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

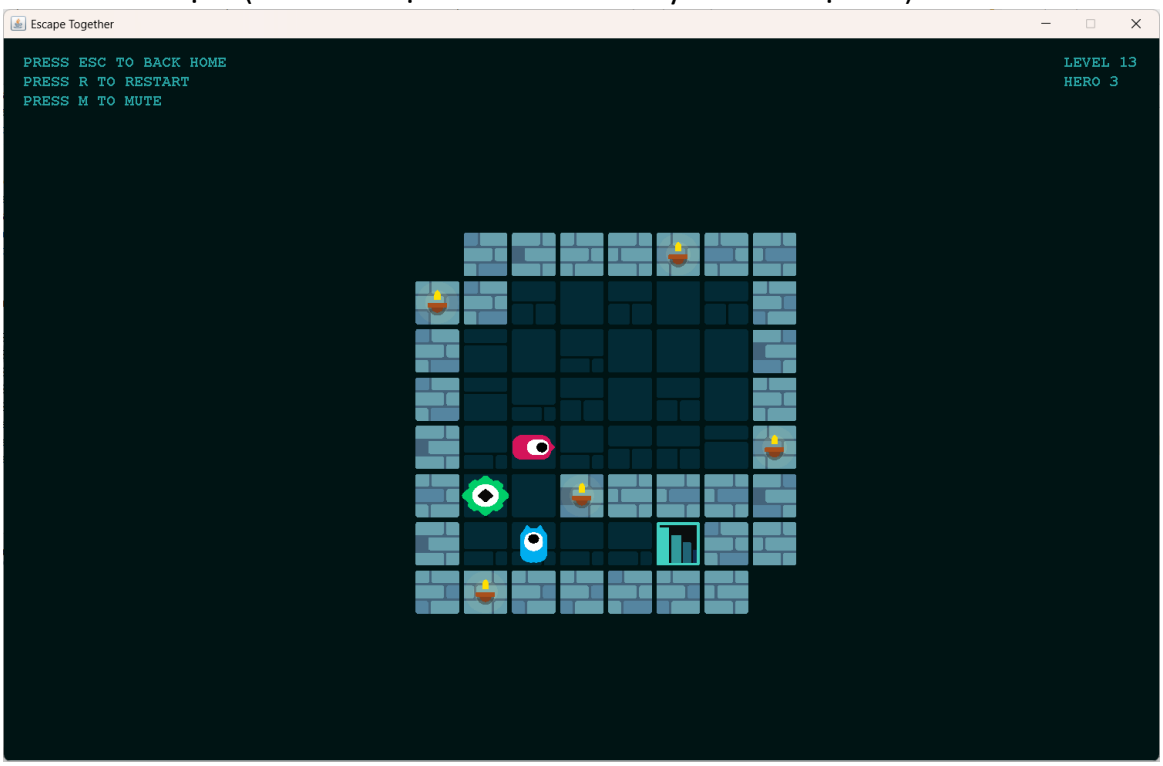
B20DCCN510 – Lê Quang Phúc

B20DCCN258 – Phạm Trung Hiếu

Hà Nội, 22 tháng 11 năm 2022

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.

2. Khảo sát

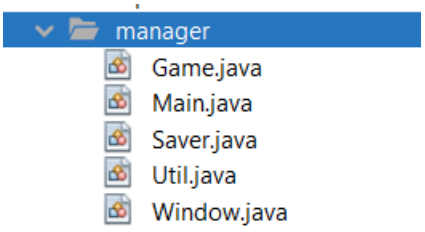
- Qua tìm hiểu ban đầu, nhận thấy game là một sản phẩm không quá khó, và nhóm bọn em có thể tìm hiểu làm được.

- 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.

3. Các module chính

a) manager



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

```
public class Main {

    public static void main(String[] args) {
        Game.getInstance().run();
    }
}
```

- **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).

```
public class Game extends Canvas {

    public static final int FPS = 60;
    public static final float DELTA_TIME = 1f / FPS;

    public Window window;
    public ImageManager imageManager;
    public SceneTransition sceneTransition;
    public KeyboardInput keyboardInput;
    public MouseInput mouseInput;

    private static Game instance;

    public static Game getInstance() {
        if (instance == null) {
            instance = new Game();
        }
        return instance;
    }
```

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

```java
public void run() {
    double drawInterval = 1e9 / FPS;
    double delta = 0;
    long lastTime = System.nanoTime();
    long currentTime;

    // calculate FPS
    long timer = 0;
    int drawCount = 0;

    while (true) {
        currentTime = System.nanoTime();
        delta += (currentTime - lastTime) / drawInterval;
        timer += (currentTime - lastTime);
        lastTime = currentTime;

        if (delta ⩾ 1) {
            update();
            render();
            delta--;
            drawCount++;
        }

        // show FPS
        if (timer ⩾ 1e9) {
            System.out.println("FPS: " + drawCount);
            timer = 0;
            drawCount = 0;
        }
    }
}
```

- 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.

```java
private void update() {
    if (GameStateMachine.getInstance().needToChangeState()) {
        GameStateMachine.getInstance().pushScreen();
    }
    GameStateMachine.getInstance().getCurrentState().update();

    keyboardInput.update();
    mouseInput.update();
    sceneTransition.update();
}
```

```java
private void render() {
    BufferStrategy bs = this.getBufferStrategy();
    if (bs == null) {
        createBufferStrategy(3);
        return;
    }
    Graphics2D g = (Graphics2D) bs.getDrawGraphics();

    // render
    window.clear(g);
    GameStateMachine.getInstance().getCurrentState().render(g);
    sceneTransition.render(g);

    g.dispose();
    bs.show();
}
```

    - **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.

```java
public class Window extends JFrame {

    public static final int WIDTH = 1200;
    public static final int HEIGHT = 750;
    public static final String TITLE = "Escape Together";
    public static final Color CLEAR = new Color(0, 20, 20);

    public Window(Game game) {
        super(TITLE);

        this.add(game);
        getContentPane().setPreferredSize(new Dimension(WIDTH, HEIGHT));
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        this.pack();
        setLocationRelativeTo(null);
        setResizable(false);
        setVisible(true);
    }

    public void clear(Graphics2D g) {
        g.setColor(CLEAR);
        g.fillRect(0, 0, WIDTH, HEIGHT);
    }
}
```

  - **Saver**: Làm 2 công việc chính là load dữ liệu (khi mới vào game) và lưu game khi có thay đổi gì đó.

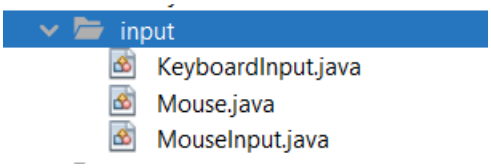
- Có 2 thông số sẽ lưu lại để khi khởi động lại app sẽ có là: Level hiện tại (để tiếp tục chơi) và Setting âm thanh (tắt hay mở).

```java
public class Saver {

    private static String fileName = "data/save.txt";

    public static void load() { ...12 lines }

    public static void save() { ...11 lines }
}
```

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

```
public class Util {

    public static int random(int n) { ...3 lines }

    public static int random(int min, int max) { ...3 lines }

    public static int clamp(int value, int min, int max) { ...9 lines }

    public static boolean inRange(int value, int min, int max) { ...3 lines }

    public static boolean inRange(float value, float min, float max) { ...3 lines }
}
```

b) input



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

```
public class KeyboardInput implements KeyListener {
```

- 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.

```
// get key state
public synchronized boolean isKeyPressed(int keycode) { ...7 lines }

// has key just been pressed?
public synchronized boolean isKeyPressedOnce(int keycode) { ...7 lines }

// has key just been released?
public synchronized boolean isKeyReleased(int keycode) { ...7 lines }
```

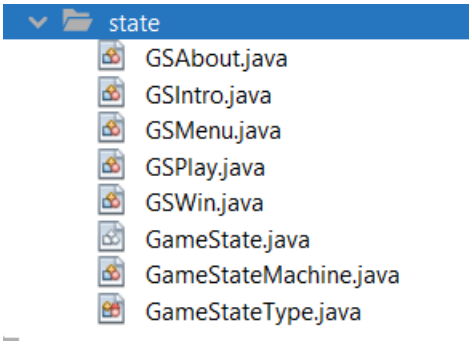
- **MouseInput**: kế thừa từ MouseInputAdapter

```
public class MouseInput extends 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, ...

```
// button codes: 1 - left, 2 - middle (wheel button), 3 - right
public ButtonState getButton(int button) { ...8 lines }

public boolean isMousePressedOnce(int button) { ...7 lines }

public MouseWheelEvent getWheel() { ...3 lines }

public Point getPosition() { ...3 lines }
```

c) 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()

```
public abstract class GameState extends JPanel {

    public static boolean IsChanging;
    protected GameStateType type;

    protected GameState(GameStateType type) { ...4 lines }

    public abstract void update();

    public abstract void render(Graphics2D g);

    public GameStateType getType() { ...3 lines }
```

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

```java
// factory method
public static GameState createState(GameStateType type) {
    GameState state = null;
    switch (type) {
        case Intro ->
            state = new GSIntro(GameStateType.Intro);
        case Menu ->
            state = new GSMenu(GameStateType.Menu);
        case Play ->
            state = new GSPlay(GameStateType.Play);
        case About ->
            state = new GSAbout(GameStateType.About);
        case Win ->
            state = new GSWin(GameStateType.Win);
    }
    return state;
}
```

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

```java
public void changeState(GameStateType type) {
    GameState screen = GameState.createState(type);
    changeState(screen);
}

public void changeState(GameState state) {
    nextState = state;
}

public void pushScreen() {
    states.add(nextState);
    Game.getInstance().add(GameStateMachine.getInstance().getCurrentState());
    nextState = null;
}

public void popState() {
    if (!states.isEmpty()) {
        states.remove(states.size() - 1);
    }
    Game.getInstance().add(GameStateMachine.getInstance().getCurrentState());
}

public boolean needToChangeState() {
    return nextState != null;
}
```

- 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:

```java
GameStateMachine.getInstance().changeState(GameStateType.Menu);
```

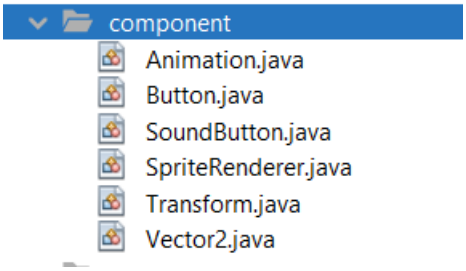
  Trong lần update tiếp theo tại class Game, ta sẽ thực hiện việc push GS mới vào, và update GS mới đó, GS cũ không còn được update nữa.

```java
// from class Game
private void update() {
    if (GameStateMachine.getInstance().needToChangeState()) {
        GameStateMachine.getInstance().pushScreen();
    }
    GameStateMachine.getInstance().getCurrentState().update();

    keyboardInput.update();
    mouseInput.update();
    sceneTransition.update();
}
```

3. 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).
4. 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

```java
GameStateMachine.getInstance().popState();
```

d) 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)

```java
public class Transform {

    private GameObject gameObject;
    public Vector2 position;
    public Vector2 rotation;
    public Vector2 size;
    public Vector2 scale;

    public Transform(GameObject gameObject) {
        this.gameObject = gameObject;
        this.position = new Vector2(0, 0);
        this.rotation = new Vector2(0, 0);
        this.size = new Vector2(0, 0);
        this.scale = new Vector2(1, 1);
    }
}
```

- **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.

```java
public class Vector2 {

    public float x, y;

    public Vector2(float x, float y) {
        this.x = x;
        this.y = y;
    }
}
```

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

```java
public class SpriteRenderer {

    private GameObject gameObject;
    public BufferedImage sprite;

    public SpriteRenderer(GameObject gameObject) {
        this.gameObject = gameObject;
    }

    public void render(Graphics2D g) {
        g.drawImage(sprite, (int) gameObject.transform.position.x, (int) gameObject.transform.position.y,
                (int) gameObject.transform.size.x, (int) gameObject.transform.size.y, null);
    }
}
```

- **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.

```java
public class Animation {

    // tạo ra 1 SpriteRenderer mới, và vẽ dựa trên vị trí của parent chứa Animation
    private SpriteRenderer sprRender;
    private BufferedImage[] spriteSheet;
    private int frameCount, currentFrame = 0;
    private float frameTime, currentTime = 0f;

    public Animation(GameObject parent, BufferedImage[] listSprite, float frameTime) {
        this.sprRender = new SpriteRenderer(parent);
        this.spriteSheet = listSprite;
        this.frameCount = listSprite.length;
        this.frameTime = frameTime;
    }

    public void renew() {
        currentFrame = 0;
        currentTime = 0f;
    }

    public void update() {
        currentTime += Game.DELTA_TIME;
        if (currentTime ⩾ frameTime) {
            currentTime -= frameTime;
            currentFrame = (currentFrame + 1) % frameCount;
        }
    }

    public void render(Graphics2D g) {
        sprRender.sprite = spriteSheet[currentFrame];
        sprRender.render(g);
    }
}
```

- **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)

```
public class Button {

    private Vector2 position, size, round;
    private IAction eventClick;

    private Color activeColor, inactiveColor;
    private boolean active;

    private String text;
    private Font textFont;
    private Color textColor;
    private Vector2 textPos;
```

- 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.

```
public void update() {
    if (!active) {
        return;
    }
    if (!GameState.IsChanging && Game.getInstance().keyboardInput.isKeyPressedOnce(KeyEvent.VK_ENTER)) {
        eventClick.excute();
        return;
    }
    if (!GameState.IsChanging && Game.getInstance().mouseInput.isMousePressedOnce(1) == true) {
        Point mousePos = Game.getInstance().mouseInput.getPosition();
        if (inRange(mousePos.x, position.x, position.x + size.x) && inRange(mousePos.y, position.y, position.y + size.y)) {
            eventClick.excute();
        }
    }
}
```

- 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)

```
public interface IAction {

    void excute();
}
```

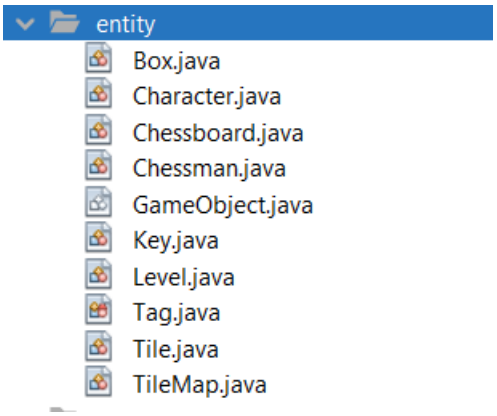
- Button:

```
// continue button from GS Menu
continueButton.setEvent(() → {
    Game.getInstance().sceneTransition.closeScene(() → {
        GameStateMachine.getInstance().changeState(GameStateType.Play);
    });
});
```

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

```
// check win
if (NumOfHero == 0 && !isGameOver) {
    System.out.println("Level: You win!");
    isGameOver = true;
    Game.getInstance().sceneTransition.closeScene(() → {
        if (GSPlay.CurrentLevel == 20) {
            GameStateMachine.getInstance().changeState(GameStateType.Win);
        } else {
            ((GSPlay) (GameStateMachine.getInstance().getCurrentState())).levelUp();
        }
    });
}
```

e) 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().

```
public abstract class GameObject {

    public GameObject parent;
    public Tag tag = Tag.Default;

    public Transform transform;
    public SpriteRenderer sprRender;

    public GameObject() { ...4 lines }

    public GameObject(Vector2 position, Vector2 size) { ...6 lines }

    public GameObject(float x, float y, float width, float height) { ...6 lines }

    public abstract void update();

    public abstract void render(Graphics2D g);
}
```

- 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.

```
public enum Tag {
    Default,
    BlueCharacter,
    RedCharacter,
    GreenCharacter,
    Box,
    Key,
}
```

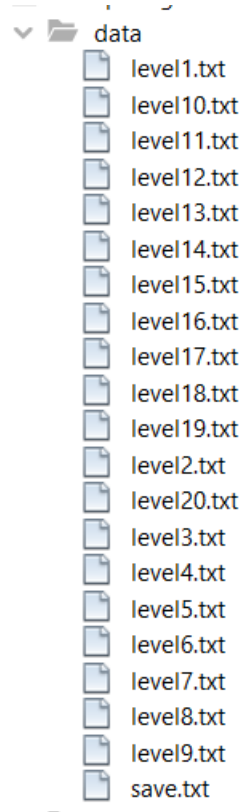
  - **Level**: GamePlay thì sẽ bao gồm level, nó sẽ nhận vào 1 số int là tên level và tiến hành đọc ghi file để lấy dữ liệu, rồi khởi tạo level đó

```java
public class Level extends GameObject {

    public static final float X = 275, Y = 50;
    public static final float WIDTH = 650, HEIGHT = 650;
    public static final int ROW = 13, COL = 13;

    // reset in each level
    public static int NumOfHero;
    public static boolean LockMove;
    public static boolean FinishMove;
    public static boolean OpenKeyDoor;
    public static boolean StandOnSwitch;
    public static boolean OpenSwitchDoor;

    private TileMap tileMap;
    private Chessboard chessboard;
    private boolean isGameOver = false;

    public Level(int index) { ...13 lines }

    private void loadAndInitLevel(int index) { ...27 lines }

    @Override
    public void update() { ...35 lines }

    @Override
    public void render(Graphics2D g) { ...4 lines }
}
```

- Load data form file:

```java
private void loadAndInitLevel(int index) {
    try {
        String fileName = "data/level" + index + ".txt";
        Scanner scanner = new Scanner(new File(fileName));

        // init map
        int[][] dataMap = new int[ROW][COL];
        for (int i = 0; i < ROW; i++) {
            for (int j = 0; j < COL; j++) {
                dataMap[i][j] = scanner.nextInt();
            }
        }
        tileMap = new TileMap(dataMap);

        // init board
        int[][] dataBoard = new int[ROW][COL];
        for (int i = 0; i < ROW; i++) {
            for (int j = 0; j < COL; j++) {
                dataBoard[i][j] = scanner.nextInt();
            }
        }
        chessboard = new Chessboard(dataBoard);

    } catch (FileNotFoundException ex) {
        ex.printStackTrace();
    }
}
```

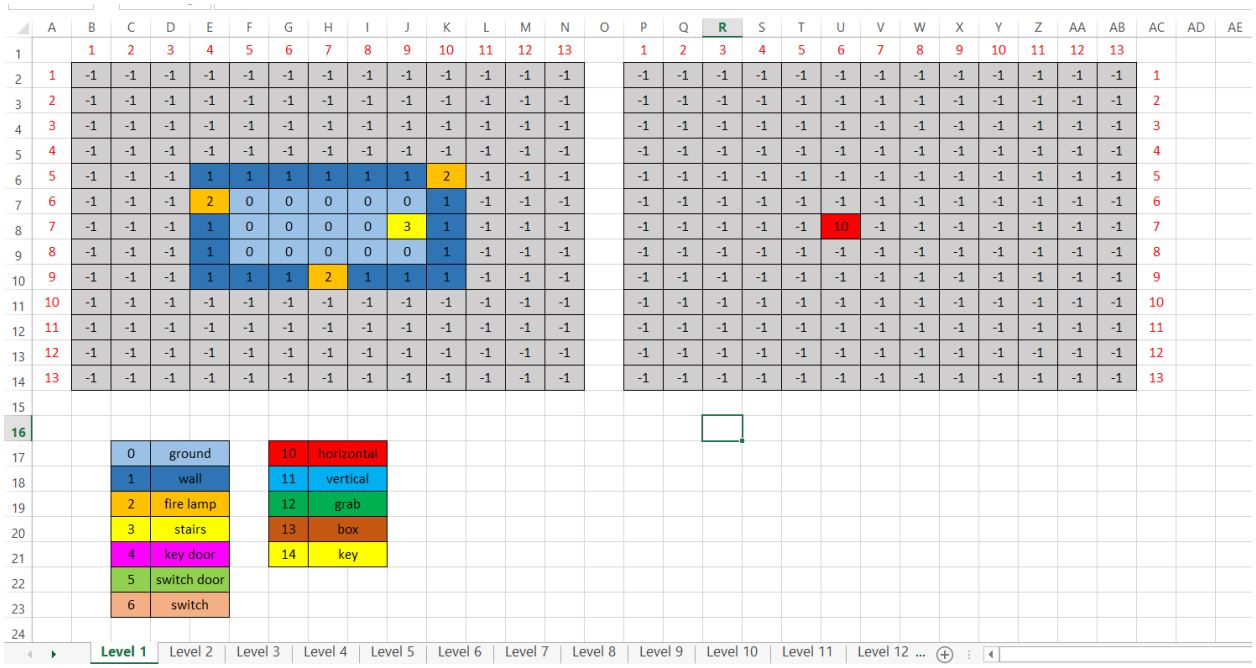
- Data được lưu file text như này:



Source History

1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
2	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
3	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
4	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
5	-1	-1	-1	1	1	1	1	1	1	2	-1	-1	-1
6	-1	-1	-1	2	0	0	0	0	0	1	-1	-1	-1
7	-1	-1	-1	1	0	0	0	0	3	1	-1	-1	-1
8	-1	-1	-1	1	0	0	0	0	0	1	-1	-1	-1
9	-1	-1	-1	1	1	1	2	1	1	1	-1	-1	-1
10	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
11	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
12	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
13	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
14													
15	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
16	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
17	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
18	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
19	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
20	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
21	-1	-1	-1	-1	-1	10	-1	-1	-1	-1	-1	-1	-1
22	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
23	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
24	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
25	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
26	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
27	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

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



0	ground	
1	wall	
2	fire lamp	
3	stairs	
4	key door	
5	switch door	
6	switch	

10	horizontal
11	vertical
12	grab
13	box
14	key

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

```java
public class Chessboard extends GameObject {

    public static int[] dirRow = {-1, 0, 1, 0};
    public static int[] dirCol = {0, 1, 0, -1};

    private Chessman[][] board;
    private Chessman hero;
    private List<Chessman> chessNeedMove;

    public Chessboard(int[][] data) { ...4 lines }

    private void initBoard(int[][] data) { ...27 lines }

    private Chessman findHero(Tag tag) { ...10 lines }

    private void swapChess(Chessman a, Chessman b) { ...6 lines }

    ////////////////////////////////////////////////////
    public void handleMoving(int dr, int dc) { ...22 lines }

    private void findConnects(int curRow, int curCol) { ...12 lines }

    private void findObstacles(int curRow, int curCol, int dr, int dc) { ...10 lines }

    private boolean canMove(int dr, int dc) { ...56 lines }

    private void moveChess(int dr, int dc) { ...48 lines }

    ////////////////////////////////////////////////////
    public void handleConnecting() { ...11 lines }

    private void connectChess(int row, int col) { ...16 lines }

    private void disconnectChess(int row, int col) { ...10 lines }

    private void reloadConnect() { ...18 lines }

    private void checkUnlockSwitchDoor() { ...14 lines }

    ////////////////////////////////////////////////////
    @Override
    public void update() { ...16 lines }

    @Override
    public void render(Graphics2D g) { ...7 lines }
}
```

- 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, ..)

```
    protected int row, col;
    protected float speed = 320;
    protected boolean moving = false;
    protected boolean connecting = false;

    public Chessman(int row, int col, Tag tag) { ...7 lines }

    private void initView() { ...10 lines }

    protected void move(int dr, int dc) { ...7 lines }

    protected void stop() { ...13 lines }

    protected void connect() { ...3 lines }

    protected void disconnect() { ...3 lines }

    @Override
    public void update() { ...22 lines }

    @Override
    public void render(Graphics2D g) { ...11 lines }
}
```

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

```
public class Character extends Chessman {

public class Box extends Chessman {

public class Key extends Chessman {
```

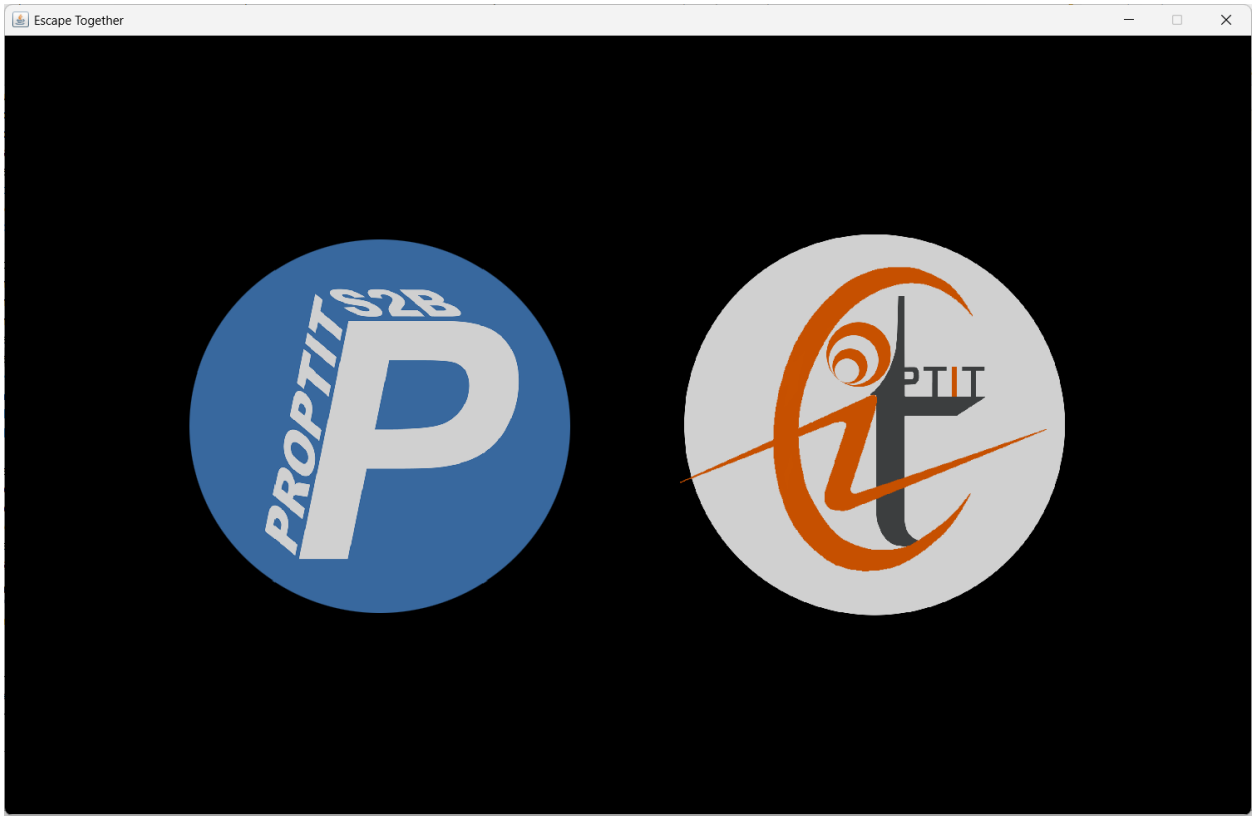
5. Phương pháp – Ngôn ngữ - Công cụ

- 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

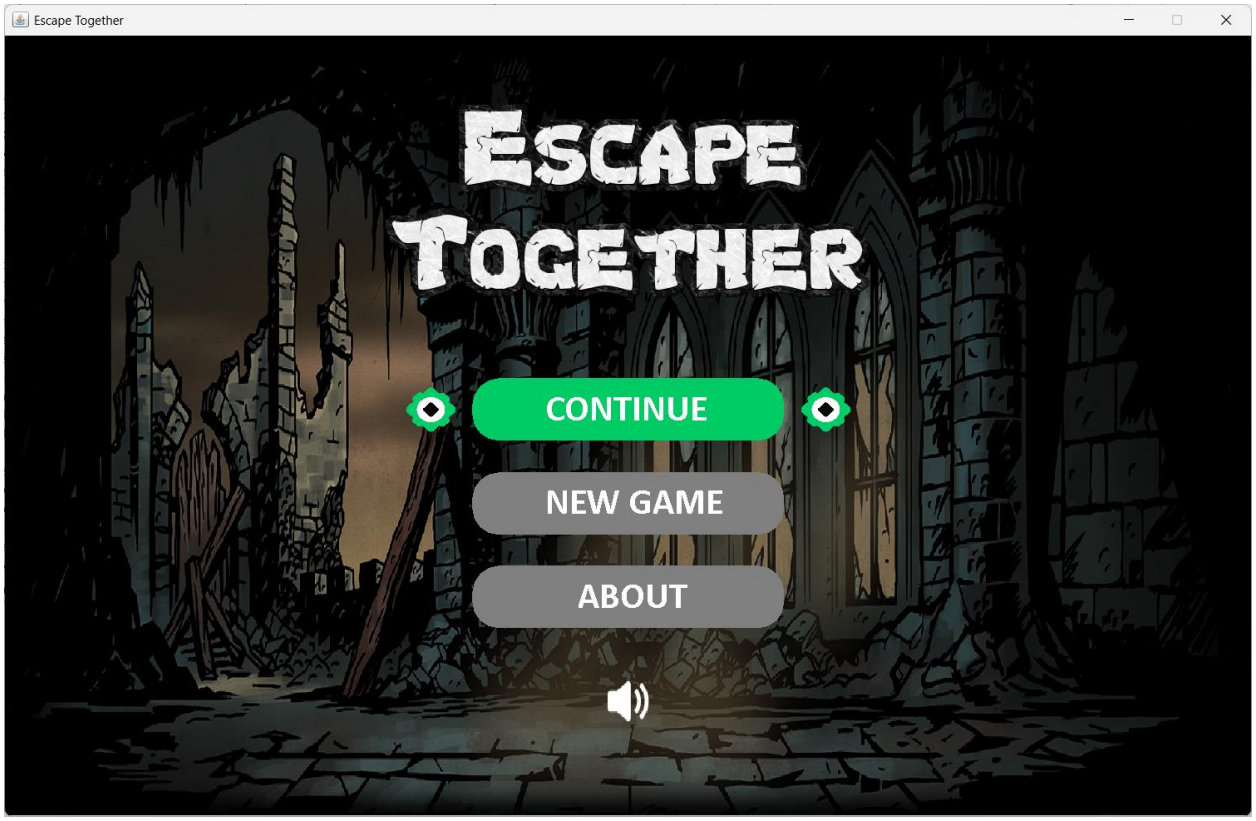
- Photoshop: chỉnh sửa 1 số ảnh

6. Giới thiệu hình ảnh kết quả

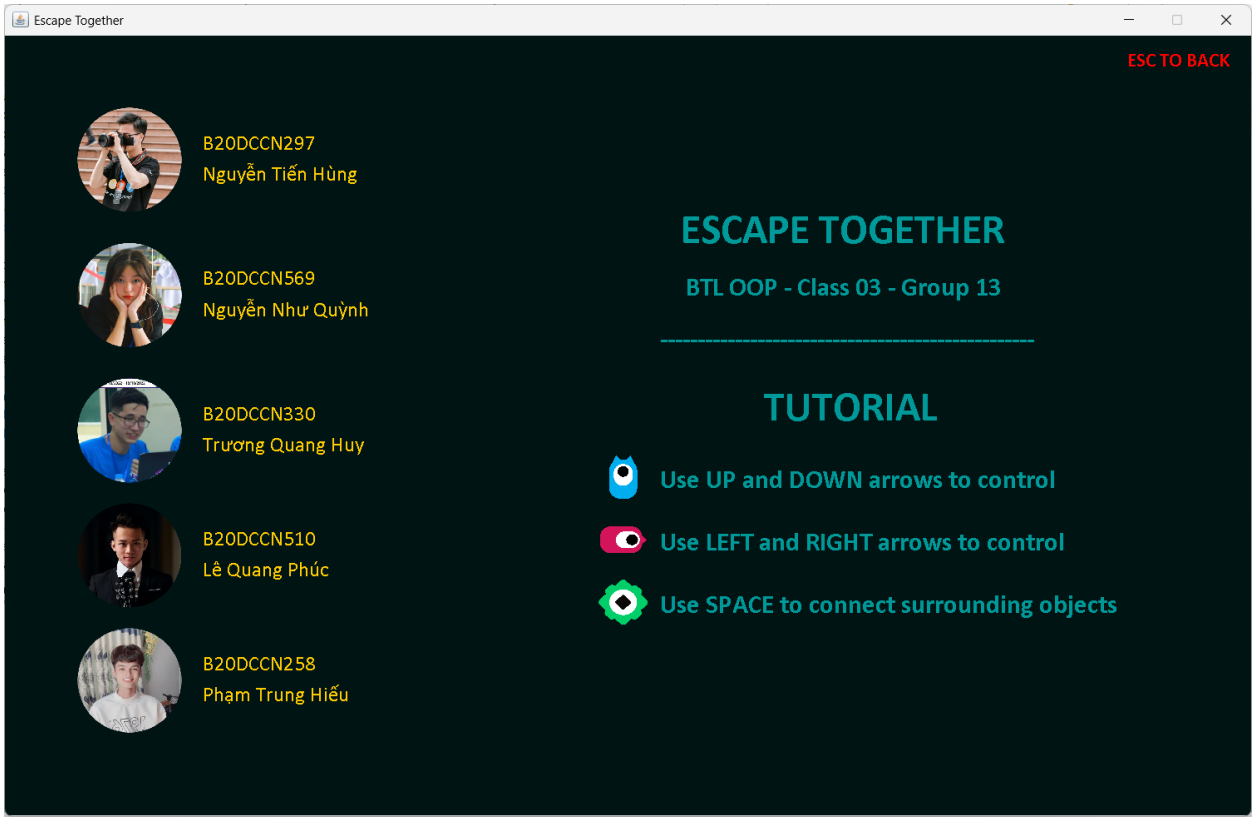
- Intro:



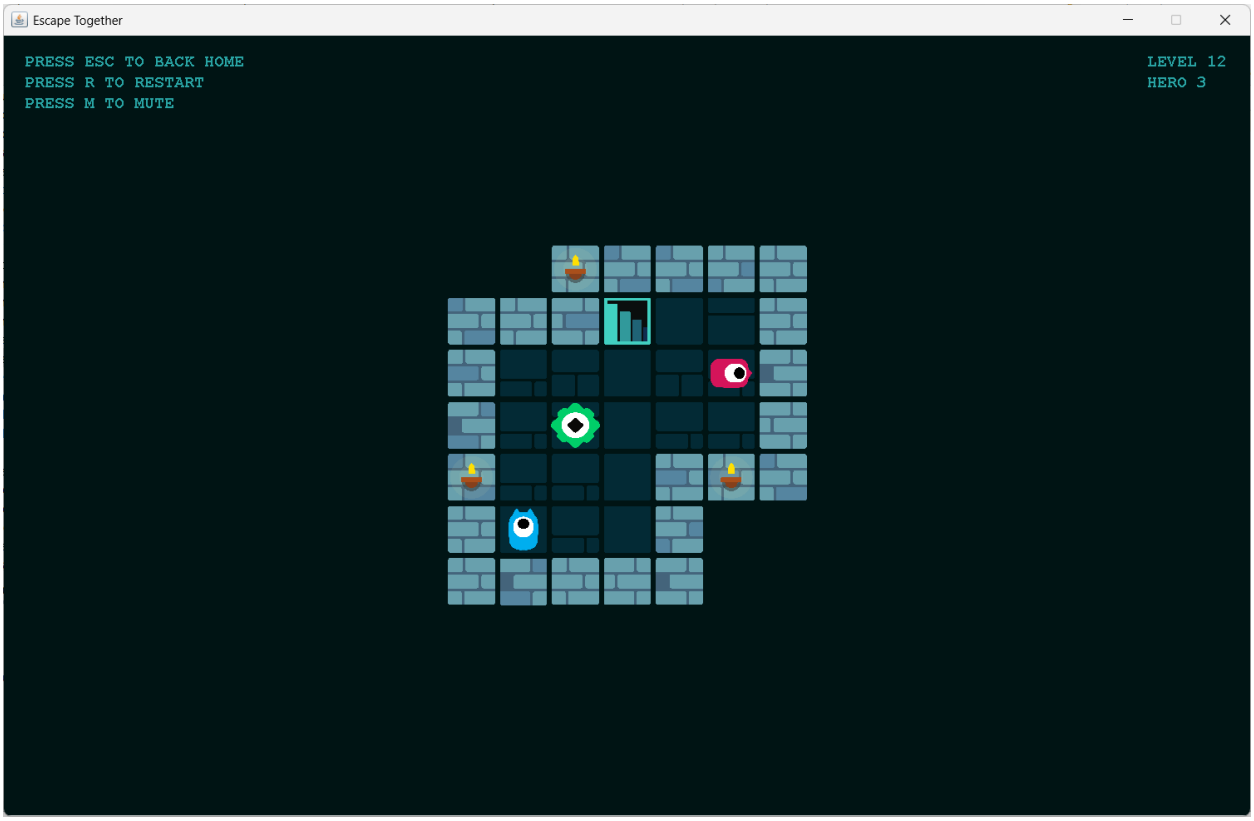
- Menu:

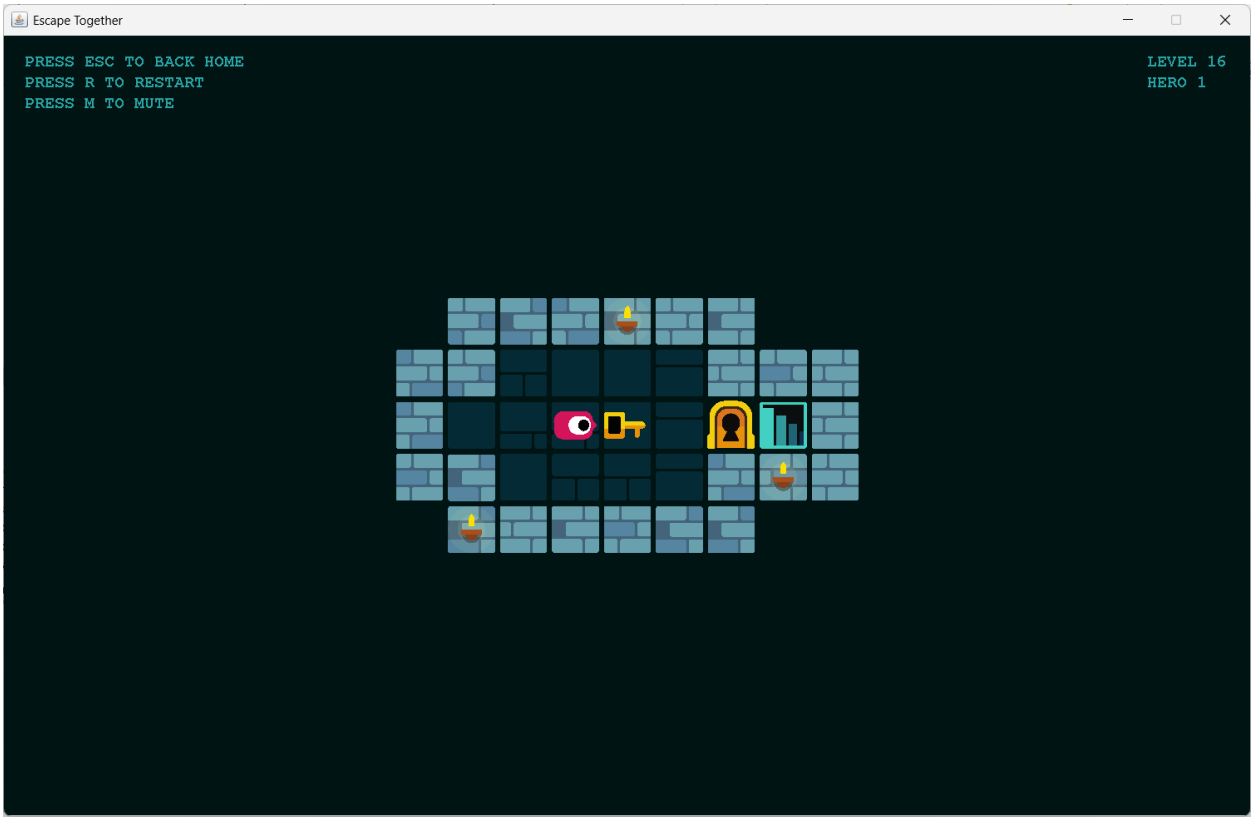


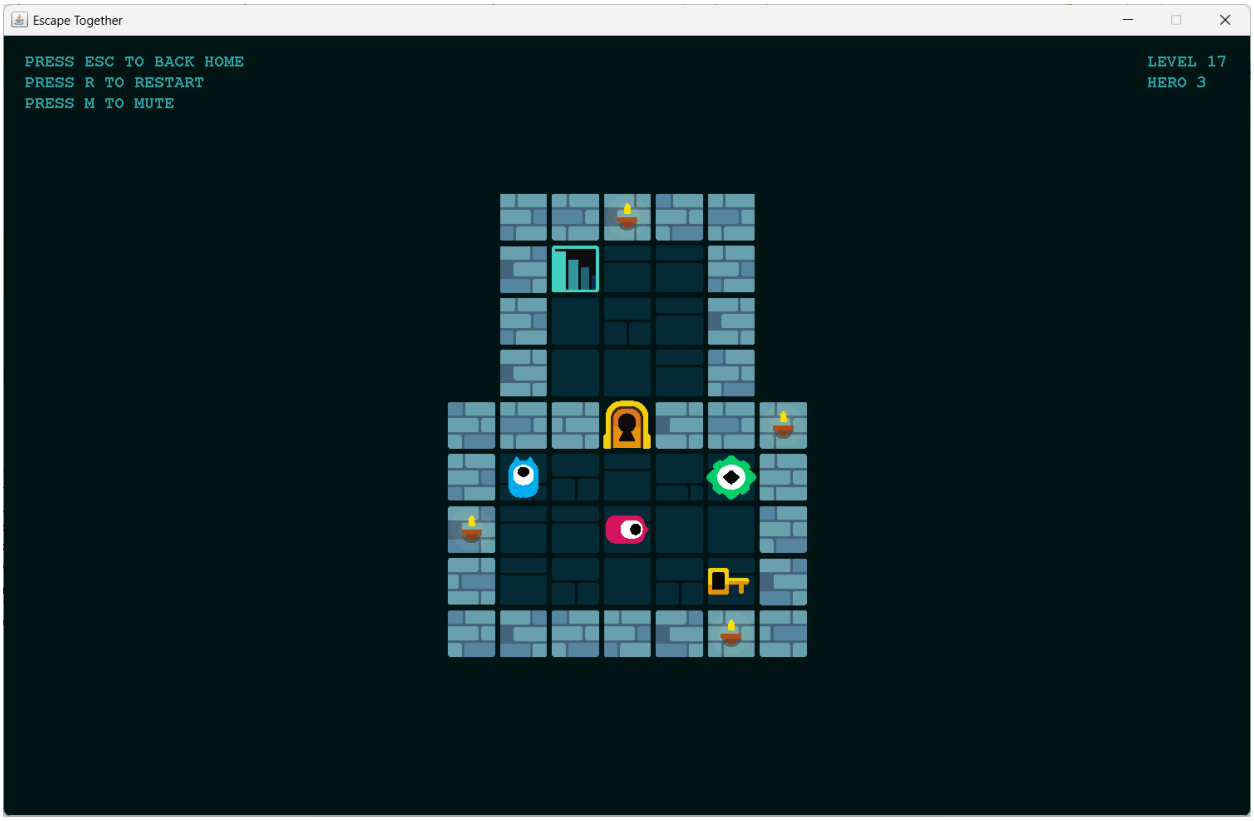
- About:


Escape Together
ESC TO BACK
B20DCCN297
Nguyễn Tiến Hùng
B20DCCN569
Nguyễn Như Quỳnh
B20DCCN330
Trương Quang Huy
B20DCCN510
Lê Quang Phúc
B20DCCN258
Phạm Trung Hiếu
ESCAPE TOGETHER
BTL OOP - Class 03 - Group 13
TUTORIAL
Use UP and DOWN arrows to control
Use LEFT and RIGHT arrows to control
Use SPACE to connect surrounding objects

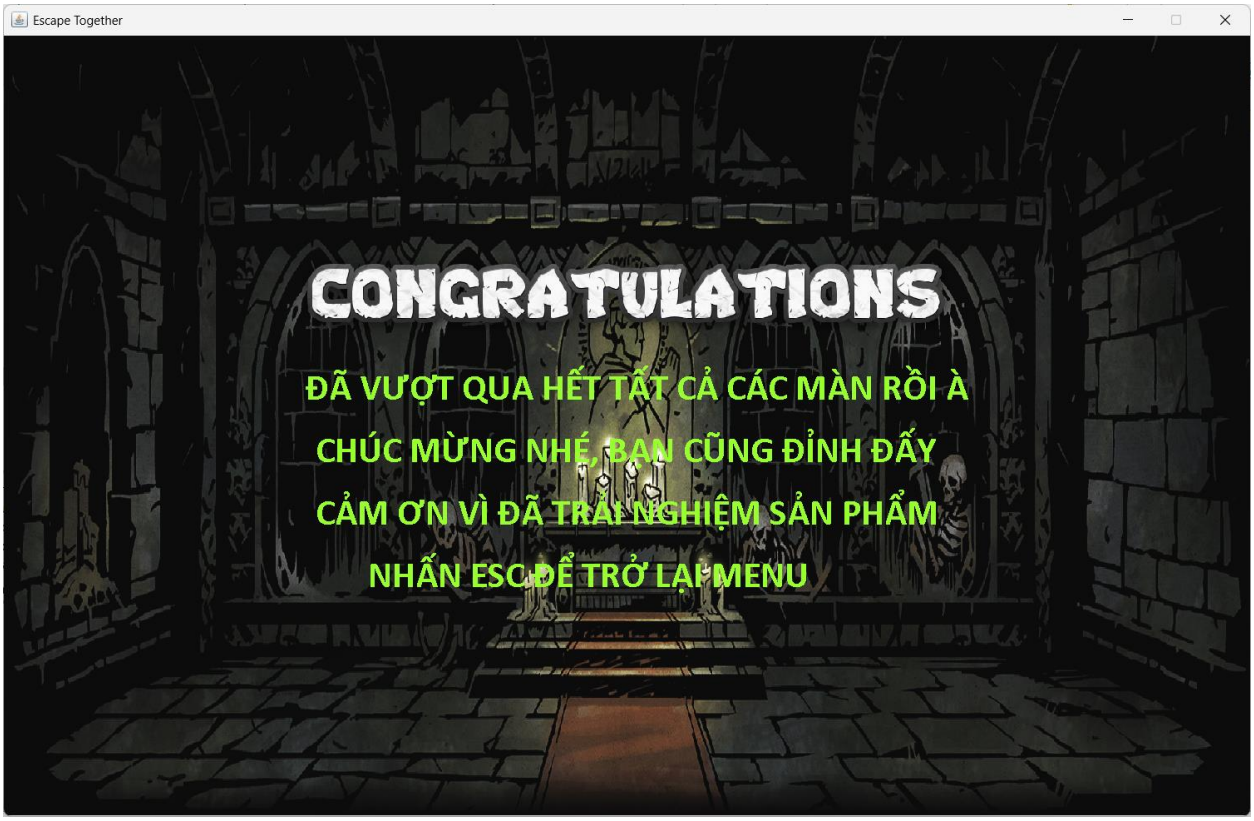
- Play:


Escape Together
PRESS ESC TO BACK HOME
PRESS R TO RESTART
PRESS M TO MUTE
LEVEL 12
HERO 3


Escape Together
PRESS ESC TO BACK HOME
PRESS R TO RESTART
PRESS M TO MUTE
LEVEL 16
HERO 1


Escape Together
PRESS ESC TO BACK HOME
PRESS R TO RESTART
PRESS M TO MUTE
LEVEL 17
HERO 3

- Win:



7. 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