T.S.I.

Práctica 1

Johanna Capote Robayna
Guillermo Galindo Ortuño

21 de abril de 2019

UNIVERSIDAD
DE GRANADA

1. Introducción

En esta práctica se nos pedía programar un controlador del entorno GVG_AI capaz de guiar a un avatar que fuera capaz de pasar los distintos niveles del juego. Este avatar aparte de un comportamiento deliberativo basado en el algoritmo A* debía ser capaz de esquivar a los enemigos mediante un comportamiento reactivo.

Esta memoria se dicidirá en tres partes, una descripción general de la solución, otra del comportammiento reactiva y la última explicara el comportamiento deliverativo.

2. Descripción general de la solución

El esquema general del comportamiento del avatar es el siguiente:

- La estrategia de búsqueda consiste en ordenar las gemas según la longitud del camino que el algoritmo A* proporciona para llegar a cada una de ellas, con esto se busca optimizar el tiempo, ya que no siempre la gema más cercana al avatar tiene el camino más corto.
- El avatar comienza con un comportamiento deliberativo, donde va recolectando las gemas en el orden proporcionado.
- En caso de que el avatar detecte que pueda morir en el siguiente movimiento se activa el comportamiento reactivo, con el que se busca que el agente sepa reaccionar a situaciones inesperadas.
- La heurística empleada es la distancia Manhatam, se ha decidido no cambiar la heuristica puesto que es admisible, lo cual hace que A* encuentre siempre un camino óptimo a la gema que buscamos en caso de que exista.
- El aspecto más relevante del controlador, es su capacidad de reaccionar a cualquier imprevisto, ya que debido a la características de los mapas proporcionados era uno de los principales problemas a resolver. El hecho de que el mapa pueda cambiar con el desprendimiento de rocas y la aprición de enemigos que puedan provocar la muerte del avatar, hacen que este sea el principal objetivo a resolver.

- Por último si pasa más de 4 turnos sin encontrar ningún camino hacia alguna gema, el avatar cambia de objetivo y se dirige a la piedra movible más cercana con la intención de que se mueva y abrá un nuevo camino hacia alguna gema.

3. Comportamiento reactivo

Como se comento anteriormente, el comportamiento reactivo es el que más importancia se le ha dado en esta práctica ya que era el principal motivo de muerte del avatar. Este se basa en una única función *esPeligrosa*, en esta función se gestiona todo el comportamiento reactivo.

En primer lugar, se crea un array de posibles acciones, de las cuales se devolverá alguna. A continuación se recrea el movimiento que el comportamiento deliberativo planeaba hacer, y se comprueba si el agente muere al realizar dicha acción. En caso de no morir, se comprueba que el movimiento no le lleve a una muerte segura, es decir que en el hipotético caso de hacer el movimiento halla al menos un movimiento que pueda hacer en el siguiente turno sin morir. Si una vez recreado el movimiento hay algún movimiento en el que no muera, se devuelve la acción planeada, de no ser así se borra de la lista de posibles acciones y se busca una acción que no lleve al agente a una muerte segura.

En la segunda parte de la función, una vez descartado el movimiento que devolvió el comportamiento deliberativo puesto que este conducía a una muerte segura, pasamos a buscar otra acción. Para ello recorremos el array de posibles acciones repitiendo el proceso anterior, recreamos cada una de las posibles acciones por orden y devolvemos aquella que no conduce a una muerte segura. En caso de no encontrar ninguna acción, se asume que el avatar va a morir y se devuelve la acción *NIL*.

```
Data: StateObservation stateObs, Types.ACTIONS siguienteaccion
Result: Types.ACTIONS siguienteaccion
posibles_acciones = Types.ACTIONS.values();
peligro = false;
aux_stateobs = stateObs.copy();
aux_sateobs.advance(siguienteaccion);
peligro = !aux_stateobs.isAvatarAlive();
if !peligro then
    muere_siempre = true;
    indice = 0;
    repeat
        accion_futura = Types.ACTIONS.values()[indice];
        aux_stateobs = stateObs.copy();
        aux_stateobs.advance(siguienteaccion);
        aux_stateobs.advance(accion_futura);
        indice ++;
    until aux_stateobs.isAvatarAlive() or ind >= Types.ACTIONS.Values().length;
    if aux_stateobs.isAvatarAlive() then
        muere_siempre = false;
    if !muere_siempre then
        return siguiente_accion ;
    else
        posibles_acciones.remove(siguienteaccion) ;
for accion_candidata in posibles_acciones do
    if aux_stateobs.isAvatarAlive() then
        muere_siempre = true;
        indice = 0;
        repeat
            accion_futura = Types.ACTIONS.values()[indice];
            aux_stateobs = stateObs.copy();
            aux_stateobs.advance(siguienteaccion);
            aux_stateobs.advance(accion_futura);
            indice ++;
        until aux_stateobs.isAvatarAlive() or ind >= Types.ACTIONS.Values().length;
        if aux_stateobs.isAvatarAlive() then
            muere_siempre = false;
        if !muere_siempre then
            return accion_candidata ;
return Types.ACTIONS.ACTION_NIL;
```

Algoritmo 1: función: *esPeligrosa*

4. Comportamiento deliberativo

El comportamiento deliberativo del agente es bastante sencillo, este se reduce a llamar al algoritmo A* para buscar un camino hacia cada gema. Estas gemas, como se comentó anteriormente, están ordenadas por la longitud del camino. En caso de no encontrar un camino (A* devuelve null) se elige como siguiente acción *NIL* y se llama a la función *esPeligrosa* para garantizar que no muera en el siguiente movimiento.

Por otro lado, si el avatar pasa más de dos movimientos devolviendo la acción *NIL*, el mapa se actualiza, por si ha habido algún desprendimiento de una piedra abriendo un nuevo camino.

Por último, si el avatar pasa más de 4 movimientos en el que el algoritmo A* devuelve *null* el avatar cambia de objetivo y busca el primer objeto movible del mapa, y se dirije a la casilla que se encuentra justo debajo. En caso de no encontrar un camino hacia esta *roca objetivo* cambia de roca (**nRocas**).

```
1 if ticks_sin_caminos >4 then
2     posiciones_rocas = stateObs.getMovablePositions();
3     for roca in posiciones_rocas do
4         pos_debajo_rocas.add( roca.position.x , roca.position.y +1) ;
5     if pos_debajo_rocas.size() >nRocas then
6         pos = pos_debajo_rocas.get(nRocas);
7         path = pf.astar.findePath(avatar, pos)
8         ifpath == null nRocas ++ ;
```