**Ataque zombi en Atomic labs**

(Prueba para Backend)

La solución se basó en una clase “Node”, manejada con estados, el algoritmo A\* pathfinding algorithm, y pygame para hacer la parte visual.

Lo primero fue entender lo requerimientos y hacer un diseño conceptual de como se comunicaría el programa.

Ya con esto pude empezar a definir la clase “Node” la cual es una representación de un lugar en la tabla. Esta clase (por simplicidad y falta de tiempo) esta manejada por estados, un Nodo puede estar en el estado vacío, o estado zombi, o estado humano, etc… Y Estos estados son los que controlan el comportamiento del nodo.

Ahora, venía hacer una representación de una tabla, la cual está hecha por una lista de listas, de 20 x 20, y cada fila llena de nodos. Y basándome en la “foto” de la oficina, (haciéndolo a mano) se crea la representación del lugar. Igual se usa un algoritmo para generar las posiciones aleatorias de los zombis.

Con la abstracción de los nodos lista, continué haciendo lo visual, pygame fue muy útil en esta parte, cada nodo se dibuja como un rectángulo y dependiendo de su estado, es su color.

Ya teniendo lo visual, venía lo interesante, mover los nodos humanos hacia la salida. La solución que se me ocurrió fue que para cada Humano hacer el algoritmo A\*, para encontrar el camino hacia una de las puertas de la salida, añadir ese camino al nodo y en cada iteración moverlo. Lo cual funciono bien, pero los problemas fueron los siguientes:

* En un punto dos humanos tendrían la misma ruta y se encimarían.
  + Esto se solucionó checando si algún nodo vecino del nodo que se quiere mover ya es humano, (pensando como en una fila para salvarse ¿?) el nodo no se mueve, así no se enciman.
* Los humanos siempre siguen la misma ruta y al final puede verse una fila enorme de humanos que quieren salvarse.
  + Solución: ¿?
* El algoritmo tiene una complejidad de O(n) en el peor de los casos, tenemos que revisar todos los nodos para encontrar el camino, y puede empeorar más a O(v), si en algún punto tenemos a todos los nodos en el open\_set; tendríamos que revisar todos los nodos vecinos de todos los nodos, por lo que para tablas más grandes puede no ser el más eficiente. Y correrlo, en este caso, veinte veces, para cada humano, puede afectar en el desempeño.
  + En este caso podría usar el algoritmo de Dijkstra's, el cuál tiene una complejidad similar, pero soló tendría que correrlo una vez para tener el camino de todos los humanos, lo cual mejoraría la eficiencia. (TODO)

Igual, ahora venía la parte de mover a los zombis, intente varias soluciones, pero la que mejor funciono, fue:

1. Checar los vecinos del zombi, (Paredes, ventanas, Humanos y zombies no cuentan como vecinos en este caso) y agarrar una posición aleatoria y moverlo a ese Nodo.
2. Repetir 1. Cuatro veces.
3. Fin

Para infectar humanos es algo similar, reviso si algún vecino es humano, si sí, infectarlo, y en la tercera iteración de estar infectado, se vuelve un zombi.

Para que la visualización se pueda ver, en el método main, hay un time.sleep(1), son los segundos que espera entre cada iteración. Se puede modificar para tener los frames deseados.

Aún quedan cosas pendientes (Por cosas de tiempo, ya que ando en exámenes finales:

TO DO:

* La espalda de los zombies no funciona bien por lo pútrida que esta, aún no pueden infectar a humanos que estén en las diagonales.
* Revisar el código y hacer optimizaciones donde sea posible.

La información del programa se guarda en un archivo, que se crea en tiempo de ejecución.