***Instituto Tecnológico de Costa Rica***

***Sede Interuniversitaria de Alajuela***

***Lenguajes de Programación***

***Profesora: Samanta Ramijan Carmiol***

***I Semestre 2021***

***Integrantes:***

***-Joan Sánchez Chinchilla -> 2015123867***

***-Tomás Acuna->2018112856***

***-Marco Madrigal->2019053902***

***Segundo Proyecto Programado: Programación Funcional en Racket***

**Datos:**

Para la implementación del proyecto decidimos utilizar 2 estructuras de datos principales. Una matriz y un grafo, donde los 2 se conforman por los mismos elementos. Por lo tanto, la matriz se podrá construir a partir del grafo o el grafo a partir de la matriz. Para la implementación de los 2, se utilizarán listas anidadas con el fin de simular la estructura de las matrices y los grafos.

Esto se debe a que requerimos mostrar un tablero, por lo que la solución óptima es construir una matriz. Sin embargo, como se detalla más adelante, el algoritmo escogido para encontrar la ruta más corta requiere de un grafo, por lo que decidimos realizar esta conversión entre estructuras para no tener inconvenientes a la hora de realizar la implementación.

Se creará también una carpeta de imágenes, en donde almacenaremos las imágenes necesarias para representar el tablero, el personaje, la meta y los obstáculos. Estas imágenes se definirán en una función para utilizarlas donde sea necesario.

Finalmente, para el principio de la ejecución del programa se necesitará definir el punto de partida y la meta. Estos 2 parámetros se usarán en los algoritmos del programa para calcular la ruta más corta.

**Descripción de los algoritmos seleccionados:**

**Dijkstra:** Para encontrar la ruta más corta entre el punto de partida y la meta, decidimos trabajar con el algoritmo de Dijkstra. Este algoritmo se considera como un algoritmo de búsqueda y trabaja utilizando un nodo de origen y un nodo final que puede ser cualquiera dentro de la misma red de nodos. Dicha red puede verse como un grafo y para su funcionamiento se utilizan los pesos de las aristas para poder determinar cada movimiento hasta encontrar el mejor (el más corto).

**MovimientoPersonaje:** Para saber hacia dónde debe ver el personaje en cada movimiento, crearemos un algoritmo que tenga como parámetros la posición actual y la posición siguiente. De esta manera podemos saber si la imagen del personaje debe mostrarse desde su perfil derecho (si gira a la derecha), izquierdo (si gira a la izquierda), de espaldas (si se mueve hacia arriba) o de frente (si se mueve hacia abajo).

**GenerarLaberinto:** Para generar los laberintos crearemos un algoritmo que reparta los obstáculos de manera aleatoria en las celdas de la matriz o nodos del grafo que representen el laberinto.

**MostrarSolucion:** Para mostrarle al usuario la solución del laberinto y cada movimiento necesario que debe realizar el personaje siguiendo la ruta más corta, crearemos un algoritmo que utilice “run-movie” con una lista que obtenga la secuencia de imágenes necesarias. Dicha lista se obtendrá haciendo uso del algoritmo MovimientoImagen descrito anteriormente.

**Lecciones aprendidas:**