SWE 265P – Reverse Engineering and Modeling

Submitted by: Anjana Krishnakumar Vellore
Date: 1/16/2020

- What is the role of EmptySprite?
  - Empty Sprite does not contain any data. It's basically used in the following scenarios:
    - While retrieving the current frame, if the index of the current frame is greater than the total number of animation frames, an empty sprite is returned as shown below. Here END_OF_LOOP is initialized as an EmptySprite object.

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

    - While creating a new sprite from its base image, if the region is not within the bounds of the base image, an empty sprite is returned as shown below.

```
@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();
}
```

    - It is also used in the test class code to verify whether an empty sprite is returned if the split is not within the actual sprite as shown below

```
@Test
public void splitOutOfBounds() {
    Sprite split = sprite.split( x: 10,  y: 10,  width: 64,  height: 10);
    assertThat(split).isInstanceOf(EmptySprite.class);
}
```

- What is the role of MOVE_INTERVAL and INTERVAL_VARIATION?
    - MOVE_INTERVAL: serves as the base or minimum time interval while calculating the delay/interval between successive moves of a ghost.
    - INTERVAL_VARIATION: serves as the maximum variation allowed from the base time interval while calculating the delay between successive moves of a ghost. Hence delay/interval between successive moves could be any value between MOVE_INTERVAL and MOVE_INTERVAL+ INTERVAL_VARIATION. This adds randomness to the movement of the ghost

```
protected Ghost(Map<Direction, Sprite> spriteMap, int moveInterval, int intervalVariation) {
    this.sprites = spriteMap;
    this.intervalVariation = intervalVariation;
    this.moveInterval = moveInterval;
}

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?
    - The implementation of fruit can be considered similar to the implementation of a pellet. Hence it can be added to all those files in the code that implements pellet as shown below.

        - We need to create a new class for Fruit similar to the Pellet class

```
public class Pellet extends Unit {

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

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

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

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

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

- PacManSprites.java – Here we should include methods to load each fruit sprite from the fruit images stored in the resources/sprite folder

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

- LevelFactory.java – We should add separate methods to create each kind of fruit.

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

- PlayerCollisons.java – Here we should add the code to handle the situation when a player gets collided with a fruit

```
@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);
    }
}
private void pelletColliding(Pellet pellet, Unit collidedOn) {
    if (collidedOn instanceof Player) {
        playerVersusPellet((Player) collidedOn, pellet);
    }
}
public void playerVersusPellet(Player player, Pellet pellet) {
    pellet.leaveSquare();
    player.addPoints(pellet.getValue());
}
```

- MapParser.java – Here we need to make changes to the addSquare method to include fruits

```
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);
    }
}
```

- DefaultPlayerInteractionMap.java – Include an entry for the collision of a player with a fruit

```
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;
}
```