Explanation on how our OOD will work

1) Board Update

Start game:

Place player on top left corner, reveal map where the player moves, gets player position from the player class and updates in the Maze Class.
Place cats on the rest of the three corners of the board, gets cat position from the cat class and updates it in the Maze Class.
Place cheese in a random passage position on the board, update cheese position in the Maze Class.

K + 1 steps:

Reveal visited and attempt to visit positions on the board, updates the CellState.
Reveal cat position, update CellState and update cat position in the Maze Class.
Update cat movement without revealing previous or next cat position.

End Game:

Player reaches the target (get cheese position from Maze Class) check if the position of the player is the same as the cheese, change the state of the game to end and update the score of cheese. If the player position is the same as the cat position or vice versa the game sate updates to end choosing the losing condition. Resets the board upon change of state to end.

2) Reveal map

Maze class generates a maze mask during generateMaze() method. This mask is a field in the maze called mazeMask and it is of type CellState[][] same as the actual maze. A CellState is an enum that contains states in a cell of a maze: PASSAGE, WALL, PLAYER, CAT, CHEESE, and HIDDEN

The mazeMask is initially filled with the HIDDEN element and is width -2 and height -2 of the actual maze.

When a player occupies a space in the maze the mazeMask will be updated according to the player’s position. To do this we copy over the elements surrounding the player in the actual maze to the mazeMask and print it to the user. Thus revealing a portion of the maze which the player sees.