Phase 4 Report

The Game

In "Dead City Chronicles," the world has changed. What was once a bustling city is now a quiet, dangerous labyrinth. Where lively streets once stood, now there are only the sounds of shuffling zombies and hidden dangers waiting for the unwary. Your mission is to collect all the keys to escape while either evading the blood-thirsty zombies or neutralizing them with vaccines. Watch the game trailer here:

https://www.youtube.com/watch?v=lvh7uye36d4

At first, we were planning to give the player the option to name the character. However, we decided to go with the static name “Carl” since it was the main character’s son’s name from The Walking Dead TV show. When Carl dies in the game, we hear Rick from the TV show calling his name.
We also drifted away from a gun animation for our character. Simply, there was not enough time at the end, and it was only an animation with no functionality, so we decided to skip that. Instead, we focused more on the functionality of the game.
Moreover, we decided not to give any health-boosting item for the player to get since the character had enough health to finish the game. We did not want the game to be too easy to finish.
Also, we decided not to include a leaderboard. This decision was also because of the limited time we had to work on the game, and we preferred to work on more important factors.

Nevertheless, there were certain elements that we decided to add midway through the project. We created sounds for every section of the game and went above and beyond with our game state functionality. We decided to be more creative.
Additionally, we decided that the game would not be as simple to win because, rather than merely gathering all the keys in the game, you would need to visit a specific house.

Most important lessons we have learned:

- Refactoring is vital to software development, and it's one of the most significant lessons we've learned. During the refactoring phase, we found out that certain functions could have been designed to be more modular and reusable. After phase 2, we discovered unexpected bugs and issues that were difficult to trace and resolve, refactoring allows us to identify and fix those bugs by breaking down the code into smaller, more manageable components.
- In a group environment, communication is crucial. A forum for exchanging concepts, opinions, and criticism is provided by regular meetings and open discussions. This constant communication promotes teamwork, guarantees goal alignment, and facilitates prompt problem-solving. All of this contributes to a cohesive and productive team environment.
- In software development, in particular, effective communication is critical in a collaborative environment. Team members can efficiently communicate goals, clarify intricate logic, and guarantee code clarity by utilizing comments and JavaDoc.
- We've learned the significance of both white box and black box testing in software development. While black box testing concentrates on outward functionality and verifies user requirements and experience, white box testing, with its internal viewpoint, assures code efficiency and logic correctness. Together, these two types of testing ensure a solid and dependable software product.

We use both white box and black box testing techniques in our testing procedures. These methods guarantee thorough coverage and offer insightful advice for efficient code reworking.

- Work in branches to improve workflow and minimize merge conflicts. Even though it is more painful in the present, it is beneficial in the long run.
- We discovered how to manage our projects using a scrum methodology, handling each stage as a sprint. With this approach, we were able to concentrate on delivering targeted features in brief iterative cycles, improving teamwork and flexibility while guaranteeing ongoing development and enhancement.

Features and Scenarios:

In the video linked above, we briefly highlight key features in our game in a trailer format. In this section of the report, we will cover each feature and scenario in depth.

Character Sprite and Animation

Main character Running

Zombie 1 Running

Zombie 2 Running

Zombie 3 Running

Map

Damaging Collisions

When the player collides with a zombie or a trap, we change the sprite for a moment to indicate player getting damaged.

Zombie AI

To create an immersive and realistic gameplay experience, we designed the zombies with a basic AI that allows them to chase the player instead of following a predetermined path. This dynamic aspect of the game introduces a layer of unpredictability and excitement, as the zombies must navigate an environment filled with obstacles, like walls, that require real-time decision-making.

The implementation of this AI was one of the more challenging tasks we undertook. The logic for controlling the zombies' movement needed to be like they are hunting for human flesh and we think that we accomplished that. While the zombie AI is not perfect, we are happy with the way it turned out.

```java
if (collisionOn) {
    // Check for viable alternative paths
    boolean canMoveVertically = !gp.cChecker.checkCollision(this, worldX, worldY + (yDistance > 0 ? speed : -speed));
    boolean canMoveHorizontally = !gp.cChecker.checkCollision(this, worldX + (xDistance > 0 ? speed : -speed), worldY);

    if ((direction.equals(anObject:"left") || direction.equals(anObject:"right")) && canMoveVertically) {
        direction = (yDistance > 0) ? "down" : "up";
    } else if ((direction.equals(anObject:"up") || direction.equals(anObject:"down")) && canMoveHorizontally) {
        direction = (xDistance > 0) ? "right" : "left";
    }
} else {
    worldX = nextX;
    worldY = nextY;
}
```