Inteligencia Artificial
Documentación práctica 2

Los extraños mundos de Belkan

Clara Mª Romero Lara

1. Nivel 1: Búsqueda en anchura

La búsqueda en anchura consiste en expandir completamente cada nodo con el que nos encontramos. Esto lo hemos implementado en una `queue<nodo>` de la STL: con el enfoque FIFO de la cola podemos recorrer los nodos en el orden que los introducimos.

La función pathfinding para este nivel consiste en:

- Primero creamos el nodo `current` y lo posicionamos en el origen
- Mientras no hayamos llegado al destino, iremos generando los nodos hijos de la siguiente manera:
    - Declaramos en nodo hijo = current
        - En los casos de girar izda/dcha, cambiamos su orientación
        - En el caso de ir hacia delante, comprobamos que no haya obstáculos delante
    - Una vez declarado, comprobamos si el nodo es el último de la rama y no ha sido generado previamente
        - Si lo es, lo añadimos a cola y a la secuencia del plan
- Finalmente, tomamos el siguiente valor de la cola (el que está más al frente)

```
nodo hijoForward = current;
  if (!HayObstaculoDelante(hijoForward.st)){
     if (generados.find(hijoForward.st) == generados.end()){
        hijoForward.secuencia.push_back(actFORWARD);
         cola.push(hijoForward);
     }
  }
```

Ejemplo de generación de nodos en anchura

2. Nivel 1: Búsqueda en costo uniforme

La búsqueda de costo uniforme pretende gastar la menor batería posible, yendo por el terreno más favorable (coste 1). Esto lo podemos tomar como una variación de la búsqueda por anchura: si todos los nodos son del mismo coste, se realizará una búsqueda por anchura. Para esta implementación hemos empleado un `multiset<nodo,ComparaNodos>`. La función `ComparaNodos` será descrita en breve. No se emplea un set debido a que no permitiría la existencia de nodos con el mismo coste.

Hemos añadido a las estructuras de datos proporcionadas un par de atributos nuevos: en el `struct estado` se añaden booleanos para el bikini y las zapatillas, y en `struct nodo` se añade el coste, un valor entero.

Se han implementado además las siguientes funciones auxiliares:

- `bool ComparaNodos:` recibe dos nodos, y devuelve true si el coste del primero es menor que el del segundo
- `int GetCosteMovimiento:` recibe dos estados y devuelve el costo de pasar de un estado a otro
- `iterator Encontrado:` recibe el multiset de nodos abiertos y un nodo, y devuelve un iterador a:
    - Si lo encuentra en el multiset, el nodo introducido
    - Si no encuentra el nodo en el multiset, el último elemento de éste

La función pathfinding para este nivel consiste en:

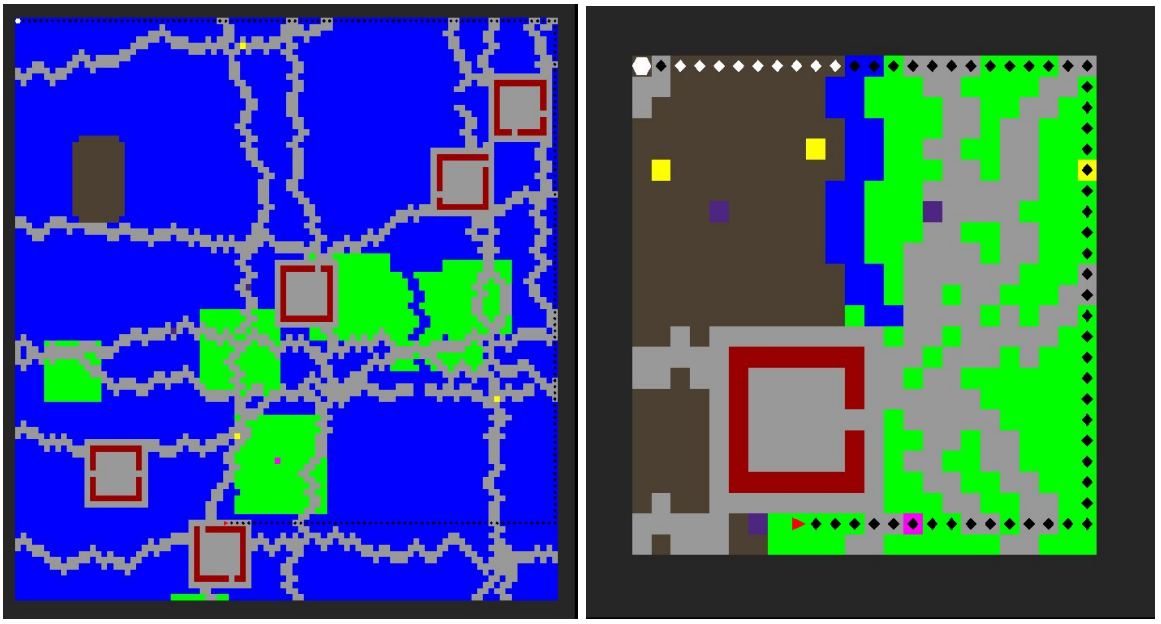
- Creamos el nodo current y lo posicionamos
- Obtenemos el coste asignado al nodo
- Mientras no hayamos llegado al destino, iremos generando los nodos hijos de la siguiente manera:
    - Declaramos el nodo hijo = current
        - En el caso de girar a izda/dcha, cambiamos su orientación
        - En el caso de ir hacia delante, comprobamos que no haya obstáculos delante
    - Buscamos el nodo con la función `Encontrado.` No podemos usar `find` porque lo que nos interesa es obtener el nodo, no el coste.
        - Si el nodo es encontrado y es la mejor opción, lo añadiremos a la secuencia del plan
- Finalmente, tomamos el siguiente valor de entre los `nodos_abiertos` (el que está el primero)

```
nodo hijoForward = current;

if (!HayObstaculoDelante(hijoForward.st))
   if (generados.find(hijoForward.st) == generados.end()){
      hijoForward.coste += getCosteMovimiento(current.st,
                           hijoForward.st);

      multiset<nodo,ComparaNodos>::iterator it;
      it = Encontrado(nodos_abiertos, hijoForward);

      if(it != nodos_abiertos.end()){
         if(hijoForward.coste <= it->coste){
            nodos_abiertos.erase(it);
            hijoForward.secuencia.push_back(actFORWARD);
            nodos_abiertos.insert(hijoForward);
       } else{
        hijoForward.secuencia.push_back(actFORWARD);
        nodos_abiertos.insert(hijoForward);
     }
   }
}
```

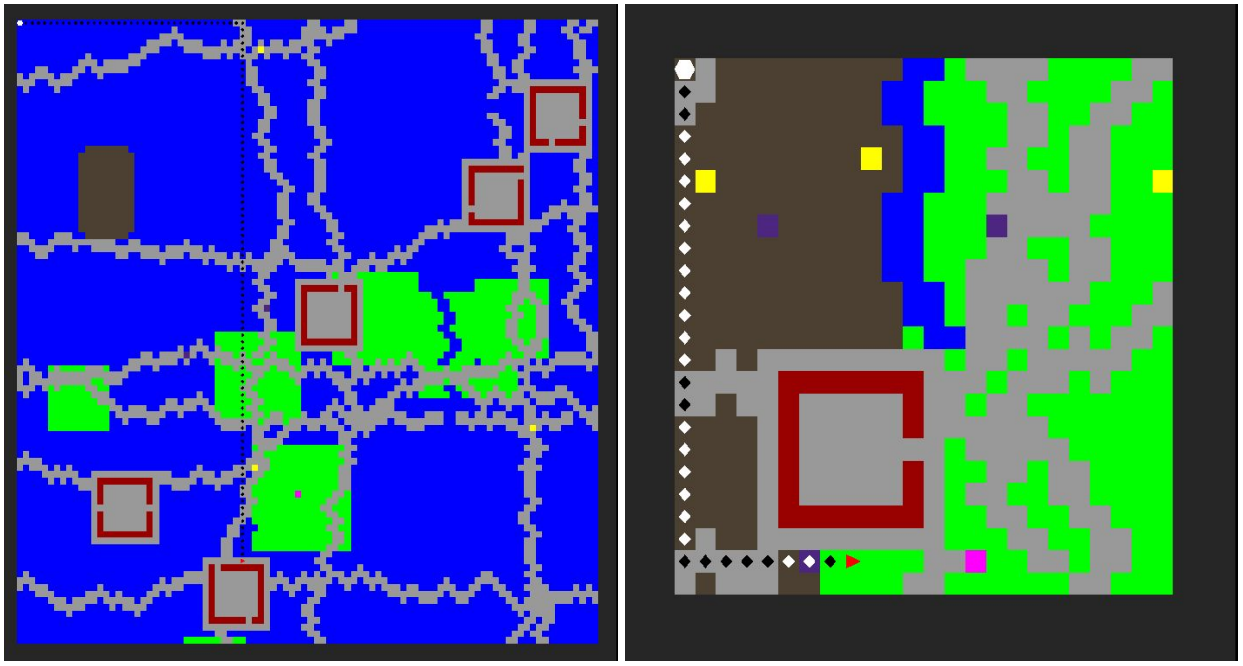
Ejemplo de generación de nodos en costo uniforme

2. Comparativa de recorridos del Nivel 1

Búsqueda en profundidad



Búsqueda en anchura



Búsqueda en costo uniforme