TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI
UNIVERSITY OF TRANSPOST AND COMMUNICATIONS

BÁO CÁO BÀI TẬP LỚN
MÔN CÔNG NGHỆ JAVA

ĐỀ TÀI: Jump Game

Giảng viên hướng dẫn: Vũ Huấn
Sinh viên thực hiện:

Họ và tên	Mã sinh viên
Nguyễn Duy Cương	211212717
Mai Tiến Mạnh	211203216

Lớp: CNTT5-K62(N07)

I. Giới thiệu về Jump Game

Jump Game là một trò chơi điện tử thuộc thể loại platformer. Trong trò chơi, người chơi sẽ điều khiển nhân vật nhảy lên các tảng đá và nhảy không giới hạn. Tuy nhiên, việc nhảy lên các tảng đá không hề dễ dàng, người chơi phải tập trung và có kĩ năng nhảy tốt để vượt qua các chướng ngại vật. Trò chơi được đánh giá là khá khó và đòi hỏi sự kiên nhẫn của người chơi.

II. Ý tưởng Jump Game

- Xây dựng Jump Game bằng công cụ Eclipse với ngôn ngữ lập trình Java.
- Luật chơi:
  + Xây dựng map và các tảng đá có vị trí hợp lí.
  + Người chơi sẽ điều khiển nhân vật sang trái sang phải bằng các phím A, D và nhảy lên bằng phím Space.
  + Mục tiêu của trò chơi là chúng ta sẽ nhảy đến điểm cao nhất có thể. Ai nhảy cao nhất sẽ là người chiến thắng

III. Phân tích và thiết kế Jump Game

1. Yêu cầu của phần mềm

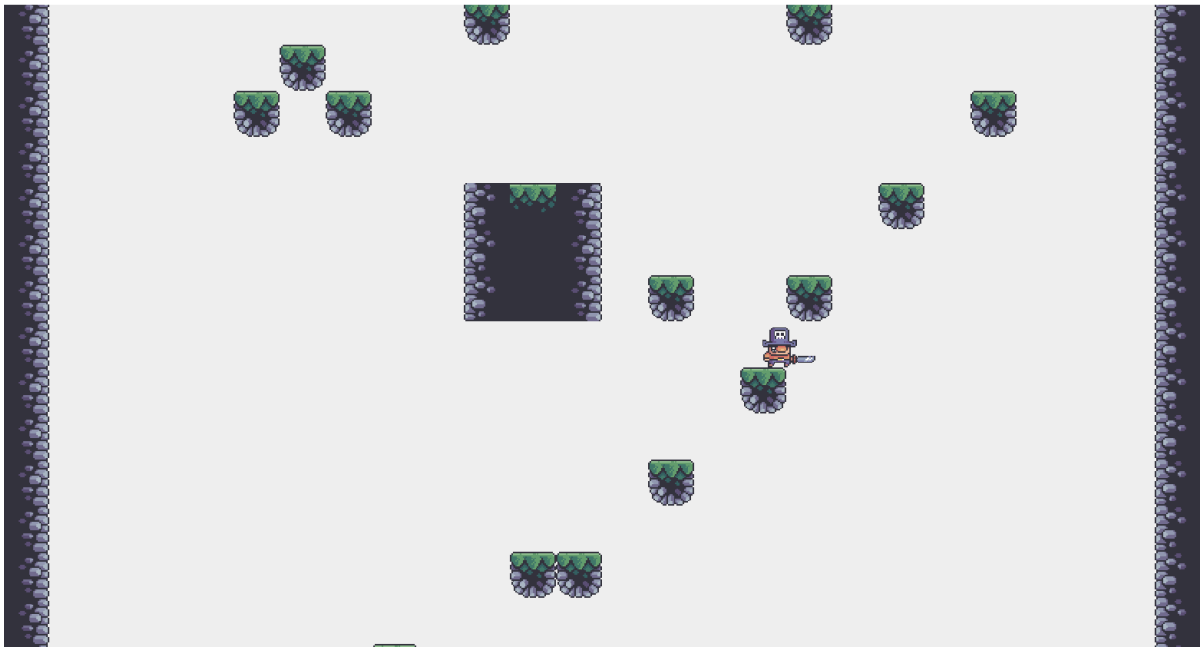
- Yêu cầu chức năng:
  + Phần mềm chạy ổn định, mượt mà.
  + Tốc độ xử lí nhanh.
  + Việc cài đặt và sử dụng phần mềm đơn giản, không yêu cầu máy phải có cấu hình cao.
- Yêu cầu phi chức năng:
  + Giao diện xử lí bắt mắt nhưng không quá cầu kì, giúp người chơi thoải mái khi sử dụng.
  + Phần mềm dễ dàng bảo trì, sửa chữa khi gặp lỗi.

2. Yêu cầu với người chơi

- Ở đây em xây dựng game này phù hợp với nhiều đối tượng chơi.
- Trong trò chơi, người chơi phải có kĩ năng nhảy tốt và tập trung để vượt qua các thử thách và tránh đường các chướng ngại vật. Ngoài ra người chơi cần có sự kiên nhẫn và sự cẩn thận để không bị rơi xuống và phải bắt đầu lại từ đầu.
- Trò chơi cũng yêu cầu người chơi phải có khả năng đánh giá và lựa chọn đúng thời điểm để nhảy lên các tảng đá và tránh các chướng ngại vật. Nếu không sẽ bị rơi xuống và bắt đầu lại từ đầu.

IV. Thiết kế giao diện

- Giao diện: Phần mềm thiết kế với giao diện đơn giản, rõ rang giúp người chơi dễ sử dụng và thực hiện thao tác chơi, trong quá trình chơi, game được tích hợp các âm thanh, hình ảnh, giúp người chơi có những phút thư giãn, thoải mái.
- Hình ảnh của Jump Game:



V. Các thuật toán sử dụng

1. Game loop

- Đây là một đoạn code trong Java để thực hiện game loop, cụ thể là vòng lặp game. Đoạn code này được chạy liên tục trong suốt thời gian chạy của game để cập nhật và vẽ các frame.
- Các biến được khởi tạo trên đầu là timePerFrame và timePerUpdate. Chúng là khoảng thời gian (tính theo nano giây) cho mỗi frame và mỗi update. FPS_SET và UPS_SET là tần số cập nhật khung hình và tần số cập nhật trạng thái (updates) được đặt sẵn để thực hiện trên mỗi giây.
- Trong vòng lặp, ta tính toán thời gian đã trôi qua kể từ khi vòng lặp bắt đầu (previousTime) bằng cách sử dụng phương thức System.nanoTime(). Ta tính delta time (deltaU và deltaF) cho mỗi frame và mỗi update bằng cách chia thời gian đã trôi qua cho timePerFrame và timePerUpdate.
- Sau đó, ta kiểm tra xem đã đến lúc cập nhật hay vẽ frame hay chưa. Nếu deltaU vượt quá giá trị 1, ta thực hiện cập nhật game và giảm deltaU đi 1. Tương tự, nếu deltaF vượt quá giá trị 1, ta vẽ một frame mới và giảm deltaF đi 1.

- Cuối cùng, ta kiểm tra xem đã đến lúc hiển thị thông tin FPS và UPS hay chưa. Nếu đã đến, ta in ra thông tin đó và reset giá trị các biến frames và updates.

```java
public void run() {

    double timePerFrame = 1000000000.0 / FPS_SET;
    double timePerUpdate = 1000000000.0 / UPS_SET;

    long previousTime = System.nanoTime();

    int frames = 0;
    int updates = 0;
    long lastCheck = System.currentTimeMillis();

    double deltaU = 0;
    double deltaF = 0;

    while (true) {
        long currentTime = System.nanoTime();

        deltaU += (currentTime - previousTime) / timePerUpdate;
        deltaF += (currentTime - previousTime) / timePerFrame;
        previousTime = currentTime;

        if (deltaU >= 1) {
            update();
            updates++;
            deltaU--;
        }

        if (deltaF >= 1) {
            gamePanel.repaint();
            frames++;
            deltaF--;
        }

        if (System.currentTimeMillis() - lastCheck >= 1000) {
            lastCheck = System.currentTimeMillis();
            System.out.println("FPS: " + frames + " | UPS: " + updates);
            frames = 0;
            updates = 0;

        }
    }

}
```

Vòng lặp này sẽ tiếp tục chạy vô hạn cho đến khi ta dừng game.

2. Camera chạy theo nhân vật

- Đây là một phương thức để kiểm tra xem nhân vật của người chơi có gần đến biên của bản đồ không và điều chỉnh vị trí của bản đồ để người chơi có thể thấy được phần bản đồ tiếp theo.

- diff là khoảng cách giữa vị trí người chơi và đường biên gần nhất của màn hình.
- yLvlOffset là độ lệch tọa độ y hiện tại của màn hình so với trục y thực sự của trò chơi.
- maxLvlOffsetX đại diện cho giá trị tối đa của yLvlOffset để đảm bảo rằng người chơi không bị di chuyển quá xa khỏi trục y thực sự của trò chơi.
- Lấy vị trí của người chơi theo trục y và tính khoảng cách giữa vị trí hiện tại của người chơi và vị trí của bản đồ.
- Nếu khoảng cách lớn hơn giá trị uptBorder (biên trên của bản đồ), điều chỉnh vị trí bản đồ sao cho phần còn lại của bản đồ được hiển thị ở phía trên nhân vật.
- Nếu khoảng cách nhỏ hơn giá trị downBorder (biên dưới của bản đồ), điều chỉnh vị trí bản đồ sao cho phần còn lại của bản đồ được hiển thị ở phía dưới nhân vật.
- Kiểm tra xem vị trí mới của bản đồ có vượt quá biên của bản đồ hay không, nếu có thì điều chỉnh lại để không vượt quá biên.

```
private void checkCloseToBorder() {
    int playerY = (int) player.getHitbox().y;
    int diff = playerY - yLvlOffset;

    if (diff > uptBorder)
        yLvlOffset += diff - uptBorder;
    else if (diff < downBorder)
        yLvlOffset += diff - downBorder;

    if (yLvlOffset > maxLvlOffsetY)
        yLvlOffset = maxLvlOffsetY;
    else if (yLvlOffset < 0)
        yLvlOffset = 0;

}
```

3. LoadAnimation

- Đoạn code này liên quan đến việc điều khiển chuyển động và hoạt hình cho nhân vật trong trò chơi.
- Hàm updateAnimationTick() được sử dụng để cập nhật chỉ số của hình ảnh hiện tại trong chuỗi hình ảnh của hoạt hình, được lưu trữ trong biến aniIndex. Nếu aniIndex vượt quá số lượng hình ảnh trong chuỗi hoạt hình, aniIndex sẽ được đặt lại thành 0 và biến attacking sẽ được đặt thành false.

```
private void updateAnimationTick() {
    aniTick++;
    if (aniTick >= aniSpeed) {
        aniTick = 0;
        aniIndex++;
        if (aniIndex >= GetSpriteAmount(playerAction)) {
            aniIndex = 0;
            attacking = false;
        }
    }

}
```

- Hàm setAnimation() được sử dụng để xác định loại hoạt hình phù hợp để hiển thị cho nhân vật dựa trên các trạng thái hiện tại của nhân vật như di chuyển, nhảy hay tấn công. Nếu trạng thái hoạt hình mới khác với trạng thái hoạt hình hiện tại, resetAniTick() sẽ được gọi để đặt lại chỉ số của hoạt hình về 0

```
private void setAnimation() {
    int startAni = playerAction;

    if (moving)
        playerAction = RUNNING;
    else
        playerAction = IDLE;

    if (inAir) {
        if (airSpeed < 0)
            playerAction = JUMP;
        else
            playerAction = FALLING;
    }

    if (attacking)
        playerAction = ATTACK_1;

    if (startAni != playerAction)
        resetAniTick();
}
```

- Hàm resetAniTick() được sử dụng để đặt lại chỉ số của hoạt hình về 0 khi hoạt hình của nhân vật được thay đổi.

```
private void resetAniTick() {
    aniTick = 0;
    aniIndex = 0;
}
```

- Hàm updatePos() được sử dụng để cập nhật vị trí của nhân vật dựa trên các hành động hiện tại của người chơi như di chuyển và nhảy. Nếu nhân vật đang trong trạng thái nhảy và không nằm trên mặt đất, airSpeed sẽ được cập nhật theo trọng lực để giảm tốc độ di chuyển của nhân vật. Nếu nhân vật chạm đất, airSpeed sẽ được đặt lại và vị trí của nhân vật sẽ được cập nhật. Nếu người chơi không di chuyển hoặc nhân vật không ở trên không, hàm sẽ không thực hiện bất kỳ thao tác nào.
- inAir là một biến Boolean (kiểu dữ liệu chỉ có thể có giá trị true hoặc false), được sử dụng để xác định xem đối tượng đang nằm trên mặt đất hay đang trong không trung.
- Trong phương thức updatePos(), inAir được kiểm tra để xem đối tượng có đang ở trên mặt đất hay không. Nếu đối tượng không ở trên mặt đất (được xác định bằng phương thức IsEntityOnFloor()), thì inAir được đặt thành true, và đối tượng được coi là đang trong không trung.
- Nếu inAir là true, thì đối tượng sẽ tiếp tục rơi xuống theo tốc độ airSpeed (được tính bằng cách cộng thêm hằng số gravity vào mỗi khung hình). Nếu đối tượng vẫn có thể di chuyển xuống dưới mà không va chạm với bất kỳ vật cản nào (được kiểm tra bằng phương thức CanMoveHere()), thì hitbox.y (vị trí đối tượng theo trục y) được tăng lên theo tốc độ airSpeed, và đối tượng được cập nhật lại vị trí theo trục x bằng updateXPos(xSpeed).
- Nếu đối tượng va chạm với một vật cản nào đó khi đang rơi xuống (được kiểm tra bằng phương thức CanMoveHere()), thì vị trí của đối tượng sẽ được điều chỉnh để đối tượng đứng trên mặt đất hoặc sàn nhà, được tính toán bằng phương thức GetEntityYPosUnderRoofOrAboveFloor(). Sau đó, nếu airSpeed vẫn lớn hơn 0 (tức là đối tượng vẫn đang rơi xuống), inAir được đặt thành false và airSpeed được đặt lại thành 0 (đối tượng đã đạt đến mặt đất). Nếu airSpeed là 0 (tức là đối tượng đang đứng yên trên mặt đất), thì airSpeed được đặt lại thành giá trị fallSpeedAfterCollision. Sau đó, đối tượng được cập nhật lại vị trí theo trục x bằng updateXPos(xSpeed).
- Nếu inAir là false (tức là đối tượng đang nằm trên mặt đất), đối tượng sẽ chỉ được cập nhật lại vị trí theo trục x bằng updateXPos(xSpeed).

```
private void updatePos() {
    moving = false;

    if (jump)
        jump();
    if (!left && !right && !inAir)
        return;

    float xSpeed = 0;

    if (left) {
        xSpeed -= playerSpeed;
        flipX = width;
        flipW = -1;
    }
    if (right) {
        xSpeed += playerSpeed;
        flipX = 0;
        flipW = 1;
    }

    if (!inAir)
        if (!IsEntityOnFloor(hitbox, lvlData))
            inAir = true;

    if (inAir) {
        if (CanMoveHere(hitbox.x, hitbox.y + airSpeed, hitbox.width, hitbox.height, lvlData)) {
            hitbox.y += airSpeed;
            airSpeed += gravity;
            updateXPos(xSpeed);
        } else {
            hitbox.y = GetEntityYPosUnderRoofOrAboveFloor(hitbox, airSpeed);
            if (airSpeed > 0)
                resetInAir();
            else
                airSpeed = fallSpeedAfterCollision;
            updateXPos(xSpeed);
        }

    } else
        updateXPos(xSpeed);
    moving = true;
    if(jump && !inAir) {
```

```
        if(jump && !inAir) {
            xSpeed = -jumpSpeed;
            inAir = true;
        }else if(jump && !doubleJumped && inAir) {
            xSpeed = - jumpSpeed;
            doubleJumped = true;
        }
    }
```

```
private void updateXPos(float xSpeed) {
    if (CanMoveHere(hitbox.x + xSpeed, hitbox.y, hitbox.width, hitbox.height, lvlData)) {
        hitbox.x += xSpeed;
    } else {
        hitbox.x = GetEntityXPosNextToWall(hitbox, xSpeed);
    }

}
```

- Hàm resetInAir() được sử dụng để đặt lại trạng thái của nhân vật về đang đứng trên mặt đất sau khi nhân vật đã kết thúc trạng thái nhảy.

```
private void resetInAir() {
    inAir = false;
    airSpeed = 0;

}
```

- jump()

  Trong hệ tọa độ của Java, tọa độ y tăng lên theo hướng xuống dưới màn hình, vì vậy giá trị của tốc độ nhảy là âm nghĩa là đối tượng sẽ di chuyển lên trên màn hình. Khi đối tượng đạt đến độ cao tối đa và bắt đầu rơi xuống, tốc độ nhảy được tăng lên dần để giảm độ cao khi rơi xuống. Sau đó, khi đối tượng chạm đất, tốc độ nhảy được đặt lại về 0 để đối tượng đứng yên trên đất.

```
private void jump() {
    if (inAir)
        return;
    inAir = true;
    airSpeed = jumpSpeed;

}
```

4. Create map

- Trong phần này của code, hàm GetLevelData() sử dụng phương thức GetSpriteAtlas() để tải một hình ảnh chứa các thông tin về bản đồ cấp độ, được lưu trữ trong biến LEVEL_ONE_DATA. Sau đó, hàm này tạo ra một mảng hai chiều lvlData với kích thước bằng với tỷ lệ kích thước của hình ảnh bản đồ cấp độ. Mỗi phần tử của mảng lvlData được khởi tạo với một giá trị số nguyên được lấy từ một giá trị màu của hình ảnh đó.
- Trong lệnh lặp for, với mỗi pixel của hình ảnh, ta lấy một đối tượng Color từ hàm getRGB() của hình ảnh, sau đó lấy giá trị màu đỏ của pixel đó bằng cách sử dụng phương thức getRed() của đối tượng Color. Nếu giá trị đó lớn hơn hoặc bằng 48 (giá trị được chọn dựa trên màu sắc của hình ảnh), giá trị của pixel sẽ được đặt thành 0, còn không thì giá trị đó được giữ nguyên.
- Sau khi các giá trị số nguyên được lưu trữ trong mảng lvlData, nó được trả về để được sử dụng cho việc vẽ bản đồ cấp độ.
- Dùng màu đỏ để lưu vị trí của vật thể. Cách lựa chọn màu để mã hóa dữ liệu của map tùy thuộc vào người viết code và các yêu cầu của ứng dụng. Trong trường hợp này, tác giả đã chọn màu đỏ vì nó có thể dễ dàng được phân biệt và dễ nhìn thấy trong hình ảnh, và vì đó là màu cơ bản trong bộ ba màu RGB.(0 - 255 in value)
- Hàm importOutsideSprites() được sử dụng để tải các sprite (hình ảnh) cho các vật phẩm và đối tượng khác nhau trong game. Các sprite này được lưu trữ trong một hình ảnh lớn, và sau đó được cắt ra thành các sprite nhỏ để sử dụng trong game.

- Hàm draw() được sử dụng để vẽ bản đồ màn chơi lên màn hình. Với mỗi phần tử trong ma trận bản đồ, hàm sử dụng một chỉ số sprite được tính toán bằng hàm getSpriteIndex() để lấy sprite tương ứng từ mảng levelSprite. Sau đó, sprite này được vẽ lên màn hình bằng hàm drawImage() của đối tượng Graphics. Vị trí của sprite trên màn hình được tính toán dựa trên vị trí của ô trên bản đồ và lvlOffset, đại diện cho sự thay đổi vị trí của màn hình khi người chơi di chuyển qua lại.

```
public LevelManager(Game game) {
    this.game = game;
    importOutsideSprites();
    levelOne = new Level(LoadSave.GetLevelData());
}

// get side atlas
private void importOutsideSprites() {
    BufferedImage img = LoadSave.GetSpriteAtlas(LoadSave.LEVEL_ATLAS);
    levelSprite = new BufferedImage[48];
    for (int j = 0; j < 4; j++)
        for (int i = 0; i < 12; i++) {
            int index = j * 12 + i;
            levelSprite[index] = img.getSubimage(i * 32, j * 32, 32, 32);
        }
}
// draw data map
public void draw(Graphics g, int lvlOffset) {
    for (int j = 0; j < levelOne.getLevelData().length; j++)
        for (int i = 0; i < levelOne.getLevelData()[0].length; i++) {
            int index = levelOne.getSpriteIndex(i, j);
            g.drawImage(levelSprite[index], Game.TILES_SIZE * i, Game.TILES_SIZE *j - lvlOffset, Game.TILES_SIZE, Game.TILES_SIZE, null);
        }
}
```

5. Hitbox Checker

- Đây là một class chứa một số phương thức hỗ trợ cho game, và đoạn code này định nghĩa các phương thức để xử lý di chuyển của đối tượng trong game.
- Phương thức CanMoveHere(): Phương thức này kiểm tra xem có thể di chuyển tới vị trí mới (x, y) được chỉ định hay không. Nó kiểm tra xem 4 góc của hitbox có nằm trên các ô không gian không, nếu không thì trả về true, ngược lại trả về false.
- Phương thức IsSolid(): Phương thức này kiểm tra xem vị trí (x, y) có nằm trên một ô không gian bất động hay không. Nó kiểm tra giá trị của ô tương ứng trong mảng lvlData, nếu giá trị này lớn hơn hoặc bằng 48, nhỏ hơn 0 hoặc không phải giá trị 11 thì trả về true, ngược lại trả về false.
- Phương thức GetEntityXPosNextToWall(): Phương thức này trả về vị trí x của đối tượng khi nó tiếp xúc với tường. Nếu đối tượng đang di chuyển sang phải, phương thức tính toán vị trí của ô tường gần nhất bên trái của hitbox, trừ đi chiều rộng của hitbox và trả về giá trị này. Nếu đối tượng đang di chuyển sang trái, phương thức tính toán vị trí của ô tường gần nhất bên phải của hitbox và trả về giá trị này.
- Phương thức GetEntityYPosUnderRoofOrAboveFloor(): Phương thức này trả về vị trí y của đối tượng khi nó tiếp xúc với nóc hoặc sàn. Nếu đối tượng đang rơi xuống (airSpeed > 0), phương thức tính toán vị trí của ô sàn gần nhất phía trên của hitbox, trừ đi chiều cao của hitbox và trả về giá trị này. Nếu đối tượng đang nhảy lên (airSpeed <= 0), phương thức tính toán vị trí của ô nóc gần nhất phía dưới của hitbox và trả về giá trị này.

- Phương thức IsEntityOnFloor(): Phương thức này kiểm tra xem đối tượng có đang đứng trên sàn hay không. Nó kiểm tra giá trị của hai ô nằm dưới góc dưới bên trái và dưới bên phải của hitbox.

```
public static boolean CanMoveHere(float x, float y, float width, float height, int[][] lvlData) {
    if (!IsSolid(x, y, lvlData))
        if (!IsSolid(x + width, y + height, lvlData))
            if (!IsSolid(x + width, y, lvlData))
                if (!IsSolid(x, y + height, lvlData))
                    return true;
    return false;
}

private static boolean IsSolid(float x, float y, int[][] lvlData) {
    int maxHeight = lvlData[0].length * Game.TILES_SIZE;
    if (x < 0 || x >= Game.GAME_WIDTH)
        return true;
    if (y < 0 || y >= maxHeight)
        return true;

    float xIndex = x / Game.TILES_SIZE;
    float yIndex = y / Game.TILES_SIZE;

    int value = lvlData[(int) yIndex][(int) xIndex];

    if (value >= 48 || value < 0 || value != 11)
        return true;
    return false;
}

public static float GetEntityXPosNextToWall(Rectangle2D.Float hitbox, float xSpeed) {
    int currentTile = (int) (hitbox.x / Game.TILES_SIZE);
    if (xSpeed > 0) {
        // Right
        int tileXPos = currentTile * Game.TILES_SIZE;
        int xOffset = (int) (Game.TILES_SIZE - hitbox.width);
        return tileXPos + xOffset - 1;
    } else
        // Left
        return currentTile * Game.TILES_SIZE;
}
```

```
public static float GetEntityYPosUnderRoofOrAboveFloor(Rectangle2D.Float hitbox, float airSpeed) {
    int currentTile = (int) (hitbox.y / Game.TILES_SIZE);
    if (airSpeed > 0) {
        // Falling - touching floor
        int tileYPos = currentTile * Game.TILES_SIZE;
        int yOffset = (int) (Game.TILES_SIZE - hitbox.height);
        return tileYPos + yOffset - 1;
    } else
        // Jumping
        return currentTile * Game.TILES_SIZE;

}

public static boolean IsEntityOnFloor(Rectangle2D.Float hitbox, int[][] lvlData) {
    // Check the pixel below bottom_left and bottom_right
    if (!IsSolid(hitbox.x, hitbox.y + hitbox.height + 1, lvlData))
        if (!IsSolid(hitbox.x + hitbox.width, hitbox.y + hitbox.height + 1, lvlData))
            return false;

    return true;

}
```

Lời cảm ơn

Cuối cùng, chúng em xin gửi lời tri ân sâu sắc đến thầy Vũ Huấn. Trong quá trình tìm hiểu và học tập bộ môn, chúng em đã nhận được sự giảng dạy và hướng dẫn tận tình, tâm huyết của thầy. Thầy đã giúp chúng em tích lũy thêm nhiều kiến thức hay và bổ ích. Từ những kiến thức mà thầy truyền đạt đã giúp em trình bày được bài báo cáo.

Tuy nhiên kiến thức về môn Công nghệ Java của em vẫn còn những hạn chế nhất định. Do đó, không thể tránh khỏi những thiếu sót trong quá trình hoàn thành bài báo cáo này. Mong thầy xem và góp ý để bài báo cáo của chúng em được hoàn thiện hơn.

Kính chúc thầy luôn nhiều sức khỏe và thành công hơn nữa trong sự nghiệp "trồng người" để tiếp tục dìu dắt nhiều thế hệ học trò đến những bến bờ tri thức.

Chúng em xin chân thành cảm ơn thầy!