UNIVERSIDAD
DE GRANADA

E.T.S. DE INGENIERÍAS INFORMÁTICA y DE
TELECOMUNICACIÓN

PRÁCTICA 3

Métodos de Búsqueda con Adversario (Juegos)

El Parchís

Inteligencia Artificial

Luis Orts Ferrer
77695861K

Análisis del problema

Esta práctica consistirá en diseñar e implementar un agente deliberativo haciendo uso de una de las técnicas de búsqueda con adversario estudiadas en la asignatura(Minimax o Poda Alfa-Beta), está técnica se hará uso en un entorno de juegos, en este caso trabajaremos con un simulador del juego Parchís, el cual simula una réplica una versión determinista en el que cada jugador jugará con dos colores alternos y los dados se van eligiendo entre un conjunto de dados posibles. El juego original permite jugar de 2 a 4 jugadores, donde estos deberán recorrer un tablero de 100 casillas y 4 casas de 4 colores diferentes(amarillo,rojo,verde y azul). El objetivo de este juego es llevar todas las fichas del color que pertenece desde su casa hasta la meta recorriendo todo el circuito, teniendo la opción de “comer” a las fichas rivales para poder avanzar más casillas y ganar ventaja sobre el resto, puesto que al “comerse” otra ficha el jugador avanzará 20 casillas. En este simulador, solamente pueden jugar 2 jugadores, teniendo cada jugador 2 colores, en este caso ganará el que consiga introducir las cuatro fichas de uno de sus colores en la meta. Además cabe destacar que en esté simulador podemos elegir el número del dado que queramos para avanzar con la casilla que le toque, pero solamente podemos usar un número hasta que se gasten todos los números y ya poder usar uno repetido de nuevo. A excepción de otros detalles explicados en el guión de esta práctica, el resto de reglas y movimientos son muy parecidos al juego original.

Para poder adentrarnos más a fondo en la descripción del algoritmo de búsqueda usado y de la heurística implementada para poder ganar explicaremos el tipo de juego que usaremos para está práctica.

Parchís es un juego bipersonal con información perfecta, es decir, un juego en el que participan dos oponentes y al tener información perfecta, se sabrá en todo momento el estado del juego y las acciones realizadas por el oponente(ambos jugadores ven lo que pasa en el tablero). Al ser un juego competitivo, los dos jugadores se enfrentarán entre sí para obtener el mayor beneficio para sus intereses. En este juego solo tenemos dos estados finales: victoria o derrota, el beneficio de un jugador al ganar significa la pérdida total del beneficio del otro(que pierda), por tanto podemos afirmar que este juego es un juego de suma nula.

Teniendo en cuenta estos detalles, se usará un árbol de exploración de juegos, que nos dará una representación explícita de todas las formas de jugar a la partida y de esta manera establecer mediante un algoritmo el ganar o perder un juego a partir de una situación inicial dada.

Descripción de la solución planteada

Uno de los objetivos para esta práctica es la implementación de uno de los algoritmos indicados, MINIMAX o PODA ALFA-BETA, para que el jugador pueda determinar el movimiento más prometedor para ganar el juego, explorando el árbol de juego desde el estado actual hasta una profundidad máxima, 6 para PODA ALF-BETA o 4 para MINIMAX.

El algoritmo MIN-MAX: teniendo un arbol de estados con nodos MAX y nodos MIN, los cuales se van alternando, los nodos MAX tienen como objetivo maximizar su objetivo a la vez que intenta minimizar el objetivo de MIN y viceversa, para ello MAX siempre buscará en sus nodos hijos el valor más alto y MIN el valor más pequeño. En este tipo de problemas MIN y MAX son jugadores, perteneciendo en este caso los nodos MAX al jugador que lleve este algoritmo y los nodos MIN al oponente, para esta práctica serán los ninjas. Para ver esta descripción de una manera más clara, mostraremos el código usado para replicar la función del algoritmo:

```
double AIPlayer::Min_Max(const Parchis &actual, int jugador, int profundidad, int profundidad_max, color &c_piece, int &id_piece, int &dice, double (*heuristic)(const Parchis &, int)) const{
    if(profundidad==profundidad_max || actual.gameOver()){
    return heuristic(actual,jugador);
    }
    color last_c_piece=none;
    //color ult_aux;
    int last_id_piece=-1, id_aux;
    int last_dice=-1, dado_aux;
        double aux = 0;
        double valor;
    Parchis hijo;

    if(actual.getCurrentPlayerId()==jugador){ //nodo max
    valor=menosinf;
    hijo=actual.generateNextMove(last_c_piece,last_id_piece,last_dice);
    while(!(hijo==actual)){

    aux=Min_Max(hijo,jugador,profundidad+1,profundidad_max,last_c_piece,last_id_piece,last_dice,heuristic);
    if(aux > valor){
    valor = aux;
        if(profundidad==0){

        c_piece = last_c_piece;
        id_piece=last_id_piece;
        dice= last_dice;
        }

    }
    hijo=actual.generateNextMove(last_c_piece,last_id_piece,last_dice);
    }
    }
    else{
    valor=masinf;
    hijo=actual.generateNextMove(last_c_piece,last_id_piece,last_dice);

        while(!(hijo==actual)){
    aux=Min_Max(hijo,jugador,profundidad+1,profundidad_max,last_c_piece,last_id_piece,last_dice,heuristic);
    if(aux < valor){

    valor = aux;
        if(profundidad==0){
    c_piece = last_c_piece;
    id_piece=last_id_piece;
    dice= last_dice;
        }
    }

    hijo=actual.generateNextMove(last_c_piece,last_id_piece,last_dice);

    }

    }

return valor;
}
```

Tenemos como parámetros de la función el estado del juego actual, la profundidad inicial y la profundidad máxima, c_piece, id_piece y dice son el color, el id de la pieza y el dado asociado al jugador, con el valor de estos 3 parámetros se moverá la pieza.

Primero se procederá a comprobar si no estamos en un estado terminal(hemos llegado a profundidad máxima o se ha acabado el juego), si es así se devolverá la heurística implementada.
Inicializamos 3 variables last_c_piece, last_id_piece y last_dice, para luego asignar estos valores a las variables para mover la pieza.

Primero recorreremos los nodos hijos de MAX, para ello comprobamos si somos el jugador(el nodo MAX) y posteriormente a la variable valor se le asignará el valor menos infinito para que cuando MAX recorra sus hijos siempre elija uno mayor, ahora con generateNexMove, recorreremos el primer hijo y para poder recorrer todos, usaremos un bucle while para explorar hasta que lleguemos al nodo actual. Dentro del bucle asignamos a aux, la función Min_Max y comprobaremos para todos los hijos si aux es mayor que la variable valor, si es así valor pasará a valer el valor de auxy justo después para que la ficha se mueva indicamos que si estamos en la profundidad 0 asignamos a c_piece, id_piece y dice los valores last_c_piece,last_id_piece y last_dice, realizamos estas acciones en cada hijo.Cuando termine el bucle pasaremos a MIN que realizará la misma función que MAX pero la variable valor valdrá más infinito y de todos los hijos del Nodo MIN irá seleccionando el de menor valor, obteniendo al final el nodo más bajo al recorrer todos los nodos.
Y finalmente devolveremos el valor, si es el turno de MAX el valor será el valor de la variable valor de MAX y si es el turno de MIN será el valor de la variable valor de MIN.

Mediante la Poda Alfa-beta se obtiene un mejor resultado que con el algoritmo MiniMax puesto que tendremos una mejor eficiencia,para ello este algoritmo realizará la misma función que el algoritmo MIN-MAX pero este ahorrará la revisión de ciertas ramas sin afectar al resultado final . En este algoritmo cada nodo tiene asociado dos variables: alfa y beta. Alfa representa el mejor valor encontrado hasta el momento por MAX, y beta representa el mejor valor encontrado hasta el momento por los nodos MIN. Alfa tendrá un valor inicial de menos infinito y beta tendrá un valor inicial de más infinito, a medida que vayamos analizando el árbol, iremos maximizando alfa y minimizando beta. Si estamos en un nodo MAX modificaremos el valor de alfa si es mayor y si es un nodo MIN modificaremos el valor de Beta si es menor. Y en cada momento revisaremos que Alfa mayor o igual que Beta, si es así, significa que el nodo actual no podrá mejorar el valor ya encontrado y por tanto es innecesario seguir evaluando el resto de ramas. Si la poda se realiza en un nodo MAX se denomina Poda Alfa y si se realiza en un nodo MIN Poda Beta.

Para ilustrar en al práctica lo explicado implementé lo siguiente:

```
double AIPlayer::Poda_AlfaBeta(const Parchis &actual, int jugador, int profundidad, int profundidad_max, color &c_piece, int &id_piece, int &dice, double alpha, double beta, double
    (*heuristic)(const Parchis &, int)) const{
    if(profundidad== profundidad_max|| actual.gameOver()){
            return heuristic(actual,jugador);

        }
        color last_c_piece=none;
        int last_id_piece=-1;
        int last_dice=-1;
        Parchis hijopoda;
        double aux2=0; //valor
     double valor;
        if(actual.getCurrentPlayerId()==jugador){ //si es una nodo max
            valor = menosinf;
            hijopoda=actual.generateNextMove(last_c_piece,last_id_piece,last_dice);
            while(!(hijopoda==actual)){
                aux2=Poda_AlfaBeta(hijopoda,jugador,profundidad+1,profundidad_max,last_c_piece,last_id_piece,last_dice,alpha,beta,heuristic); //
                if(aux2 > alpha){
                    alpha = aux2;
                    if(profundidad == 0){
                    c_piece = last_c_piece;
                    id_piece=last_id_piece;
                    dice=last_dice;
                    }
                }
                //hijopoda=actual.generateNextMove(last_c_piece,last_id_piece,last_dice);
                if(alpha>=beta)

                    break;
              hijopoda=actual.generateNextMove(last_c_piece,last_id_piece,last_dice);


            }

            return alpha;

        }else{

             hijopoda=actual.generateNextMove(last_c_piece,last_id_piece,last_dice);
            while(!(hijopoda==actual)){
                 aux2=Poda_AlfaBeta(hijopoda,jugador,profundidad+1,profundidad_max,last_c_piece,last_id_piece,last_dice,alpha,beta,heuristic);
                if(aux2 < beta){
                    beta = aux2;
                    if(profundidad == 0){
                    c_piece = last_c_piece;
                    id_piece=last_id_piece;
                    dice=last_dice;
                    }
                }
                //hijopoda=actual.generateNextMove(last_c_piece,last_id_piece,last_dice);
                if(alpha >= beta)
                    break;

                hijopoda=actual.generateNextMove(last_c_piece,last_id_piece,last_dice);

            }

            return beta;
```

Si nos fijamos realiza la mayoría de acciones que realiza el algoritmo MINMAX implementado pero en este caso añadimos los parámetros alpha y beta que valdrán, inicialmente más infinito y menos infinito. Dentro del while para la comprobación de cada hijo, tenemos otro condicional el cual nos indica que si alpha es mayor que beta saldremos del bucle, si no es así seguiremos recorriendo el bucle. Si acaba el bucle nos devolverá alpha si estamos en el nodo MAX y beta si estamos en el nodo MIN.

En mi caso he usado para la Heurística el algoritmo MIN-MAX puesto que pese a pensar que tengo bien implementado Poda Alfa-Beta, puedo obtener mejores resultados con MIN-MAX. Deduzco que por algún fallo del código que he pasado por alto no realiza del todo bien el algoritmo o mi heurística no está bien estructurada para el algoritmo.

Heurística

La función heurística es la parte principal del algoritmo y es la que determina cómo se obtiene el beneficio y cómo ganar. Esta heurística consiste esencialmente en un valor absoluto de lo cerca que está el jugador de ganar - lo cerca que está el oponente de ganar. Para ello describimos los siguiente casos:

En esta Heurística nos basamos en la primera parte de la heurística demo que nos incluía la práctica, obteniendo más infinito si ganamos o menos infinito si gana el oponente. Para ello valoraremos positivamente o negativamente acciones del jugador y lo mismo pero con el oponente y luego de volveremos la resta de ambas puntuaciones Las acciones serían:

Valoro muy positivamente que se llegue a la meta, ya que le doy un valor de 20 puntos.

Lo mismo si como una ficha,añado 10 puntos, o si añado piezas a la meta,añado 15 puntos.

Valoro de manera positiva que el jugador contrario tenga fichas en su casa, añado 7 puntos.

También valoro de una manera positiva estar en las casillas seguras, añado 1 punto.

Valoro de una manera negativa tener fichas en casa, resto 30 puntos.

Y para los puntos rivales realizó la misma acción ya que es un juego de suma nula.