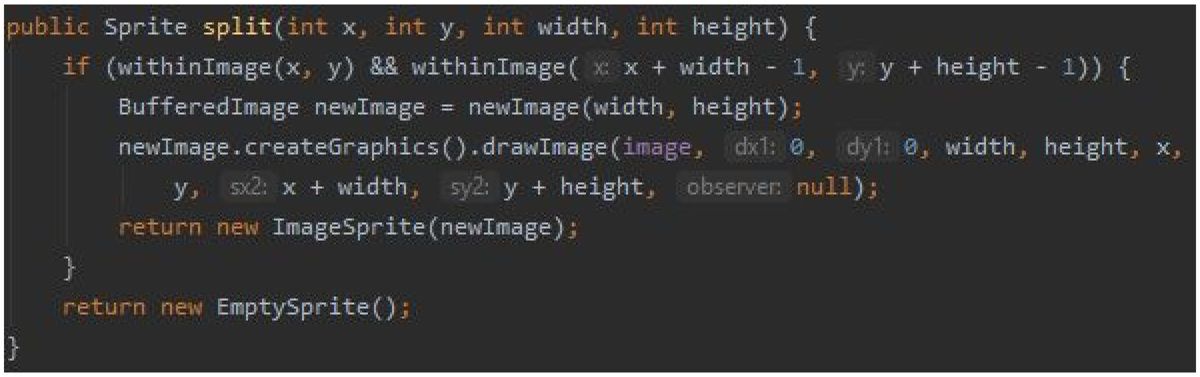
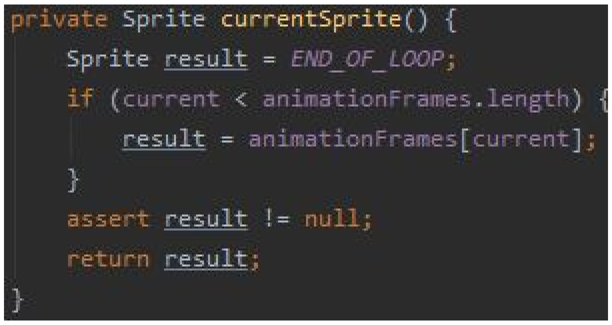
what is the role of EmptySprite?

EmptySprite is a placeholder sprite that draws nothing when used primarily to prevent errors. It is used for when splitting the source sprite sheet goes wrong and out of bounds.



```java
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();
}
```

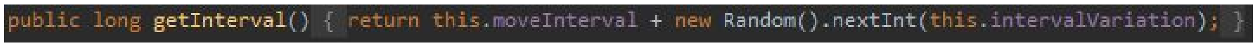
It is also used as an end of loop sprite when the current sprite index goes beyond the length of animation frames available.



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

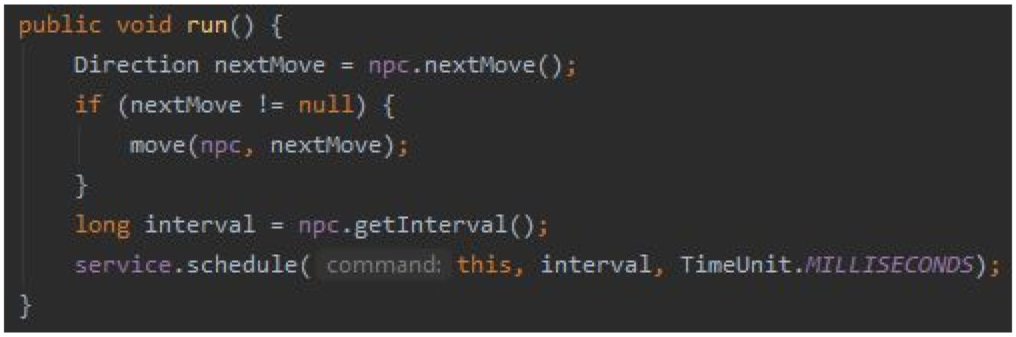
what is the role of MOVE_INTERVAL and INTERVAL_VARIATION?

They are movement stats for each ghost.
MOVE_INTERVAL is their base move speed (base time that is taken between moves).
INTERVAL_VARIATION is a random interval added to their base interval to make them more random/human like in the getInterval() method below.



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

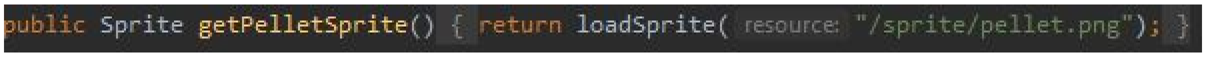
The interval is then used in the run statement below to set their move delay.



```java
public void run() {
    Direction nextMove = npc.nextMove();
    if (nextMove != null) {
        move(npc, nextMove);
    }
    long interval = npc.getInterval();
    service.schedule( command: this, interval, TimeUnit.MILLISECONDS);
}
```

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

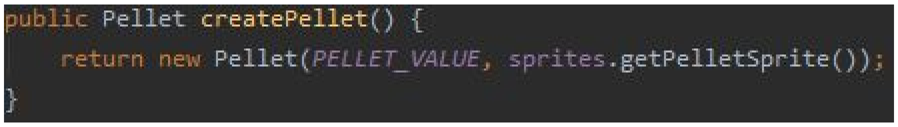
Edit the file called PacManSprites and add get methods that would get the fruits similar to getPelletSprite().



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

getAppleSprite(), getOrangeSprite(), etc.

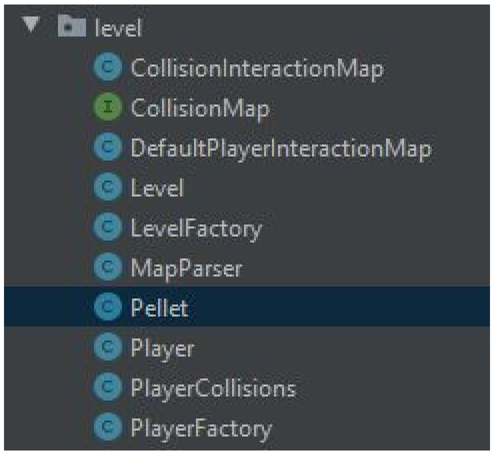
Edit the file called LevelFactory and add create methods that would create the fruits.



```
public Pellet createPellet() {
    return new Pellet(PELLET_VALUE, sprites.getPelletSprite());
}
```

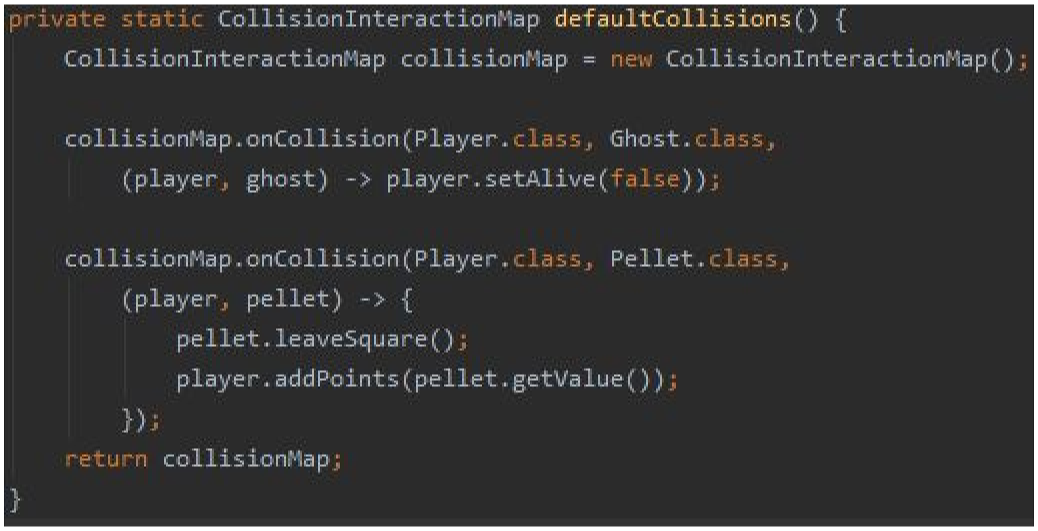
createApple(), createOrange(), etc.

Continue using the Pellet class for the fruits, **OR**
Create files and classes for each fruit just like how the Pellet class is implemented inside the level directory.



```
▼ level
    CollisionInteractionMap
    CollisionMap
    DefaultPlayerInteractionMap
    Level
    LevelFactory
    MapParser
    Pellet
    Player
    PlayerCollisions
    PlayerFactory
```

Add cases for collision with the fruits in the DefaultPlayerInteractionMap file.

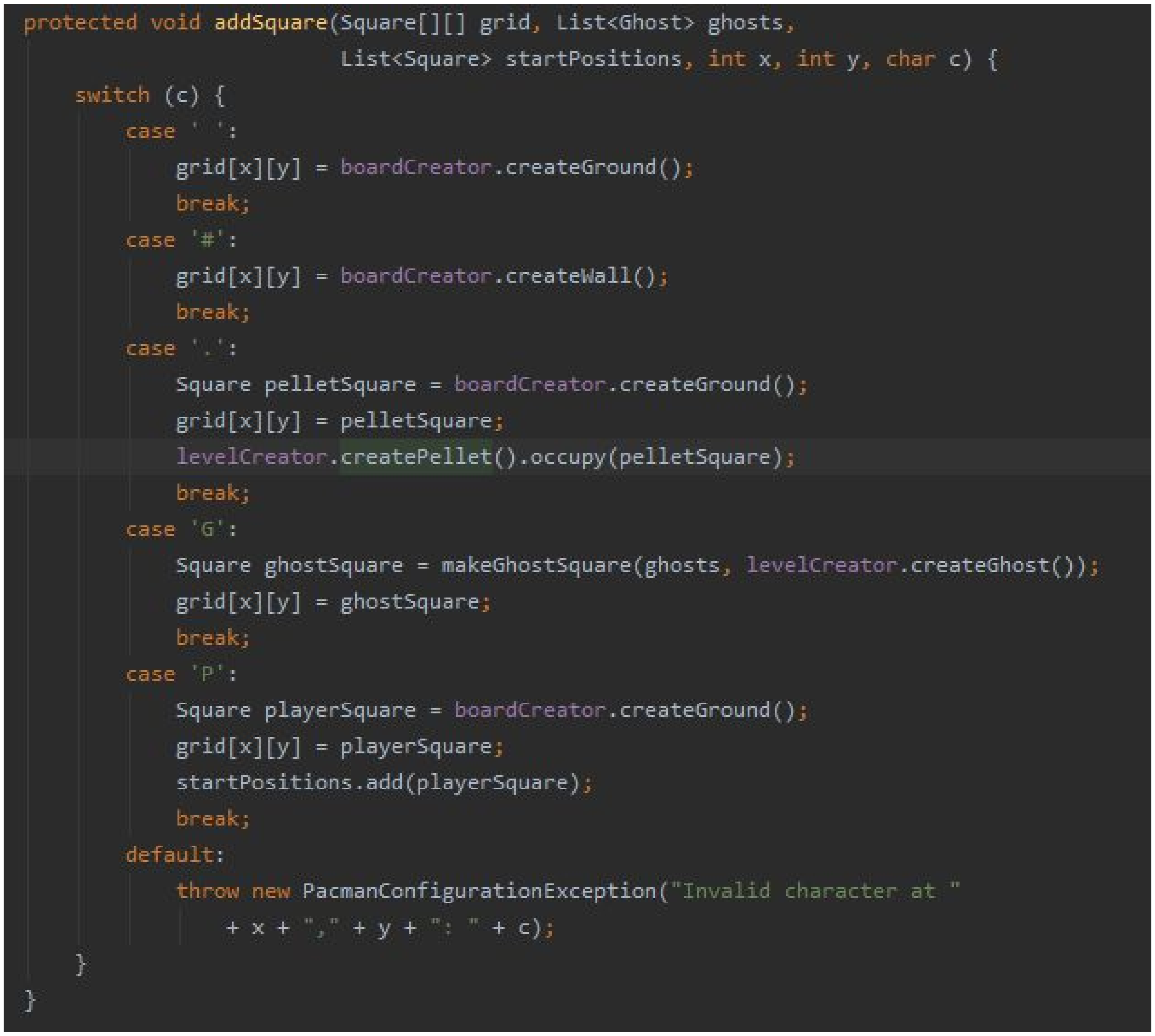


```
private static CollisionInteractionMap defaultCollisions() {
    CollisionInteractionMap collisionMap = new CollisionInteractionMap();

    collisionMap.onCollision(Player.class, Ghost.class,
        (player, ghost) -> player.setAlive(false));

    collisionMap.onCollision(Player.class, Pellet.class,
        (player, pellet) -> {
            pellet.leaveSquare();
            player.addPoints(pellet.getValue());
        });
    return collisionMap;
}
```

Add cases for all the fruits in the file MapParser, that calls their respective create call.



```
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);
    }
}
```

Finally add whatever letter that represents fruits into the board.txt file in the resources folder.