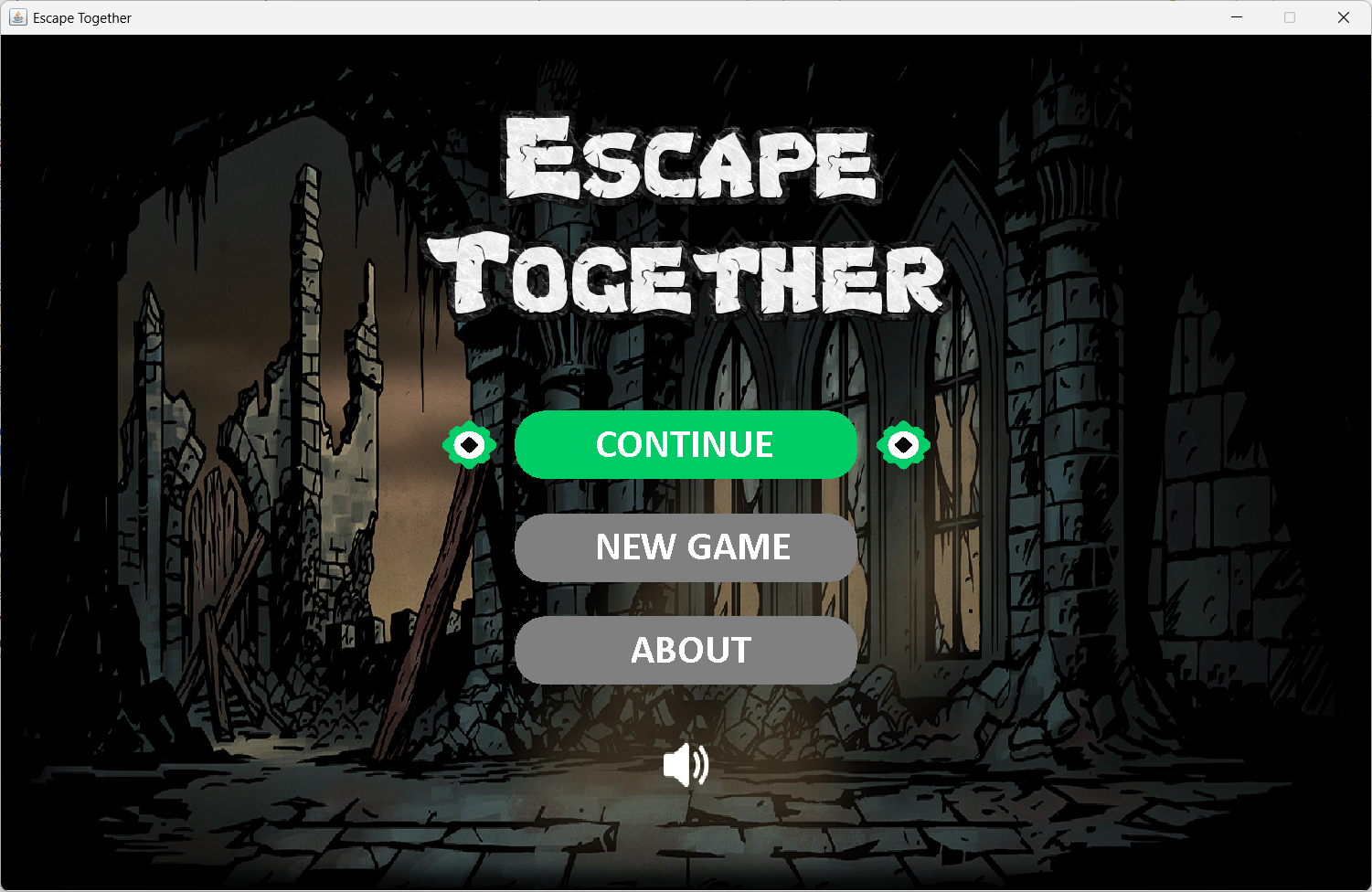
* Phương pháp:
  + Lập trình hướng đối tượng oop
  + Design pattern: singleton, factory method, state machine
* Ngôn ngữ: Java
* Công cụ:
  + Apache Netbeans IDE: code
  + Graphics2D cùng các thư viện khác của java.awt
  + Adobe Ilustator: thiết kế asset cho game
  + Photoshop: chỉnh sửa 1 số ảnh

1. **Giới thiệu hình ảnh kết quả**

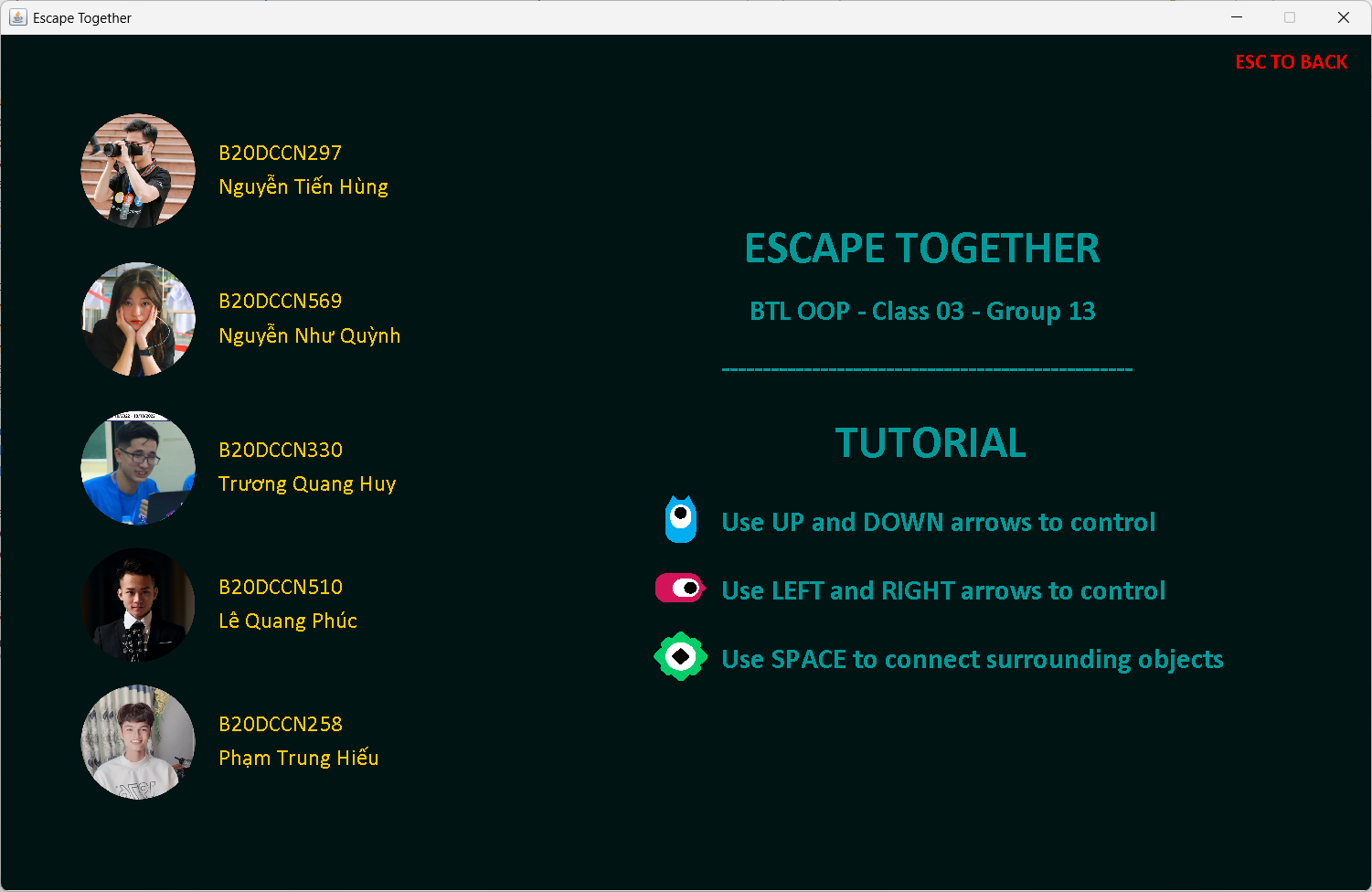
* Intro:



* Menu:

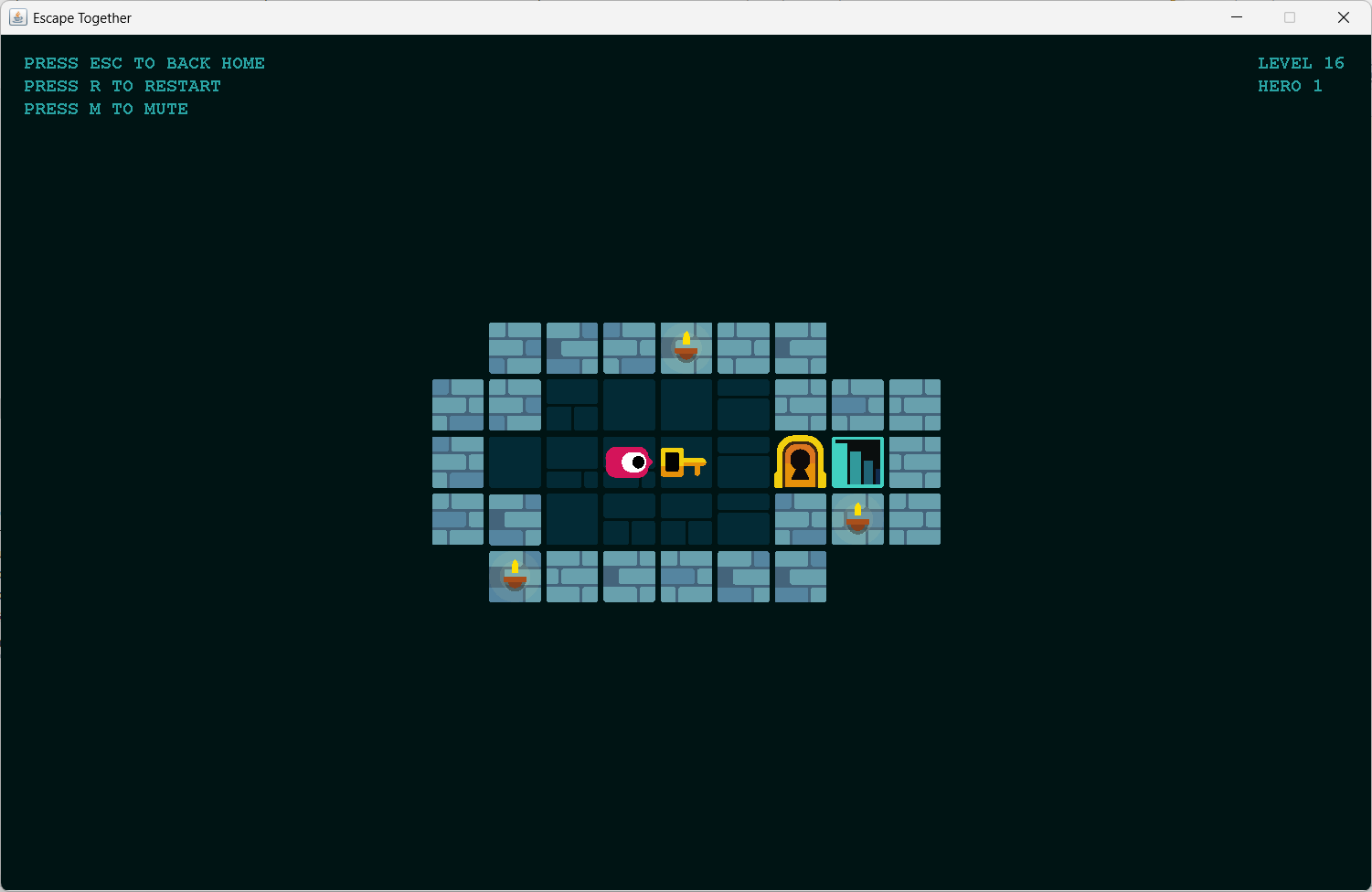


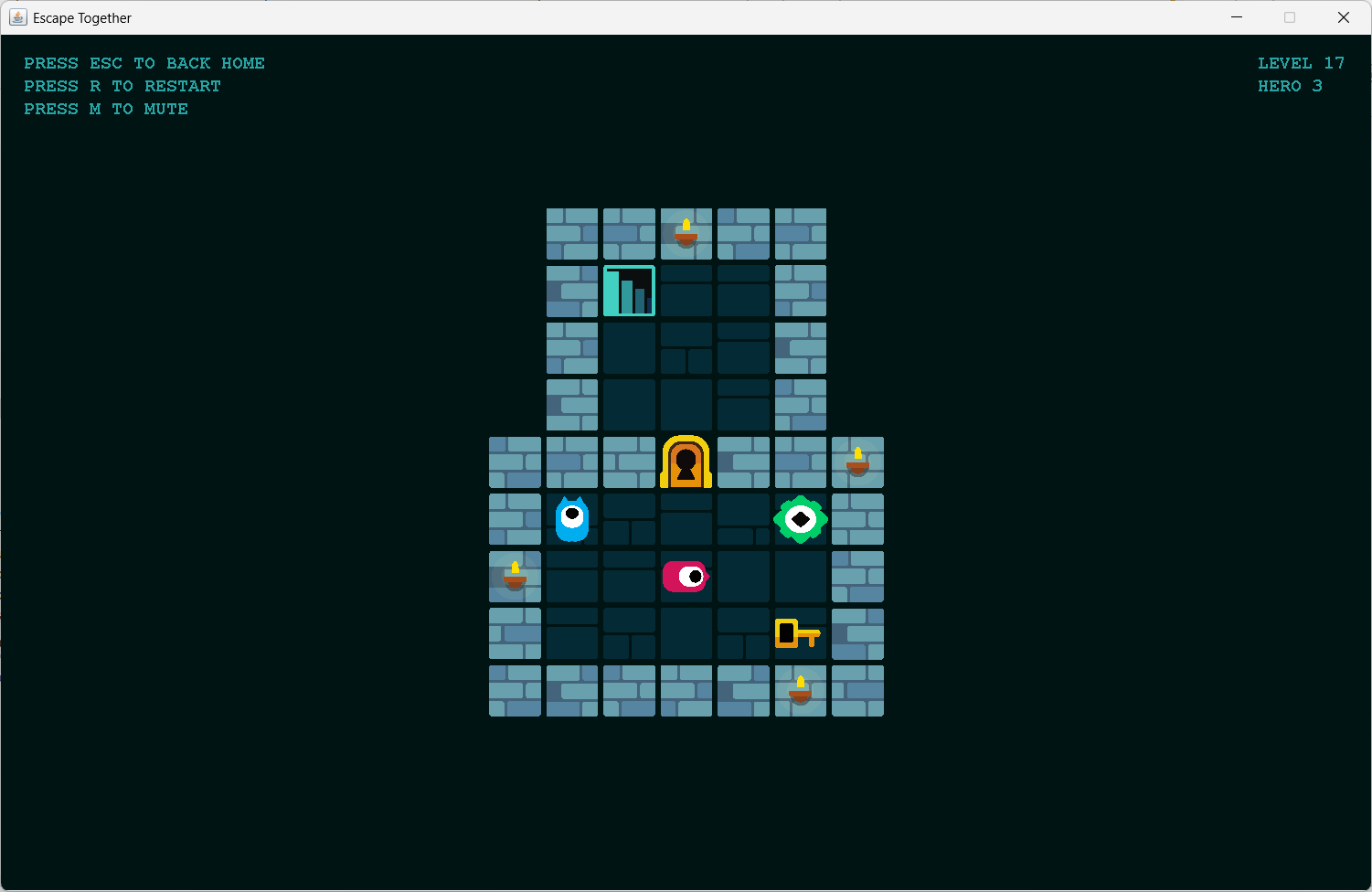
* About:



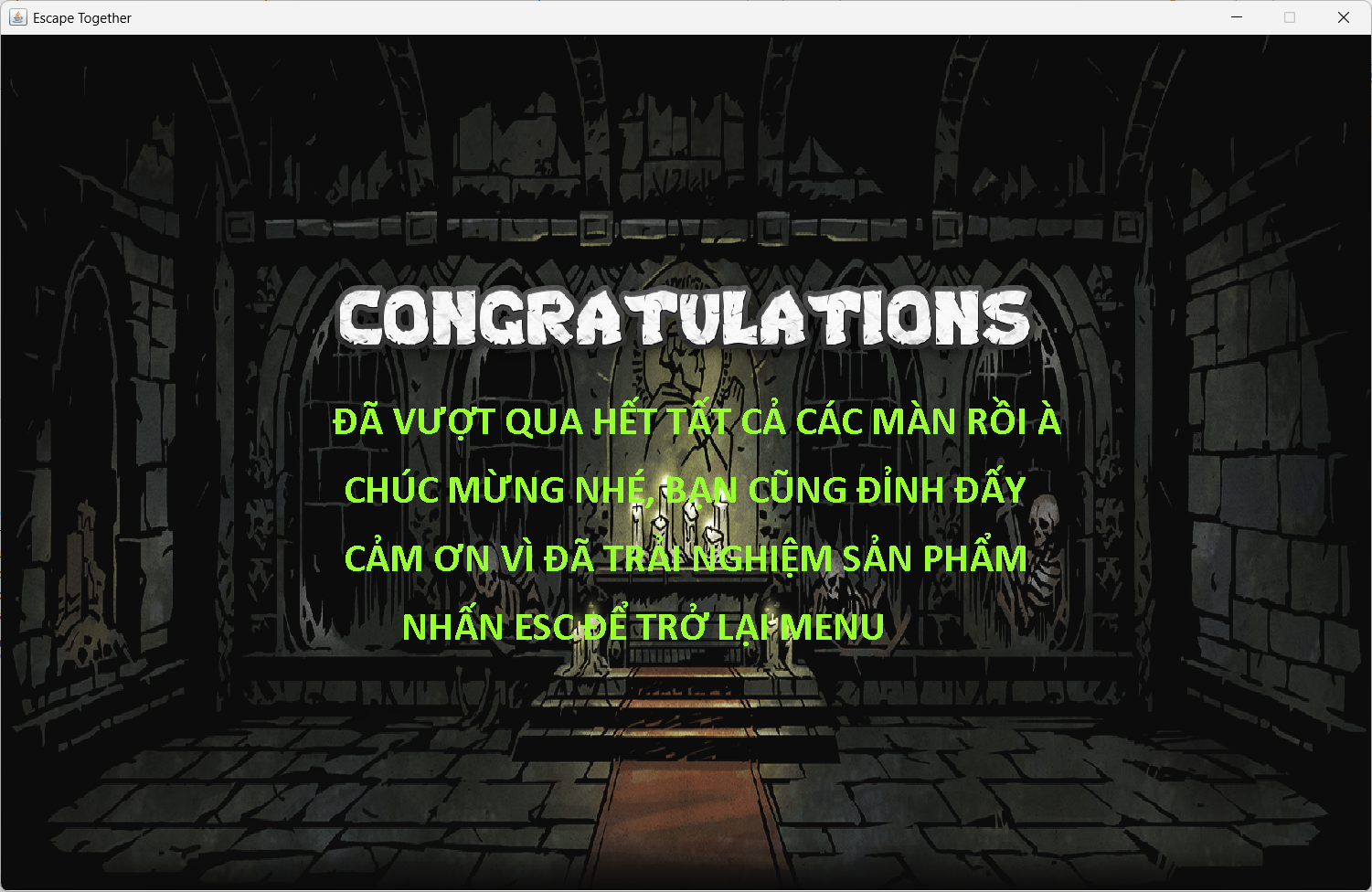
* Play:







* Win:



1. Các tài liệu tham khảo:

* Series lập trình game 2d với java: <https://youtu.be/om59cwR7psI>
* MouseInput pooling system: <https://youtu.be/_zNh-BJPrb4>
* Delta time handling: <https://youtu.be/p7X64g6cOgQ>
* Github tham khảo: <https://github.com/NgTienHungg/SimpleMaze>
* Github tham khảo: <https://github.com/Dynamicslvl/Beat-em-up>