INTERNATIONAL UNIVERSITY
VIETNAM NATIONAL UNIVERSITY, HCM CITY

School of Computer Science & Engineering

PACMAN GAME PROJECT

Advisor: Tran Thanh Tung
Course: Object – Oriented Programming

Group members:

Trần Khánh Tài	ITITIU21300
Trần Khôi Nguyên	ITITIU21264
Huỳnh Đạt Minh Tâm	ITITIU21305
Mai Đức Thiện	ITITIU21319

Contents

I. Introduction

Nowadays, with the speed up rapidly of Software Technology industry, the elevated level in programming skill is more necessary. So, the classic Procedural-Oriented Programming method cannot be response completely to all requirements. That leads the invention of a new method follow the principal ***Alan – Kay*** named **"Object – Oriented Programming"** to overcome these cases.

This project was designed based on Object – Oriented Programming method by Java language. So, this has solved some problems that occurred while building by common Procedural – Oriented Method:

- The code is clear, easy to understand and concise.
- The project is a unified logic system by many related - Classes combine together.
- Each Class has many Methods which take on different behaviors of own class.
- The ability to reuse resources.

The goal of our project is to design a basic game built on the principles of object-oriented programming (OOP). As a group, we decided to develop a modified version of the classic game "Pacman", showing key OOP properties such as class interactions, object behaviors, and relationship between them. Pacman, a timeless arcade game, serves as an excellent foundation for exploring and implementing core OOP concepts like encapsulation, inheritance, polymorphism, and abstraction. By leveraging OOP, we aim to build a modular and maintainable codebase, emphasizing the interaction between classes, objects, and their respective behaviors.

Moreover, beyond meeting the project requirements, this endeavor allows us to practice the techniques we have learned throughout this course, such as designing and structuring classes, managing objects, and logically solving problems using OOP methodologies.

While we are aware of challenges and potential difficulties of utilizing knowledge out of the university syllabus, we see this as an opportunity to prepare for real world pressures in the software development industry.

II. Property of Pacman Game

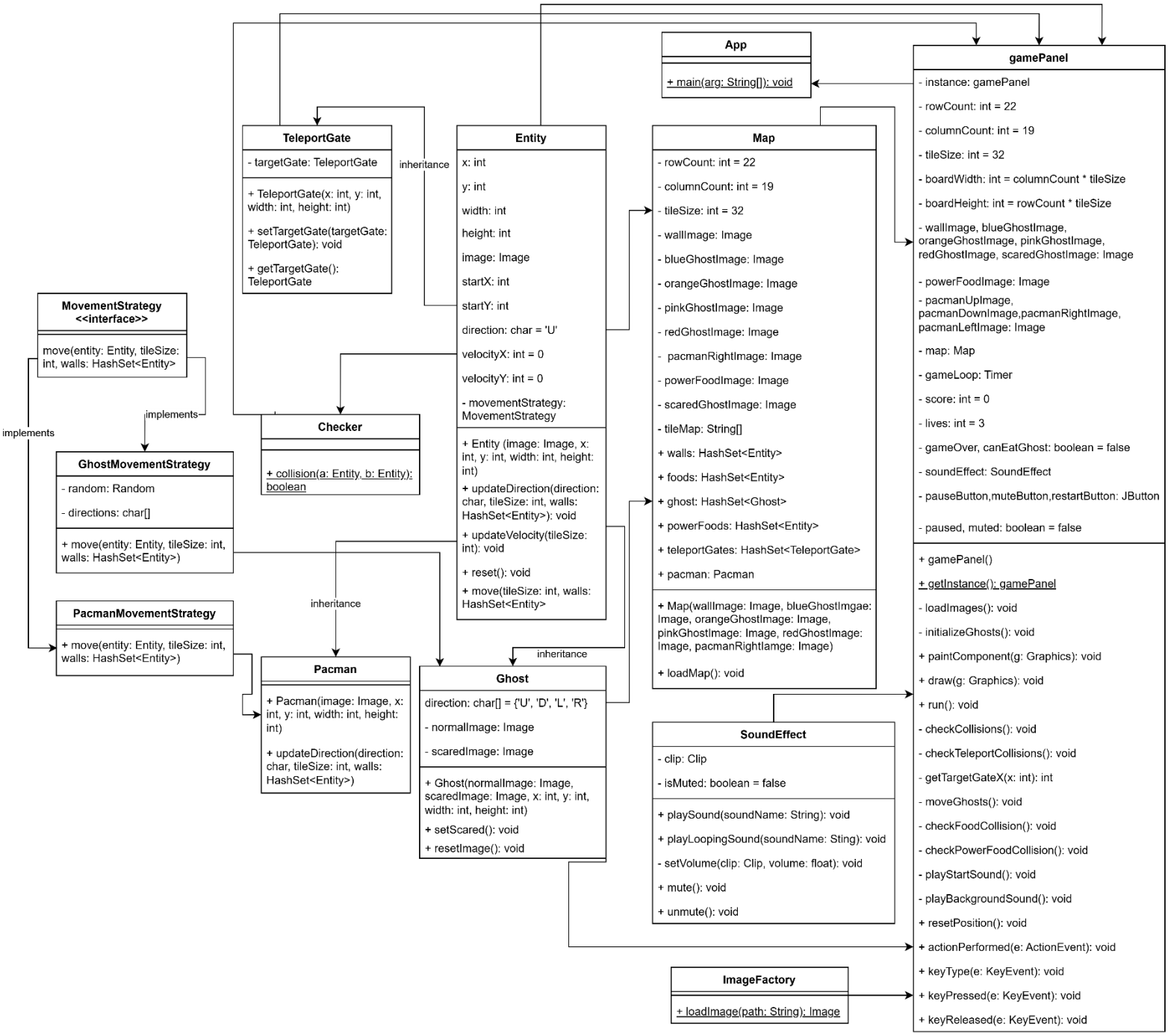
1. Goal

The aim in Pacman game is simple. You will control a Pacman in game, run around the maze to eat food, each food will increase 10 points, and a special food is power food that gives Pacman a special ability and increases 50 points. If Pacman eats all the food on the map, you win.

2. Rule

Pacman has very simple rules: that player have to control Pacman through a maze to eat all food while avoiding ghosts, but after Pacman eat power food Pacman can chase and eat ghost to increase score. And the mission of player is how to reach the highest score as they can.

III. UML Diagram of Pacman Game



IV. Methodology:

This project is combined into 13 classes. There are App, gamePanel, Map, SoundEffect, ImageFactory, Entity, Ghost, Pacman, Checker, TeleportGate, MovementStrategy, GhostMovementStrategy, and PacmanMovementStrategy. We built the game based on the object-oriented method, so we focus on the gamePanel and its dependencies.

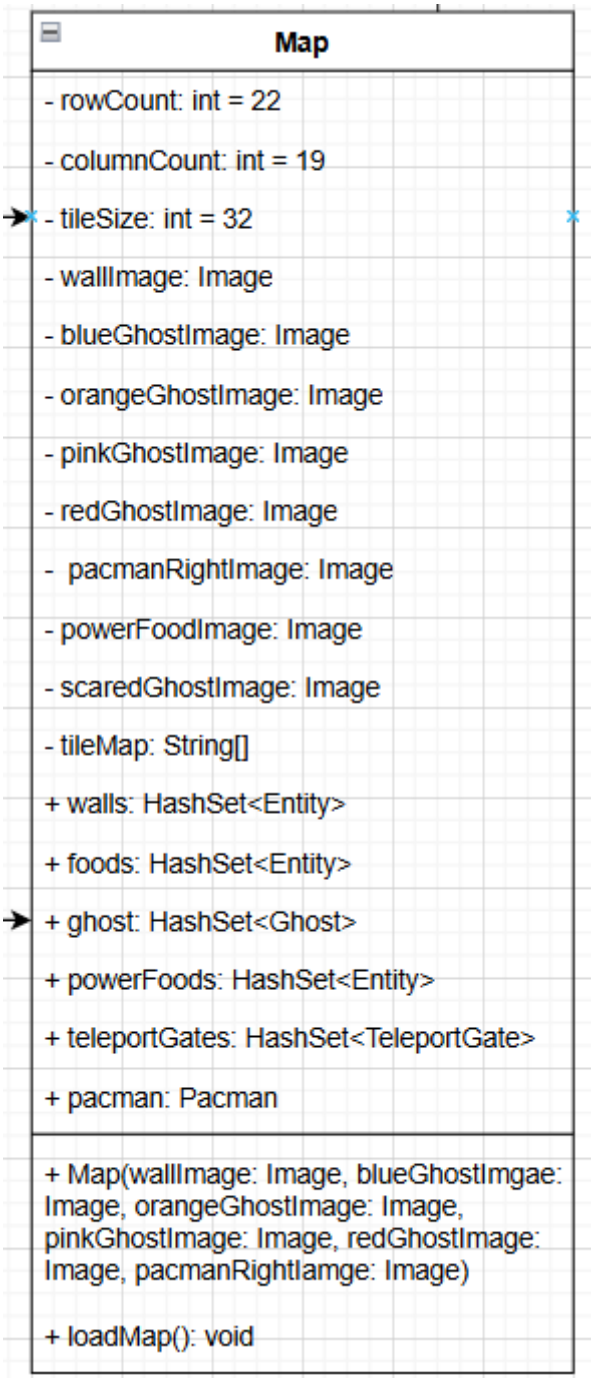
First, we create the Entity and MovementStrategy class of the Pacman and Ghost class. Pacman and Ghost inherit from Entity and each class has unique behavior, such in Pacman we handle user control, and in Ghost is the movement logic.

Second, we build a supporting class that is Checker class for collision detection. Then we add SoundEffect and ImageFactory classes to manage audio files and load images.

Finally, we created the gamePanel class to handle rendering and user input and set up game logic. Then, we connect all the classes to a logical system and put it into the App class that contains the main function to run the game.

V. Main Class – Function – Explain

1. Map Class



Map class was created to design the game map for Pacman. It holds information about the layout of the game board (the window console), including the positions of walls, ghosts, Pacman, teleport gate, power food, and food.

In Map class we define the number of rows (rowCount) and columns (columnCount)

in the map, and the size of each tile (tileSize). And the string array (tileMap) represents the layout of map.

```
    private int rowCount = 22;
    private int columnCount = 19;
    private int tileSize = 32;

    private Image wallImage, blueGhostImage, orangeGhostImage,
pinkGhostImage,
            redGhostImage, pacmanRightImage, powerFoodImage,
scaredGhostImage;
    private String[] tileMap = {
            "0000000000000000000",
            "XXXXXXXXXXXXXXXXXXX",
            "X                 X",
            "XfXX XXX X XXX XXfX",
            "X                 X",
            "X XX X XXXXX X XX X",
            "X    X   X   X    X",
            "XXXX XXX X XXX XXXX",
            "OOOX X       X XOOO",
            "XXXX X X r X X XXXX",
            "T      XbpoX      T",
            "XXXX X XXXXX X XXXX",
            "OOOX X       X XOOO",
            "XXXX X XXXXX X XXXX",
            "X        X        X",
            "X XX XXX X XXX XX X",
            "Xf X     P     X fX",
            "XX X X XXXXX X X XX",
            "X    X   X   X    X",
            "X XXXXXX X XXXXXX X",
            "X                 X",
            "XXXXXXXXXXXXXXXXXXX"
    };
```

In `loadMap()` method, we set each character in the String corresponds to a different type of game element:

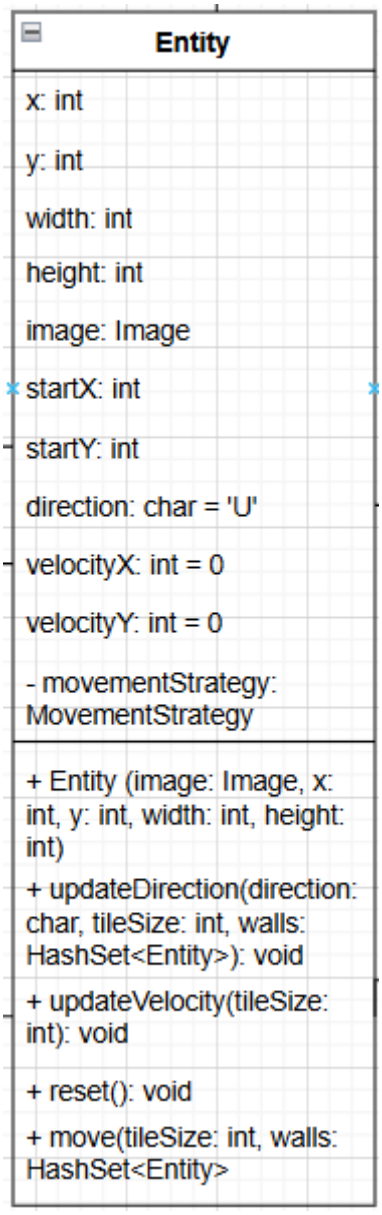
- "X": wall
- "f": power food
- "T": teleport gate
- "P": Pacman
- "r": red ghost

- "b": blue ghost
- "o": orange ghost
- "p": pink ghost
- "O" and "0": skip (it means nothing will be there)

```
public void loadMap() {
        walls = new HashSet<>();
        foods = new HashSet<>();
        ghosts = new HashSet<>();
        powerFoods = new HashSet<>();
        teleportGates = new HashSet<>();

        TeleportGate firstGate = null;

        for (int r = 0; r < rowCount; r++) {
            for (int c = 0; c < columnCount; c++) {
                char tileMapChar = tileMap[r].charAt(c);
                int x = c * tileSize;
                int y = r * tileSize;

                switch (tileMapChar) {
                    case 'X' -> walls.add(new Entity(wallImage, x, y,
tileSize, tileSize));
                    case 'b' -> ghosts.add(new Ghost(blueGhostImage,
scaredGhostImage, x, y, tileSize, tileSize));
                    case 'o' -> ghosts.add(new Ghost(orangeGhostImage,
scaredGhostImage, x, y, tileSize, tileSize));
                    case 'p' -> ghosts.add(new Ghost(pinkGhostImage,
scaredGhostImage, x, y, tileSize, tileSize));
                    case 'r' -> ghosts.add(new Ghost(redGhostImage,
scaredGhostImage, x, y, tileSize, tileSize));
                    case 'P' -> pacman = new Pacman(pacmanRightImage, x,
y, tileSize, tileSize);
                    case ' ' -> foods.add(new Entity(null, x + 14, y +
14, 4, 4));
                    case 'f' -> powerFoods.add(new Entity(powerFoodImage,
x + 8, y + 8, tileSize - 16, tileSize - 16));
                    case 'T' -> {
                        TeleportGate gate = new TeleportGate(x, y,
tileSize, tileSize);

                        if (firstGate == null) {
                            firstGate = gate;
```

```
                    } else {
                        gate.setTargetGate(firstGate);
                        firstGate.setTargetGate(gate);
                        teleportGates.add(gate);
                        teleportGates.add(firstGate);
                    }
                }
            }
        }
    }
```

2. Entity Class



Class Entity is a base class of Pacman and Ghost classes, it is applied one of four OOP properties – inheritance. There are four important algorithms in this class.

Update Velocity Algorithm

We need to update entity's velocity due to it's direction:

```
public void updateVelocity(int tileSize) {
        if (this.direction == 'U') {
            this.velocityX = 0;
            this.velocityY = -tileSize / 4;
        } else if (this.direction == 'D') {
            this.velocityX = 0;
            this.velocityY = tileSize / 4;
        } else if (this.direction == 'L') {
            this.velocityX = -tileSize / 4;
            this.velocityY = 0;
        } else if (this.direction == 'R') {
            this.velocityX = tileSize / 4;
            this.velocityY = 0;
        }
    }
```

Here we use "`tileSize / 4`" instead of "`tileSize`", because we want to control the speed of movement for the entity in the game.
The "`tileSize`" stands for the size of one tile in grid. If the velocity we set to "`tileSize`", each time we call this method Pacman will move a full tile for each update, it make us feel like Pacman is teleporting.

Update Direction Algorithm

```
public void updateDirection(char direction, int tileSize, HashSet<Entity>
walls){
        char prevDirection = this.direction; // save the previous
direction
        this.direction = direction;
        updateVelocity(tileSize); // update the velocity based on the new
direction
        this.x += this.velocityX;
        this.y += this.velocityY;
```

```
        for (Entity wall : walls) {
            if (Checker.collision(this, wall)) {
                // if there is a collision, revert the position and
direction
                this.x -= this.velocityX;
                this.y -= this.velocityY;
                this.direction = prevDirection;
                updateVelocity(tileSize); // update the velocity based on
the previous direction
            }
        }
    }
```

This method updates the direction of the entity and handles collision with walls. First, we save the current direction of the entities to prevDirection, then update the direction of the entities. After that we update the velocity of entities based on the new direction.

```
        this.x += this.velocityX;
        this.y += this.velocityY;
```

This block is used for moving entities.
Finally, we check for collisions with walls. If a collision is detected, the entities position is revered by subtracting the velocity from its coordinates, and set the direction to prevDirection.

Moving Algorithm

The `move()` method use a strategy pattern to implement the movement logic to a MovementStrategy object.

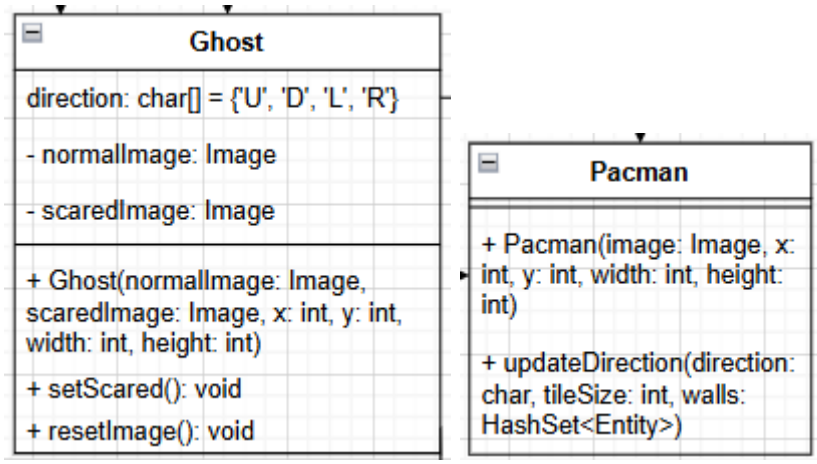
```
public void move(int tileSize, HashSet<Entity> walls) {
        if (movementStrategy != null) {
            movementStrategy.move(this, tileSize, walls);
        }
    }
```

Reset Entities position Algorithm

If we want to reset the position of entity's we can use `reset()` method. The method set the current x, y coordinate to the initial `startX`, `startY` coordinate.

```
public void reset() {
        this.x = this.startX;
        this.y = this.startY;
    }
```

3. Ghost and Pacman class



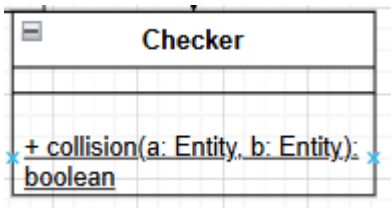
The Ghost and Pacman class is a subclass of Entity class so they inherit all the method of Entity class. And in Ghost class we have 2 more methods

setScared() method and resetImage() method

We need to set the ghost's image to scary after Pacman eat power food. Then, after 7 seconds we will call `resetImage()` method to reset ghost's image to normal.

```
    public void setScared() {
        this.image = scaredImage;
    }

    public void resetImage() {
        this.image = normalImage;
    }
```

4. Checker class



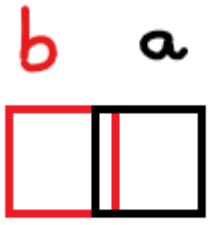
The collision method of `Checker` class checks if two entities are colliding. This work by comparing their position and dimension.

```
public static boolean collision(Entity a, Entity b) {
        return a.x < b.x + b.width &&
                a.x + a.width > b.x &&
                a.y < b.y + b.height &&
                a.y + a.height > b.y;
    }
```

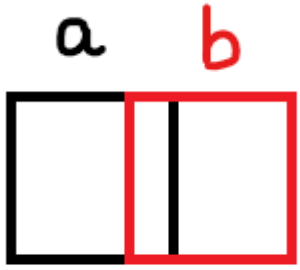
We use axis-aligned bounding box algorithms to determine of two entities overlapping.

Checker Algorithm

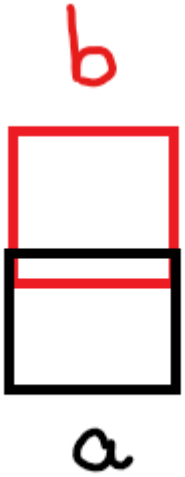
- a.x < b.x + b.width : check if the left side of a is to the left of the right side of b.



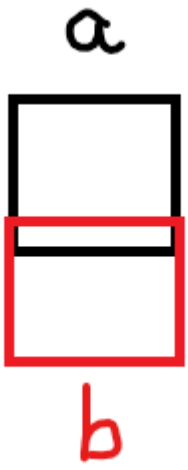
- a.x + a.width > b.x: check if the right side of a is to the right of left side of b.



- a.y < b.y + b.height: check if the top side of a is above the bottom side of b.

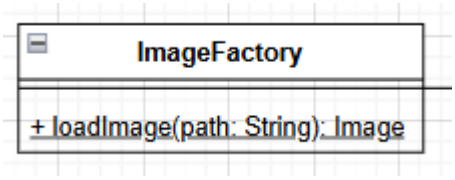


- a.y + a.height > b.y: check if the bottom side of a is below the top side of b.



If all these conditions are true, it means that the rectangle of entities are overlapping, indicating a collision.

5. ImageFactory class



This class is designed to load image from a specified path.

```
// Load image from the path
public static Image loadImage(String path) {
    return new ImageIcon(Objects.requireNonNull(ImageFactory.class.getResource(path))).getImage();
}
```

- `Objects.requireNonNull`: to make sure that the resource path is not null.

Usage:

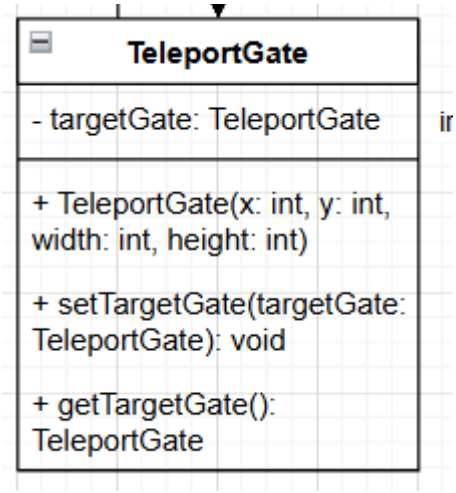
The `loadImage(String path)` method is used in the `gamePanel` class to load images for the game.

```
private void loadImages() {
    wallImage = ImageFactory.loadImage("resource/wall.png");
    blueGhostImage = ImageFactory.loadImage("resource/blueGhost.png");
    orangeGhostImage =
ImageFactory.loadImage("resource/orangeGhost.png");
    pinkGhostImage = ImageFactory.loadImage("resource/pinkGhost.png");
    redGhostImage = ImageFactory.loadImage("resource/redGhost.png");
    scaredGhostImage =
ImageFactory.loadImage("resource/scaredGhost.png");

    powerFoodImage = ImageFactory.loadImage("resource/powerFood.png");

    pacmanUpImage = ImageFactory.loadImage("resource/pacmanUp.png");
    pacmanDownImage = ImageFactory.loadImage("resource/pacmanDown.png");
    pacmanLeftImage = ImageFactory.loadImage("resource/pacmanLeft.png");
    pacmanRightImage =
ImageFactory.loadImage("resource/pacmanRight.png");
```

```
}
```

Depending on the `loadImage()` method of `ImageFactory` class we can make the `loadImage()` method look very clean.

6. TeleportGate class



This class provides the methods to Pacman and Ghosts when hit the Teleport gate will appear in another Teleport gate.

```
public class TeleportGate extends Entity {
    private TeleportGate targetGate;

    public TeleportGate(int x, int y, int width, int height) {
        super(null, x, y, width, height);
    }

    public void setTargetGate(TeleportGate targetGate) {
        this.targetGate = targetGate;
    }

    public TeleportGate getTargetGate() {
        return targetGate;
    }
}
```

Like in the Map class we have a snippet code that:

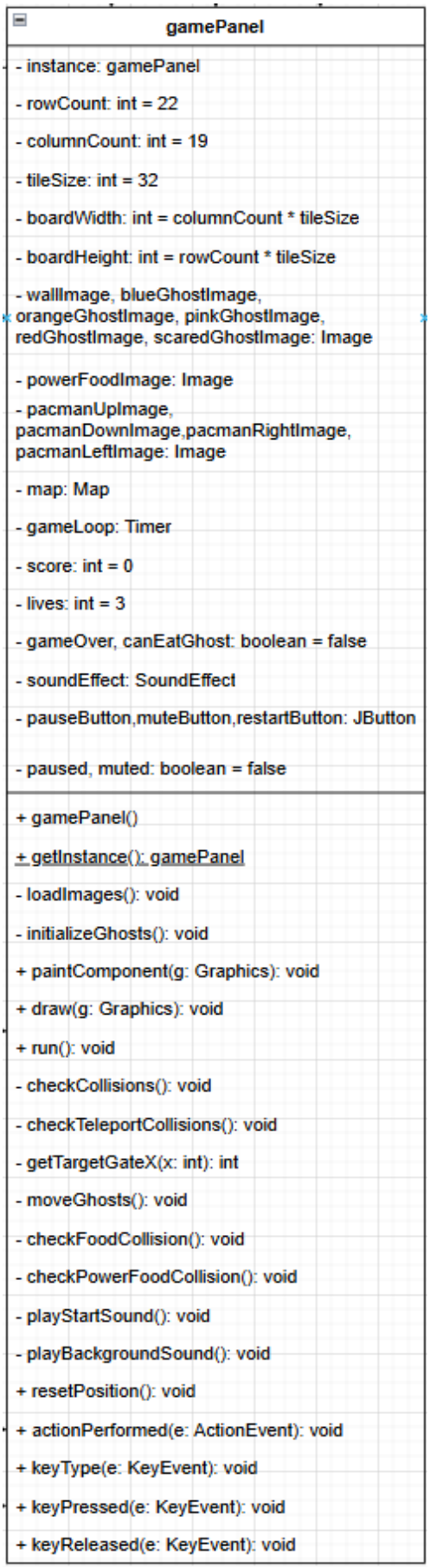
```
                    case 'T' -> {
```

```
                        TeleportGate gate = new TeleportGate(x, y,
tileSize, tileSize);

                        if (firstGate == null) {
                            firstGate = gate;


                        } else {
                            gate.setTargetGate(firstGate);
                            firstGate.setTargetGate(gate);
                            teleportGates.add(gate);
                            teleportGates.add(firstGate);
                        }
                    }
```

First, we have created a new `TeleportGate` object. Then, we check if the `firstGate` is null, meaning that the first teleport gate is created, so it can assign to `firstGate`.

Second, when the for-loop loops into the another 'T' in `tileMap` we check this condition again. Now the firstGate is not null, which means that a teleport gate is created. The new gate is linked to the firstGate by `setTargetGate()` method.

7. gamePanel class



gamePanel is a core class of this project. We put all the important algorithms in this class.

First, we have a singleton design pattern in this class. Singleton design pattern ensures that class has only one instance and provides a global access point to it.

```
// instance of gamePanel
    private static gamePanel instance;
```

This variable holds the single instance of the gamePanel class.

```
// singleton pattern
    public static synchronized gamePanel getInstance() {
        if (instance == null) {
            instance = new gamePanel();
        }
        return instance;
    }
```

This method checks if the instance is null, it creates a new instance of `gamePanel`. The `synchronized` keyword ensures that the method is thread safe, preventing multiple threads from creating multiple instances simultaneously.

Check Collision Algorithms

So, Pacman can't go over the edge of the map and can't eat ghosts in normal condition.

```
private boolean canEatGhost = false;
```

First, we declare a boolean variable canEatGhost and set initial value to false.

```
private void checkCollisions() {

        for (Entity wall : map.walls) {
            if (Checker.collision(map.pacman, wall)) {
                map.pacman.x -= map.pacman.velocityX;
                map.pacman.y -= map.pacman.velocityY;
                break;
            }
        }
```

```
        for (Ghost ghost : map.ghosts) {
            if (Checker.collision(ghost, map.pacman)) {
                if (canEatGhost) {
                    ghost.reset();
                    score += 200;
                    soundEffect.playSound("gs_eatghost.wav");
                    soundEffect.playSound("gs_returnghost.wav");
                } else {
                    lives--;
                    soundEffect.playSound("gs_pacmandies.wav");
                    if (lives == 0) {
                        gameOver = true;
                    } else {
                        resetPacmanPositions();
                    }
                }
            }
        }
    }
```

We use Checker class to check the collision between Pacman and other entities. If Pacman collides with walls, we will reverse Pacman to the last position (before colliding with wall), then break the loop after the first wall collision.

Besides, if Pacman collides with Ghost, the first thing we do is check if canEatGhost is false or true. If it is true now Ghost will reset to the initial position as when we started the game and increase the scores. In the other hand, live of Pacman will minus one, and we check if the lives is equal zero, gameOver will be set to true, otherwise, we will reset the position of Pacman to the initial by using `resetPacmanPosition()` method. This algorithm also is similar to the `checkFoodCollision()` method, just one thing different that when collide with food we will remove food.

```
private void resetPacmanPositions() {
        map.pacman.reset();
        map.pacman.velocityX = 0;
        map.pacman.velocityY = 0;
    }
```

Checking collides with Power Foods Algorithm

```
private void checkPowerFoodCollision() {
        for (Entity powerFood : map.powerFoods) {
            if (Checker.collision(map.pacman, powerFood)) {
                map.powerFoods.remove(powerFood);
                score += 50;
                canEatGhost = true;
                soundEffect.playSound("gs_siren_soft.wav");
                for (Ghost ghost : map.ghosts) {
                    ghost.setScared();
                }
                // 7 seconds
                Timer powerFoodTimer = new Timer(7000, e -> {
                    for (Ghost ghost : map.ghosts) {
                        // return ghost to normal
                        ghost.resetImage();
                    }
                    canEatGhost = false;
                });

                powerFoodTimer.setRepeats(false);
                powerFoodTimer.start();
                break;
            }
        }
    }
```

It is also quite similar with check food collision, but after Pacman eat power food, we will give Pacman a special ability that can eat Ghost. So, we need to add some algorithms for that.

Firstly, after eating power food we set the value of `canEatGhost` to true, then we set the image of Ghosts to scared ghost by using `setScared()` method in Ghost class.

Secondly, we need a Timer to count the time that Pacman can eat Ghosts, here we set the timer to 7000 milliseconds (7 seconds), after 7 seconds we will bring them back to normal, using `resetImage()`, and set `canEatGhost` to false.

Check Teleport Collision

```
    private long lastTeleportTime = 0;
    private final long teleportDelay = 200; // 0.2s

    private void checkTeleportCollision() {
        long currentTime = System.currentTimeMillis();
        if (currentTime - lastTeleportTime < teleportDelay) {
            return;
        }

        for (TeleportGate teleportGate : map.teleportGates) {
            if (Checker.collision(map.pacman, teleportGate)) {
                TeleportGate targetGate = teleportGate.getTargetGate();
                map.pacman.x = getTargetGateX(targetGate.x);
                map.pacman.y = targetGate.y;

                lastTeleportTime = currentTime;
                break;
            }
        }
    }
```

In this method, we will check if Pacman collides with `teleportGate`, Pacman will appear in target gate like we mention above.

Firstly, we set `lastTeleportTime` to zero, and a fixed variable to control the time between two times of teleport, `teleportDelay`, to 200 milliseconds (0.2 seconds).

```
        long currentTime = System.currentTimeMillis();
        if (currentTime - lastTeleportTime < teleportDelay) {
            return;
        }
```

This block checks if the time between two teleports is less than `teleportDelay`, the method will return nothing. For example, if we set the `teleportDelay` to 0.1 seconds, Pacman will stuck in the gate because in 0.1 seconds Pacman can't move out of the gate, or if `teleportDelay` too high, Pacman will move out of the map when the time between two teleports is less than `teleportDelay,` without teleport to the target gate.

Secondly, if a collision is detected, we will retrieves the `targetGate` of the current teleport gate using `getTargetGate()` method. Then we updated the position of Pacman to the target gate position.

In the updated position algorithms, we have a special case, in our map we just set two teleport gates in the same horizontal line. If we control Pacman into the teleport, and as soon as it appears and we control it go to the opposite direction, Pacman will go out of the map. So we need `getTargetGateX()` method to handle this problem.

```
private int getTargetGateX(int x) {
    if (x == 0) {
        return x + tileSize;
    } else {
        return x - tileSize;
    }
}
```

This method adjusts the x-coordinate of target gate to ensure Pacman is correctly positioned. For example, if the target x-coordinate is zero, we will add tileSize.

Here we set:

```
private int tileSize = 32;
```

So, if the x-coordinate of target gate is zero, Pacman will appear at `x = 0 + 32,`

otherwise `x = x - 32`.

Run game method

```
private void run() {
        map.pacman.move(tileSize, map.walls);
        moveGhosts();
        checkCollisions();
        checkFoodCollision();
        checkPowerFoodCollision();
        checkTeleportCollision();

        if (map.foods.isEmpty()) {
            gameLoop.stop();
            JOptionPane.showMessageDialog(this, "You win!");
            resetPacmanPositions();
            map.loadMap();
            gameLoop.start();
        }
    }
```

The `run()` method executed the main game logic like move Pacman, move Ghosts, check for various collisions (walls, ghosts, foods,…), and check of the food is empty, it stop the game loop and show a “You win!” message, reset Pacman position, reloads the map, and restart the game loop

An override method actionPerformed handle actions events by running the game logic, repaint the game panel, and stop the game loop if the game is over.

```
@Override
    public void actionPerformed(ActionEvent e) {
        if (!paused) {
            run();
            repaint();
            if (gameOver) {
                gameLoop.stop();
            }
        }
```

```
    }
```

Key Control

```
@Override
public void keyReleased(KeyEvent e) {
    if (gameOver) {
        map.loadMap();
        resetPacmanPositions();
        lives = 3;
        score = 0;
        gameOver = false;
        gameLoop.start();
    }

    switch (e.getKeyCode()) {
        case KeyEvent.VK_UP, KeyEvent.VK_W -> map.pacman.updateDirection(direction:'U', tileSize, map.walls);
        case KeyEvent.VK_DOWN, KeyEvent.VK_S -> map.pacman.updateDirection(direction:'D', tileSize, map.walls);
        case KeyEvent.VK_LEFT, KeyEvent.VK_A -> map.pacman.updateDirection(direction:'L', tileSize, map.walls);
        case KeyEvent.VK_RIGHT, KeyEvent.VK_D -> map.pacman.updateDirection(direction:'R', tileSize, map.walls);
    }

    switch (map.pacman.direction) {
        case 'U' -> map.pacman.image = pacmanUpImage;
        case 'D' -> map.pacman.image = pacmanDownImage;
        case 'L' -> map.pacman.image = pacmanLeftImage;
        case 'R' -> map.pacman.image = pacmanRightImage;
    }
}
```

This method handles key release events and updates the game state based on the key pressed.

Firstly, we check if gameOver is true, we can press any key to restart game.

Secondly, we have a switch statement that checks which key was released and update the direction of Pacman. And the second switch statement used to update the image of Pacman based on his current direction.

VI. Result – Limited – Conclusion

1. Result

Based on the principle of Object-oriented programming method, our project has been completely built with basic rules and properties. The combination of classes and objects in the system relatively logical. We also do successfully the display to connect the user with the program, it can be controlled by other input devices like a keyboard. The player can use arrow keys or WASD to control the character.

Moreover, there are some new features out of the basic rule is the background sound while playing, and some button users can interact with (restart game button, mute button, and pause game button).

2. Limitation

Besides the success of building the game with basic rules, our project still has many cases that cannot be solved:

- Do not have input name for user, leaderboard, game level system, and animation of entities.
- This game is just for one player, still cannot be modified for two players.
- There is only one map.

3. Conclusion

Pacman Game that was build by Object-oriented programming method is more easier and logical than the traditional method. This shows clearly polymorphism, inheritance, encapsulation, abstraction of OOP, and the relationship between classes and objects is linked tightly and systematically. Besides that, learning more knowledge outside the limits of this course is one of the important things to do while performing this project.

Github link: https://github.com/KhanhTaiTran/PacMan-Project.git