Grau d'Enginyeria Informàtica

Intel·ligència Artificial

Pràctica 1: Agents Intel·ligents

Jaume Adrover Fernández
jaume.adrover3@estudiant.uib.cat

Joan Balaguer Llagostera
joan.balaguer2@estudiant.uib.cat

17 de novembre de 2022

Índex

1 Introducció

La realització d'aquesta pràctica té l'objectiu d'implementar i analitzar el rendiment dels diferents agents segons el seu sistema de cerca. Dintre del nostre directori arrel de la pràctica (anomenat "Pràctica1") hi trobarem, a part dels fitxers que ens varen proporcionar, una carpeta anomenada "Agents". Dintre d'aquesta, s'hi troben els 4 fitxers corresponents als 4 tipus de cerca qie s'han dut a terme en aquesta pràctica. En el nostre cas, hem desenvolupat els següents algoritmes de cerca:

- **Cerca no informada (fitxer agentNoInfor.py):** aquest algoritme consisteix en realitzar la cerca sense tenir en compte si un estat és millor o no. Aquesta metodologia és, en general, la més lenta i costosa.
- **Cerca informada A* (fitxer agentAE.py):** al contrari de la anterior, la cerca informada consisteix en assignar a cada estat una heurística, que ens ajuda a seleccionar els fills més pròxims a la solució. La nostra heurística es calcula en funció de la distància de l'agent a la solució.
- **Cerca MiniMax (fitxer agentMinMax.py):** quan ens trobem en un joc on hi ha un adversari, el nostre objectiu és guanyar aquest en el millor marge possible. En aquesta pràctica, es cerca maximitzar la distància a la qual queda l'oponent de la meta. És a dir, escollir la nostra millor jugada i guanyar amb més avantatge.
- **Cerca Algoritme Genètic (fitxer agentGenetic.py):** com a darrer algoritme, aquest es centra en simular la selecció natural i en trobar una solució amb individus, que representen una formulacióm del problema actual. S'establirà una fitness function que representarà la correctesa de l'individu.

Per executar-ne un o un altre s'ha de, necessariament, copiar el contingut d'un dels fitxers de la carpeta "Agents", copiar-lo al fitxer anomenat "agent.py", que es troba al inserit dintre del directori arrel de la pràctica, i executar el fitxer main que ens han proporcionat.

1.1 Mesures de rendiment

Per a comparar el rendiment entre tots els algoritmes de cerca utilitzarem aquestes mesures:

- **Distància inicial pizza(quadres):** una mesura que pot afectar en la mesura del rendiment de l'algoritme consisteix en la distància inicial de la pizza a l'agent. Pot haver-hi casos en que el temps de cerca sigui igual ja que la solució és molt pròxima. Per aquest motiu, agafarem diferents casos de prova i obtenir un conjunt de dades més fiable.
- **Cost total(punts):** tots els moviments del joc tenen un cost, específicament 1 en el cas de moure's una casella i 6 si es volen botar dues. Aquest cost canviarà en funció de quin algoritme s'utilitzi.
- **Temps cerca(segons):** aquesta mesura consisteix en analitzar els segons que tarda la cerca, ja que hi haura algoritmes que trobin la solució abans que altres, tot i no ser òptims.
- **Fills Generats(unitats):** aquesta mesura és més indirecta, tot i això, serveix per a veure quants de camins diferents ha explorat el mètode de cerca.

En el cas de l'algorisme MinMax, és completament diferent als altres 3 (ja que és un algorime on hi participen 2 agents en canvi de 1) i per tant, no té sentit comparar-lo amb les mateixes mesures de rendiment. Com que és un algorisme aillat, hem decidit obtenir una mesura de rendiment distinta als altres:

- **% de victòries/total partides jugades:** si el nostre algoritme és correcte, els agents sempre escolliran la seva millor opció. Per tant, com que no hi ha forma de comparar aquest algorisme amb la resta, avaluarem la quantitat de partides que guanya cada agent.

1.2 Casos de prova

Els casos de prova que hem utilitzat per a totes les proves de rendiment han estat els següents:

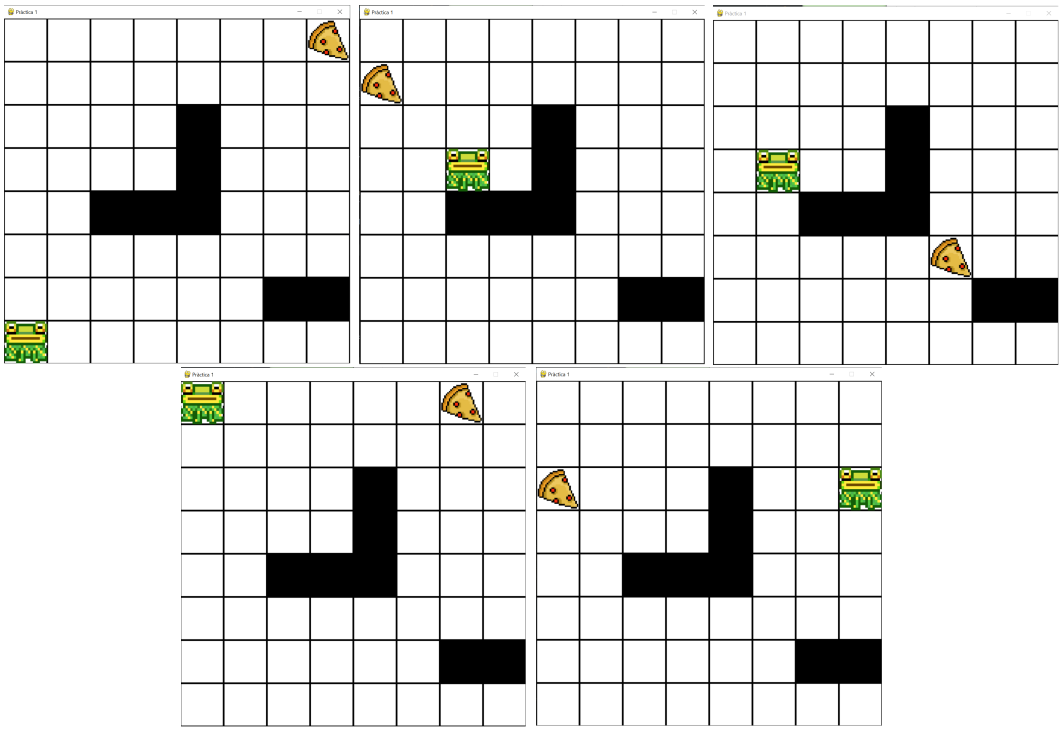

Figura 1: Casos de prova tractats

S'han utilitzat per a realitzar comparacions justes entre els algoritmes, per a observar el seu rendiment en unes situacions comuns d'igualtat.

2 Cerca no informada

Aquest algorisme consisteix en fer un recorregut per l'espai d'estats sense tenir en compte cap criteri adicional que no sigui si s'ha arribat a la meta o no. Per tant, no es té en compte cap tipus d'informació "extra"a l'hora de fer el recorregut. Principalment en trobem de dos tipus:

- **Cerca en Amplada:** en aquest, es desenvolupen els estats d'esquerra a dreta i els que es troben a una profunditat d es desenvolupen abans que els que es troben a una profunditat $d+1$. A més, aquest tipus de cerca no informada, utilitza una coa auxiliar de tipus FIFO per anar desenvolupant els estats. El primer que entri dintra de la coa serà el primer que se n'anirà de la mateixa.
- **Cerca en Profunditat:** en aquest, es desenvolupen els estats d'esquerra a dreta i els que es troben en una profunditat de l'abre p, es desenvoluparà abans que els estats que es troben a una profunditat $p+1$. A més, aquest tipus de cerca no informada, utilitza una coa auxiliar de tipus LIFO per anar desenvolupant els estats. El darrer que hagi entrat es el primer que serà desenvolupat.

Cal comentar que en el cas de la nostra pràctica, hem utilitzat l'algorisme de cerca en amplada. Degut a aquest motiu, l'agent sempre sol fer botar de 2 en 2 per una raó molt curiosa. Si en un arbre es generen els fills, tant els desplaçaments adjacents a la casella inicial com els bots de 2 en 2 estan en el mateix nivell. Estan en el mateix nivell pero els bots estan desplaçant-se més caselles. Per aquesta tendència l'algoritme és més probable a trobar una solució si bota, ja que ha de recórrer el doble de nivells per a trobar el mateix camí que els bots amb moviments de 1 amb 1.

2.1 Rendiment

A continuació es mostren els resultats obtinguts de 5 execucions realitzades amb l'algorisme de cerca no informada:

	Dist. inicial pizza	Cost Total	Temps cerca(s)	Fills Generats
Cas de Prova 1	14	38	0.013	332
Cas de Prova 2	4	12	0.002	125
Cas de Prova 3	6	18	0.004	236
Cas de Prova 4	6	18	0.002	131
Cas de Prova 5	7	19	0.006	229

3 Cerca Informada A*

Aquest algoritme fa un recorregut fins la solució mitjançant un sistema d'heurístiques. És a dir, tenim una estimació de la distància a la meta dins cada estat ($h((n))$ i un cost que hem anat acumulant fins arribar a aquest n ($g(n)$). D'aquesta forma, a cada iteració de l'algorisme anirem elegint els estats successors de l'actual que tinguin el valor $g(n) + h(n)$ més petit. Així, no s'exploraran tants de camins i es centrarà en obtenir el camí més òptim en funció de cost acumulat i del cost estimat fins a la meta.

3.1 Rendiment

A continuació es poden observar els resultats obtinguts a 5 execucions de la pràctica utilitzan l'algorisme de cerca informada A*. Els resultats obtinguts han estat els següents:

	Dist. inicial pizza	Cost Total	Temps cerca(s)	Fills Generats
Cas de Prova 1	14	14	0.009	300
Cas de Prova 2	4	4	0.001	56
Cas de Prova 3	6	6	0.001	59
Cas de Prova 4	6	6	0.001	37
Cas de Prova 5	7	9	0.002	117

4 Cerca MiniMax

Aquest algoritme consisteix en competir contra un adversari i en base a la predicció de les possibles jugades del contrari escollir la teva millor. En el nostre cas, podem preveure fins a 2 jugades del contrari, és a dir, el nostre arbre té una profunditat de 2. Per a poder avaluar els estats, establirem un sistema de puntuació:

$$\psi_i = p_{min} - p_{max} \tag{1}$$

Tenim que ψ_i és la puntuació de l'estat i, que s'obte restant la puntuació del contrari menys la pròpia. D'aquesta manera tenim aquests dos paràmetres que ens deixen amb 3 casos possibles:

$$\psi_i(p_{min}, p_{max}) = \begin{cases} \psi_i(p_{min}, p_{max}) > 0 & \text{MAX guanya} \\ \psi_i(p_{min}, p_{max}) = 0 & \text{Empat} \\ \psi_i(p_{min}, p_{max}) < 0 & \text{MIN guanya} \end{cases} \tag{2}$$

Quan major sigui la diferència, amb més avantatge està guanyant l'agent. Per exemple:

$$\psi_i(p_{min}, p_{max}) = 6 \tag{3}$$

En aquest cas, tenim que MAX està guanyant amb una diferència de 6 caselles. Si aquest nombre fos menor, el contrari estaria més pròxim a nosaltres, i, en conseqüència, en una pitjor situació.

4.1 Rendiment

En el cas de l'algorisme MinMax, hem executat 5 vegades l'algorisme a cada un dels taulells de prova mencionats a l'inici. Els agents s'anomenen M̈iquelï J̈aumeï quan apareix el seu nom a una de les cel·les de la taula que es descriu a continuació, vol dir que han arribat a la pizza abans que el contrincant. Obviament que un agent guanyi o perdi depèn molt de la posició inicial aleatòria de cada agent. Per aquest motiu hem executat 5 vegades cada taulell. Els resultats obtinguts han estats els següents:

	Execució 1	Execució 2	Execució 3	Execució 4
Cas de Prova 1	Miquel	Miquel	Jaume	Jaume
Cas de Prova 2	Jaume	Jaume	Miquel	Jaume
Cas de Prova 3	Jaume	Jaume	Miquel	Miquel
Cas de Prova 4	Mquel	Jaume	Jaume	Miquel
Cas de Prova 5	Miquel	Miquel	Miquel	Jaume

Com es pot observar, encara que les posicions d'inici siguin aleatories, % de victories dels dos agents es del 50% en aquests casos de prova i execucions.

5 Algoritme genètic

Els algorismes genètics es basen en els fonaments de l'evolució biològica. En resum, una certa quantitat d'individus (anomenada població), competeixen pels recursos i s'aparellen. Aquells individus més aptes, són els que s'aparelleran per crear més descendencia que els menys aptes. En el nostre cas, els individus els hem definit com una seqüència de moviments que farà al granota:

$$\text{Individu} = (m_0, m_1, m_3 ... m_{30}) \tag{4}$$

Aquesta seqüencia de moviments serà de, com a màxim, 30 moviments. Això es deu a el nostre agent Rana, en en el mètode **actua**() farà més d'una crida al mètode **cerca**() (encarregat de dur a terme la logística de l'algorisme genètic). Per tant, el camí final fins arribar a la pizza, tal vegada s'acabarà formant per la concatenació dels moviment dels millors individus generats per distintes crides al mètode **cerca**(). Amb aquests fragments de codi quedarà millor explicat:

```
1   def _cerca(self, posPizza, posAg, posParets):
2       individus = PriorityQueue()
3       fills = []
4       solucio=False
5       gen=0
6          .
7          . #Hi ha codi pel mitg
8          .
9       #mentres no soluciuó
10      while(not Rana.solucio):
11                 .
12                 . #Hi ha codi pel mitg
13                 .
14          #mirar si algun individu.calFitness == 0
15          if list(individus.queue)[0][0] == 0:
16              Rana.solucio = True
17          gen+=1
18          #Cada 5 generacions, retorna el millor individu generat
19          if(gen==5):
20              break
21      return list(individus.queue)[0][1]
```

El codi aterior pertany al mètode cerca. Aquest, una vegada ha generat la població inicial d'individus, el que feim es que, com a màxim, pugui reproduïr 5 generacions. Una vegada s'en generin 5 i no

s'hagi trobat una solució (posició final = posició pizza), farem un break, aturarem el bucle (que es farà mentre no s'hagi trobat una solució) i retornarem el millor indicidu generat fins aleshores.

```
    def actua(self, percep: entorn.Percepcio
    ) -> entorn.Accio | tuple[entorn.Accio, object]:
        percepcions = percep.to_dict()
        claus = list(percepcions.keys())
        # Assignam a la llista d'accions les accions del millor individu generat pel mètode cerca.
        # Quan la llista d'accions de self.__accions sigui None, tornarem a entrar al condicional
        if (self.__accions is None):
            individu=self._cerca(percep[claus[0]], percep[claus[1]], percep[ClauPercepcio.PARETS])
            self.__accions=individu.getAccionsConvert()

        if self.__accions:
            if (self.__jumping > 0):  # esta botant
                self.__jumping -= 1
                return AccionsRana.ESPERAR
            else:
                acc = self.__accions.pop(0)
                if(self.__accions is []):
                    self.__accions=None
                if (acc[0] == AccionsRana.BOTAR):
                    self.__jumping = 2
                return acc[0], acc[1]
        else:
            return AccionsRana.ESPERAR
```

Aquest codi pertany al mètode actua. Com es pot observar, aquest cridarà al mètode **cerca**() sempre i quan l'atribut *accions* de l'objecte Rana sigui None (que es com està definit inicialment). Per tant, si la primera crida al mètode **cerca**() retorna un individu el qual fa aporta una seqüencia d'accions que no arriba a la pizza, quan la Rana hagi executat els n moviments i l'atribut *accions* torni a ser None, entrarem al primer condicional una altre vegada i en conceqüencia cridarem de nou al mètode **cerca**() per generar nous individus (tenint en compte que ara els nous individus calcularan les seves funcions de fitness amb la nova posició inicial actualitzada de la Rana després de haver realitzar els moviments del millor individu anterior).

Per altre banda, per determinar l'aptitut d'un individu, hem definit la funció de fitnes ($f(n)$). Aquesta, donat un individu n amb l'estructura mostrada anteriorment, retorna l'aptitud d'aquest. Pel cas d'aquesta pràctica, hem escollit com a funció de fitnes la distancia de Manhattan des de la posició actual fins a la pizza. Per tant, un individu serà millor o més apte que un altre quant menor sigui el valor que retorni $f(n)$.

Una vegada conegudes les bases del nostre algorisme genètic, com es comportarà? Idò començarà generant n individus aleatoriament i cada un serà avaluat per $f(n)$. Cal comentar que he definit una funció anomenada **corregir**() que just després de crear els individus, els escurça fins a trobar el primer moviment il·legal. Aquests, s'introdroduiran a una coa de prioritat, on s'ordenaran de menor a major segons el valor retornat per la funció de fitness. Amb aquesta estrura de dades, podem anar fent crides per parelles d'individus al mètode **reproduce**(). Aquest, donat dos individus pares, els dividirà aleatoriament en dues meitats i retornarà els dos fills generats per la combinació de les dues meitats dels seus progenitors. Cada generació reproduirà n individus per tant, quan els hem reproduit tots, la nostra coa de prioritat serà de tamany $2n$. En aquest punt, hem de fer la selecció dels n millors individus. Això ho farem extraient els n primers individus de la coa de prioritat i eliminant els n individus menys aptes restants a la coa. D'aquesta forma, els individus amb millor funció de fitness, sobreviuran més generacions.

Finalment, els fills també es podran mutar. Una mutació és la variació aleatoria d'un gen (en el nostre cas un gen serà una de les accions de l'individu, x_i) d'un individu resultant del process de reproducció. Aquestes, es produeixen de forma aleatoria quan es combinen les dues meitats dels progenitors. Hem decidit que les mutacions tenen una probabilitat del 10% de produïr-se.

5.1 Rendiment

A continuació es mostren els resultats obtinguts de 5 execucions realitzades amb l'algorisme genètic:

	Dist. inicial pizza	Cost Total	Temps cerca	Fills Generats
Cas de Prova 1	14	38	0.048	1200
Cas de Prova 2	4	12	0.039	900
Cas de Prova 3	6	18	0.047	700
Cas de Prova 4	6	18	0.055	900
Cas de Prova 5	7	19	0.048	900

6 Comparació CNI-A*-GEN

En aquesta darrera secció compararem tots el rendiments dels algoritmes genètics, no informada(amplada) i informada(A estrella).Hem recopilat 5 execucions de cada i realitzat les mitjanes aritmètiques.

	Cost Total	Temps cerca(s)	Fills Generats
CNI	21	0.00474	920
A*	7.8	0.0028	113.8
GEN	21	0.0054	203.2

D'aquests 3 algoritmes, la cerca informada és el millor en termes de tots els indicadors.

$$A*_{cost} = 0.37 \cdot GEN_{cost} = 0.37 \cdot CNI_{cost} \tag{5}$$

$$A*_{temps} = 0.52 \cdot GEN_{temps} = 0.59 \cdot CNI_{temps} \tag{6}$$

$$A*_{fills} = 0.56 \cdot GEN_{fills} = 0.123 \cdot CNI_{fills} \tag{7}$$

Vegent aquestes correspondències podem elaborar una taula amb el % de millora de A* respecte cada algoritme i la seva mesura de rendiment.

	Cost total	Temps cerca	Fills Generats
CNI	169%	69.3%	78%
GEN	169%	93%	708%

D'aquests càlculs matemàtics podem concloure que:

1) A* és l'algoritme més ràpid i eficient de tots.

2) GEN és l'algoritme que més fills genera ja que té una dinàmica més aleatòria.

3) Tant CNI com GEN troben un cami amb cost més elevat ja que no tenen en compte aquest factor.