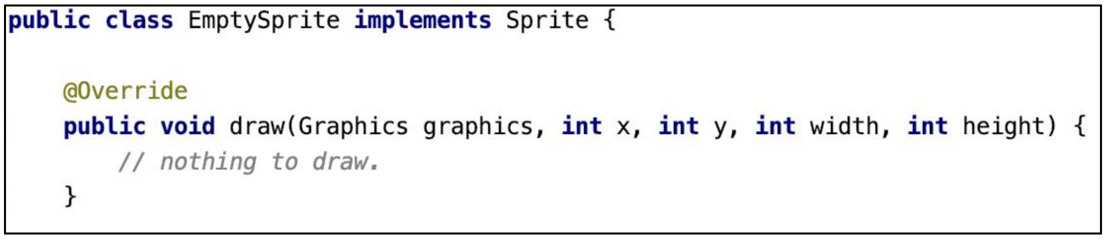
SWE 265P Assignment - Continue to explore JPacman:

<u>What is the role of EmptySprite?</u>

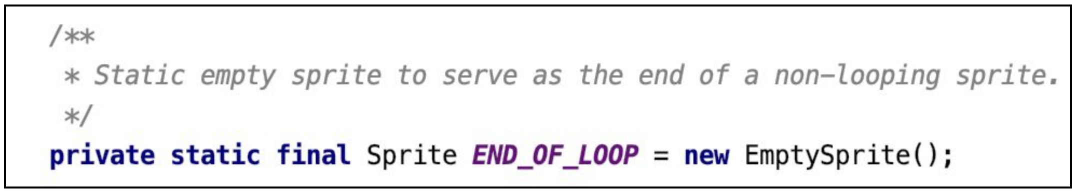
As observed from Code Snippet (1), EmptySprite draws nothing.



```java
public class EmptySprite implements Sprite {

    @Override
    public void draw(Graphics graphics, int x, int y, int width, int height) {
        // nothing to draw.
    }
```

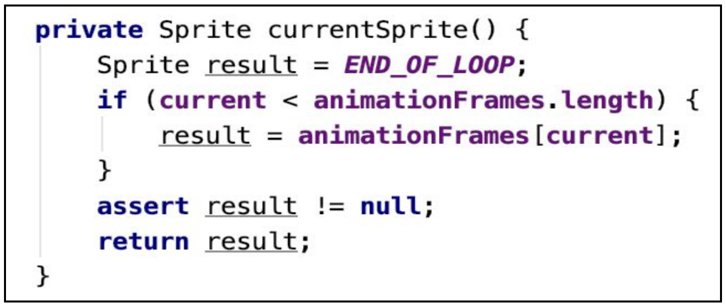
Code Snippet (1) : EmptySprite.java [Lines 11 - 16]

On further searching the project, we notice that the EmptySprite instance is returned in AnimatedSprite and ImageSprite classes.



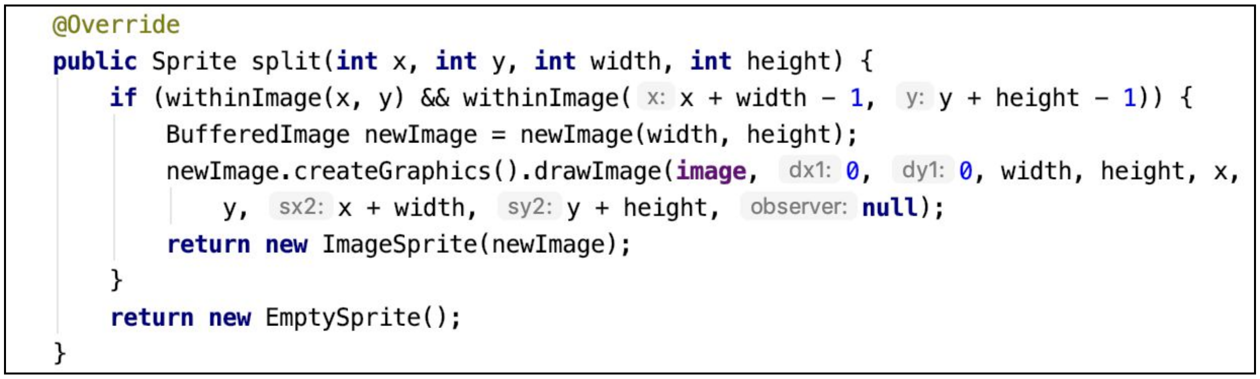
```java
    /**
     * Static empty sprite to serve as the end of a non-looping sprite.
     */
    private static final Sprite END_OF_LOOP = new EmptySprite();
```

Code Snippet (2) : AnimatedSprite.java [Lines 13 - 16]



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

Code Snippet (3) : AnimatedSprite.java [Lines 90 - 97]



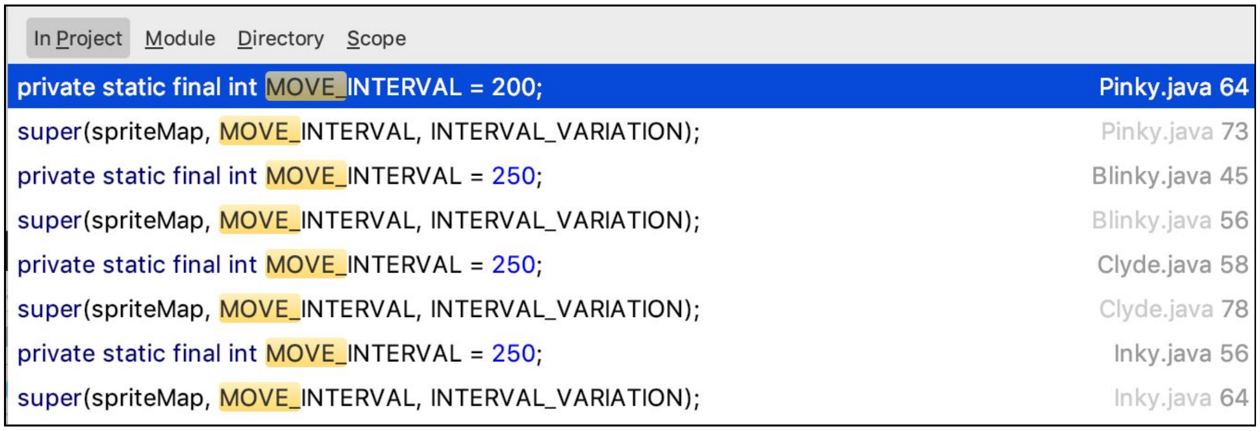
```java
@Override
public Sprite split(int x, int y, int width, int height) {
    if (withinImage(x, y) && withinImage( x: x + width - 1,  y: y + height - 1)) {
        BufferedImage newImage = newImage(width, height);
        newImage.createGraphics().drawImage(image,  dx1: 0,  dy1: 0, width, height, x,
            y,  sx2: x + width,  sy2: y + height,  observer: null);
        return new ImageSprite(newImage);
    }
    return new EmptySprite();
}
```

Code Snippet (4) : ImageSprite.java [Lines 39 - 47]

From Code Snippet (2), we notice that the instance of EmptySprite class is used as the end of non-looping sprite. Further analyzing Code Snippet (3), we can conclude that after reaching the length of the animation frames, it will invoke the EmptySprite, which will display nothing. This happens when the Pacman dies. One can observe this in the animation, after the ghost catches the Pacman, we can see some frames of the pacman before completely disappearing.

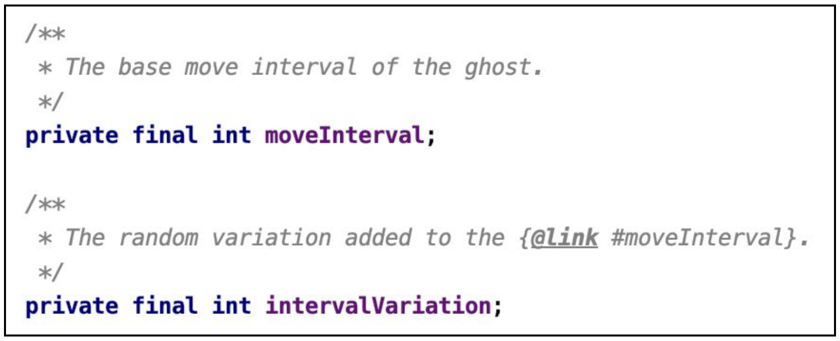
<u>What is the role of MOVE_INTERVAL and INTERVAL_VARIATION?</u>

As observed from figure (1), we notice that MOVE_INTERVAL and INTERVAL_VARIATION is present in all of the ghosts files.



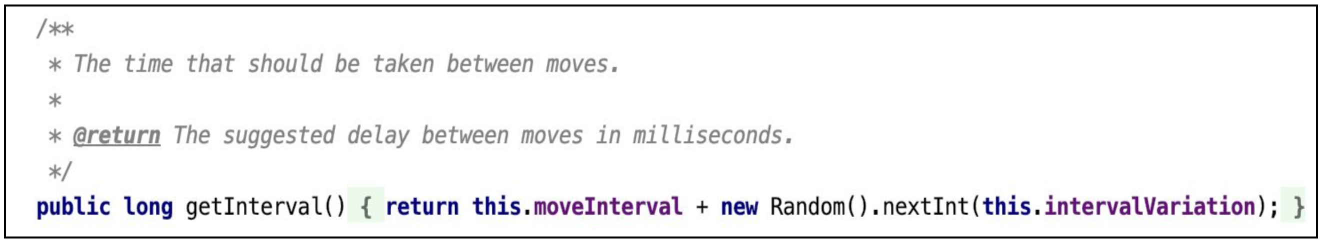
```
private static final int MOVE_INTERVAL = 200;                          Pinky.java 64
super(spriteMap, MOVE_INTERVAL, INTERVAL_VARIATION);                   Pinky.java 73
private static final int MOVE_INTERVAL = 250;                          Blinky.java 45
super(spriteMap, MOVE_INTERVAL, INTERVAL_VARIATION);                   Blinky.java 56
private static final int MOVE_INTERVAL = 250;                          Clyde.java 58
super(spriteMap, MOVE_INTERVAL, INTERVAL_VARIATION);                   Clyde.java 78
private static final int MOVE_INTERVAL = 250;                          Inky.java 56
super(spriteMap, MOVE_INTERVAL, INTERVAL_VARIATION);                   Inky.java 64
```

Figure (1) : Search results for “MOVE_” [in Project]



```java
/**
 * The base move interval of the ghost.
 */
private final int moveInterval;

/**
 * The random variation added to the {@link #moveInterval}.
 */
private final int intervalVariation;
```

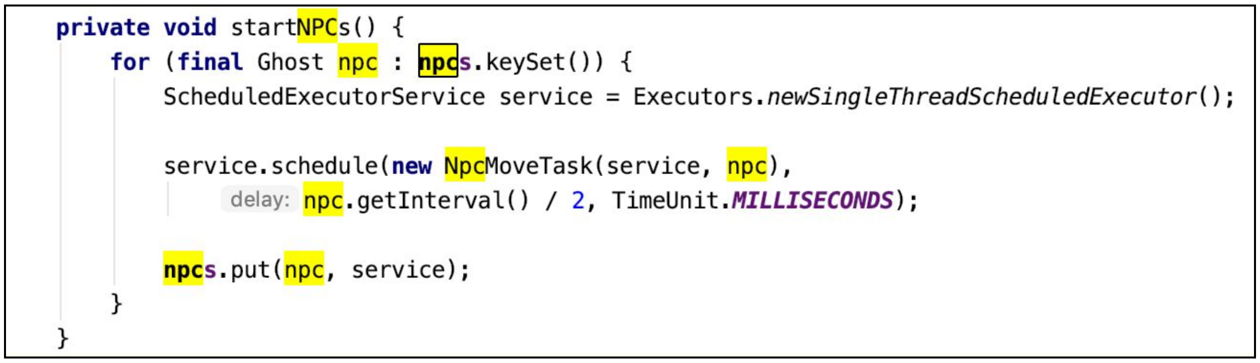
Code Snippet (5) : Ghost.java [Lines 25 - 33]



```java
/**
 * The time that should be taken between moves.
 *
 * @return The suggested delay between moves in milliseconds.
 */
public long getInterval() { return this.moveInterval + new Random().nextInt(this.intervalVariation); }
```

Code Snippet (6) : Ghost.java [Lines 74 - 79]

We notice that the MOVE_INTERVAL is stored in moveInterval variable of the Ghost class. On further analysing this variable using Code Snippet (6), we notice that it contributes to the time taken between the moves and the suggested delay between the moves. We use a random value generator to randomly assign movements among the ghosts. The getInterval() is called as shown in the Code Snippet (7) below. Here, the non-player characters are the ghosts which is stored as a HashMap.

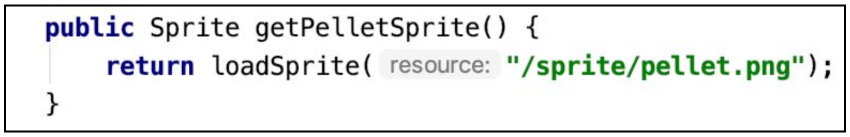


```java
private void startNPCs() {
    for (final Ghost npc : npcs.keySet()) {
        ScheduledExecutorService service = Executors.newSingleThreadScheduledExecutor();

        service.schedule(new NpcMoveTask(service, npc),
            delay: npc.getInterval() / 2, TimeUnit.MILLISECONDS);

        npcs.put(npc, service);
    }
}
```

Code Snippet (7) : Level.java [Lines 229 - 238]

<u>If you wanted to add a fruit, which files would you need to change?</u>

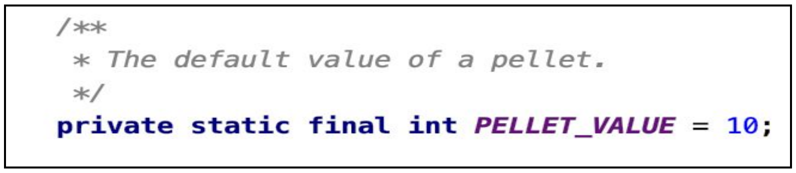
One can look at the fruit similar to the pellet in JPacman Version 3. This would mean that we have to add a similar file to Pellet.java. As noticed from Code Snippet (8), we should have a similar getFruit() in PacManSprites class.



```java
public Sprite getPelletSprite() {
    return loadSprite( resource: "/sprite/pellet.png");
}
```

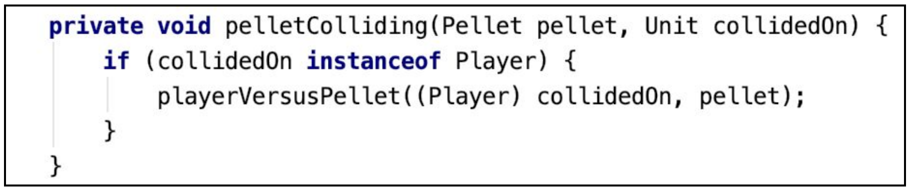
Code Snippet (8) : PacManSprites.java [Lines 132 - 134]

As observed from Code Snippet (9), one should modify LevelFactory class to include the point distribution for the fruits.



```java
/**
 * The default value of a pellet.
 */
private static final int PELLET_VALUE = 10;
```

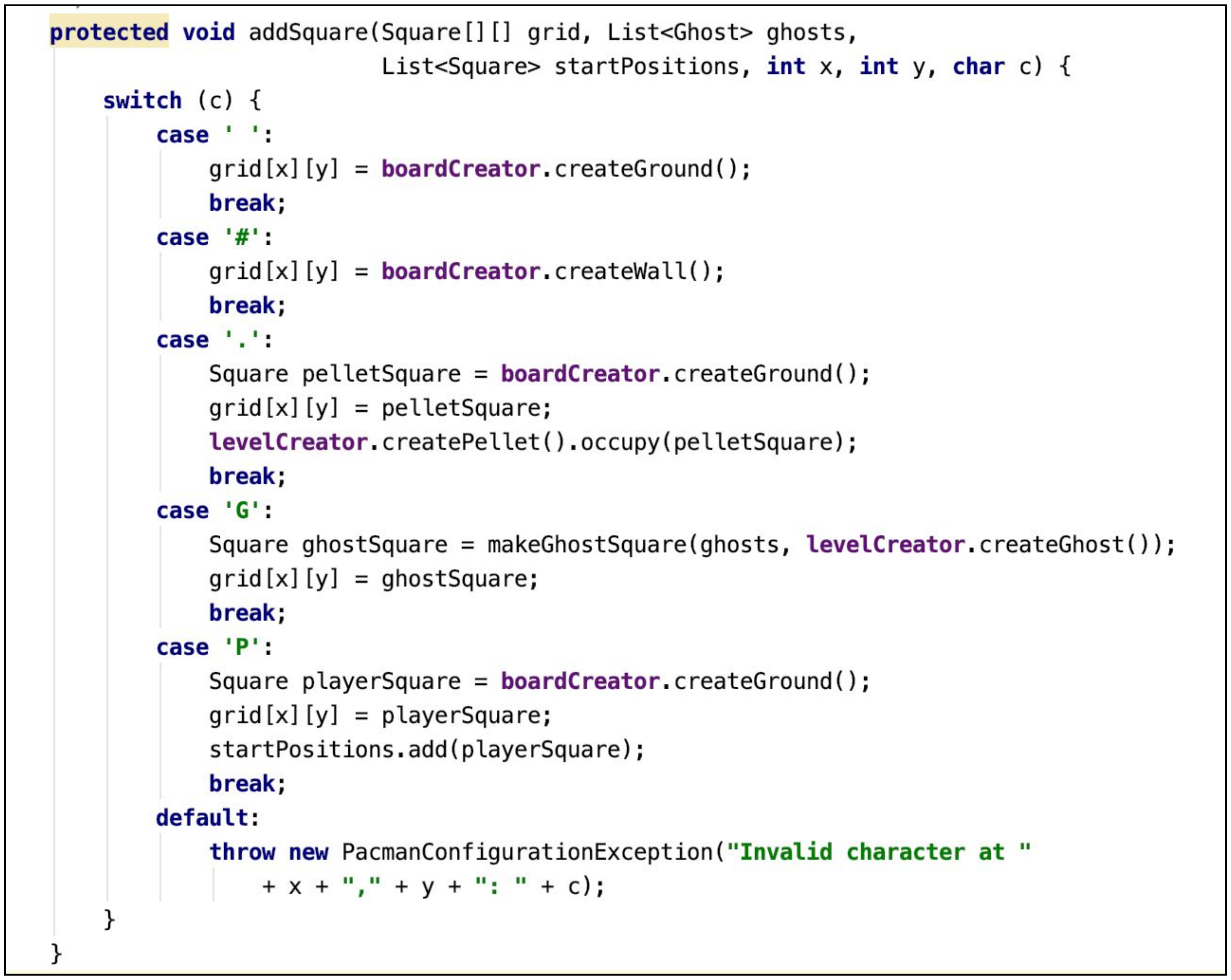
Code Snippet (9) : LevelFactory.java [Lines 29 - 32]



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

Code Snippet (10) : PlayerCollisions.java [Lines 47 - 51]

One should also change the PlayerCollisions class to define the collision as such. Other classes that has to be taken into consideration is the MapParser Class as shown in the Code Snippet (11). On analyzing the project, we notice that the associated map is present in Launcher class with the help of board.txt file. We might have to introduce new variables into this file and associate it in the MapParser class.



```java
protected void addSquare(Square[][] grid, List<Ghost> ghosts,
                         List<Square> startPositions, int x, int y, char c) {
    switch (c) {
        case ' ':
            grid[x][y] = boardCreator.createGround();
            break;
        case '#':
            grid[x][y] = boardCreator.createWall();
            break;
        case '.':
            Square pelletSquare = boardCreator.createGround();
            grid[x][y] = pelletSquare;
            levelCreator.createPellet().occupy(pelletSquare);
            break;
        case 'G':
            Square ghostSquare = makeGhostSquare(ghosts, levelCreator.createGhost());
            grid[x][y] = ghostSquare;
            break;
        case 'P':
            Square playerSquare = boardCreator.createGround();
            grid[x][y] = playerSquare;
            startPositions.add(playerSquare);
            break;
        default:
            throw new PacmanConfigurationException("Invalid character at "
                + x + "," + y + ": " + c);
    }
}
```

Code Snippet (11) : MapParser.java [Lines 109 - 136]

[Reference: Code Snippets from jpacman3 GitHub repository of SWE-265P.]