# Assignment:

Continue to explore JPacMan3 by answering the following questions:

- what is the role of EmptySprite?

1. In the *AnimatedSprite* class, the *EmptySprite* is used as the end of a non-looping sprite. When the current frame number is larger than the animation frame length, the EmptySprite is used to stop the animation of the sprite and make it disappear. The code related is as below:

```
private Sprite currentSprite() {
    Sprite result = END_OF_LOOP;
    if (current < animationFrames.length) {
        result = animationFrames[current];
    }
    assert result != null;
    return result;
}
```

2. In the *ImageSprite* class, the *EmptySprite* is used when splitting sprites as new sprites. If the new sprite is out of the panel boundary, the *EmptySprite* is used to make the new sprite disappear from the panel. The code related is as below:

```
@Override
public Sprite split(int x, int y, int width, int height) {
    if (withinImage(x, y) && withinImage(x + width - 1, y + height - 1)) {
        BufferedImage newImage = newImage(width, height);
        newImage.createGraphics().drawImage(image, 0, 0, width, height, x,
            y, x + width, y + height, null);
        return new ImageSprite(newImage);
    }
    return new EmptySprite();
}
```

- what is the role of MOVE_INTERVAL and INTERVAL_VARIATION?

  1. The MOVE_INTERVAL is the basic interval of the ghost movement.
  2. The INTERVAL_VARIATION is a scope of random number added to MOVE_INTERVAL used to make the ghost more dynamic and not predictable.

  The code related are:

```
public long getInterval() {
    return this.moveInterval + new Random().nextInt(this.intervalVariation);
}
```

- if you wanted to add a fruit, which files would you need to change?

  1. Add a class named Fruit like class Pellet.

```java
package nl.tudelft.jpacman.level;


import nl.tudelft.jpacman.board.Unit;
import nl.tudelft.jpacman.sprite.Sprite;

/**
 * A fruit, one of the little dots Pac-Man has to collect.
 *
 * @author Jeroen Roosen
 */
public class Fruit extends Unit {

    /**
     * The sprite of this unit.
     */
    private final Sprite image;

    /**
     * The point value of this fruit.
     */
    private final int value;

    /**
     * Creates a new fruit.
     * @param points The point value of this fruit.
     * @param sprite The sprite of this pellet.
     */
    public Fruit(int points, Sprite sprite) {
        this.image = sprite;
        this.value = points;
    }

    /**
     * Returns the point value of this fruit.
     * @return The point value of this fruit.
     */
    public int getValue() {
        return value;
    }

    @Override
```

```
    public Sprite getSprite() {
        return image;
    }
}
```

2. In PacManSprites class, create a method called getFruitSprite similar to getPelletSprite.

```
/**
 * @return The sprite for the fruit
 */
public Sprite getFruitSprite() {
    return loadSprite("/sprite/apple.png");
}
```

3. In LevelFactory class, add a method createFruit similar to createPallet.

```
/**
 * Creates a new fruit.
 *
 * @return The new fruit.
 */
public Fruit createFruit() {
    return new Fruit(Fruit_VALUE, sprites.getFruitSprite());
}
```

And set Fruit_VALUE.

```
/**
 * The default value of a fruit.
 */
private static final int Fruit_VALUE = 50;
```

4. In PlayerCollision.java, add code to handle collision with fruit.

```
package nl.tudelft.jpacman.level;

import nl.tudelft.jpacman.board.Unit;
import nl.tudelft.jpacman.npc.Ghost;

/**
 * A simple implementation of a collision map for the JPacman player.
 * <p>
 * It uses a number of instanceof checks to implement the multiple dispatch for the
 * collisionmap. For more realistic collision maps, this approach will not scale,
 * and the recommended approach is to use a {@link CollisionInteractionMap}.
 *
 * @author Arie van Deursen, 2014
 *
 */

public class PlayerCollisions implements CollisionMap {
```

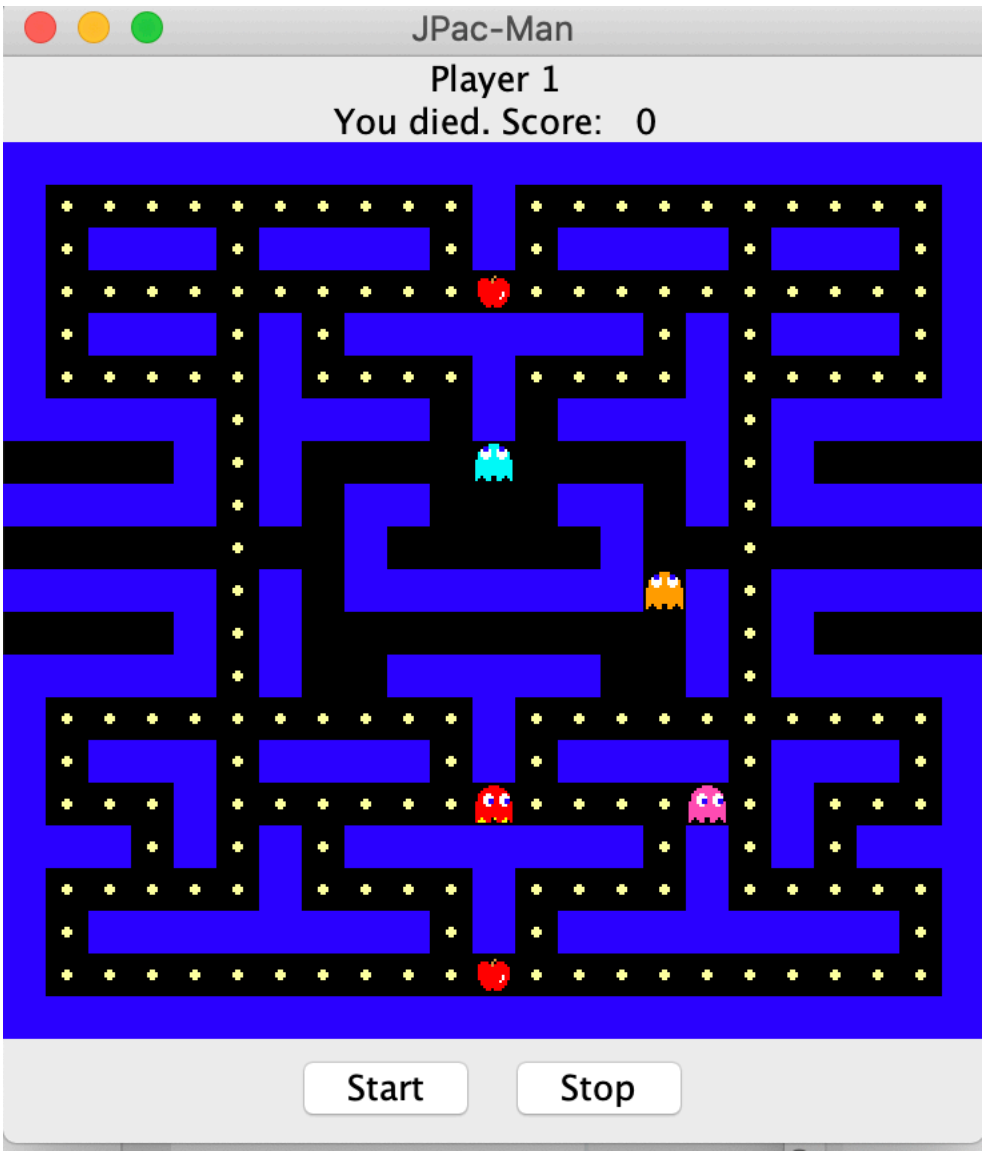
```
@Override
public void collide(Unit mover, Unit collidedOn) {
    if (mover instanceof Player) {
        playerColliding((Player) mover, collidedOn);
    }
    else if (mover instanceof Ghost) {
        ghostColliding((Ghost) mover, collidedOn);
    }
    else if (mover instanceof Pellet) {
        pelletColliding((Pellet) mover, collidedOn);
    }
    else if (mover instanceof Fruit) {
        fruitColliding((Fruit) mover, collidedOn);
    }
}

private void playerColliding(Player player, Unit collidedOn) {
    if (collidedOn instanceof Ghost) {
        playerVersusGhost(player, (Ghost) collidedOn);
    }
    if (collidedOn instanceof Pellet) {
        playerVersusPellet(player, (Pellet) collidedOn);
    }
}

private void ghostColliding(Ghost ghost, Unit collidedOn) {
    if (collidedOn instanceof Player) {
        playerVersusGhost((Player) collidedOn, ghost);
    }
}

private void pelletColliding(Pellet pellet, Unit collidedOn) {
    if (collidedOn instanceof Player) {
        playerVersusPellet((Player) collidedOn, pellet);
    }
}

private void fruitColliding(Fruit fruit, Unit collidedOn) {
    if (collidedOn instanceof Player) {
        playerVersusFruit((Player) collidedOn, fruit);
    }
}


/**
 * Actual case of player bumping into ghost or vice versa.
```

```
     *
     * @param player The player involved in the collision.
     * @param ghost The ghost involved in the collision.
     */
    public void playerVersusGhost(Player player, Ghost ghost) {
        player.setAlive(false);
    }

    /**
     * Actual case of player consuming a pellet.
     *
     * @param player The player involved in the collision.
     * @param pellet The pellet involved in the collision.
     */
    public void playerVersusPellet(Player player, Pellet pellet) {
        pellet.leaveSquare();
        player.addPoints(pellet.getValue());
    }

    /**
     * Actual case of player consuming a fruit.
     *
     * @param player The player involved in the collision.
     * @param fruit The fruit involved in the collision.
     */
    public void playerVersusFruit(Player player, Fruit fruit) {
        fruit.leaveSquare();
        player.addPoints(fruit.getValue());
    }

}
```

5. Change some ".' In board.txt to "F", and in mapParser.java, add parser to parse "F" to fruit.

```
case 'F':
    Square fruitSquare = boardCreator.createGround();
    grid[x][y] = fruitSquare;
    levelCreator.createPellet().occupy(fruitSquare);
    break;
```

6. Result:


JPac-Man
Player 1
You died. Score: 0
Start
Stop