Trabajo final de Complejidad Temporal, Estructuras de Datos y Algoritmos

Emil Hromek

Complejidad Temporal, Estructuras de Datos y Algoritmos,
curso 2020 del primer cuatrimestre, Instituto de Ingeniería y Agronomía,
Universidad Nacional Arturo Jauretche

Resumen

En el presente trabajo se diseñó e implementó un sistema de inteligencia artificial, basándose en el algoritmo MiniMax, para un juego que se ejecuta por consola. Las clases ya estaban provistas para facilitar el armado. El objetivo del trabajo era implementar la inteligencia artificial.

1. Introducción

En el ámbito de las ciencias computacionales, hay cada vez más demanda de sistemas de inteligencia artificial. Si bien se pueden crear tales sistemas para diversidad de aplicaciones, puntualmente se puede hacerlo para crear videojuegos. Una forma de crear una IA para un juego, de suma cero, en el que participen dos jugadores y en donde la información es completa (es decir, por ejemplo, que no haya cartas ocultas, como en el póker) es utilizando el algoritmo MiniMax. El mismo consiste en que la computadora elija el mejor movimiento (el más favorable para la misma), suponiendo que el humano puede elegir uno que la perjudique. Para implementar el algoritmo, se utilizan varios elementos:

1. Estado inicial del juego
2. Operadores: son los movimientos válidos
3. Condicion terminal: determina cuando el juego se acabó
4. Función de utilidad/heurística: da un valor a una configuración del juego, según qué tan favorable sea a la computadora

Para implementar el algoritmo se crea un arbol con todos los posibles estados de juego (nodos), los cuales incluyen la jugada y el valor de la función heurística. Para los estados finales (nodos hojas) se asigna el valor de la función y luego se va para arriba, asignando a cada nodo el valor máximo que hay entre los hijos, si es el turno del humano, o el valor mínimo si es el turno de la computadora.

La IA tratará de avanzar en el juego según la información heurística que haya en el arbol, tratando en cada paso de elegir el camino con el valor máximo. [1]

En el caso de este trabajo se implementó la IA para un juego en donde se reparten cartas numeradas del 1 al 12, dándole seis al humano y seis a la

computadora, de manera azarosa. Se fija un limite que oscila entre 25 y 35 y el juego consiste en descartas cartas por turno. El primer participante que supera dicho límite pierde.

2. Detalles de implementación

Para el armado del juego ya estaban provistas varias clases, las cuales se detallan a continuación:

- Juego: contiene la función Main, que crea una instancia de Game (detallada abajo).
- Game: contiene los campos y métodos para inicializar una partida. Puntualmente esta clase es la que elige el puntaje límite y que reparte seis cartas a cada jugador.
- Jugador: es una clase abstracta que contiene tres métodos, inicializar, descartarUnaCarta y cartaDelOponente. El primer método se usa para que cada jugador reciba las cartas que le corresponden al inicio, el segundo se usa durante los turnos y el tercero se usa para que recibir información sobre la carta descartada por el oponente.
- ComputerPlayer y HumanPlayer: son subclases de Jugador, que implementan los métodos anteriores. HumanPlayer ya tenía los métodos inicializar y descartarUnaCarta implementados.

Para el armado de la AI se agregaron las siguientes clases:

- ArbolGeneral, NodoGeneral y Cola: se utilizan para el armado del árbol MiniMax.
- Dupla: es una clase que contiene dos campos int, el primero se utiliza para almacenar una carta y el segundo para almacenar el valor de la función heurística.
- Estatus: esta clase se utiliza con un propósito estructural; se aprovecha en el armado del árbol MiniMax, para pasarle la información necesaria a los hijos de cada nodo. Contiene cuatro campos: uno para guardar el limite actual, otro para guardar las cartas del humano, otro para las cartas de la computadora y un booleano que especifica si es el turno del humano o no. A medida que se avanza en la profundidad del árbol, el estado se va actualizando.
- ComputerPlayer y HumanPlayer: son subclases de Jugador, que implementan los métodos anteriores. HumanPlayer ya tenía los métodos inicializar y descartarUnaCarta implementados.

Además:

- En la clase Game se agrego un menú de consultas durante el juego, el cual contiene las siguientes opciones:
    - Comenzar nuevo juego.

- Desde el punto actual, imprimir todos los resultados posibles (según las cartas que tenga el humano).
- Desde el punto actual, y dadas determinadas cartas, imprimir todos los resultados posibles para cada carta.
- Elegir una profundidad e imprimir las jugadas a dicha profundidad (retornar la duplas contenidas en los nodos a dicha profundidad).
- Jugar (tirar una carta).

- A la clase Jugador se le agregaron dos campos del tipo ArbolGeneral: minimax y jugadaActual. El primer árbol contiene todos los estados posibles del juego. El segundo árbol inicialmente comienza como una copia de minimax y se va a actualizando a medida que se avanza en el juego (el nodo raíz es reemplazado constantemente por uno de sus hijos, el cual se corresponde con la carta jugada anteriormente). Es aprovechado por la IA, para saber que carta tirar en el siguiente turno. Vale aclarar que estos dos campos son utilizados solamente por ComputerPlayer.

- En ComputerPlayer se implementaron los métodos inicializar, descartarUnaCarta y cartaDelOponente. El primero recibe los datos de cartas de cada jugador y límite, crea una instancia de la clase Estado (la cual recibe esos datos) y finalmente crea el árbol MiniMax, aprovechando esa instancia inicial de Estado. Tambien crea el árbol jugadaActual, como copia del anterior. El segundo utiliza el árbol jugadaActual para decidir qué carta jugar. Simplemente lo que hace que elegir, de entre la lista de hijos, el primero que encuentre (mediante busqueda secuencial) con función heurística +1 y jugar la carta correspondiente a ese nodo. En caso de que ningún hijo tenga +1, se elige el primero de los nodos hijos (todos tienen -1 en ese caso). Además actualiza jugadaActual. El tercero recibe la carta jugada por el humano y actualiza jugadaActual.

- Además, se agregó un metódo armarArbolMinimax (y uno auxiliar) para crear el árbol a partir de los datos de estado inicial.

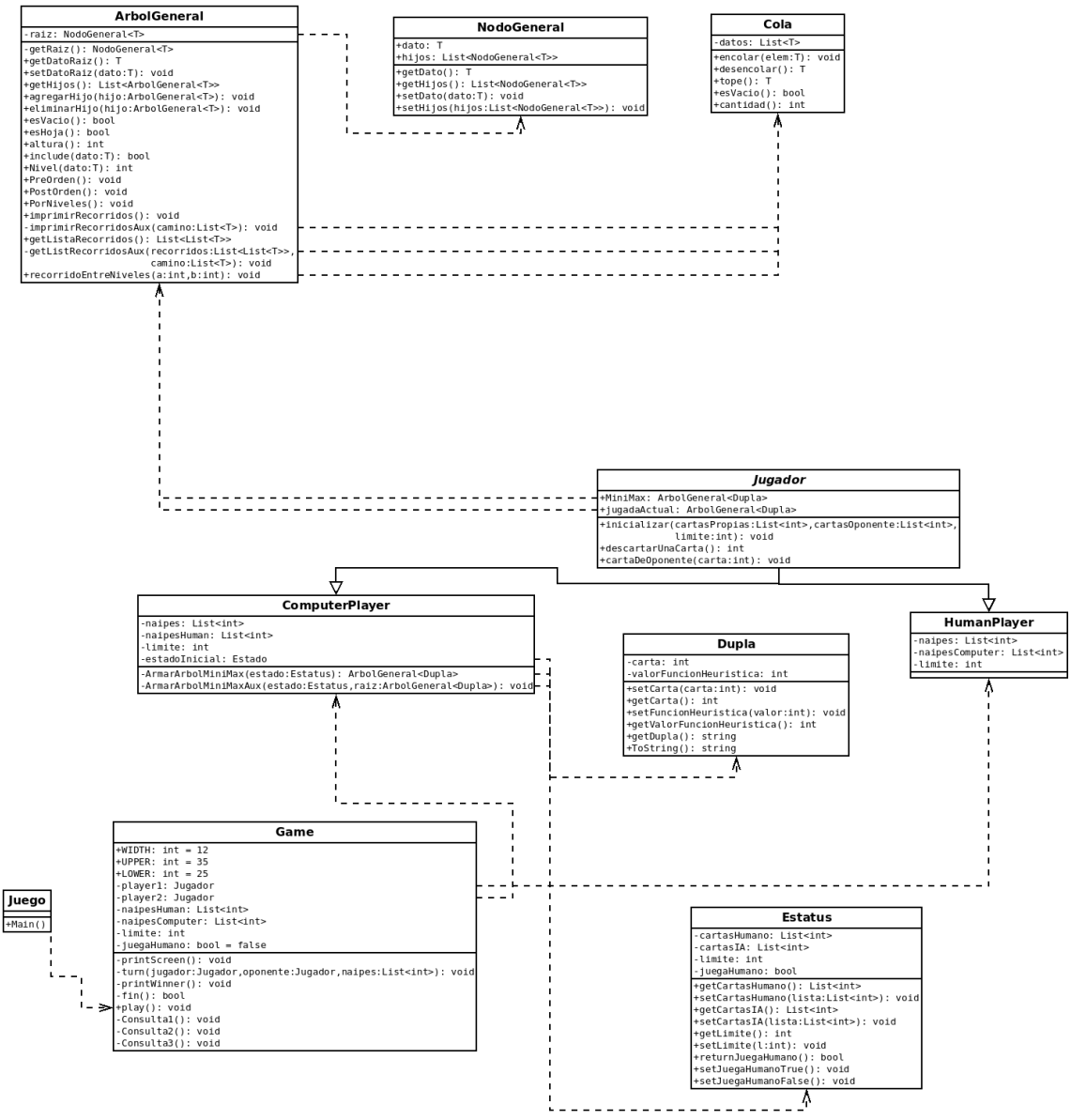

Figura 1: Diagrama UML de la aplicación

3. Problemas que surgieron durante la implementación

No hay casi nada que destacar. Al principio costó lograr que el árbol se armara correctamente (es decir que se crearan los nodos esperados), hasta que se pudo dar con los pasos correctos. También si bien al principio se había optado por utilizar la clase Tuple, que viene incluida de forma predeterminada, se decidió no hacerlo, ya que no permite modificar el contenido de la misma, una vez instanciada, y en su lugar se decidió crear la clase personalizada Dupla, que sí lo permite (era necesario para poder asignar el valor de la función heurística en cada nodo).

4. Capturas de pantalla

A continuación se muestran capturas del sistema con la descripción correspondiente:

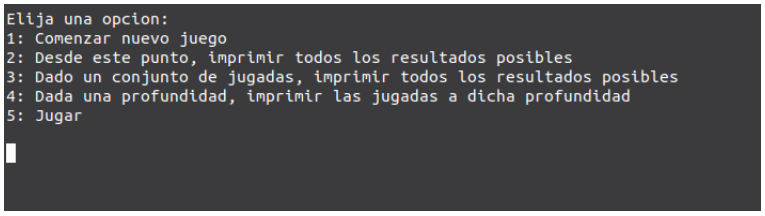

Figura 2: Menú principal

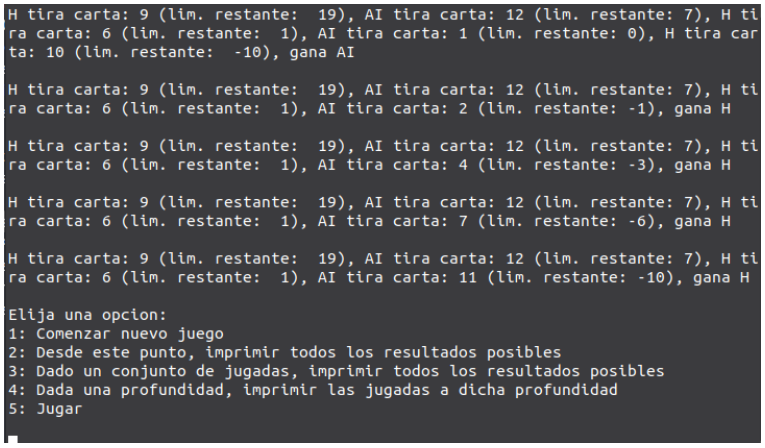

Figura 3: Consulta 2

```
Ingrese las posibles cartas a jugar o escriba 0 para terminar:

8
5
10
6
9

3

8

Ingrese las posibles cartas a jugar o escriba 0 para terminar:

5
10
6
9

3
8
```

Figura 4: Consulta 3

```
2: Desde este punto, imprimir todos los resultados posibles
3: Dado un conjunto de jugadas, imprimir todos los resultados posibles
4: Dada una profundidad, imprimir las jugadas a dicha profundidad
5: Jugar

4

Ingrese la profundidad o 0 para salir:

3

(4, -1) (5, -1) (7, -1) (8, -1) (11, -1) (12, -1) (4, -1) (5, -1) (7, -1) (8, -1
) (11, -1) (12, -1) (4, -1) (5, -1) (7, -1) (8, -1) (11, -1) (12, -1) (4, -1) (5
, -1) (7, -1) (8, -1) (11, -1) (12, -1) (4, -1) (5, -1) (7, -1) (8, -1) (11, -1)
 (12, -1) (4, -1) (5, -1) (7, -1) (8, -1) (11, -1) (12, -1)

Elija una opcion:
1: Comenzar nuevo juego
2: Desde este punto, imprimir todos los resultados posibles
3: Dado un conjunto de jugadas, imprimir todos los resultados posibles
4: Dada una profundidad, imprimir las jugadas a dicha profundidad
5: Jugar
```

Figura 5: Consulta 4

```
4: Dada una profundidad, imprimir las jugadas a dicha profundidad
5: Jugar

5


Limite: 29
Naipes disponibles (usuario):
10, 1, 3, 2, 6, 9
Ingrese naipe: 10

Limite: 19
Naipes disponibles (AI):
4, 5, 7, 8, 11, 12
La AI jugo la carta 4

Elija una opcion:
1: Comenzar nuevo juego
2: Desde este punto, imprimir todos los resultados posibles
3: Dado un conjunto de jugadas, imprimir todos los resultados posibles
4: Dada una profundidad, imprimir las jugadas a dicha profundidad
5: Jugar
```

Figura 6: Partida en curso

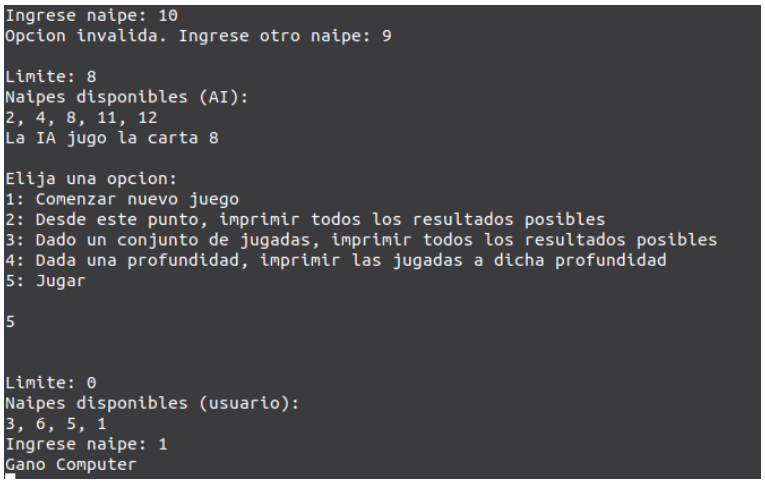

Figura 7: Fin de partida: ganó la computadora

5. Ideas y sugerencias para mejorarlo

Se podría implementar una opción para modificar el número de cartas que están en juego, así como también el puntaje límite. También se podría agregar un función para seleccionar la dificultad de juego, ya que en este caso la computadora juega a ganar. En un caso así, se podrían agregar más valores para la función heurística, y según la dificultad elegida la computadora elegiría nodos con valores más o menos favorables a ella misma, a la hora de jugar. También para hacerlo más agradable visualmente se podría implementar una interfaz gráfica.

6. Conclusión

Se pudo armar el sistema, cumpliendo los requisitos pedidos por la cátedra. Además, se pudo ver como se puede trabajar con árboles en algo de índole aplicado y ver ello en funcionamiento (en este caso, en un videojuego).

7. Referencias

[1] "*El algoritmo MiniMax y su aplicación en un juego*" `https://devcode.la/tutoriales/algoritmo-minimax/` [Fecha de consulta: 18 de julio de 2020]