Reverse Engineering HomeWork – Jan 16

NAME: AMAN BHATIA

1. What is the role of empty sprite?

A. Any class that implements the sprite interface uses the draw method, to draw a sprite on a given Graphics context, based on certain parameters like height, width and starting coordinates. This draw method uses the split method to retrieve, a region from an image file. If however, the draw is unable to get that region from the file, an empty sprite is returned, denoting nothing, I believe this is done, so that in the code, whenever there is any reference to sprite, we can check if the sprite is empty, or does it have values and exists on the graphics context.

The following code snippet shows that EmptySprite does nothing on draw

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

@Override
public Sprite split(int x, int y, int width, int height) {
    return new EmptySprite();
}
```

The following code snippet is an example of how an Empty Sprite is returned while splitting in ImageSprite

```
public Sprite split(int x, int y, int width, int height) {
    if (withinImage(x, y) && withinImage(x + width - 1, y + height - 1)) {
        BufferedImage newImage = newImage(width, height);
        newImage.createGraphics().drawImage(image, 0, 0, width, height, x,
            y, x + width, y + height, null);
        return new ImageSprite(newImage);
    }
    return new EmptySprite();
}
```

2. What is the role of MOVE_INTERVAL and INTERVAL_VARIATION?

A. MOVE_INTERVAL denotes the initial or default delay between each move of the ghost. INTERVAL_VARIATION is a number which is used to get a random additional delay on top of the MOVE_INTERVAL. The minimum delay that can be achieved by the ghost is MOVE_INTERVAL, and the maximum would be MOVE_INTERVAL + INTERVAL_VARIATION.

The following code snippet shows how an additional interval can be added to base delay This is from the base ghost class

```
/**
 * 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);
}
```

3. If you wanted to add fruits where would you add them?

A. Firstly, I believe I could have an entire class inside the sprite package, called "FruitSprites" which would extend SpriteStore, that would behave in a similar manner to PacManSprites, but its main purpose would be to call the loadsprites method in the superclass.

I believe having a separate class here for FruitSprites, would make the code more modularized, but I could also have the code for loading FruitSprites inside the PacManSprites itself.

The following code snippets shows how a sprite is loaded, the code to load fruits would be similar

```
/**
 * @return The sprite for the ground.
 */
public Sprite getGroundSprite() {
    return loadSprite("/sprite/floor.png");
}

/**
 * Overloads the default sprite loading, ignoring the exception. This class
 * assumes all sprites are provided, hence the exception will be thrown as a
 * {@link RuntimeException}.
 *
 * {@inheritDoc}
 */
@Override
public Sprite loadSprite(String resource) {
    try {
        return super.loadSprite(resource);
    } catch (IOException e) {
        throw new PacmanConfigurationException("Unable to load sprite: " +
resource, e);
    }
}
```

Also, since there are many kinds of fruits, we could also have a FruitFactory, kind of like the other factory classes, the purpose of which would be to generate fruits. This will help us separate the “creation code” from other functionality. Now since each fruit can also have some points associated with them, each fruit class generated by FruitFactory could be modeled like the pellet class, with variations, for instance Apple could have 10 points, whereas Cherry could have 20.