**Design Check-in #1**

**Detecting if cat has path to edge**

We are planning on making an array that holds the path length from the cat to each one of the edge vertices. We will be weighing walls with an edge weight of 1000 (or a similar large number). Any edge that ends up at a wall vertex will have a weight of 1000, while edges between two non-wall vertices will have a weight of 1. Using Djikstra’s algorithm to keep track of the path length from each edge vertex to the cat, as well as the last node traveled to, we can find the *shortest path* from each edge vertex to the cat. After every move (and at the beginning of the game) we will check the path length from the cat to each exterior vertex, and if every path length is over 1000, that means the cat would have to go through a wall to get out, which is not possible. When all the path lengths are over 1000, the game is over, and the player wins. If the cat escapes, the cat player wins.

**Knowing what spaces are on the edge of the board**

We will have an ArrayList<ArrayList<Blocks>>. Each row in the board will correspond to an ArrayList<Blocks>. There will be a bigger ArrayList that holds each row (technically the board as a whole). The program will know whether or not it is at an edge by checking the position in within the row. If it is in index 0 or index size-1 for a particular row, it is at one end or the other of the row, and consequently, one edge or the other of the gameboard.

**Knowing what valid neighbors of each space are**

Due to the nature of the gameboard not being a nice square, some measures will have to be taken in order to make sure the cat moves in a legal manner. This is where the first instance of special case code comes into play. Through a series of if-statements, we can relegate the cat to only legal moves by understanding the following.

Suppose row i, position j in a particular row. Let i+1 be the row below (moving down the board) and let i-1 be the row above (moving up the board). Then one of the following will occur.

If cat is in middle row, legal moves are:

i , j-1 i, j+1 i-1, j-1. i-1, j. i+1, j-1. i+1, j

If the cat is in the top half, legal moves are:

i, j-1. i, j+1 i-1, j-1. i-1, j i+1, j. i+1, j+1

If the cat is in the bottom half, legal moves are:

i, j-1 i, j+1 i-1, j. i-1, j+1 i+1, j-1 i+1, j

**Overall game flow**

The majority of the logic for the game will operate within a while loop. While GameOver = false, keep going. If the cat’s position reaches an edge, or the game is won by the non-cat player, GameOver will be set to true at the end of that iteration and the game will conclude.

* This is your list of all classes, interfaces, enums, attributes, and methods (with return types and parameter list).

**Block Class**

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Method** | **Return Type** | **Parameters** |
| containsWall (boolean) |  |  |  |
| containsCat (boolean) |  |  |  |
|  | Block() default const. |  |  |
|  | Block() constructor | N/A | containsWall, containsCat |
|  | getContainsWall | boolean |  |
|  | getContainsCat | Boolean |  |
|  | setContainsWall | Void | containsWall |
|  | setContainsCat | Void | containsCat |

**Model Class**

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Method** | **Return Type** | **Parameters** |
| center(int[])  cat starting spot |  |  |  |
| blocks (ArrayList<ArrayList<Block>> |  |  |  |
|  | getBlocks | ArrayList<ArrayList <Block>> |  |
|  | initializeGrid  (makes game board) | Void |  |
|  | toString | String—this method allows us to test it in the model completely separate from the view |  |
|  | CharNoirModel  (Constructor) |  |  |
|  |  |  |  |

**View Class**

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Method** | **Return Type** | **Parameters** |
| Center(int [])  NOTE: the center attribute as well as the toString will not be in the final project, it is simply for testing the logic separate from the view |  |  |  |
| ArrayList<ArrayList  <Rectangle>> blocks  NOTE: blocks will not exist in the model class after testing the logic is complete. This array will stay however as it is the view |  |  |  |
|  | chatNoirView | N/A (constructor) | gameModel |
|  | initializeBlocks  (makes game board) | Void |  |
|  | createBlock  (creates individual blocks in the board) | Rectangle | xCoord(int) yCoord(int) |
|  | addBlocks | Void |  |
|  | formatBlocks  (method devoted to formatting blocks to make board) | GridPane | blockToFormat (Rectangle) |
|  | Handle | Void | e (Mouse event) to handle user clicks |

**Main Class**

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Method** | **Return Type** | **Parameters** |
| root (VBox) |  |  |  |
| Button (resetButton) |  |  |  |
| Label (feedback) |  |  |  |
| gameModel (ChatNoirModel) |  |  |  |
|  | Start (runs the GUI) |  |  |
|  | Main |  |  |
|  | Handle | Void | ActionEvent e |
|  | Redraw  This method redraws the view of the game after every move | Void |  |