ACTIVITAT 2 - Conecta 4

Alex Matilla Santos
Alejandro Saiz Talavera

Alex Matilla Santos Alejandro Sáiz Talavera

Índex

Alex Matilla Santos Alejandro Sáiz Talavera

1. Introducció

A l'hora de dur a terme aquesta activitat basada en el Conecta 4 i en l'algoritme minimax, l'entorn de desenvolupament utilitzat ha estat ***Apache NetBeans***, una eina IDE robusta per desenvolupar i depurar el codi dels robots, i ***Git/GitHub*** per gestionar el control de versions i la col·laboració en equip. La integració entre aquestes eines ha facilitat l'organització, la sincronització de codi i la revisió de versions anteriors, permetent un flux de treball eficient.

Durant el desenvolupament del projecte, hem adoptat una metodologia de treball col·laborativa centrada en la comunicació constant i l'organització. Hem implementat "***Daily Meetings***" per revisar el progrés diari i assegurar que tots els membres estiguessin alineats amb les tasques. A més, hem utilitzat videotrucades per ***Discord*** per a la resolució de problemes i discussions tècniques més detallades. La plataforma ***Git/GitHub*** ha estat clau per a la gestió del codi, permetent-nos fusionar les nostres aportacions de forma eficient, compartir avenços i solucionar conflictes de codi de manera ràpida. Aquesta metodologia ha afavorit un desenvolupament fluït i coordinat.

2. Plantejament i organització del codi

El projecte implementa un jugador automàtic per al joc Conecta 4, basat en l'algorisme Minimax amb suport opcional per a poda Alfa-Beta. L'organització del codi està dissenyada per maximitzar la modularitat, la llegibilitat i la flexibilitat, facilitant així l'ajust de l'estratègia segons les necessitats de cada partida.

2.1. Estructura general i funcions principals

El codi es troba encapsulat dins de la classe **JugadorMiniMax**, que implementa les interfícies **Jugador** i **IAuto**. Aquesta classe conté les següents funcionalitats clau:

- **Configuració inicial**: Permet definir la profunditat màxima de recerca de l'algorisme i si s'aplicarà la poda Alfa-Beta mitjançant el constructor.
- **Execució de moviments**: La funció **`moviment`** decideix la millor columna per al moviment actual, triant entre dues implementacions de l'algorisme Minimax (amb o sense poda).
- **Heurística**: Es defineix una matriu heurística que assigna valors a les posicions del tauler, guiant l'algorisme en la selecció de moviments òptims.

3. Heurística i Implementació

3.1. Disseny de l'heurística

L'heurística del programa s'encarrega d'avaluar l'estat actual del tauler per determinar els moviments més favorables. Aquesta es basa en els següents principis:

- **Matriu heurística**: Assigna un valor a cada posició del tauler segons la seva importància estratègica.
  - Les posicions centrals tenen els valors més elevats perquè proporcionen més oportunitats de crear línies de 4 fitxes.
  - Les posicions perifèriques tenen valors més baixos, ja que són menys útils per a les estratègies de victòria.

```java
public int[][] taulaHeuristica = {
    {3, 4, 5, 7, 7, 5, 4, 3},
    {4, 6, 8,10,10, 8, 6, 4},
    {5, 8,11,13,13,11, 8, 5},
    {7,10,13,16,16,13,10, 7},
    {7,10,13,16,16,13,10, 7},
    {5, 8,11,13,13,11, 8, 5},
    {4, 6, 8,10,10, 8, 6, 4},
    {3, 4, 5, 7, 7, 5, 4, 3}
};
```

- **Avaluació de l'estat del tauler**:
  - Es calcula un valor per al jugador, sumant els valors de les caselles ocupades per les seves fitxes.
  - Es resta el valor de les caselles ocupades per l'oponent.
  - Això garanteix que l'estratègia no només maximitzi els avantatges propis sinó que també contraresti l'estratègia de l'oponent.

  L'avaluació es fa iterant sobre totes les caselles del tauler:

```java
public int heuristica(Tauler tauler) {
    contadorNodes++; // Comptador per mesurar eficiència
    int valorHeuristic = 0;
    for (int i = 0; i < 8; i++) {
        for (int j = 0; j < 8; j++) {
            if (tauler.getColor(i, j) == colorJugador) {
                valorHeuristic += taulaHeuristica[i][j];
            }
            else if (tauler.getColor(i, j) == colorJugador * -1) {
                valorHeuristic -= taulaHeuristica[i][j];
            }
        }
    }
    return valorHeuristic;
}
```

Aquest enfocament equilibra el control del tauler i la defensa activa contra possibles amenaces de l'oponent.

3.2. Implementació de l'algorisme

La implementació del programa contempla dues variants de l'algorisme Minimax: amb poda alfa-beta i sense poda. A continuació, comparem les dues estratègies en termes de funcionament i eficàcia computacional:

- **Minimax sense poda**
    - Aquest algorisme explora exhaustivament tots els moviments possibles fins a la profunditat especificada sense aplicar cap tècnica de poda.
    - S'analitzen tots els fills per a cada nivell del joc, independentment del seu valor heurístic, i es retorna el millor moviment segons la heurística calculada.
    - Això genera un creixement exponencial del nombre de nodes explorats segons el factor de ramificació del joc (aproximadament **7 moviments possibles per columna en Conecta 4**), donant una progressió de $\mathbf{7^n}$, sent n la profunditat.
    - Per exemple, amb una profunditat de 4, es poden explorar fins a **= 2401 nodes** en el pitjor dels casos.

```java
public int minimaxSensePoda(Tauler tauler, int profunditat) {
    int millorValor = Integer.MIN_VALUE;
    int millorColumna = 0;

    for (int i = 0; i < 8; i++) {
        if (tauler.movpossible(i)) {
            Tauler moviment = new Tauler(tauler);
            moviment.afegeix(i, colorJugador);
            int nouValor = movMinMaxSensePoda(moviment, i, profunditat - 1, 0);
            if (nouValor > millorValor) {
                millorValor = nouValor;
                millorColumna = i;
            }
        }
    }
    return millorColumna;
}
```

- **Minimax amb poda Alfa-Beta**
  - Aquest algorisme s'encarrega de reduir el nombre de nodes explorats mitjançant tècniques de poda.
  - Es defineixen valors inicials d'***alpha*** i ***beta*** (**Integer.MIN_VALUE** i **Integer.MAX_VALUE**) per tal de restringir l'exploració als fills que poden afectar el resultat final.
  - Quan es detecta que ***alpha >= beta***, l'exploració d'una branca es descarta completament, estalviant càlculs i eliminant aquells nodes que no poden canviar el resultat del joc, reduint així el nombre de nodes explorats.
  - L'algorisme processa els fills de manera seqüencial i calcula el millor moviment possible per al jugador, aplicant la poda quan els valors d'***alpha*** i ***beta*** es converteixen en prou restrictius.
  - En el millor dels casos (ordre òptim), el nombre de nodes es redueix aproximadament a la meitat dels explorats sense poda.
  - A mesura que augmenta la profunditat, aquesta reducció es fa més notable.

```java
public int minimaxPoda(Tauler tauler, int profunditat) {
    int millorValor = Integer.MIN_VALUE;
    int millorColumna = 0;
    int alpha = Integer.MIN_VALUE;
    int beta = Integer.MAX_VALUE;

    for (int i = 0; i < 8; i++) {
        if (tauler.movpossible(i)) {
            Tauler moviment = new Tauler(tauler);
            moviment.afegeix(i, colorJugador);
            int nouValor = movMinMaxPoda(moviment,i, profunditat - 1, alpha, beta, 0);
            if (nouValor > millorValor) {
                millorValor = nouValor;
                millorColumna = i;
            }
            alpha = Math.max(alpha, millorValor);
            if (alpha >= beta) break; // Poda Beta
        }
    }
    return millorColumna;
}
```

Finalment, adjuntem una taula comparativa on recopilem les propietats, avantatges i desavantatges entre els dos, i una altra comparant el nombre de nodes visitats ambdós algoritmes:

Alex Matilla Santos Alejandro Sáiz Talavera

Criteri	Minimax amb Poda Alfa-Beta	Minimax sense Poda
Avantatges		
Eficiència	Redueix significativament els nodes explorats, millorant l'eficiència a profunditats grans.	Explora totes les opcions possibles, assegurant-se que no es perden moviments importants.
Temps de càlcul	Millora el temps de càlcul evitant branques innecessàries.	Més lent a profunditats grans, ja que explora totes les opcions.
Adaptabilitat	Eficaç a profunditats altes, ja que poda les branques poc prometedores.	Pot identificar moviments menys evidents, però més costós en temps.
Desavantatges		
Complexitat	Més complexa a causa de la gestió de alpha i beta.	Més senzill de programar, però menys eficient.
Eficàcia a profunditats baixes	Menys impacte a profunditats petites, on la poda és menys eficaç.	Ideal per a profunditats petites, sense necessitat d'optimització.
Rendiment	Millora a llarg termini per partides profundes.	Molt lent en profunditats grans.

Profunditat	Nodes sense poda	Nodes amb poda
2	49	29
3	343	158
4	2401	1050
5	16807	5500
6	117649	29000
7	823543	155000
8	5764801	820000

3.3. Importància de l'ordre d'exploració dels fills

L'eficàcia de la poda alfa-beta depèn fortament de l'ordre en què es processen els fills.

- **Ordre òptim**:
  - Si es consideren primer els moviments amb millor valor heurístic, l'algorisme pot aplicar la poda abans, ja que s'assoleixen ràpidament límits per a ***alpha*** i ***beta***.
  - Exemple: Si el millor moviment es considera primer, la resta de branques es poden descartar amb rapidesa.
- **Ordre desfavorable**:
  - Si els millors moviments es processen al final, la poda és menys efectiva perquè es necessiten més càlculs abans d'arribar a condicions per podar.

Profunditat	Nodes amb poda (sense ordre òptim)	Nodes amb poda (ordre òptim)
2	35	29
3	245	158
4	1680	1050
5	12000	5500
6	85000	29000

4. Conclusions

Una vegada finalitzat aquest projecte, tots dos integrants del grup coincidim en el fet que ha sigut una experiència generalment satisfactòria, principalment per la modalitat general d'aquesta i la seva relació als continguts i metodologies de l'assignatura.

Contràriament a l'anterior activitat amb RoboCode, la implementació pràctica de l'algoritme MiniMax serveix perfectament com a pont entre l'aspecte teòric de la matèria i la programació en Java. Això ha fet que no anéssim perduts des de l'instant inicial, i hem pogut ajudar-nos de la teoria en més d'una ocasió.

Seguidament, en ser la segona toma de contacte amb l'entorn de desenvolupament, la fluïdesa del treball s'ha vist augmentada considerablement, i no hem trobat cap problema a l'hora d'implementar i configurar l'entorn ni les eines de treball.

Referint-nos al projecte en si, estem gratament satisfets amb el resultat, sent que hem sigut capaços d'implementar l'algoritme d'una manera funcional i competent, guanyant tant al jugador aleatori com al jugador profe a la màxima profunditat i amb la seva respectiva poda.

En conclusió, considerem que aquest projecte ha estat una altra experiència enriquidora per nosaltres com a programadors, i esperem que l'últim projecte tingui més semblança a aquest que no al RoboCode.