Estructures de dades i Algorismes utiltizants

Per dur a terme els dos algorismes bàsics relacionats amb les funcionalitats de calcular un moviment per la màquina i validar un problema hem fet servir l'algorisme Minimax. Aquest algorisme, utilitzat freqüentment en la Teoria de Jocs, es usat en jocs amb adversari on els usuaris tenen la informació completa sobre l'estat del taulell en qualsevol moment donat, com ara el Chess o el Go.

Per fer la funcionalitat de crear una màquina amb qui l'usuari pot jugar hem aplicat el minimax per trobar quin és el millor moviment que pot fer la màquina en funció de l'estat del taulell. Generem tots els possibles moviments i apliquem recursivament l'algorisme per cada un d'ells, quan arribem a un estat final (ja pot ser perquè hem matat el rei, ens l'han matat o perquè hem arribat a la profunditat màxima donada) avaluem mitjançant un heurístic la puntuació del taulell resultat d'aplicar tota aquesta seqüència de moviments. D'entre tots ells agafem el més òptim, tenint en compte que l'adversari agafarà el moviment que més li afavoreix a ell. Hem utilitzat un heurístic bàsic, tenint en compte que de cara a la segona entrega una de les optimitzacions serà fer un heurístic més complet donant així més dificultat a la màquina. Aquest heurístic consisteix en puntuar cada peça en funció del seu valor i retornar un nombre que serà el resultat de sumar les teves peces restant les de l'adversari. Clarament s'observa que és un heurístic senzill ja que en cap cas avalues les amenaces a les teves peces, per exemple. Aquest algorisme retorna un Pair que dona la Piece que mourem i la posició a on anirà.

Com a estructures de dades usades en l'algorisme que calcula el moviment hem treballat bàsicament amb una matriu de peces de tamany 8*8 que representa el taulell d'escacs. Només amb el taulell inicial amb les peces, el jugador que vol moure (blanques o negres) i la profunditat màxima que li donem ho calculem. També utilitzem un vector que donat un taulell i un jugador conté tots els possibles moviments de cada peça del jugador.

Recordem que el Minimax genera un arbre de manera exponencial, així que la profunditat ha de ser limitada si volem que l'algorisme acabi en un temps raonable. Per fer-se una idea, suposant que ambdós jugadors tenen 20 moviments possibles (factor de ramificació) i la profunditat amb la que volem treballar és de 10, generariem 20^20 possibles moviments, inassumible amb un ordinador estàndard i un temps raonable. L'usuari haurà d'anar en compte amb la profunditat i els moviments amb els que vol executar el programa.

El segon algorisme, que permet validar un problema (dir si és solucionable en els moviments proposats), és similar a l'anterior ja que també utilitza el Minimax. En aquest cas podriem haver utilitzat un backtracking senzill, que en cost és similar, pero tenint fet el Minimax de la màquina vam trobar coherent fer-ho també amb Minimax. Aquest algorisme però simplement farà backtrack fins que matis el rei de l'advserari o bé s'acabin els moviments posssibles. Hem aprofitat per realitzar una variant extra de la funcionalitat principal, que a més a més de validar el problema et retorna els moviments òptims amb els que pots resoldre'l. Com és de suposar aquesta variant el que fa és recorre totes i cada una de les branques generades i retornar el mínim de moviments, en canvi el validar "normal" a la que trobem una branca de l'arbre amb la que pots solucionar el problema retornem que

es solucionable. En aquest segon algorisme tampoc utilitzem pràcticament cap estructura de dades extra, únicament un vector on emmagatzem tots els possibles moviments i per cada un d'ells cridem recursivament.