Yijia Zhang SWE265P

What is the role of EmptySprite?

EmptySprite implements the interface Sprite.

```
public class EmptySprite implements Sprite {   Dreef, 1
```

At the time of creation, fields will not actually be populated (or be filled with 0), as it role is to only provide a base structure without actually drawing anything (as noted in comment, "// nothing to draw".

Then we proceed to find the usages. One is found in ImageSprite.java, in function split().

```
@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();   Dreef, 1/16/20, 11:32 AM • JPacman v3 for SWE265p
}
```

What is does is to return an empty sprite if, when creating a new sprite, the target coordinate does not fall in the frame.

Another usage is in

```
 */
private Sprite currentSprite() {
    Sprite result = END_OF_LOOP;
    if (current < animationFrames.length) {
        result = animationFrames[current];   Dreef, 1
    }
    assert result != null;
    return result;
}
```

, where an instance of empty sprite is declared with name END_OF_LOOP. But then result is updated to either null or other status.

Thus the conclusion is that it basically does not do anything, or that it serves as a placeholder where nothing is supposed to be drawn.

What is the role of MOVE_INTERVAL and INTERVAL_VARIATION?

MOVE_INTERVAL: Thanks to the good naming we can make a guess that it defines certain boundaries for the movements of the ghosts.

```
public Inky(Map<Direction, Sprite> spriteMap) { super(spriteMap, MOVE_INTERVAL, INTERVAL_VARIATION); }
```

Looking further into the code where behavior of Ghosts are defined (in the abstract class Ghost), we find out from its constructor

```
/**
 * Creates a new ghost.
 *
 * @param spriteMap         The sprites for every direction.
 * @param moveInterval      The base interval of movement.
 * @param intervalVariation The variation of the interval.
 */
protected Ghost(Map<Direction, Sprite> spriteMap, int moveInterval, int intervalVariation) {
    this.sprites = spriteMap;
    this.intervalVariation = intervalVariation;
    this.moveInterval = moveInterval;
}
```

We know that it bounds interval of movement (again, thanks for the method header)

Same procedure for INTERVAL_VARIATION (which is just next to move interval), we know it defines a variation, which is "The random variation added to the {**@link** #moveInterval}".

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

```
/**
 * Creates a new level from the provided data.
 *
 * @param board
 *            The board with all ghosts and pellets occupying their squares.   Dreef, 1/16/20, 11:32 AM • JPacman v3 for SWE265p
 * @param ghosts
 *            A list of all ghosts on the board.
 * @param startPositions
 *            A list of squares from which players may start the game.
 * @return A new level for the board.
 */
public Level createLevel(Board board, List<Ghost> ghosts,
```

So, if we want to add fruit, when the description here would then become “The board with all ghosts and pellets AND FRUITS occupying their squares”, as we may want certain number of fruits to replace the pellets. So we would want to create new instances of fruits where new instances of pellets are created, so **LevelFactory** should definitely be modified.

Also, how does the program read the board, which is drawn in an txt file?

```
############################
#............##............#
#.####.#####.##.#####.####.#
#..........................#
#.####.##.##########.##.####.#
#......##....##....##......#
######.#### ## ####.######
   #.##    G     #.##
######.## ###  ### ##.######
    .  #G G G#  .
######.## ######### ##.######
   #.##         #.##
######.##  ######  ##.######
#............##............#
#.####.#####.##.#####.####.#
#...##......P......##...#
###.##.##.##########.##.##.###
#......##....##....##......#
#.########.##.########.#
#..........................#
############################
```

The board looks something like this, and it is read by functions in “**MapParser.java**”, in which there’s switch-case clauses. We would want to add cases of characters representing the fruits.

We will also need to take care of what happens when players eat the fruit. This lead us to “**PlayerCollisions.java**”, where collision logics are defined.

Further searches for pellets leads to “**PackManSprites.java**”, which creates “Sprite Store containing the classic Pac-Man sprites”. We would want to make sure “fruit.png” is added to the store.