

jpacman3

Wen-Chia, Yang



What is the role of EmptySprite?

EmptySprite is an object that will serve as a purpose of empty sprite which does not hold any data. When EmptySprite is called, nothing will be drawn. There are three places in the project that will use EmptySprite class.

1. constant END_OF_LOOP in AnimateSprite.java

```
/**
 * Static empty sprite to serve as the end of a non-looping sprite.
 */
private static final Sprite END_OF_LOOP = new EmptySprite();
/**
 * @return The frame of the current index.
 */
private Sprite currentSprite() {
    Sprite result = END_OF_LOOP;
    if (current < animationFrames.length) {
        result = animationFrames[current];
    }
    assert result != null;
    return result;
}
```

- AnimateSprite.java a class that updates drawing animation by time.
- END_OF_LOOP is an empty sprite that serves as the end of a non-looping sprite.
- Every time draw(Graphics, int, int, int, int) in AnimateSprite.java is called, currentSprite() will be triggered to check if current is less than the length of animation frames. If true, it will return the current frame of the animated sprite. If not, it will return the empty sprite(END_OF_LOOP).

2. split(int, int, int, int) in ImageSprite.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();
}
```

- ImageSprite.java a class that creates a sprite with image.
- split(int, int, int, int) is a method that takes x, y, width and height as parameters and checks if a split sprite is beyond the boundary of the image. If true, it will return an empty sprite. If not, it will return the partial image.

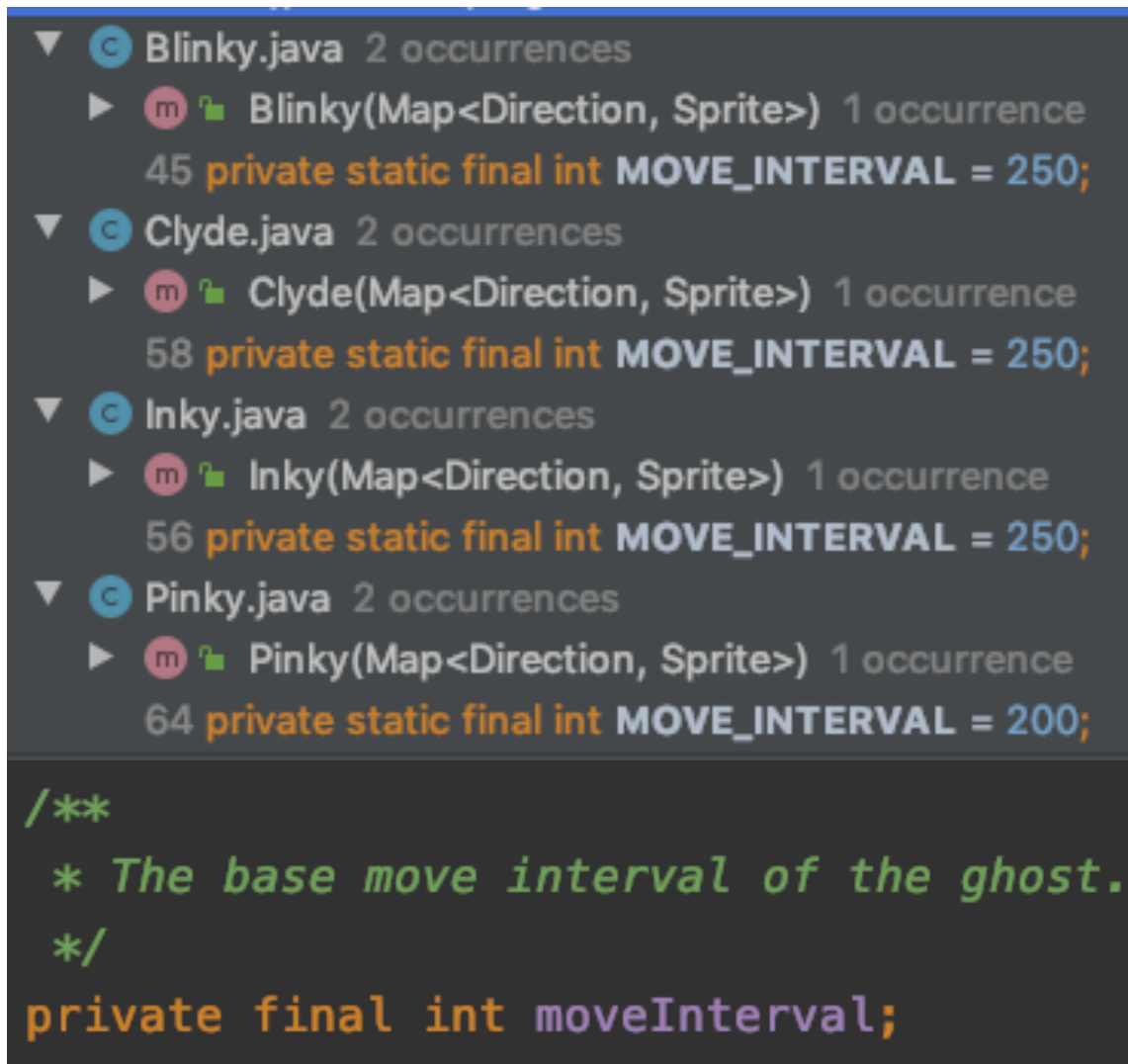
3. splitOutOfBounds() in SpriteTest.java

```
/**
 * Verifies that a split that isn't within the actual sprite returns an empty sprite.
 */
@Test
public void splitOutOfBounds() {
    Sprite split = sprite.split(x: 10, y: 10, width: 64, height: 10);
    assertThat(split).assertInstanceOf(EmptySprite.class);
}
```

- SpriteTest.java a class that verifies the loading state of sprites.
- splitOutOfBounds() is a method that verifies the empty sprite case when calling split(int, int, int, int) in ImageSprite.java with parameters that are outside the boundary.

What is the role of MOVE_INTERVAL and INTERVAL_VARIATION?

1. Constants int MOVE_INTERVAL in Blinky.java, Clyde.java, Inky.java and Pinky.java



```
▼ Blinky.java 2 occurrences
  ► Blinky(Map<Direction, Sprite>) 1 occurrence
    45 private static final int MOVE_INTERVAL = 250;
▼ Clyde.java 2 occurrences
  ► Clyde(Map<Direction, Sprite>) 1 occurrence
    58 private static final int MOVE_INTERVAL = 250;
▼ Inky.java 2 occurrences
  ► Inky(Map<Direction, Sprite>) 1 occurrence
    56 private static final int MOVE_INTERVAL = 250;
▼ Pinky.java 2 occurrences
  ► Pinky(Map<Direction, Sprite>) 1 occurrence
    64 private static final int MOVE_INTERVAL = 200;

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

- Constants int MOVE_INTERVAL in Blinky.java(250), Clyde.java(250), Inky.java(250) and Pinky.java(200) are extended from the super class Ghost.java and denote the different speed of movement of these ghosts.

2. Constants int INTERVAL_VARIATION in Blinky.java, Clyde.java, Inky.java and Pinky.java

```
▼ Blinky.java 2 occurrences
  ► Blinky(Map<Direction, Sprite>) 1 occurrence
    40 private static final int INTERVAL_VARIATION = 50;
▼ Clyde.java 2 occurrences
  ► Clyde(Map<Direction, Sprite>) 1 occurrence
    53 private static final int INTERVAL_VARIATION = 50;
▼ Inky.java 2 occurrences
  ► Inky(Map<Direction, Sprite>) 1 occurrence
    51 private static final int INTERVAL_VARIATION = 50;
▼ Pinky.java 2 occurrences
  ► Pinky(Map<Direction, Sprite>) 1 occurrence
    59 private static final int INTERVAL_VARIATION = 50;

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

- Constants int INTERVAL_VARIATION in Blinky.java(50), Clyde.java(50), Inky.java(50) and Pinky.java(50) are extended from the super class Ghost.java and denote the variation in the given intervals and makes the ghosts look more dynamic and less predictable.

3. getInterval() in Ghost.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);
}

▼ Level.java 2 occurrences
  ▼ startNPCs() 1 occurrence
    234 npc.getInterval() / 2, TimeUnit.MILLISECONDS);
  ▼ run() 1 occurrence
    351 long interval = npc.getInterval();
```

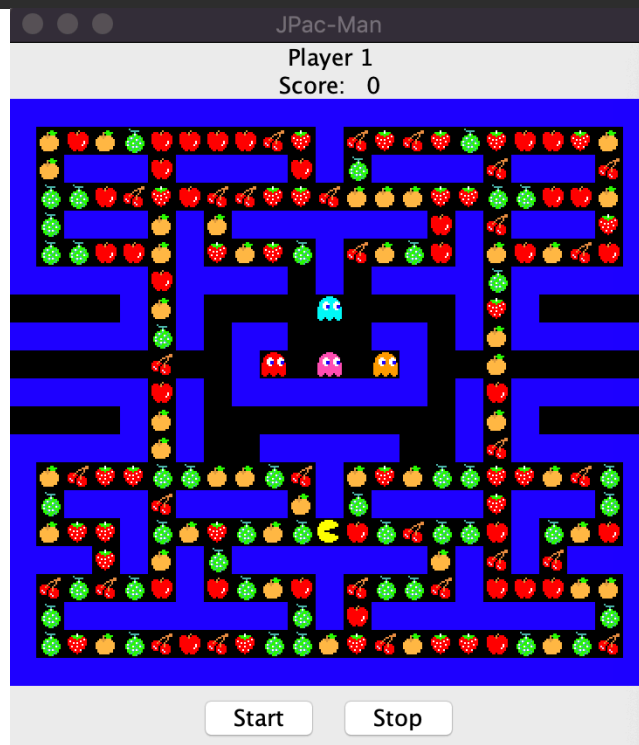
- Ghost.java is an abstract class that can be considered as a non-player unit.
- getInterval() is a method that will randomly return the time of the movement of ghosts by calculating variables of moveInterval and intervalVariation.
- getInterval() is called in Level.java when setting up all NPC movements.

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

For this question, I think there are multiple answers.

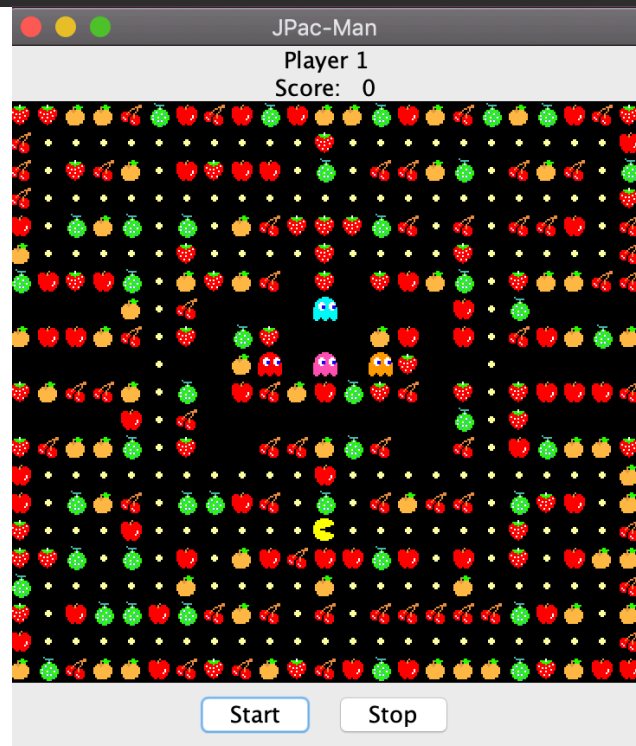
1. Replace pellet image with fruit image in `getPelletSprite()` in `PacManSprites.java`

```
/**
 * @return The sprite for the
 */
public Sprite getPelletSprite() {
    String[] fruits = {
        "apple", "cherry", "melon", "orange", "strawberry"
    };
    String fruit = fruits[new Random().nextInt(fruits.length)];
    return loadSprite(resource: "/sprite/" + fruit + ".png");
//    return loadSprite("/sprite/pellet.png");
}
```



2. Replace wall with fruit image in `getWallSprite()` in `PacManSprites.java`

```
/**
 * @return The sprite for the wall.
 */
public Sprite getWallSprite() {
    String[] fruits = {
        "apple", "cherry", "melon", "orange", "strawberry"
    };
    String fruit = fruits[new Random().nextInt(fruits.length)];
    return loadSprite(resource: "/sprite/" + fruit + ".png");
//    return loadSprite("/sprite/wall.png");
}
```



3. I can also replace `floor.png` as fruit image like above in `getGroundSprite()` in `PacManSprites.java`...

4. Consider a fruit is a super pellet(3 times pellet value).

- Step1: Create a super pellet class(SuperPellet.java) extends pellet

```
/**
 * Creates a new pellet.
 *
 * @param points The point value of this pellet.
 * @param sprite The sprite of this pellet.
 */
public SuperPellet(int points, Sprite sprite) {
    super(points, sprite);
}
```

- Step2: Add fruits mapping and getSuperPelletSprite() method in PacManSprites.java

```
/**
 * @return The sprite for the super pellet.
 */
private static final Map<Character, String> fruits;
static
{
    fruits = new HashMap<>();
    fruits.put('A', "apple");
    fruits.put('C', "cherry");
    fruits.put('M', "melon");
    fruits.put('O', "orange");
    fruits.put('S', "strawberry");
}

public Sprite getSuperPelletSprite(char key) {
    String fruit = fruits.get(key);
    return loadSprite(resource: "/" + fruit + ".png");
}
```

- Step3: Define constant of SUPER_PELLET_VALUE and add createSuperPellet(char) in LevelFactory.java

```
private static final int SUPER_PELLET_VALUE = PELLET_VALUE * 3;

/**
 * Creates a new super pellet.
 *
 * @return The new super pellet.
 */
public Pellet createSuperPellet(char key) {
    return new SuperPellet(SUPER_PELLET_VALUE, sprites.getSuperPelletSprite(key));
}
```

- Step4: Add new switch cases for apple('A'), cherry('C'), melon('M'), orange('O') and('S') in MapParser.java

```
case 'A':
case 'C':
case 'M':
case 'O':
case 'S':
    Square superPelletSquare = boardCreator.createGround();
    grid[x][y] = superPelletSquare;
    levelCreator.createSuperPellet(c).occupy(superPelletSquare);
    break;
```

- **Step5: Add new fruits in board.txt**

. # #
. ### . ### . # . ### . ### . #
. #
. ### . # . ##### . # . ### . #
. . . . # # #
. ##### # ##### . #####
. # ## ## # . #####
#G G G#
. # ##### # . #####
. # # . #
. # ##### # . #####
. # #
. ### . ### . # . ### . ### . #
. . . # PACMOS | . . . #
. # . # . ##### . # . # . ###
. . . . # # #
. ##### . # . ##### . #
. #
#####

