Tarea Corta 3

paCE Man

Instituto Tecnológico de Costa Rica

Área Ingeniería en Computadores

Lenguajes, Compiladores e intérpretes (CE3104)

Primer Semestre 2020

Preparado por:

Kevin Acevedo Rodríguez
2018148661
Hazel Martínez Loría
2018002084

Profesor:

Ing. Marco Rivera Meneses

TEC | Tecnológico de Costa Rica

Contenido

Descripción de las estructuras de datos desarrolladas

Para esta tarea fue necesaria la implementación de dos estructuras de datos, una **lista directora** que mantenga el control de las posiciones en las que puede haber artículos en el mapa como los son pac dots, frutas y pastillas, la otra estructura es el **grafo fantasma** que permite validar las posiciones y movimientos de los fantasmas.

Grafo Fantasma: Aunque su nombre hace referencia a un grafo, en realidad es una lista que contiene la información útil con respecto a un grafo no dirigido que se formó para generar las rutas más cortas mediante el algoritmo Dijkstra. Esta lista contiene elementos de tipo **GhostNode**.

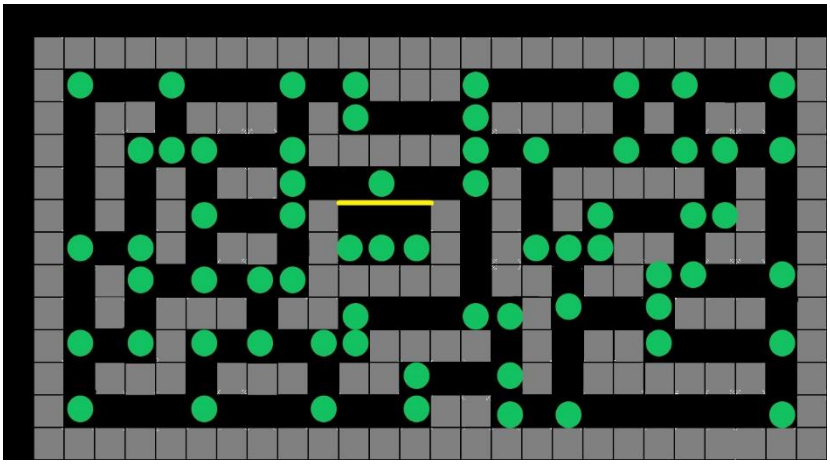
GhostNode: Esta es una subestructura del grafo fantasma, almacena la información más importante de un nodo perteneciente al grafo.

Lleva el control de las posiciones X y Y del nodo, de su nombre (los nombres de los nodos son del tipo → "node_numeroDeNodo"), y además guarda el nombre de los nodos vecinos.

Un **GhostNode** o **nodo fantasma** puede tener máximo cuatro vecinos y mínimo 2. Esto indica que desde cualquier nodo se puede tomar varias rutas para ir a otros nodos del grafo.

Ahora, volviendo al **grafo fantasma**, esta lista está conformada por **64 nodos fantasma**. Cada uno de ellos se encuentran en posiciones específicas del mapa, generalmente en donde los fantasmas pueden tomar varias direcciones desde un mismo punto.

En la siguiente figura se puede observar la localización de cada uno de los nodos que componen al **grafo fantasma**:

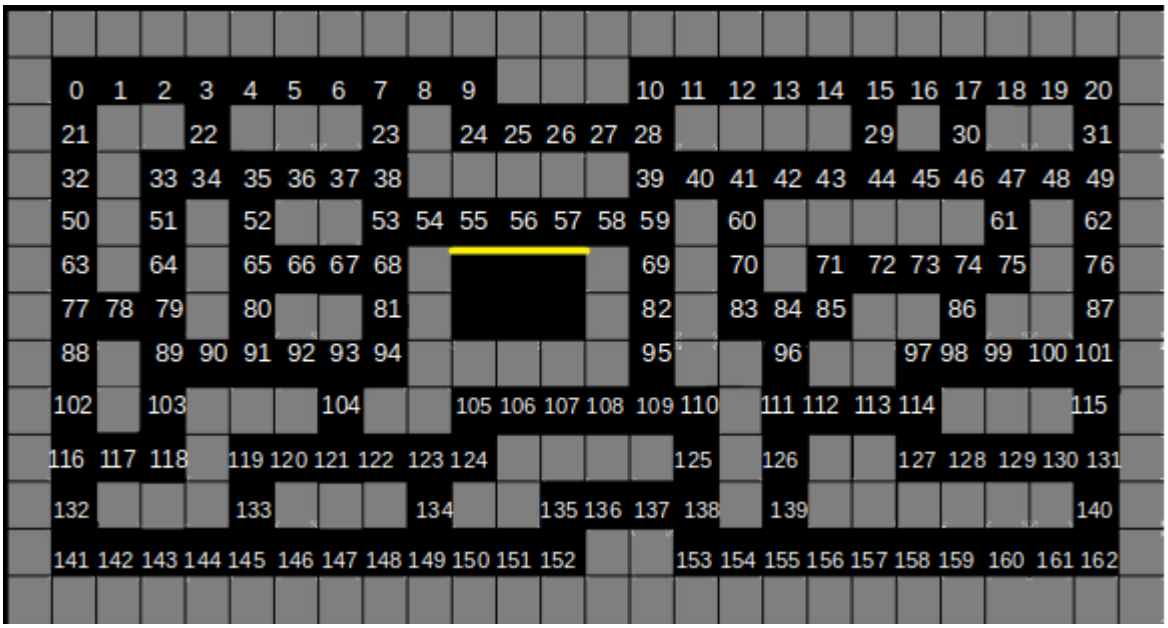


Cada uno de los puntos verdes son los nodos o las posiciones que reconocen los fantasmas. Además, el **grafo no dirigido** implementado para usar el algoritmo Dijkstra está hecho formando aristas entre estos nodos.

El uso principal de la lista es poder generar las **rutas fantasmas**, estas rutas indican las direcciones que deben llevar los fantasmas en el juego. Usando la lista se puede verificar qué dirección se puede llevar a cabo estando en cualquiera de los nodos del grafo.

Lista: Se utiliza una lista para representar cada cuadrante de posición en el tablero de juego, esta lista va desde la posición 0 hasta la posición 162.

Esta lista es llenada en inicio con pac-dots, y en el momento en que el servidor desee agregar una pastilla cuyo comando corresponde a pill,# siendo el segundo parámetro la posición en la lista es decir el tablero se colocaran las pastillas, frutas y fantasmas, siendo esta una manera eficiente de enviar los datos a la estructura lógica, y menos pesada para la RAM.



Descripción detallada de los algoritmos implementados de los fantasmas

En el juego, el personaje principal (Pac Man) tiene 4 enemigos principales que van a tratar de impedir que Pac Man se coma todos los pac dots que aparecen alrededor de todo el mapa. Estos enemigos son los 4 fantasmas del juego clásico y se describen a continuación:

- **Shadow:** Es el fantasma rojo, se le considera como el fantasma más peligroso debido a que en el momento en que el **administrador** decida colocarlo en el juego, este persigue a Pac Man de inmediato. La única manera de dejar de perseguirlo es que Pac Man coma las **pastillas**.

- **Bashful:** Es de color celeste, este fantasma no tiene objetivo fijo como Shadow, sino que sus movimientos son impredecibles, en otras palabras, sus movimientos se ejecutan de manera aleatoria.

- **Pokey:** Es de color naranja, tiene un objetivo claro, a diferencia de Shadow, Pokey no persigue a Pac Man al aparecer en el juego, sino que siempre busca las **esquinas** del mapa, ya que estos son los lugares en los que aparecen las **pastillas** iniciales.

- **Speedy:** Es de color rosa, se puede decir que es el segundo fantasma más peligroso, su comportamiento cambia cada cierto tiempo. Cuando sus ojos son de color rosado Speedy rodea la casa fantasma impidiendo que Pac Man se acerque, pero cuando sus ojos se tornan color amarillo se vuelve violento y en seguida persigue al jugador.

Antes de explicar cada uno de los algoritmos es necesario introducir un término nuevo, el cuál es la **ruta fantasma**. Es una lista que almacena 4 tipos de Strings → ("R", "L", "U", "D"), hacen referencia a las direcciones que pueden tomar los fantasmas, Right, Left, Up y Down. Esta lista indica a los fantasmas qué dirección deben tomar comenzando desde el primer elemento almacenado en la lista hasta que ya no haya elementos en la lista (esto significa que el fantasma se debe detener) o cambien las direcciones (en algunos momentos la ruta fantasma puede cambiar las direcciones porque los algoritmos reconocieron una ruta mejor o más corta para lograr el objetivo).

Ahora se van a explicar los diferentes **modos fantasmales** que se implementaron usando 4 algoritmos:

Modo Persecución: Este modo es intrínseco de Shadow, sin embargo, Speedy lo aplica cada cierto tiempo. Este algoritmo permite que los fantasmas obtengan la posición de Pac Man para luego generar una **ruta de persecución**.

Cada uno de los algoritmos fantasmales utilizan el **Grafo Fantasma** explicado en la sección anterior para poder localizar a Pac Man. Cada uno de los **Nodos Fantasma** tienen una posición definida en la pantalla, de esta manera es posible reconocer la posición de Pac Man para notificar a los fantasmas que la necesitan para perseguirlo.

Cada vez que Pac Man toque un **nuevo nodo fantasma** se le envía el número del nodo a Shadow y Speedy. Seguidamente empieza ejecutarse el algoritmo:

Paso 1: Una vez que se haya actualizado al **nodo fantasma** que Pac Man toca, se guarda el número de dicho nodo (recordar que cada nodo fantasma del grafo tiene un número particular y específico) como el **nodo destino**.

Paso 2: Se espera a que el fantasma toque un nuevo **nodo fantasma**. La idea es ejecutar los procesos cada vez que el fantasma ejecute las direcciones de la **ruta fantasma**.

Paso 3: Se procede a **vaciar la ruta fantasma**, esto debido a que la posición de Pac Man puede variar con facilidad y velocidad, por lo que se espera que el fantasma cambie la ruta.

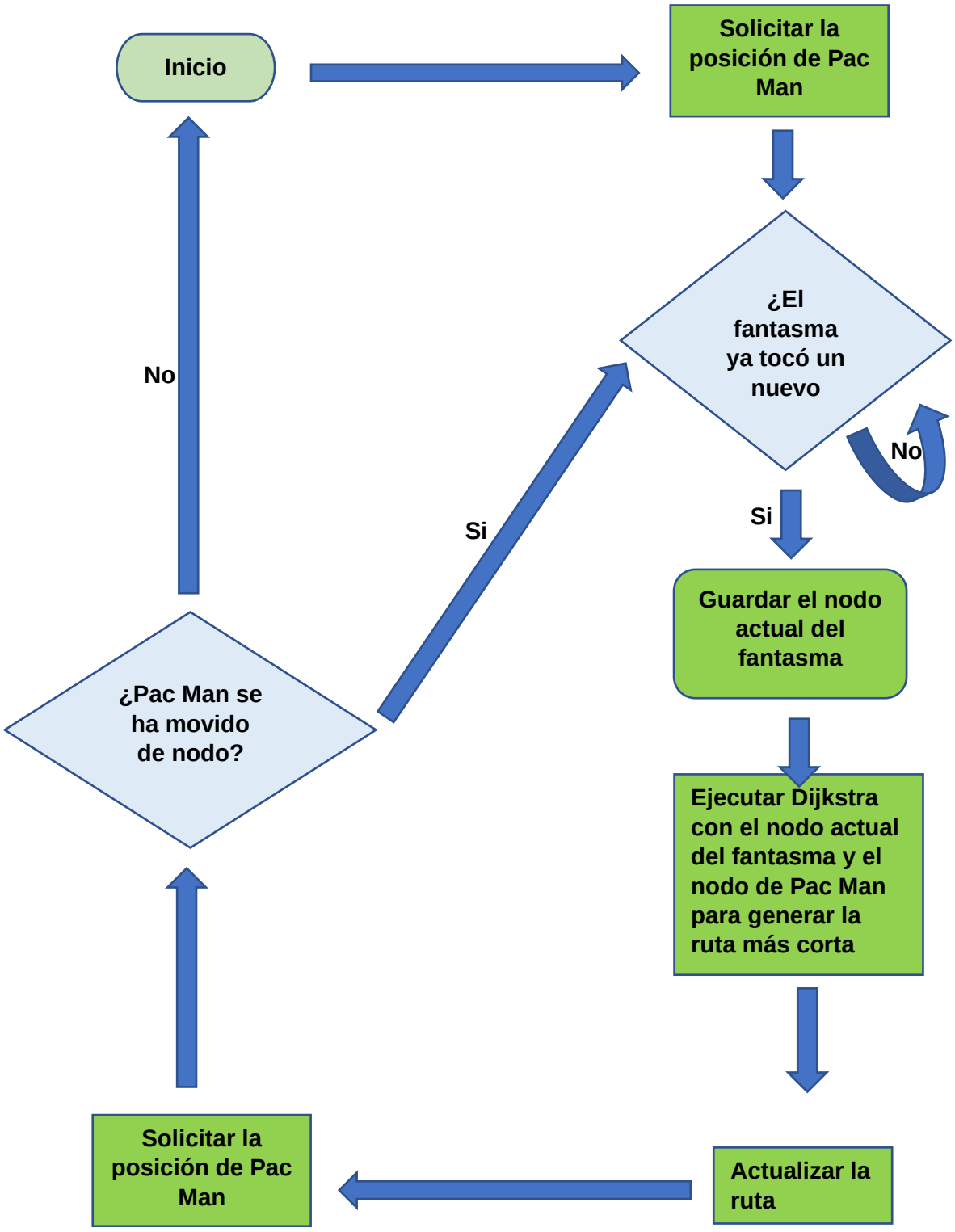
Paso 4: En este punto el fantasma justo acaba de tocar un **nodo fantasma**, por esta razón, se guarda el **número del nodo** en el que se encuentra. Ahora ya se tiene el **nodo de partida** (en el que se encuentra el fantasma) y el **nodo destino** (El nodo en el que se actualizó la posición de Pac Man).

Paso 5: Con la ayuda del algoritmo Dijkstra aplicado en el **grafo fantasma** y usando el **nodo de partida** y el **nodo destino** solicitamos la **ruta más corta** para poder llegar hasta Pac Man, esta ruta se guarda por medio de las direcciones →R, L, U, D en la **ruta fantasma**.

Paso 6: El fantasma reconoce las direcciones que se encuentran en la lista y se mueve hasta que la lista esté vacía. Cada vez que el fantasma toque un nuevo **nodo fantasma** significa que ha ejecutado una dirección de la lista, por eso, se elimina la posición anterior y se ejecuta la siguiente hasta que la lista quede vacía o cambie la **ruta fantasma**.

Para poder reconocer cuando Pac Man o alguno de los fantasmas toca cualquiera de los **nodos fantasmas** del grafo, se usan rectángulos invisibles que rodean a los personajes del juego. El uso de los rectángulos permite también validar colisiones entre Pac Man y elementos coleccionables como pac dots, pastillas y frutas, entre Pac Man y muros y además con fantasmas.

Ahora se va a mostrar un diagrama de flujo que permite visualizar el algoritmo de persecución de manera sencilla:


Inicio
Solicitar la posición de Pac Man
¿El fantasma ya tocó un nuevo
No
Si
Guardar el nodo actual del fantasma
Ejecutar Dijkstra con el nodo actual del fantasma y el nodo de Pac Man para generar la ruta más corta
Actualizar la ruta
Solicitar la posición de Pac Man
¿Pac Man se ha movido de nodo?
Si
No

Modo Dispersión: Este modo fantasma solamente lo poseen los fantasmas Pokey y Speedy. La dispersión consiste en que los fantasmas tengan como objetivo cuatro nodos del grafo, de esta manera solamente se van a dirigir a cualquiera de estos nodos constantemente. El fantasma Pokey tiene como objetivo los nodos que se encuentran en las cuatro esquinas del mapa (en donde aparecen las pastillas iniciales) y el fantasma Speedy en modo dispersión tiene como objetivo las esquinas de la casa fantasma.

La ejecución de este algoritmo empieza cuando el administrador solicita colocar a Pokey en el juego o cuando Speedy cambia a este modo. Ahora se van a explicar los pasos respectivos:

Paso 1: Se debe guardar el nodo al que llega el fantasma (este es una de las cuatro esquinas del mapa), o en caso de que el fantasma recién haya aparecido se omite este paso. Este nodo será el **punto de partida** que se va a utilizar para calcular la ruta más corta usando Dijkstra. Además, se debe tener una variable que indique la **última esquina visitada**.

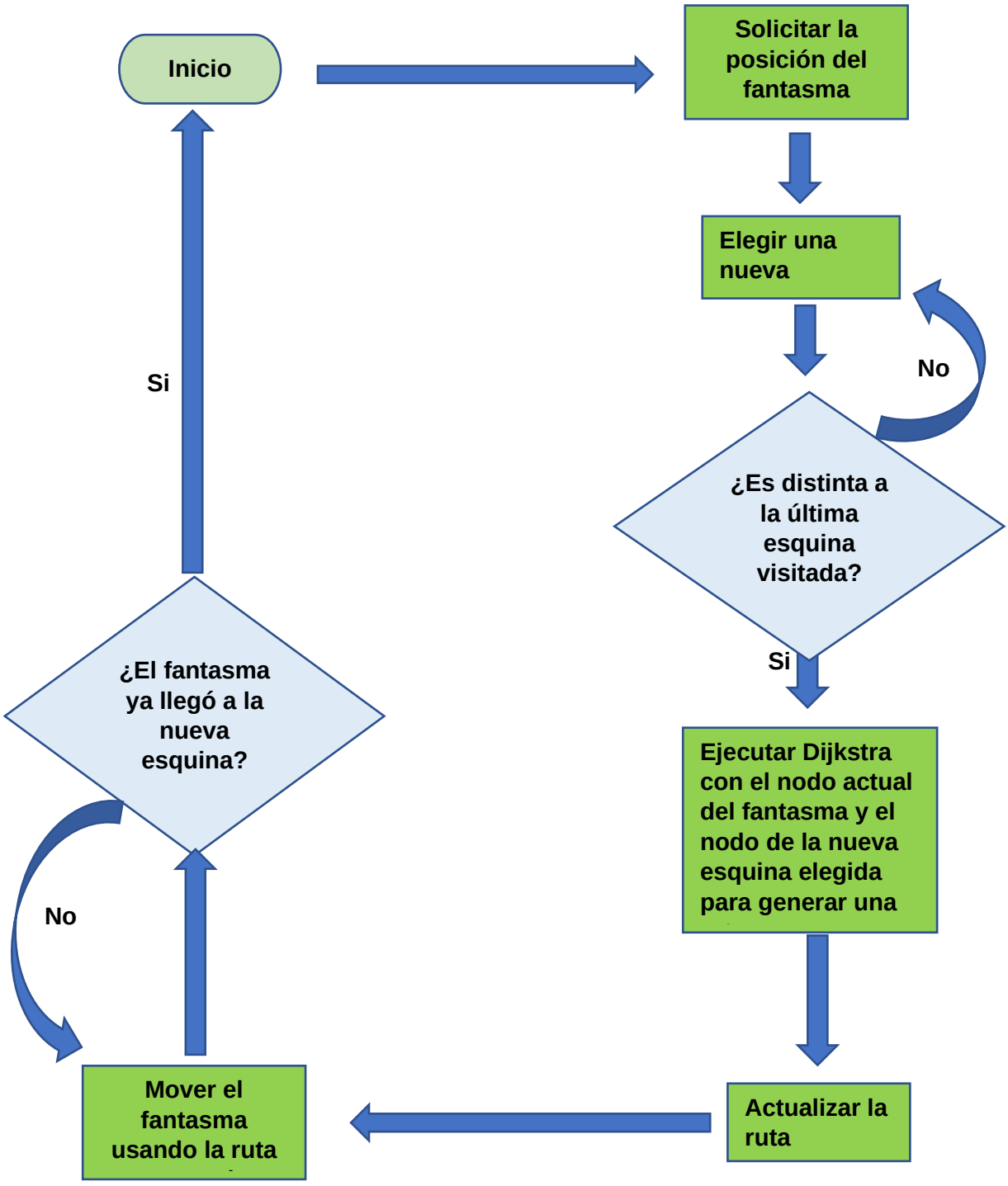
Paso 2: Se elije aleatoriamente una de las esquinas que el fantasma tiene como objetivo. Para ello se genera un número random entre 0 y 3 representando las 4 esquinas a las que el fantasma puede ir. Si ya se eligió una esquina random se debe verificar que esa esquina sea distinta a la **última esquina** guardada en el paso anterior. En caso de que la esquina generada sea igual a la **última esquina** guardada, se debe solicitar otra esquina aleatoria hasta que sea distinta a la última (es necesario para evitar que el fantasma se cicle en la misma esquina).

Paso 3: Se genera una **ruta fantasma** utilizando el algoritmo Dijkstra y pasándole el **nodo de partida** y el **nodo de la nueva esquina generada de manera aleatoria**. De esta manera se agregan **direcciones** a la **ruta fantasma** que le van a indicar a los fantasmas qué direcciones deben de tomar desde la esquina en la que están para llegar hasta la nueva esquina.

Paso 4: Se espera a que el fantasma llegue a la esquina destino. En este momento la **ruta fantasma** tendrá un tamaño de 0 elementos. Ahora se debe regresar al **paso 1** e ir a otra esquina.

Como ya se mencionó anteriormente, Speedy tiene la capacidad de cambiar de modo. Cada 10 segundos puede cambiar a **modo persecución** o **modo dispersión**.

Ahora se va a mostrar un diagrama de flujo que ejemplifica la ejecución básica del algoritmo de dispersión:

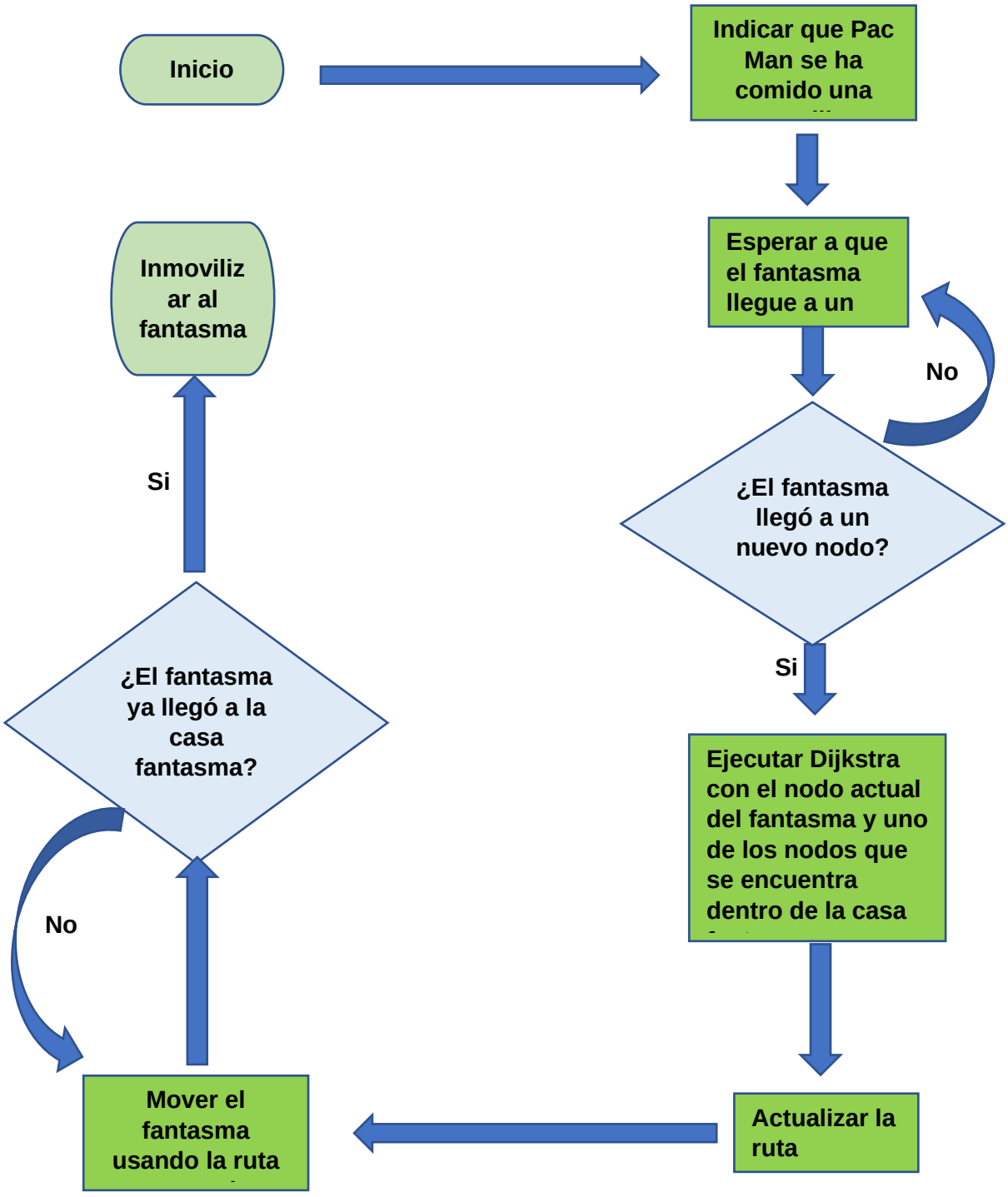

Inicio
Solicitar la posición del fantasma
Elegir una nueva
No
¿Es distinta a la última esquina visitada?
Si
Ejecutar Dijkstra con el nodo actual del fantasma y el nodo de la nueva esquina elegida para generar una
Actualizar la ruta
Mover el fantasma usando la ruta
¿El fantasma ya llegó a la nueva esquina?
No
Si

Modo Escape: Este modo les pertenece a todos los fantasmas y se ejecuta en el momento en que Pac Man coma una de las pastillas que aparecen en el juego. Pac Man puede comerse a los fantasmas cuando está bajo el efecto de la pastilla y para notarlo se puede observar que el color de Pac Man se torna a gris y los fantasmas cambian de apariencia.
Este modo permite que los fantasmas se olviden del objetivo actual y modifiquen su ruta de movimiento (la ruta fantasma) para evitar que Pac Man se los coma. Si los fantasmas se encuentran dentro de la casa fantasma evitan que sean comidos, esto debido a que Pac Man no puede ingresar a este lugar del mapa.
La ejecución de este algoritmo se lleva a cabo de la siguiente manera:

Paso 1: En el momento en que Pac Man come una pastilla se activa una bandera en el juego, esta les indicará a los fantasmas que deben escapar.
Paso 2: Se espera a que el fantasma toque un nuevo **nodo fantasma,** ya que solo se puede cambiar la dirección hasta que se encuentre en uno de los nodos pertenecientes al **grafo fantasma**.
Paso 3: Cuando ya haya llegado a uno de los nodos del grafo, se guarda dicha posición como el **nodo de partida**.
Paso 4: Ahora se debe vaciar y generar una nueva **ruta fantasma** por medio del algoritmo Dijkstra y pasándole el **nodo de partida** y uno de los nodos que se encuentran dentro de la casa fantasma como **nodo destino**. De esta manera nos aseguramos de que el fantasma obtenga una ruta de escape desde su posición actual.
Paso 5: Esperar a que el fantasma llegue a su **destino**. Si ya llegó, se debe inmovilizar al fantasma y liberarlo hasta que termine el tiempo de funcionamiento de la pastilla.

Si por algún motivo Pac Man logra comerse alguno de los fantasmas se deben eliminar del juego. El administrador puede volver a colocar otro fantasma del mismo tipo en el momento que él desee.

Seguidamente, para lograr la mejor comprensión del algoritmo de huida, se va a mostrar su diagrama de flujo respectivo:


Inicio
Indicar que Pac Man se ha comido una
Esperar a que el fantasma llegue a un
No
¿El fantasma llegó a un nuevo nodo?
Si
Ejecutar Dijkstra con el nodo actual del fantasma y uno de los nodos que se encuentra dentro de la casa
Actualizar la ruta
Mover el fantasma usando la ruta
¿El fantasma ya llegó a la casa fantasma?
No
Si
Inmoviliz ar al fantasma

Modo Impredecible: Este modo le pertenece únicamente al fantasma Bashful. Es un comportamiento impredecible porque no es posible saber qué dirección tomará el fantasma.

La ejecución de este algoritmo inicia desde el momento en que el **administrador** solicita colocar al fantasma Bashful en el juego. Los pasos son sencillos y se explican a continuación:

Paso 1: Se espera a que el fantasma toque un nuevo **nodo fantasma**. Una vez se haya cumplido la condición se debe guardar la posición del nodo al que llegó.

Paso 2: Se selecciona aleatoriamente un número del 0 a 3, representando a las 4 direcciones que puede tomar el fantasma. Cuando se haya seleccionado una dirección se debe consultar al **grafo fantasma**. Primero se toma la posición actual del fantasma para indicarle al **grafo** en qué nodo se encuentra. Seguidamente, usando la **dirección aleatoria** se debe asegurar que al tomar dicha dirección se puede llegar a un nodo. En caso de que no se cumpla dicha condición, se debe iniciar el **paso 2**.

Paso 3: Mover al fantasma en dicha dirección y esperar a que llegue a un nuevo **nodo fantasma**. Cuando llegue al nuevo nodo, se devuelve al **paso 1**.

Problemas sin solución

- El cliente observador implementado en Java no muestra los artículos (pac dots, pastillas y frutas) que hay en el Cliente jugador. La comunicación entre ambos programas se lleva a cabo mediante Sockets, pero también exista una comunicación entre el Cliente jugador y el servidor en C. Se decidió mostrar la posición actual de Pac Man, la posición de los fantasmas que se encuentren en el momento, los puntos obtenidos en el momento, el nivel en el que se encuentra el cliente jugador y las vidas que tiene Pac Man en el momento, sin embargo, para no saturar bruscamente la transferencia de datos por medio de los Sockets se omite la información referente a los artículos que aparecen en el mapa ya que no son elementos esenciales para observar el estado principal del juego observado.

Plan de actividades realizados por estudiante

Plan de Actividades realizadas por estudiante				
Descripción de la tarea	**Tiempo estimado**	**Responsable a cargo**	**Estado**	**Fecha de entrega**
Matrices Lógicas	2 días	Kevin	Entregado	25/07
Matrices Gráficas	2 días	Kevin	Entregado	25/07
Sprites pacman, fantasmas ,items	7 horas	Kevin	Entregado	25/07
Game Loop	9 horas	Kevin	Entregado	25/07
Investigación JSON Java/C		Kevin y Hazel	Entregado	25/07
Inicio servidor y cliente		Kevin y Hazel	Entregado	26/07
Servidor	8 horas	Hazel	Entregado	26/07
Control de niveles	4 horas	Hazel	Entregado	26/07
Mover a pacman y actualizar posición en matriz	2 días	Kevin	Entregado	26/07
Crear fantasmas	2 horas	Hazel	Entregado	26/07
Crear pastillas (recibe datos del administrador)	3 horas	Hazel	Entregado	26/07
Frutas con un valor asignado (el administrador lo decide)	3 horas	Hazel	Entregado	2707
Asignación de vidas	2 horas	Hazel	Entregado	27/07
Control de velocidad de fantasmas	2 horas	Hazel	Entregado	27/07
Control de la puntación del jugador	2 horas	Hazel	Entregado	27/07
Algoritmo de colisiones entre Pac Man y paredes del mapa	14 horas	Kevin	Entregado	28/07
Movimiento del fantasma Shadow	16 horas	Kevin	Entregado	30/07
Movimiento de todos los fantasmas	24 horas	Kevin	Entregado	31/07
Control de los artículos coleccionables en el juego	24 horas	Kevin	Entregado	31/07
Creación de el cliente observador implementando el patrón de diseño Observer	12 horas	Kevin	Entregado	01/08
Creación de los sockets para poder comunicar la interfaz en Java y el servidor en C	8 horas	Hazel	Entregado	01/08
Ejecución y pruebas de la interfaz en el sistema operativo Linux	4 horas	Hazel	Entregado	01/08

Pruebas usando el cliente jugador y el cliente observador al mismo tiempo	6 horas	Kevin	Entregado	02/08
Conexión y pruebas entre el servidor en C y la interfaz en Java	24 horas	Kevin y Hazel	Entregado	03/08
Pruebas finales y generación de los ejecutables	12 horas	Kevin y Hazel	Entregado	06/08

Problemas encontrados

- **Problema encontrado:** El IDE utilizado para escribir y compilar el código de Java no reconoce la ruta de las imágenes utilizadas en el programa. Cuando se le especifica la ruta de las imágenes y se compila muestra un error de punteros vacíos y se cae el programa.

  Intentos de solución: 1.

  Soluciones encontradas: Después de investigar acerca del IDE Intellij Idea, nos dimos cuenta de que el proyecto cargado tenía una estructura modificable. La solución encontrada fue indicar que la carpeta que contenía todas las imágenes se usaría como un módulo de recursos.

  Recomendaciones: Se recomienda conocer muy bien las características exclusivas de los IDEs que se utilicen a la hora de programar.

  Conclusiones: El apartado de **proyect structure** que tiene Intellij Idea organiza mejor nuestros proyectos.

  Bibliografía consultada: https://www.jetbrains.com/help/idea/content-roots.html#adding_content_root

- **Problema encontrado:** No se ha logrado implementar colisiones entre Pac Man y las paredes del mapa. Es necesario que el jugador solo pueda moverse por las zonas del mapa en donde no hay paredes, sin embargo, no se ha logrado de ninguna manera.

  Intentos de solución: 2.

  Soluciones encontradas: Se utilizó la teoría de los famosos rectángulos que forman parte de la programación de juegos 2D. Pero para poder validar la colisión con las paredes se usaron 4 rectángulos pequeños e invisibles que rodean a Pac Man. Además, se implementó una clase Wall que forma rectángulos simbolizando las paredes que se observan en el mapa. Para validar estas colisiones se le pregunta a Pac Man cuáles de sus cuatro rectángulos intersecan a las paredes, ya que si Pac Man tiene un rectángulo que no interseca significa que en esa dirección hay un camino disponible, en caso contrario significa que hay una pared y por ende no hay paso.

  Recomendaciones: Se recomienda usar rectángulos para cualquier validación de colisiones en juegos 2D.

Conclusiones: No reinventar la rueda: En este caso solo hizo falta modificar un poco la teoría de los rectángulos para así implementar un algoritmo de colisiones con paredes.

Bibliografía consultada: No aplica.

- **Problema encontrado:** El algoritmo de persecución de los fantasmas no muestra un comportamiento eficiente. Cuando los fantasmas ejecutan el algoritmo de persecución, en efecto llegan hasta la posición de Pac Man, pero esa posición puede cambiar en cualquier momento cuando Pac Man se mueva y los fantasmas no logran darse cuenta de que el jugador ya no está en la misma posición, lo que provoca que sea muy fácil evadirlos.

  Intentos de solución: 1.

  Soluciones encontradas: Se creó una variable que guarde las posiciones que toma Pac Man durante la ejecución del juego. Esta posición se actualiza 60 veces por segundo, así que los fantasmas logran darse cuenta del cambio de posición de Pac Man de inmediato y de esta manera pueden cambiar de dirección e ir hacia la nueva posición del jugador para atraparlo.

  Recomendaciones: Hay que aprovechar la velocidad con que las máquinas ejecutan el código y usan los recursos.

  Conclusiones: Ahora los fantasmas se han vuelto una verdadera amenaza.

  Bibliografía consultada: No aplica.

- **Problema encontrado:** No se ha logrado ejecutar el cliente jugador en el sistema operativo de Linux. Se usa Intellij Idea y un JDK con versión 11.0.8 en Linux para correr el juego, sin embargo, el proyecto se había programado usando la versión 13.0.2 del JDK y no es posible compilar el código en Linux para hacer pruebas porque las versiones de compilación no son compatibles.

  Intentos de solución: 1.

  Soluciones encontradas: Después de investigar acerca de los JDKs y Intellij Idea nos dimos cuenta de que existe un apartado del IDE en donde se puede cambiar el JDK que se usará para compilar el proyecto específico. Luego de cambiarlo por la versión 11.0.8 que se tenía en Linux se pudo compilar el código con éxito.

Recomendaciones: Asegurarse de que los integrantes del proyecto cuenten con los mismos requerimientos.

Conclusiones: Un IDE con buenas características requiere de conocimientos no solo de programación, sino que se debe investigar también las configuraciones del IDE.

Bibliografía consultada: https://mkyong.com/intellij/how-to-change-the-intellij-idea-jdk-version/

Conclusiones

Se concluye que la lista como el manejo de estructura de datos permitió de una manera eficiente enviar y recibir datos por medio de sockets ya que no se recargaba de manera innecesaria al enviar estructuras más grandes.

La estructuración de los comandos permitió una comunicación sencilla debido a que, por medio de mensajes estructurados previamente, la segunda posición siempre indicará la posición donde se realizaran cambios.

Finalmente se reconoce la importancia del manejo de las estructuras de datos ya que estas permitieron el manejo cada una de las partes del juego, permitiendo que se unificara la parte lógica con la parte gráfica de videojuego paCE man.

Recomendaciones

Se recomienda utilizar una lista para el manejo del tablero, y estructurar mensajes de manera que su segunda posición indique el lugar donde se realiza el cambio. La lógica del juego realizada en el servidor de esta manera el recibirá la información para procesar los datos y de ser necesario informar de cambios en la interfaz. Así no se tendrá que pasar una lista por medio de sockets cada vez que haya un cambio sino solamente la posición donde se dio el cambio.