# CMPT276 Assignment 3: Code Review

By Tawheed Sarker Aakash and Josh Le

Tawheed and Josh have worked on the engine section of their final project. While Tawheed worked on the maze generation, Josh has worked on integrating the maze with other aspects of the game. They both collaborated jointly to work out the inner functionalities of the game as well as all required logic to make the game run smoothly. As this is the first collaboration, things didn't go as smoothly as planned. What initially seemed to be good code blocks looked like major code smells in multiple revisions. We have listed 6 of the worst code smells we had on the project with their commits and the challenges that were encountered during the refactoring process.

## Code Smell 1: God method in Maze Generation

### Before Commit: a5beceaf3514012f3a70670187bccf67a7352255

### After Commit: d701dd6389e7b19539a62c09ac7642a85f6b197b

One code smell we noticed was in our maze generation class. Previously, maze generation was managed by two methods: createCanvas() and createMaze(). The latter was responsible for constructing the pathways using Prim's Algorithm and ensuring that it resulted in a valid maze. However, after refactoring, we decomposed the methods even further and integrated more helper methods to support us in creating the maze. This improved the readability and maintainability of the code which allows for future extensions.

## Code Smell 2: X and Y problem

### Before Commit: d6e1608c4ce6aa15be7cd3263a75ceaf489fe900

### After Commit: e4cb1c586c89eaa2e39169224ebbcda69272a234

In our prior iteration, we utilized x and y as coordinates for the maze. However, as we progressed with building other components of the game (such as entities, maze, etc.), miscommunication among collaborators led to confusion between x and y. Consequently, we encountered problems with player movement and maze orientation. To address these issues, we had to make some illogical adjustments to ensure the game ran smoothly. However, we eventually replaced the x and y coordinates with height and width, which offered better clarity on which field represented what. Additionally, we established some

general guidelines for utilizing the Point class. This change enabled us to work effectively with the Point class and carry out the necessary refactoring.

## Code Smell 3: Hard-coded values

### Before Commit: 3baab7e60f34e5d12b38914dbeab8c4e82a843b6

### After Commit: 01853ee394781fb2a65ec346fe84ecd91b696aaa

Another code smell that we noticed was that during the game engine development, we utilized hard-coded values extensively. This was partly due to the X and Y issue mentioned previously. This is a problem because this didn't allow us to . To address this problem, we modified the code to use the Maze class' getter methods instead. One significant benefit of this approach was that it allowed for future level creation, which could be seamlessly integrated into the game.

## Code Smell 4: Lack of Documentation

### Before Commit: 371c2476cf676a5793779f85164dc3d479bb8580

### After Commit: 487e544741f4e8b16356a3dd4fd5a95bb5d36c4a

One crucial code smell that was particularly noticeable early on was the absence of documentation. This presented a significant challenge during the collaboration process. We frequently encountered roadblocks in our work because we lacked clarity on the purpose of specific methods. To mitigate this issue, we added documentation to the code, which proved to be extremely beneficial. The documentation enabled us to leave notes for our teammates, facilitating better understanding and utilization of the methods.

## Code Smell 5: Use of god method for trap and enemy

### Before Commit: 45bac4f2e3d68ee6f45032b636ca25ebf1ee3f15

### After Commit: 3baab7e60f34e5d12b38914dbeab8c4e82a843b6:

The use of polymorphism for entity generation resulted in repetitive code blocks. To minimize code duplication, we refactored these common code blocks into helper methods. After each player move, the traps and enemies were checked. However, this process was previously executed through a single large "god" method, which made the code difficult to read and maintain. To improve the code quality, we decomposed the god class into smaller methods, which improved the readability and maintainability of the code.

## Code Smell 6: Public helper methods

### Before Commit: 0c0fde4781fd6f7ef4010a4e18aba56e8a2650ca

### After Commit: 8018243b9a465c80dc62d1fc16c7b28bcc312b8a

The helper methods were made public, which was likely due to the tendency of methods being more public than private by default, as well as muscle memory. To resolve this issue, we had to monitor the usage of each method and convert them to private if they were only used within the class. This allowed for encapsulation to be kept and also allowed for other members of the team to not be overwhelmed by the public methods that a particular class had.