APPROACH:

- Use of Java's libraries and classes: JFrame, JPanel, Graphics, KeyHandler, MouseHandler to control the graphics of the game.
- Implementing the interfaces that were imported by overriding their functions to include features of our elements of the game.
- Sub-classing and packaging to divide responsibility and use modularity as an effective technique to improve the layout of code.
- Using Thread to run the loop of the game.

Title: Catch the Code

Player: Coder

Movements: UP, DOWN, RIGHT, LEFT, p for pause, r for resume, s for save, mouse-click for start and quit

Enemies: Backspace Key

Rewards: Pieces of code:

- **Regular:** HTML tags, Semicolon, curly brackets
- **Bonus:** laptop

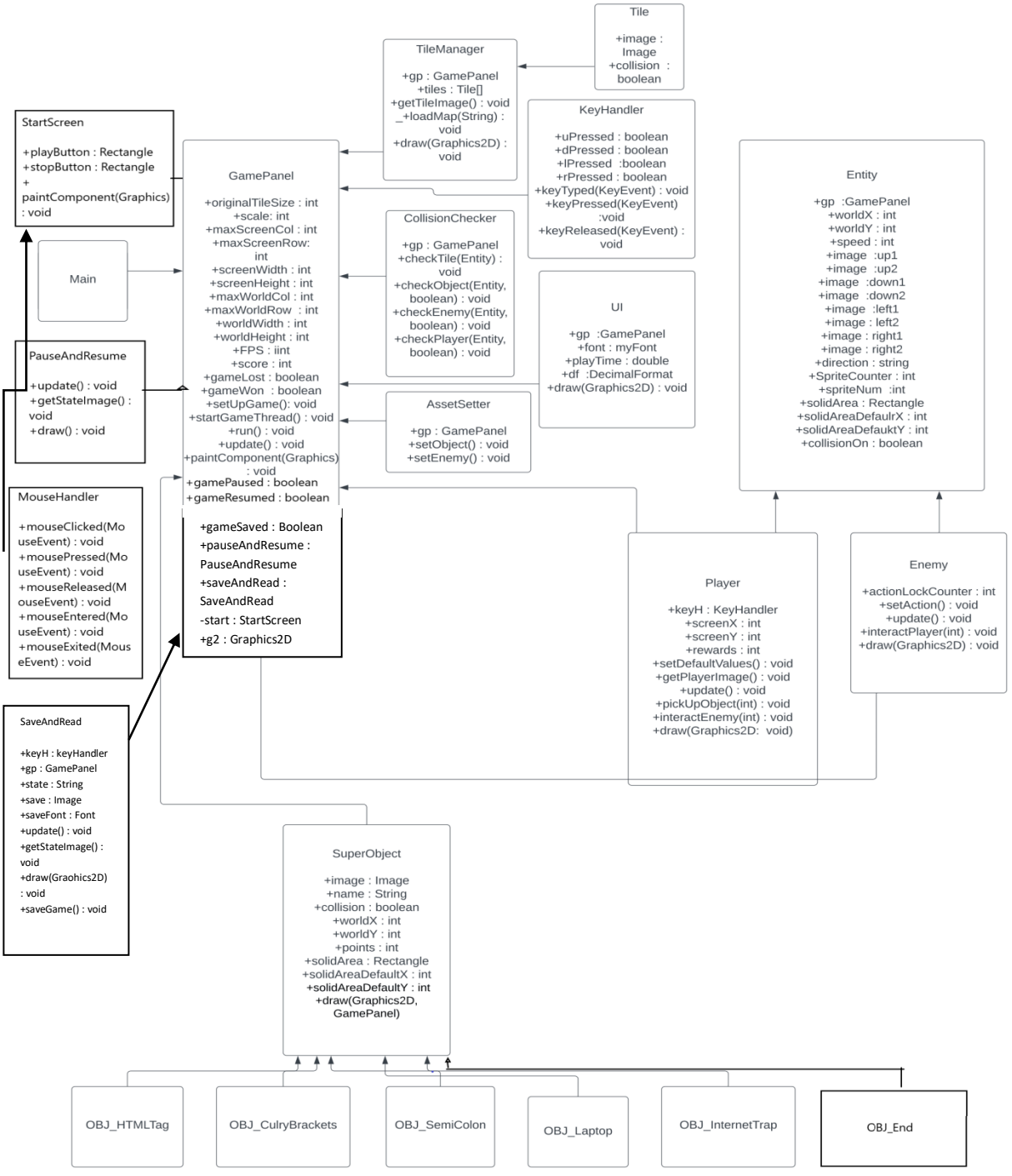
Punishments/Penalties:, InterNet Trap

Barriers: binary code tiles

Board: Grid/Maze – some cells contain rewards (+ points), and others contain punishments (- points). If we move to a cell that has an enemy or if the enemy moves towards us we lose the game.

Story/Background: The coder falls asleep and finds themselves caught in a program. They have to collect the pieces of code in the right order, before the enemies get to them or they come across their punishments and lose points.

MODIFICATIONS: The entire UML game of Phase 1 has changed as we there were many components of the code that we hadn't accounted for.



Also, certain visual aspects of the game were changed:

- The barriers were made into binary code instead of comments.
- The bonus reward was turned into a laptop instead of functions.
- The rewards became code's pictorial representation and not pieces of code.

Reason: These were easier to draw and looked better on the board.

MANAGEMENT & DIVISION:

- The initial division was:
  - Swagi Desai: Board, Tiles, GamePanel, Main, Time and Score – use case
  - Estella Deng: Enemy (Entity), Start and Quit – use case
  - Hojung Park: Rewards and Punishments, Pause and Resume – use case
  - Ray Liu: Player (Entity) – Save – use case

But as our requirements and UML diagram changed, the division changed and each member contributed towards the code of the game.

EXTERNAL LIBRARIES: None used. Only Java libraries were used for this project.

IMPROVE CODE QUALITY:

- Class hierarchy and inheritance was used to reuse common code and clearly divide each element of the game.
- Java documents and other comments made understanding the code easier.
- Kept the colors of different elements such that they appear on the board.

CHALLENGES:

- None of us had experience with Maven or creating a 2D game before, and thus searching the right tools to create the game was difficult at the start.
- We had a hard time integrating the classes initially, but we figured it out finally.