PROYECTO FINAL DE IA
Juego de damas

Antinori, Nicolás
Eyherabide, Natalia
Ríos, Alejo

28 de mayo de 2014

Resumen

Las damas es un juego muy popular en todo el mundo. El primero intento de construir un programa que juegue a las damas fue en el año 1951. En los 90's, el programa más fuerte fue Chinook, escrito en 1989 por un equipo en la universidad de Alberta y ganador del Torneo Nacional de EEUU en 1996. En 2007 los desarrolladores anunciaron que el programa se había mejorado hasta el punto de que no podía perder un juego.

En este documento presentamos nuestros enfoques para la implementación del juego de damas realizado como proyecto final de la materia.

Índice

1. Motivación

La motivación del proyecto fue crear un juego de damas con la inteligencia suficiente para mantener una partida contra un contrincante promedio. A la hora de implementar el mismo, se investigaron distintos algoritmos de IA con el fin de encontrar el que mejor se adecuara a nuestras necesidades. Finalmente se eligió el algoritmo MiniMax, detallado a continuación.

Min-Max es un método de decisión para minimizar la pérdida máxima esperada. Alternativamente puede ser pensado como maximizar la ganancia mínima. Formulado para teoría de juegos con adversario, información perfecta y suma cero. Esto quiere decir juegos donde se enfrentan dos contrincantes y cada jugador conoce de antemano los movimientos posibles de su oponente, sus consecuencias y donde la ganancia o pérdida de un participante se equilibra con exactitud con las pérdidas o ganancias del otro participante.

Las damas es considerado un juego complejo con 10^{20} posibles posiciones legales en un tablero de 8x8 y un game-tree de aproximadamente 10^{40} nodos. Nuestra implementación se basa en las reglas de las damas inglesas que están descritas en Wikipedia[1] , y algunas otras reglas o modificaciones de las dadas que describimos a continuación:

- Cuando una ficha llega a la última línea y es coronada, esta puede continuar moviendo sólo en caso de que este llevando a cabo una comida múltiple.
- En las comidas múltiples, sólo las coronadas pueden continuar comiendo en dirección opuesta. Las fichas regulares deben ir siempre en un mismo sentido.

2. Objetivos

El objetivo del proyecto es crear un juego de damas que a través del minimax pueda simular las acciones de un jugador humano, y como tal llevar adelante una partida de damas entretenida y compleja, dando así la sensación de que la computadora tiene una cierta inteligencia y que "piensa" las jugadas.

Comenzamos por implementar un jugador básico sin inteligencia que pueda realizar los movimientos válidos en el tablero. Luego con la implementación del algoritmo y con una heurística de evaluación sencilla, creamos un jugador-máquina que lleva a cabo las decisiones en cada turno. Paso a paso lo fuimos mejorando implementando primero una optimización en el minimax llamada "poda alpha-beta" y luego mejorando las heurísticas para tener mejores aproximaciones de la posición de nuestro jugador. Dependiendo la heurística seleccionada, el jugador máquina varía su forma de juego.

3. Proyecto

3.1. Decisiones de diseño

La idea básica cuando se busca vencer computacionalmente un adversario es intentar predecir todos los posibles movimientos, desde el turno actual hasta el final de la partida, y elegir el que prometa mejores resultados. Esta idea se basa en generar todo el árbol de búsqueda, aunque dicha estrategia la mayoría de las veces es algo imposible e inaceptable, pues una partida no puede durar siglos. Es por esto que hay que tener en cuenta la complejidad temporal de la estructura a utilizar para representar los elementos del juego (tablero, fichas, etc), ya que, el tiempo para explorar los nodos dependerá directamente de la implementación elegida. Por otro lado, simplificar demasiado las estructuras utilizadas generaría un impacto negativo a la hora de recurrir a las mismas para implementar la parte gráfica. Debido a esto, a lo largo de proyecto el juego tuvo dos implementaciones distintas. En una se usó una representación "compleja", con la cuál se obtuvo una performance bastante pobre, y una más simple obtenida a partir de modificar la primera representación encapsulando la información en un arreglo de bytes ahorrando así procesamiento. A continuación explicaremos ambas más detalladamente para destacar sus diferencias.

Representación compleja.

- En el juego de damas sólo se utilizan los casilleros negros, por lo tanto nos podemos abstraer de los casillero blancos en la representación del tablero y de esta forma ahorrar tiempo de procesamiento.
- Los casilleros a los cuales no podemos llegar con ninguna ficha, podemos no tenerlos en cuenta en el estado actual del juego.
- Los casilleros vacíos, si bien son necesarios para saber las posiciones posibles de una ficha, no es necesario tenerlos representados en todo momento, simplemente cuando se genera el árbol de posibles movimientos

Representación simple.

- El tablero es representado como un array de bytes para que las operaciones sobre el mismo sean más eficientes.
- Tanto el color de las fichas como su tipo (coronada o no coronada) son representados como un único número dentro del array y su índice hace referencia a la posición en el tablero.

3.2. Estructura del juego

La estructura del juego se divide en tres partes: el motor, el algoritmo utilizado (min-max) y la GUI. El motor se encarga de la parte lógica del juego. Es quien verifica que los movimientos sean válidos, los efectúa y además elimina del juego las fichas comidas. En síntesis, es el encargado de hacer que se cumplan las reglas del juego. La GUI es la encargada de

realizar la interfaz gráfica usada por el usuario, y el min-max es la implementacion del algoritmo de inteligencia artificial. El haber estructurado el código de esta forma nos da la posibilidad de futuros cambios sobre el mismo sin la necesidad de grandes modificaciones. A continuación se listan las clases utilizadas en cada representación.

Representación simple

- **Tablero** - Clase que representa el tablero. Consiste en un arreglo de números donde cada uno representa una ficha en particular o un espacio vacío, y el índice su posición.
- **ListaSoplo** - Clase dedicada a manejar la regla del soplido.

Representación compleja

- **Fichas** - Clase que representa una ficha. Contiene su posición en el tablero, su color y si está coronada o no.
- **Tablero** - Clase que representa el tablero. Consiste en dos listas de fichas (rojas y negras).
- **ListaSoplo** - Clase dedicada a manejar la regla del soplido.
- **FichaGrafica** - Clase utilizada para le representación gráfica de las fichas.
- **TableroGrafico** - Representación del tablero en la interfaz gráfica.

3.3. Explicación del minimax

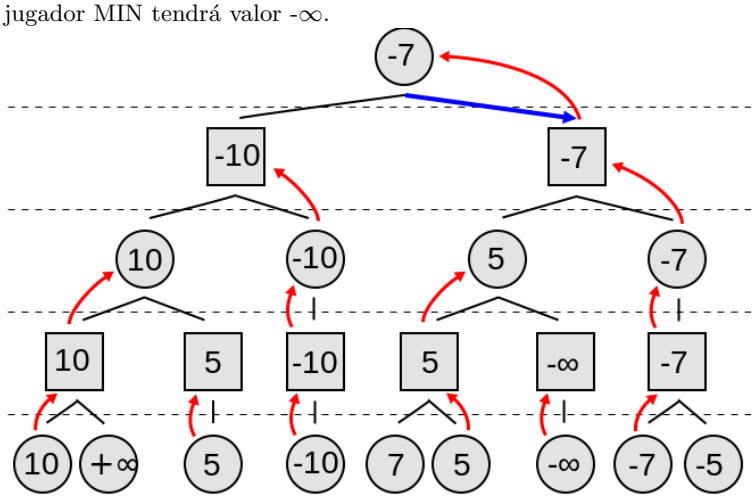
El algoritmo Min-Max es aplicado para juegos como el tic-tac-toe, damas, ajedrez, etc. Todos estos juegos tienen una cosa en común, son juegos lógicos de dos jugadores. Esto significa que pueden ser descritos por una serie de reglas y premisas. Con estas es posibles saber a partir de un punto del juego cuales son los próximos movimientos posibles.

Identificaremos a cada jugador como el jugador MAX y el jugador MIN. MAX será el jugador que inicia el juego y tendrá como objetivo encontrar el conjunto de movimientos que proporcionen su victoria independientemente de lo que haga el juegador MIN. Podemos darnos cuenta que deberá existir una función de evaluación heurística que devuelva valores elevados para indicar buenas situaciones y valores negativos para indicar situaciones favorables para el oponente.

Se genera un árbol de búsqueda en profundidad de todas las posibles posiciones empezando desde el estado actual del juego hasta el final. Cada nivel corresponde con una jugada de uno de los jugadores (MAX o MIN). MAX a través de la función evaluadora le asigna un valor a las hojas del árbol, luego el nivel anterior es evaluado por MIN y así hasta que se evalúa el árbol entero. Los nodos que pertenecen al jugador MAX reciben el máximo valor de sus hijos y los nodos que pertenecen al jugador MIN el mínimo.

Los valores representan qué tan buena es la jugada, así MAX intentará elegir los movimientos con los valores más altos en el final. Pero MIN también intentará escoger lo que sea mejor para el por lo que buscará el mínimo, minimizando de esta forma el resultado del MAX.

Todo esto nos lleva a que una jugada vencedora del jugador MAX (lo que vendría a ser un "jaque mate") tendrá valor $+\infty$, y una jugada del jugador MIN tendrá valor $-\infty$.



3.4. Optimizaciones Posibles

Las optimizaciones del algoritmo tienen un costo adicional. Cuando optimizamos estamos negociando la información completa sobre los estados del juego por una vista imperfecta del juego creada a través de probabilidades y atajos. De esta forma en lugar de conocer el camino completo que lleva a una victoria segura, tendremos un camino que posiblemente nos lleve a una victoria y las decisiones serán tomadas en base a dicho camino. Si la optimización no es bien elegida o está mal aplicada podríamos terminar con un jugador-máquina incompetente.

3.4.1. Limitar la búsqueda

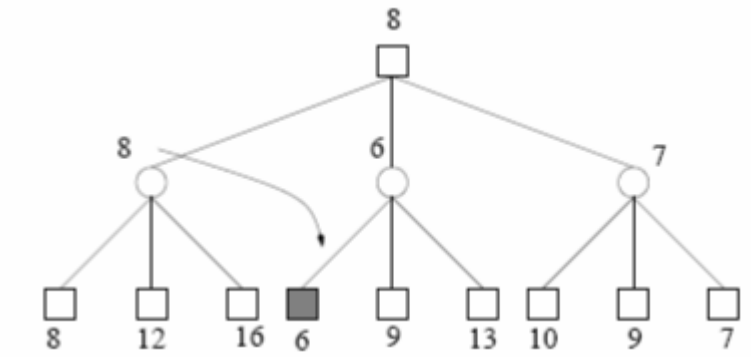
Una de las optimizaciones básicas es limitar la profundidad del árbol de búsqueda generado. Crear el árbol entero conlleva demasiado tiempo. En el caso de las damas podría llevar años e incluso podría no entrar en la memoria de la PC. Al limitar la profundidad a 4, por ejemplo, tendremos un algoritmo que podrá predecir los próximos 4 movimientos.

3.4.2. Poda Alfa-Beta

Un punto clave que hemos de tener en cuenta es que cuanto más profunda sea la exploración, mejor podremos tomar la decisión sobre qué jugada debemos escoger. Para juegos con un factor de ramificación elevado, esta profundidad no podrá ser muy grande ya que el cálculo necesario para cada decisión será muy costoso. Para reducir este tiempo se han estudiado los árboles de búsqueda y se ha llegado a la conclusión de que es muy factible usar heurísticas con poda. En otras palabras, usaremos una técnica de ramificación y poda (branch and bound) con la cual una

solución parcial se abandonará cuando se compruebe que es peor que otra solución ya conocida.

La estrategia denominada alfa-beta aprovecha que el algoritmo del MiniMax hace una exploración en profundidad para guardar información sobre cuál es el valor de las mejores jugadas encontradas para cada jugador. Concretamente, guarda dos valores denominados alfa y beta. El valor alfa representa la cota inferior del valor que puede asignarse en último término a un nodo maximizante, y análogamente el valor beta representará la cota superior del valor que puede asignarse como última opción en un nodo minimizante. De esta manera el algoritmo podrá acotar el valor de una exploración, pues gracias a los valores de alfa y beta conocerá si vale la pena explorar un camino o no.



En el árbol superior podemos ver que el valor que propagaría MiniMax al nodo raíz sería 8. Pero si nos fijamos en el nodo marcado, en el segundo descendiente de la raíz ya sabe que el mejor nodo del primer descendiente tiene valor 8. Puesto que el desdendiente es un nodo Min, y se ha encontrado por debajo un valor de 6, el algoritmo interpretará que por ese camino como mejor resultado habrá un 6 (pues el nodo inferior elegirá o el 6 o algo menor). Como ya conocemos el posible 8, todos los nodos que cuelgan del Min que elegiría el 6 ya no hace falta explorarlos, pues seguro que ahí no se encuentra nuestra mejor jugada. En este momento se realizaría la poda de esa rama e iría a explorar el tercer posible camino. Podemos observar también que esta circunstancia se repetirá en el tercer descendiente al explorar su último nodo, pero al no quedar más descendientes la poda no nos ahorraría nada. Esto significa que la efectividad de esta heurística dependerá del orden de los nodos, pues cuanto antes encuentres un valor penalizado por el umbral, antes podrás podar la rama.

3.5. Heurísticas

Como ya mencionamos, el algoritmo utiliza una función de evaluación heurística para saber qué tan buena es la jugada y dependiendo de dicha función la máquina varía su forma de juego. Se han creado varias de estas funciones en busca de un comportamiento aceptable teniendo en cuenta las distintas estrategias del juego. A continuación describimos las distintas opciones elegidas:

3.5.1. Heurística 1 - Cantidad de Fichas

Se considera la cantidad de fichas de cada jugador y se le asigna un valor a cada una, de esta forma se ve reflejado el hecho de que un jugador está en desventaja si tiene menos fichas que el contrincante. Así, la máquina buscará la forma de comer tantas fichas como sea posible intentando minimizar la cantidad de fichas perdidas. A la hora de asignarle valor a las fichas tenemos en cuenta que las coronadas tienen mayor libertad de movimientos y por lo tanto tendrá un valor superior.

Valores:

- Fichas Comunes: 3
- Fichas Coronadas: 10

3.5.2. Heurística 2 - Cantidad de Fichas y Distancia a la Corona

Esta heurística sigue la idea anterior sólo que agrega el hecho de que es más conveniente ubicar las fichas de la mitad del tablero para adelante puesto que están más próximas a ser coronadas. Es por esto que también se le asigna un valor a los casilleros, de esta forma el valor de una ficha quedará determinado por una suma entre su valor y el valor del casillero.

Valores:

- Fichas Comunes: 3
- Fichas Coronadas: 10
- Casilleros en posiciones no avanzadas: 7
- Casilleros en posiciones avanzadas: 10

3.5.3. Heurística 3 - Cantidad de Fichas y Ubicación

Se considera tanto la cantidad de fichas como las posiciones de cada una. Es más conveniente ubicar una ficha en una fila adyacente al borde del tablero ya que de esta manera es imposible que la misma sea comida. También es una buena estrategia no mover las fichas en la primera fila pues en esas posiciones es donde corona sus fichas el oponente.

4. Discusión y Conclusiones

4.1. Conclusiones de Diseño

Como ya se dijo anteriormente, a lo largo de proyecto se probaron dos implementaciones distintas. En esta sección se tratarán las diferencias entre ambas.

Haciendo un intento por buscar una buena estructura a utilizar para el juego nos pusimos a pensar en ciertas características del mismo y ver si podemos llegar a un diseño óptimo. Lo que se buscaba era abstraerse de los datos innecesarios para lograr una representación lo mas simple posible. La filosofía utilizada era desechar aquellas situaciones que no son pertinentes para el estado actual del juego, por ejemplo los espacios en el tablero a los cuales no podíamos llegar con ningún movimiento. La estructura resultante que llamamos "representación compleja"se muestra en la sección 3.2.

Los resultados de esta estrategía no fueron satisfactorios, si bien logramos lo propuesto (abstracción de ciertos datos) no fue suficiente ya que el tiempo entre cada jugada era muy extenso por lo que decidimos enfocarnos en una mejora desde lo computacional. Lo que se hizo fue representar todos los datos de la implementación anterior como un arreglo de bytes, lo cual derivó en una mejora en cuestión de tiempos entre jugada y jugada. Esta representación la llamamos "representación simplez es la que finalmente fue elegida para implementar el juego. La misma se muestra en la sección 3.2

4.2. Estructura del juego

La estructura del proyecto esta dividida en tres grandes partes: motor, algoritmo y GUI. Esta modularización nos fue muy útil durante el transcurso del proyecto ya que nos permitio modificar las estructuras del motor y probar diferentes variantes del algoritmo de IA sin mayores modificaciones. Como conclusión podemos decir que es altamente recomendable estructurar de forma eficiente el código en proyectos de este tipo.

4.3. Heurísticas

Se probaron tres heurísticas diferentes, todas ellas explicadas en la sección 3.5. La forma de juego de la máquina cambia notablemente dependiendo cuál se seleccione, es por eso que finalmente optamos por desarrollar el código de manera que sea facil de cambiar entre una y otra. En particular creemos que la primer heurística probada (heurística 1) es la que mejores resultados obtiene.

4.4. Poda Alfa-Beta

Un planteo que surgió durante el desarrollo del trabajo fue decidir si la implementación de la poda Alfa-Beta generaba cambios significativos en la performance. A continuación se detallan los resultados obtenidos al evaluar el algoritmo con y sin la optimización. La prueba fue realizada con los distintos niveles de dificultad disponibles en el juego, ejecutando 10 movimientos desde el inicio del mismo. Los parámetros tenidos en cuenta fueron la cantidad de nodos recorridos, y el tiempo que le llevo hacer cada jugada.

Evaluación en nivel inicial

N Jugada	Nodos Recorridos		Tiempo	
	Con Poda	Sin Poda	Con Poda	Sin Poda
1	7	7	0.0078	0.0089
2	7	7	0.0102	0.0118
3	6	6	0.0117	0.0121
4	8	8	0.0104	0.0133
5	16	16	0.0139	0.0289
6	16	16	0.0130	0.0191
7	11	11	0.0034	0.0076
8	10	10	0.0071	0.0100
9	11	11	0.0117	0.0152
10	10	10	0.0046	0.0127

Evaluación en nivel intermedio

N Jugada	Nodos Recorridos		Tiempo	
	Con Poda	Sin Poda	Con Poda	Sin Poda
1	165	435	0.1775	0.2269
2	180	463	0.2012	0.2095
3	479	1295	0.5770	0.5890
4	702	1442	0.5850	0.6842
5	1035	2767	0.9719	1.0676
6	790	2542	0.9611	0.9627
7	517	1152	0.4858	0.4590
8	641	1650	0.5578	0.6003
9	1132	1878	0.6528	1.0739
10	666	1434	0.6107	0.5840

Evaluación en nivel experto

N Jugada	Nodos Recorridos		Tiempo	
	Con Poda	Sin Poda	Con Poda	Sin Poda
1	1663	26889	2.0152	11.8925
2	2333	30394	2.7076	13.5331
3	2755	27773	2.3983	14.7608
4	4436	55048	4.4762	23.0274
5	7543	108221	8.2743	46.8519
6	94256	10524	10.1986	39.2855
7	5708	177293	4.7880	71.1817
8	6250	93212	6.5481	40.2141
9	5906	95673	5.9879	38.9721
10	41938	58685	22.3642	25.5294

A partir de los datos presentados podemos concluir que la optimización si bien en el nivel inicial no genera cambio alguno, la mejora se ve reflejada en los sucesivos niveles que recorre el algoritmo donde la cantidad de nodos a visitar aumenta exponencialmente.

4.5. Objetivos del proyecto

Se pretendía implementar un juego de damas que simule un jugador humano y que sea "inteligente", es decir que no realize los movimientos al azar sino que dichos movimientos tengan sentido.

El juego fue probado contra varios jugadores con distintos niveles de habilidad, y el resultado obtenido es que puede ganar contra jugadores de nivel intermedio e incluso mantener una partida de damas entretenida contra un jugador experimentado. La dificultad del jugador máquina se puede seleccionar según se desee teniendo tres opciones, fácil, intermedio y difícil y las heurísticas se pueden cambiar facilmente.

Para nuestro criterio el objetivo general del proyecto es satisfactorio y el desarrollo del mismo productivo pues tuvimos que sortear varias dificultades, en particular la regla del soplido, y se realizo en un lenguaje de programación que en ese entonces era poco conocido por los integrantes del grupo lo que derivo en un aprendizaje del mismo.

Referencias

[1] The Proceedings of the Association for Computing Machinery Meeting, Toronto, 1952.

[2] Richards, D.J. and Hart, T.P. ,1961 to 1963. The Alpha-Beta Heuristic (AIM-030)

[3] MIT Artificial Intelligence Memo

[4] Computer Chess Methods from Encyclopedia of Artificial Intelligence. S. Shapiro

[5] Inteligencia artificial, Un enfoque moderno - Stuart Russel, Peter Norving