



UNIVERSIDAD DE GRANADA

TÉCNICAS DE LOS SISTEMAS INTELIGENTES
GRADO EN INGENIERÍA INFORMÁTICA

PRÁCTICA 1

TÉCNICAS DE BÚSQUEDA

Autores

Vladislav Nikolov Vasilev
Carlos Núñez Molina

Rama

Computación y Sistemas Inteligentes



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE
TELECOMUNICACIÓN

CURSO 2018-2019

Índice

1. DESCRIPCIÓN GENERAL DE LA SOLUCIÓN	2
2. COMPORTAMIENTO REACTIVO	4
3. COMPORTAMIENTO DELIBERATIVO	4

1. DESCRIPCIÓN GENERAL DE LA SOLUCIÓN

El aspecto fundamental de la práctica es cómo elegir qué gemas coger y en qué orden. Hay en total 23 gemas por nivel, de las cuales solo se necesitan coger 9. Si se hacen los cálculos, hay $\binom{23}{9} \cdot 9! = 296541907200$ combinaciones posibles. Este número es inabarcable para el A*, sin importar la estrategia usada, por lo que no podemos usarlo para que resuelva el nivel desde cero: hace falta simplificar el problema.

Para reducir el número de posibilidades se ha usado una estrategia de *clustering*, técnica de aprendizaje no supervisado. La heurística detrás de esto es la siguiente: si nos encontramos en un *cluster* (grupo) de gemas, al estar estas gemas todas juntas, generalmente será una buena idea (un buen plan) coger todas las gemas del *cluster* antes de irse a otro. Por tanto, hemos transformado el problema de qué gemas coger y en qué orden al problema de qué *clusters* de gemas coger y en qué orden. Como el número de *clusters* es mucho menor que el de gemas, este problema sí que es abordable. Para generar los *clusters* se ha usado un algoritmo llamado *DBSCAN*. Su funcionamiento (implementado) es el siguiente: se va iterando por todas las gemas del nivel; si esa gema no pertenece a un *cluster* y no hay otra gema de algún *cluster* cerca suya se crea un nuevo *cluster* y se asigna a él; después se ve qué otras gemas sin *cluster* están cerca de ésta y se asignan al mismo *cluster*. Una gema está cerca de otra si su distancia Manhattan es menor o igual a un parámetro ε del método. En la práctica se ha usado $\varepsilon = 3$, que es el que genera mejores *clusters*. Para elegir el *tour* (camino) a través de los *clusters* se ha usado un simple algoritmo de *Branch&Bound*, que devuelve un camino a través de *clusters* de forma que en total se consigan el número de gemas necesarias para abandonar el nivel, siendo el camino elegido en función de la distancia entre los *clusters* y la “dificultad” de cada *cluster* (el número de rocas, muros y enemigos en el *cluster* y cómo de alejadas están sus gemas).

De esta forma, esta es la estrategia fundamental usada en la resolución de la práctica: agrupar las gemas en *clusters* e ir yendo de un *cluster* a otro hasta tener 9 gemas, en cuyo caso se planifica para abandonar el nivel.

La integración del comportamiento reactivo y deliberativo, a grandes rasgos y en pseudocódigo, es la siguiente:

Algorithm 1 Integración del comportamiento reactivo-deliberativo (I)

```

1: procedure ACT()
2:   if primer_turno then                                ▷ Esto se hace en el constructor
3:     crearClusetsYCircuitos()
4:     cluster_actual  $\leftarrow$  0
5:     buscarPlan(cluster_actual)

```

Algorithm 2 Integración del comportamiento reactivo-deliberativo (II)

```

6:   end if
7:       ▷ Plan creado cuando el jugador puede o va a morir en próx. turnos
8:   if plan_no_morir.isEmpty() then
9:       return plan_no_morir.first()
10:  end if
11:  if num_gems ≥ 9 then                                ▷ Dirigirse a la salida si se puede salir
12:      buscarPlanAbandonarNivel()
13:  end if
14:  if busqueda_no_terminada then ▷ Búsqueda puede tardar múlt. turnos
15:      seguirBuscandoPlan()
16:  end if
17:  if busqueda_terminada and camino_no_encontrado then
18:      hay_que_replanificar ← true
19:      if num_gems < 9 then
20:          removeCluster(cluster_actual) ▷ Eliminar cluster y recrear circuito
21:          crearClusterYCircuito()          ▷ porque el cluster es inaccesible
22:      end if
23:  end if
24:  if hay_que_replanificar then
25:      buscarPlan()
26:  end if
27:  if busqueda_terminada then
28:      if plan_vacio then
29:          cluster_actual ← cluster_actual + 1
30:          buscarPlan(cluster_actual)
31:      else
32:          accion ← plan.first()
33:      end if
34:  end if
35:       ▷ Parte reactiva: ver si ejecutar la acción del plan o no
36:  if enemigos_cercanos or muerte_por_roca then
37:      crearPlanNoMorir()
38:      return plan_no_morir.first()
39:  end if
40:  if jugador_choca_con_roca_cayendo then
41:      return quedarse_quieto
42:  end if
43:  return accion
44: end procedure

```

2. COMPORTAMIENTO REACTIVO

3. COMPORTAMIENTO DELIBERATIVO