TEC | Tecnológico de Costa Rica

Proyecto 1 – Tron

Algoritmos y Estructuras de Datos I Grupo 2

Profesor Leonardo Araya Martínez

Esteban Campos Abarca

2022207705

Introducción

El presente proyecto trata de la realización de un videojuego cuya implementación requiera de las estructuras de datos, lista, pila y cola. Se utilizó Unity como motor gráfico. A continuación, se presenta la documentación del proyecto con el objetivo de catalogar las herramientas y código utilizados.

Contenidos

Breve Descripción del problema

El videojuego es un juego en 2D programado en C# en el que el jugador controla una entidad que deja a su paso un rastro y esta se mueve por una malla rectangular de dimensiones variables y coordenadas discretas. Chocar con el rastro de cualquier jugador destruye la entidad. La idea se basa en la película “Tron” en la que competidores manejan motos que dejan un rastro de luz sólido en un entorno virtual.

Descripción de la solución

Previo a la implementación del proyecto. Se contaba con las clases propias LinkedList, LinkedListQueue, LinkedListStack, y MatrixLinkedList. El nombre LinkedList se le agregó a todas para diferenciarlas de las clases propias de C#. Para enlazar el proyecto programado con Unity se utilizó un GameManager. Este es un GameObject, cada uno de los elementos en una escena en Unity, cuyo script puede ser accedido por cualquier otro script de la escena de forma global.

El proyecto cuenta con 11 requerimientos:

1. ***Las motos de luz se implementan como una lista enlazada simple.***

Se creó una clase Player con un atributo trail de tipo LinkedList de objetos Point2D. Esta es una clase que guarda una coordenada bidimensional. En el programa un trail de largo n se interpreta como una cabeza en la posición 0 y un rastro de luz entre 1 y n-1. La coordenada guardada por cada elemento de la lista indica en qué posición del mapa se halla la cabeza o rastro de luz. A los rastros de luz se les denota en las variables y comentarios relacionados como “LP”.

2. ***Atributos del jugador.***

Los jugadores tienen cinco atributos según las especificaciones:

- Velocidad: int speed. Su valor es dado por el GameManager en la instanciación del objeto Player.
- Tamaño de la estela: int LP_size. Su valor comienza en cuatro, esto permite una cabeza y tres rastros de luz.

- Combustible: float fuel. Comienza en 100.
- Items: LinkedList de enteros items_queue. Las especificaciones indican que debe ser una cola, con la excepción de que los items de celda de combustible se deben de añadir al inicio de la cola y no al final, por lo que se optó por usar una lista como cola para manejar la excepción.
- Poderes: LinkedList de enteros power_ups_stack. Del mismo modo se indica usar una pila, pero en este caso existe la excepción de que la pila debe poder rotar, lo cual es más fácil con una cola o una lista.

Además de los indicados, la clase Player tiene 15 atributos más:

- int player_ID.
- int current_speed. La variable speed guarda la velocidad inicial o base del jugador, y current_speed depende de esta y del estado actual del poder de hiper velocidad.
- El atributo trail previamente mencionado.
- int direction. Representa la dirección siendo 0, 1, 2, 3 derecha, arriba, izquierda y abajo, respectivamente. Comienza en 1, hacia arriba.
- int shield_remaining. Comienza en 0. Representa el tiempo en frames que el jugador cuenta con el poder de escudo.
- int hyperspeed_remaining. Comienza en 0. Representa el tiempo en frames que el jugador cuenta con el poder de hiper velocidad.
- MatrixLinkedList de MapCells map. Es el mapa del juego. El Player lo recibe al construirse para actualizarlo cuando añade y quita elementos del trail.
- int item_timer. Su valor por defecto es 90. Decrece una unidad por frame cuando el jugador posee algún item. Esto funciona como un contador de 1,5 s para usar los items.
- bool character_destroyed. Comienza en false. Le indica al GameManager y a algunas funciones que el jugador ya fue destruido y no debe ser más actualizado.
- int max_items. Por defecto vale 6.
- int max_PU. Por defecto vale 6.
- bool UI_is_updated. Se vuelve falso cada vez que el jugador cambia una variable mostrada por la interfaz de usuario. Estas son el combustible, la lista de items y la de poderes. Permite que la UI no se actualice cada frame.

- bool has_moved. Se vuelve true cada vez que el jugador se mueve; y false, cuando el usuario presiona una tecla WASD. Permite a la función de movimiento, Controls(), tomar en cuenta inputs muy rápidas por parte del usuario de forma más cómoda.
- int last_fuel_increase. Comienza en 0. Cuando una celda de combustible se usa, el jugador recibe una cantidad aleatoria de combustible calculada por el mismo objeto Player. Esta variable guarda el último incremento de combustible para que la UI pueda mostrarlo.
- bool previous_direction. Comienza en true. Indica si el último movimiento del jugador fue vertical u horizontal. Esto permite a ChangeDirection() alternar entre movimiento horizontal y vertical una vez por turno cuando el jugador intenta moverse en diagonal.

3. Distribución de los items y poderes de un jugador al destruirse.

Existe una función del GameManager que crea items y poderes aleatorios y los coloca en una posición aleatoria del mapa llamada SpawnItemsAndPowerUps(). Similarmente una versión modificada de esta función llamada SpawnDroppedItemsAndPowerUps(LinkedList items_or_PU_list) coloca objetos que ahora no son aleatorios sino los de una lista de objetos (lista de enteros). También es la encargada de vaciar la lista de objetos. Cuando un jugador se destruye llama a esta función dos veces para descargar su lista de items y la de poderes.

4. Ejecución de los poderes.

Usuario y bots.

La pila de poderes se muestra horizontalmente en la esquina inferior derecha de la UI, y el elemento agregado más recientemente es el de más a la izquierda. Para usar el poder del tope de la pila se llama la función de la clase Player UsePowerUp(). Esta función guarda el ID del objeto al tope de la pila y borra este elemento (hace pop). Según el ID del objeto modifica las variables shield_remaining o hyperspeed_remaining según min_shield y max_shield o min_hyperspeed y max_hyperspeed, según corresponda.

Para rotar la pila de poderes existen las funciones RotatePULeft() y RotatePURight() de la clase Player. Estas remueven el elemento del tope y lo recolocan al inicio o viceversa, respectivamente.

Usuario.

Los poderes se ejecutan en cualquier momento del juego al presionar la barra espaciadora.

Bots.

Los poderes son ejecutados por los bots cuando una función booleana dependiente de un número aleatorio es verdadera. Esta función se ejecuta antes del turno de cada bot. La probabilidad de que un bot use un poder está dada por:

$$\frac{1}{7 - cantidad\ de\ poderes\ del\ bot}$$

Según max_PU de la clase Player los bots no pueden tener más de seis poderes.

5. Ejecución de los items.

Los items se utilizan automáticamente según las especificaciones. De esto se encarga la función UpdatePowerUps() de la clase Player. Esta función se llama una vez por frame desde el GameManager y decrementa en uno a shield_remaining y hyperspeed_remaining. En cuanto a los items, la función decrementa en uno a item_timer si la items_queue tiene al menos un objeto. Cuando item_timer llega a cero UpdatePowerUps() llama a la función UseItem(). De forma similar a UsePowerUp(), esta función hace un dequeue y según el ID del item realiza una acción. Las celdas de combustible dan entre 10 y 30 de combustible aleatoriamente; el incremento de estela aumenta LP_size entre 1 y 4; y las bombas llaman a la función Delete() del jugador.

6. Jugabilidad.

Movimiento.

Los jugadores están diseñados alrededor de un sistema de turnos. Un jugador realiza una serie de acciones cuando le corresponde un turno, cuyo periodo medido en frames depende de la velocidad del jugador según

$$\frac{120}{current\_speed}$$

Esto equivale a una frecuencia de movimiento de

$$\frac{1}{2} current\_speed$$

En turnos por segundo.

Un turno consiste en llamar a la función Update() del objeto Player esta depende del valor de character_destroyed. Cuando el jugador no ha sido destruido ejecuta lo siguiente:

1. Resta 0,2 de fuel.
2. Si fuel es menor o igual a 0,2, llama a Delete().
3. Según el valor de direction, genera un nuevo Point2D y lo añade a trail. Esto significa que el jugador genera una nueva cabeza. Además, verifica si esta adición supera el límite de tamaño según LP_size, en cuyo caso elimina el último nodo de trail. Seguidamente actualiza las posiciones de nueva cabeza, cabeza anterior y posible cola eliminada en el mapa de juego.
4. Si hyperspeed_remaining es mayor que 0, calcula current_speed según esta función:

$$current\_speed = (speed + flat\_speed\_increase) * 2 - \frac{speed + flat\_speed\_increase}{hyperspeed\_remaining}$$

   Esto significa que el aumento de velocidad disminuye gradualmente con el tiempo y que al menos es mayor que la velocidad base, speed, en una cantidad igual a flat_speed_increase, el cual tienen un valor fijo de 8.

Colisiones.

La detección de colisiones se hace en conjunto por los objetos Player y el mapa. Los objetos Player actualizan las celdas del mapa con información de sus posiciones y las de los rastros de luz. Por otro lado, desde el GameManager, la función UpdateMap() revisa cada celda del mapa y verifica uno de tres tipos de colisión. Las colisiones que UpdateMap() detecta son las siguientes:

1. Colisión entre jugadores. Si la lista de jugadores en cualquier celda es mayor que uno, llama a Delete() para cada uno de los jugadores.
2. Colisión entre un jugador y un rastro de luz. En este tipo de colisión ya se ha descartado que haya más de un jugador, sin embargo, si hay uno al mismo tiempo que un rastro de luz, llama a Delete() en este.
3. Colisión entre un jugador y un item o poder. Si hay un jugador y la lista de items y poderes de la celda de mapa no está vacía, llama la función GiveItemPU() para cada objeto encontrado.

7. ***Mapa.***

El mapa es un objeto de la clase LinkedListMatrix con dimensiones m x n cuyos nodos almacenan un objeto de la clase MapCell. La clase MapCell tiene seis atributos:

- LinkedList de enteros player_IDs.
- LinkedList de enteros item_PU_IDs.
- int LP. La cantidad de rastros de luz en la celda. Puede haber varios rastros de luz en una misma celda cuando un jugador con poder de escudo atraviesa un rastro de luz.
- int LP_direction. Similar a direction de un jugador. Su función es indicar a la función que instancia gráficamente al rastro de luz con qué orientación hacerlo.
- bool LP_particle_is_instantiated. Indica a la función que instancia gráficamente al rastro de luz que debe actuar en esta celda. Este valor se vuelve verdadero cuando un Player actualiza la celda del mapa y falso, cuando la función gráfica se ejecuta.
- int item_PU_particles_instantiated. Indica a la función gráfica cuantos objetos han sido instanciados gráficamente en la celda de mapa. La función contrasta este valor con el largo de la lista de objetos para saber si hay objetos sin instanciar.

8. ***Generación de items y poderes.***

La aparición de items y poderes es controlada por el GameManager mediante la función SpawnItemsAndPowerUps(). Está función genera un objeto de índice aleatorio en una posición aleatoria. El periodo de aparición en frames está dado por:

$$base\_spawn\_time \;/\; m \;/\; n$$

Donde base_spawn_time es una constante de valor 21600, esto son seis minutos. Esta fórmula indica que el periodo es inversa y linealmente proporcional al número de celdas del mapa. Esto implica además que en una sola celda el periodo de generación de objetos es siempre seis minutos en promedio, sin importar el tamaño del mapa.

Los items tienen los siguientes índices:

1, Celda de combustible.
2, Crecimiento de estela.
3, Bomba.

Los poderes, los siguientes:

4, Escudo
5, Hiper velocidad.

9. Bots.

Los bots son objetos Player que se manejan automáticamente según una de tres funciones de comportamiento. La función de comportamiento se asigna según el player_ID del bot. Dos player_IDs comparten una variante de la misma función de comportamiento, por lo que hay cuatro distintos comportamientos. Los bots realizan su primer turno uno por uno unos momentos después del inicio del juego. Esto da una pequeña ventaja al usuario al inicio. Las funciones de comportamiento son:

1. *BotBehaviorFollow().*

Está función hace que el bot se mueva hacia el usuario hasta que la distancia entre estos sea menor que el rastro de luz del usuario, en cuyo caso el bot se mueve perpendicular a la línea entre ambos aproximadamente en un círculo. Además, el bot evita chocar con rastros de luz.

2. *BotBehaviorFollow(), variante.*

Está función se ejecuta sobre el bot de índice 2, pero se calcula con la posición del de índice 1. Esto significa que este bot se mueve aproximadamente al unísono con el de índice 1.

3. *BotBehaviorItems().*

Está función, similar a la de seguimiento, hace que el bot se mueva hacia los objetos. En este caso no hay distancia mínima, sino que colisiona con estos para recogerlos. Ignora bombas y evita rastros de luz.

4. *BotBehaviorRandom().*

Está función hace que el bot vire a su relativa derecha o izquierda aproximadamente un sexto de sus turnos.

10. Programado en C#.

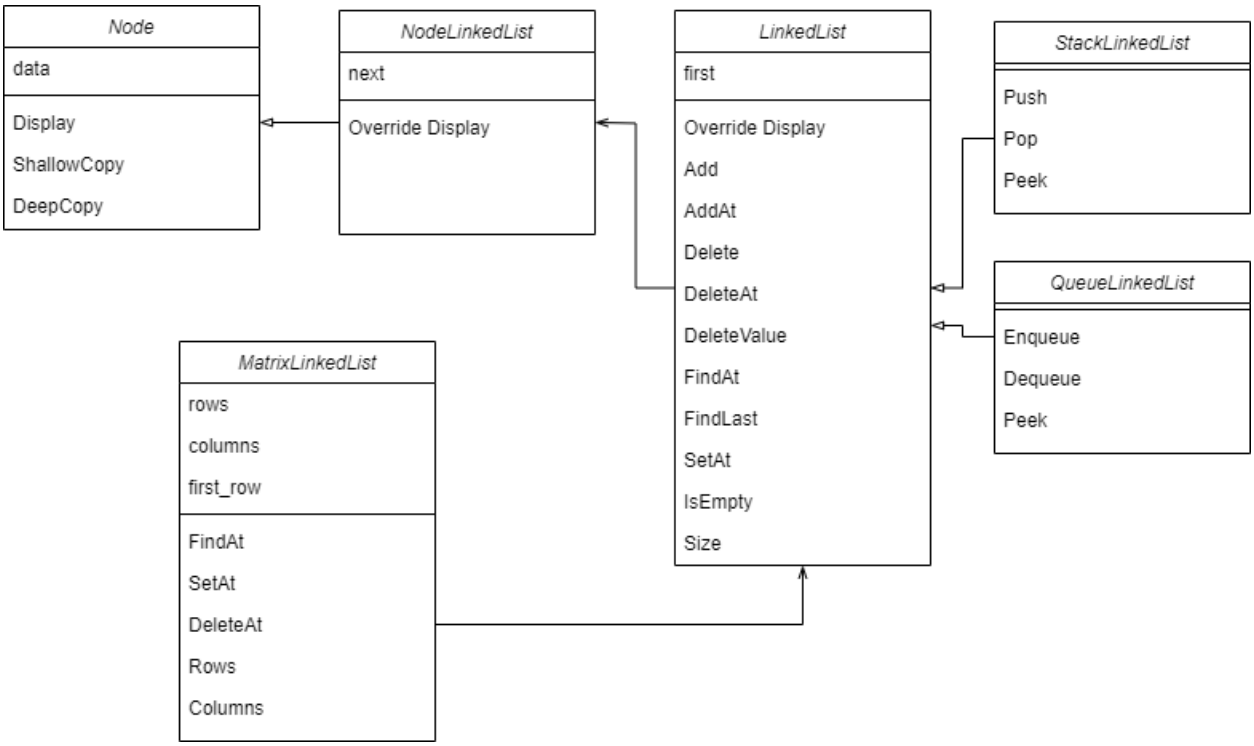
Para incorporar el código en C# a Unity según las especificaciones, se utilizó la capacidad de Unity de añadir código a los GameObjects. El código del GameManager, se colocó sobre un

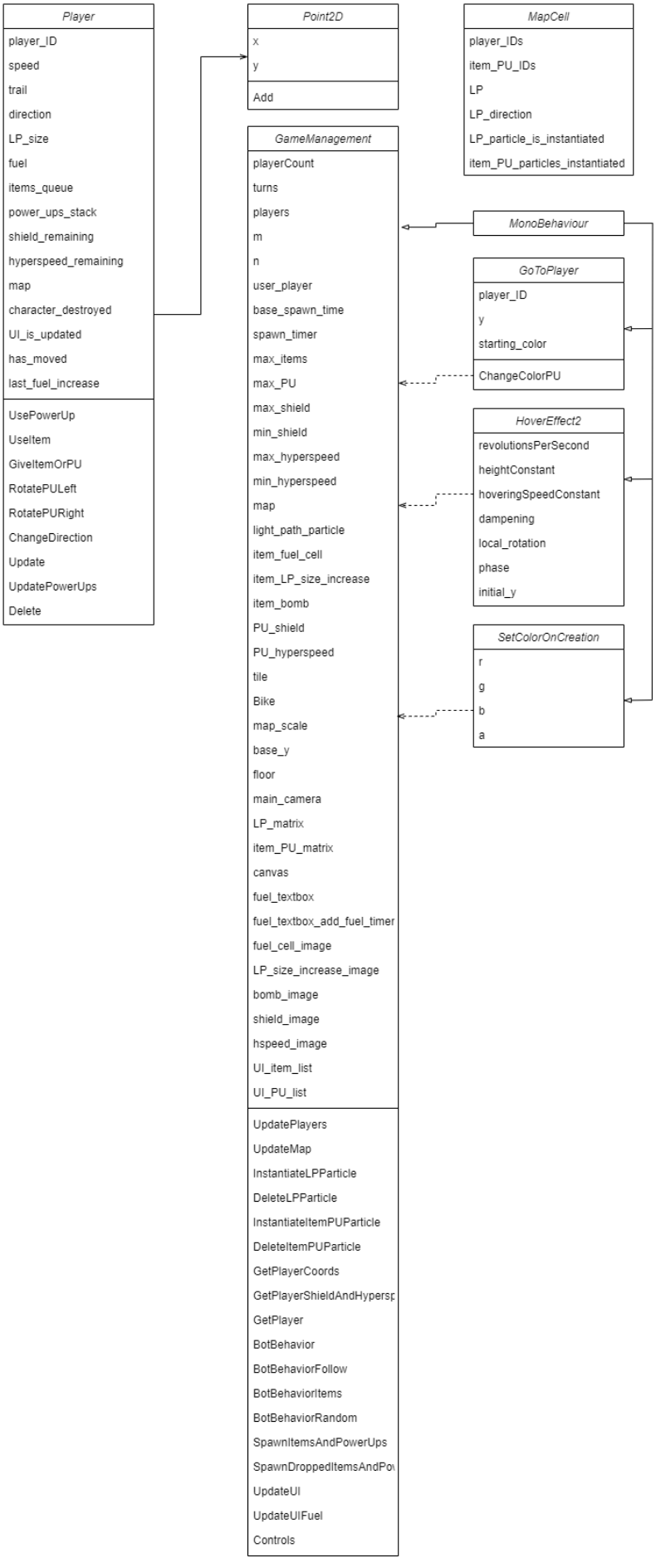
GameObject del mismo nombre. Este instancia la representación gráfica de las celdas del mapa y un fondo. También instancia por cada jugador un GameObject con un script que se encarga de ponerlo en la posición de este. El GameManager, por tanto, se encarga por un lado de instanciar los objetos que interactúan en el programa y, por otro, de instanciar los GameObjects de Unity que permiten ver lo que ocurre.

11. Implementación de las estructuras de datos.

Si bien las colas y pilas se encuentran entre los scripts del proyecto, estos no se usan en favor de listas que se usan de forma similar, pero también manejas comportamientos excepcionales a las colas y pilas.

Diagrama de clases




Player
player_ID
speed
trail
direction
LP_size
fuel
items_queue
power_ups_stack
shield_remaining
hyperspeed_remaining
map
character_destroyed
UI_is_updated
has_moved
last_fuel_increase
UsePowerUp
UseItem
GiveItemOrPU
RotatePULeft
RotatePURight
ChangeDirection
Update
UpdatePowerUps
Delete
Point2D
x
y
Add
GameManagement
playerCount
turns
players
m
n
user_player
base_spawn_time
spawn_timer
max_items
max_PU
max_shield
min_shield
max_hyperspeed
min_hyperspeed
map
light_path_particle
item_fuel_cell
item_LP_size_increase
item_bomb
PU_shield
PU_hyperspeed
tile
Bike
map_scale
base_y
floor
main_camera
LP_matrix
item_PU_matrix
canvas
fuel_textbox
fuel_textbox_add_fuel_timer
fuel_cell_image
LP_size_increase_image
bomb_image
shield_image
hspeed_image
UI_item_list
UI_PU_list
UpdatePlayers
UpdateMap
InstantiateLPParticle
DeleteLPParticle
InstantiateItemPUParticle
DeleteItemPUParticle
GetPlayerCoords
GetPlayerShieldAndHypersp
GetPlayer
BotBehavior
BotBehaviorFollow
BotBehaviorItems
BotBehaviorRandom
SpawnItemsAndPowerUps
SpawnDroppedItemsAndPo
UpdateUI
UpdateUIFuel
Controls
MapCell
player_IDs
item_PU_IDs
LP
LP_direction
LP_particle_is_instantiated
item_PU_particles_instantiated
MonoBehaviour
GoToPlayer
player_ID
y
starting_color
ChangeColorPU
HoverEffect2
revolutionsPerSecond
heightConstant
hoveringSpeedConstant
dampening
local_rotation
phase
initial_y
SetColorOnCreation
r
g
b
a