Instituto Tecnológico de Costa Rica

Ingeniería en Computación

IV Tarea Programada

Steven Peraza Porras

Sede San Carlos

Septiembre 2017

Introducción

Este tarea programada trata sobre la creación de una adaptación del popular juego llamado “Battle City” de la consola Nintendo Entertaiment System (NES), el cual trata sobre tal y como lo indica su nombre, de una batalla de tanques, los cuales tienen el objetivo de destruir la base del jugador, por lo cual la meta del jugador es destruir los tanques enemigos antes que la base sea destruída. Esta que recorre una matriz buscando una serie de objetos denominados “manzanas” alrededor de la misma, con la intención de “comerlos” e incrementar su tamaño.

Para lograr esto, se debe implementar un algoritmo de búsqueda en profundidad en el lenguaje de programación Python, mediante la utilización de árboles y/ grafos enlazados, con este algoritmo se podrá recorrer la matriz de juego buscando las manzanas utilizando la menor cantidad de movimientos.

Análisis del Problema

El problema de este proyecto radica en la utilización de árboles y/o grafos enlazados, los cuales deben ser recorridos mediante un algoritmo de profundidad, por ejemplo, se debe crear grafo de pares ordenados, los cuales representarán posiciones dentro de la matriz gráfica, donde dentro de cada grafo irán entre 2 o 4 nuevos nodos hijos denominados “vecinos”, y éstos a su vez tendrán más vecinos, así hasta recorrer completamente todos los nodos del árbol y/o grafo.

El verdadero problema reside en la necesidad de recorrer este grafo/árbol de nodos (vecinos de vecinos), extender estos nodos, recorrerlos y descartarlos para elegir un “mejor vecino”; ya que para esto, se debe comparar cada “vecino” con el destino actual y de alguna manera realizar una función que obtenga un rasgo que diferencie y determine cuál nodo es mejor.

    Además, se deben de crear las funciones que permitan crear el grafo e insertar los nodos extendidos por cada par ordenado de la matriz de 18 filas y 20 columnas, y una función (o funciones) que determinen el mejor nodo y lo inserten en la ruta final, la cual tomará la serpiente del juego. Del mismo modo, es necesario utilizar alguna librería gráfica para la implementación de la solución del problema en un apartado gráfico, es decir, la creación del juego de “Snake”.

    Para terminar, cabe a destacar que la funcionalidad del programa debe ser automática, es decir, no debe existir interacción alguna entre el usuario y la aplicación, el juego debe jugarse por sí mismo. Por esta razón, debe de realizarse otra (s) funciones que se encarguen de este apartado.

Solución del Problema

Para resolver los problemas antes citados, se tuvieron que crear una serie de funciones específicas para grafos, vecinos, fitness y el apartado gráfico.

Primero, para crear el grafo se decidió realizar un diccionario de tuplas, donde cada llave será una tupla con las posiciones que tomarán dentro de la matriz gráfica, y sus respectivos valores serán más tuplas, representando a los posibles vecinos, tal y como se ejemplifica en la figura 1.



Figura 1. Diccionario de tuplas con una lista de vecinos.

Sin embargo, a la hora de crear los posibles vecinos, se debe de comprobar la posición de la tupla; ya que dependiendo de esta, posee 2, 3 o 4 vecinos, por lo que se crearon dos funciones auxiliares que crean vecinos dependiendo de estas posiciones.

A la hora de realizar el recorrido en profundidad se necesitó de una función que recorriera recursivamente el grafo, para poder realizar el descarte de opciones y retornar la ruta correcta mediante backtracking, por lo que se creó un ciclo dentro de la lista de vecinos y una recursividad por cada vecino, esto para lograr la extensión del grafo.

    Para el caso de la elección del mejor “vecino”, se decidió crear una función fitness, la cual se encarga de realizar una pequeña resta o suma entre las filas o columnas de los pares ordenados: el nodo actual y el nodo donde se encuentra la manzana, dependiendo de la posición de la manzana, la función fitness se adapta y elije una de sus 3 funciones auxiliares (una por cada caso, si la manzana está arriba, abajo o en el mismo nivel que la cabeza de la serpiente) para comparar su “porcentaje de proximidad”, en este caso es un entero menor o mayor que los demás vecinos, dependiendo de los casos anteriormente brindados y retorna el nodo que cumple mejor este “porcentaje de proximidad” para que se continúe el recorrido en profundidad en la función principal.

    Para el apartado gráfico, se decidió utilizar la librería gráfica de Python llamada “PyGame”, la cual está especializada para la realización de aplicaciones de juegos; ya que provee de muchas facilidades para el programador al ofrecer gran libertad y cantidad de operaciones soportadas, desde imágenes, sonido y música, ventanas, botones, etc, los cuales le dan un toque más agradable al programa.

Dentro del proyecto se incluyen muchas imágenes y pistas de sonido, esto para realizar una aplicación que en serio pareciera un juego; ya que al fin y al cabo de esto se trataba el trabajo, además, se utilizaron archivos .png para los cuadros de juego, desde el vacío, la serpiente y las manzanas.

Para conectar los nodos retornados por el recorrido en profundidad con la matriz gráfica, solo se necesitó ingresar a los elementos 1 y 2 de cada tupla de la mejor ruta para extraer la fila y columna de la siguiente posición de la serpiente, y a esa posición se le asignó un número que se representa como el vacío, la serpiente o las manzanas y realizó un ciclo sin fin que refrescara la pantalla y mantuviera el juego en funcionamiento hasta el correcto termino del mismo.

Análisis de Resultados

|  |  |  |
| --- | --- | --- |
| Descripción | Estado | Observaciones |
| Algoritmo de Recorrido por Profundidad | 100 % completado | Se utiliza un grafo global con todas las combinaciones posibles de nodos y vecinos. |
| Creación del Grafo | 100 % completado | Se utilizan 2 funciones auxiliares dependiendo de la posición de la tupla |
| Algoritmo de Elección del mejor vecino | 100 % completado | Se utilizan 3 funciones aux, una para cada caso particular (arriba, abajo o en el mismo nivel) |
| Creación de la mejor ruta para las manzanas definidas. | 100 % completado | Se hace un append a una lista de tuplas con los movimientos realizados y por realizar. |
| Creación del apartado gráfico | 100 % completado | Se utilizó la librería gráfica de Python: Pygame |
| Creación de la cola de la serpiente | 0 % completado |  |

Conclusiones

Gracias a la realización de este proyecto, se pudo comprobar las ventajas de la programación funcional, como por ejemplo, el proceso de depuración es menos problemático; ya que se puede tener más control al ser solamente llamadas a funciones, programas más confiables, al trabajar solamente con funciones, las probabilidades de que surjan resultados incongruentes es menor que con otros paradigmas y una mayor facilidad para la ejecución concurrente, es decir, es más fácil llamar a funciones que permitan la ejecución continua del programa.

Luego de haber realizado este proyecto, el nivel de conocimiento adquirido en el tema de grafos, árboles, interfaz gráfica y temas relacionados se vio notoriamente incrementados. Para llevar a cabo algunas de las tareas necesarias para realizar funciones, como por ejemplo, realizar el recorrido en profundidad; ya que por medio de este conocimiento se pudo recorrer el grafo y sus nodos hijos (vecinos) en una función recursiva, se tuvo que reunir conocimiento e investigar sobre el tema, lo que aumentó nuestros conocimientos.

Para terminar, se experimentó con la interfaz gráfica, la cual es entretenida e interesante de utilizar, además, de que estos nuevos conocimientos en este apartado puede que nos sirvan más adelante en nuestra carrera universitaria.

Bibliografía

Búsqueda en profundidad. (2017). Wikipedia, La enciclopedia libre. Recuperado desde https://es.wikipedia.org/w/index.php?title=B%C3%BAsqueda\_en\_profundidad&oldid=99055873.

Stackoverflow, (2017). Depth First Search. Recuperado de: https://stackoverflow.com/questions/43430309/depth-first-search-dfs-code-in-python