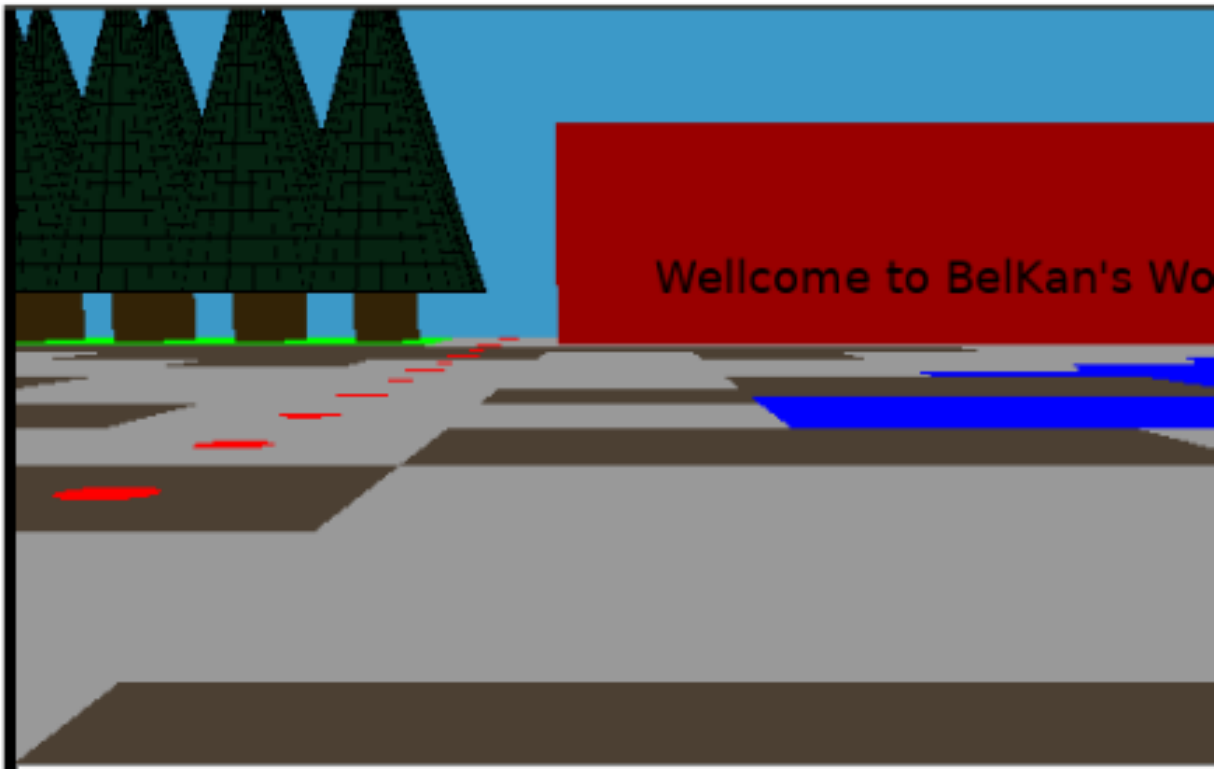
Los mundos de Belkan

Pablo Borrego Megías 2ºA, subgrupo A1



Introducción.

A continuación, procederé a describir lo que he realizado por cada nivel que he implementado, es decir, los niveles 0, 1, 2 y 4.

Nivel 0.

Este nivel no tiene mucho que explicar ya que para completarlo simplemente había que hacer el tutorial que se nos había proporcionado.

Nivel 1. Encontrar el camino con menor número de pasos.

Para implementar este nivel se han realizado un par de cambios con respecto al algoritmo de búsqueda en profundidad (ya viene implementado).
Para hacer el algoritmo de búsqueda en anchura simplemente he cambiado el contenedor de “Abiertos” a una cola (en profundidad es una pila) y de este modo, la búsqueda que se realizará será en anchura.

El otro cambio que se ha realizado ha sido para optimizar este algoritmo, para ello, el la función “pathFinding_Nivel1” se ha añadido “if(Cerrados.find(current.st)==Cerrados.end())” lo cual comprueba si el nodo actual no está explorado, porque se supone que si está explorado no hay que volver a comprobarlo porque no se va a sacar de cerrados. De esta manera los tiempos se verán bastante mejorados.

Además de estos pequeños cambios no he hecho nada más en el nivel 1 y comprobando con los ejercicios que se nos han adjuntando se obtienen los resultados estimados.

Nivel 2. Encontrar el camino con menor coste.

Para este nivel ya si ha habido más cambios notables con respecto a el Nivel 1. Lo primero que he cambiado ha sido el struct nodo, añadiendo un atributo tipo int llamado coste el cual es el que me va a ayudar a comparar los costes de cada nodo y así elegir el de menor (he sobrecargado el operador < de nodo para poder realizar estas comparaciones).

Después se han añadido las funciones “calcularCosteF” y “calcularCosteLR” las cuales se encargarán de asignar el coste de cada nodo dependiendo del tipo de casilla que es.

Otro cambio más a tener en cuenta antes de adentrarme en el “pathFinding_CostoUniforme” es que se han añadido dos variables tipo Bool a el struct “Estado”, de forma que podamos

tener en cuenta cuando se coja el bikini o las zapatillas para realizar el camino con menor costo. De esta forma, en la función CompararEstados se han añadido un par de condiciones más al if, para tener en cuenta como he dicho antes el bikini y las zapatillas.

Ahora sí, comentaré los últimos aspectos que he hecho para este nivel. En la función "pathFinding_costoUniforme" se ha vuelto a cambiar el contenedor, esta vez por una cola con prioridad la cual estará organizada por los nodos con coste más pequeño. Esta función también incluye la optimización descrita para el nivel 1. A la hora de calcular los hijos, el cambio que he hecho ha sido que dependiendo de si es un hijo con el movimiento "Forward" o un hijo con los movimientos "TurnL" y "TurnR", se añade el valor de coste llamando a la función "calcularCosteF" para "Forward" y "CalcularCosteLR" para los demás. He distinguido entre unos y otros ya que en el guión distingue algunos cambios en los costes dependiendo de si es un movimiento hacia delante o de giro. Finalmente, se comprueba si dicho nodo es una casilla de bikini o de zapatillas, y si es así le asigna el valor true al atributo que toque y false al otro.

Y con esto se acaba la explicación de lo implementado en el nivel 2. Se ha comprobado su funcionalidad con los ejercicios de comprobación que nos han proporcionado y los resultados son los esperados.

Nivel 4. Reto.

Para el nivel 4 he realizado una gran cantidad de cambios y he añadido muchas cosas, por ello, voy a intentar no hacer muy larga la explicación de todo lo que he hecho en este nivel.

Lo primero que he podemos ver es que el think está muy cambiado, pero antes de explicar que he añadido iré a el nuevo struct que he diseñado, el struct nodo_Aestrella, el cual tiene implementado el algoritmo A* y una función que calcula la distancia a los objetivos con distancia Manhattan. Este será el algoritmo y el método de recorrido que se usará para este nivel, por supuesto se ha vuelto a sobrecargar el operador <. Este nuevo struct es el que se usará en el pathFinding_nivel4.

Continuando por el pathfinding, es muy parecido al del nivel 2, salvo que se ha cambiado todo "nodo" por nodoAestrella y además, a la hora del cálculo de los hijos, se calcula también la distancia Manhattan de cada uno a el destino, de esta manera se selecionarán los nodos que más se aproximen al destino y los de menor costo. Esta vez no vamos a tener en cuenta si un nodo está en una casilla de bikini o zapatillas, se explicará más adelante en el think. otra cosa a añadir es que las funciones de calcular el coste se han cambiado y adaptado a este nivel, por lo tanto tenemos 4 funciones en total, 2 para nivel 2 y otras 2 para el nivel 4. No las he usado igual en ambos niveles, y para el nivel 4 he tenido que hacer multitud de pruebas para que en todos los mapas me dieran resultados los más buenos posibles, por ello, estas funciones son muy distintas en este nivel. De esas funciones el mayor cambio está en lo que he tenido en cuenta para coger o no los bikinis o las zapatillas, que en resumidas cuentas lo que he hecho es que en los primeros 800 instantes de tiempo esas casillas cuenten 1, y a partir de ese momento se tendrá en cuenta la casilla del objetivo, si está en agua cogerá un bikini y si está en bosque unas zapatillas.

Eso de los 800 instantes de tiempo he tenido que hacerlo porque en el mapa de las islas no me funcionaba bien si no estaba implementado, y al añadirlo, en el resto de mapas no ha variado mucho la eficacia (en algunos ha mejorado incluso).

Ahora pasemos a la función “actualizarMapa” la cual se encarga de ir actualizando las casillas del mapa para saber qué terreno son, y lo más importante, guardar la posición de la casilla de recarga. En resumen, esta función, según la orientación del personaje va actualizando los valores de terreno de las casillas en el orden especificado en el guión. Antes de pasar a explicar el think debo decir que se han añadido muchas variables globales que serán usadas en el think() y que además se ha añadido un nuevo atibuto del struct estado llamado “destinosPendientes” la cual será explicada ya mismo.

pasemos al think, en esta función el primer cambio ha sido distinguir cuando es el nivel 4 de los demás, lo cual se ha hecho mediante los sensores de nivel. Después, he ordenado los destinos por distancia a el personaje, es decir, los más cercanos, de esta manera se gana mucha eficiencia ya que no se pierde tiempo en ir en el orden predeterminado. La ordenación de los objetivos es muy sencilla, se ha hecho en variables auxiliares locales y en resumen, el proceso de ordenamiento es:
1.Se rellena un vector de objetivos auxiliar.
2.Después, según el objetivo, se rellena un vector de pair <indice , distancia> de cada objetivo.
3. Se hace un sort de este vector.
4.Se rellena el vector objetivos.
5. Si el vector “destinosPendientes” está vacío se rellena con “objetivos”.

De esta manera se realiza la ordenación por proximidad de los objetivos. Después entramos en el plan, y aqui hay algunas cosas que especificar primero. Mi forma de pensar el funcionamiento para este reto ha sido la siguiente, el personaje irá buscando objetivos mientras su batería esté bien (por encima del mínimo que yo he establecido), si no está bien, se pondrá modo “Búsqueda y recarga” de batería, entonces el objetivo pasará a ser la casilla de recarga y una vez llegue recargará hasta estar por encima del maximo de batería que he impuesto yo. De esta manera evitaremos la gran mayoria de veces que el personaje se quede sin batería. Una vez especificado esto, ahora pasaré a explicar el funcionamiento de cuando está buscando un objetivo o la batería. Yo lo he pensado así, el personaje replanificará el plan cada 3 pasos que dé, de esta manera evitará entrar en obstáculos o casillas que no debe, además de que al ser 3 pasos siempre verá lo que tiene por delante sin llegar a entrar a una casilla que no debe o colisionar con algo. Todo esto se realiza mediante las comprobaciones (que creo que están bastante claras) y las variables globales pasos y maxPasos. Además se tiene en cuenta las casillas de bikini y de zapatillas y a los aldeanos, lo cual he decidido que espere cuando tenga un aldeano delante, ya que al intentar replanificar en algunos casos no iba del todo bien, y esperando a que el aldeano se quite solo se han conseguido mejores resultados.

Además, siempre comprobará si la batería está bien, si no lo está se pasará al modo antes descrito, en el cual estará la misma comprobación pero al revés, cuando esté bien la batería pasará al modo “búsqueda de objetivos”.

Después, cuando encuentre un objetivo, lo borrará de “destinosPendientes”, comprobará si era el último (si se queda vacío después de borrar) y si es así, lo volverá a rellenar con objetivos, para evitar diferentes errores. Después volverá a llamar a pathfinding para seguir con la búsqueda de objetivos.

Con esto finaliza la explicación del Nivel 4. He de añadir que tal y como está implementado, en la mayoría de mapas me hace un desempeño normal y alto (conforme a la calificación de objetivos por mapa que nos han proporcionado) excepto en pinkworld y medieval, ya que al intentar innumerables opciones, configuraciones de costes de casillas, implementaciones, etc, no he conseguido que llegue a un nivel normal de eficacia, pero por lo menos realiza bien algunos objetivos de dichos mapas.

Realizado por Pablo Borrego Megías DNI: 26504976Y subgrupo: A1