Universidad del Rosario

Automatizando el juego de las mini-damas
Reporte del primer problema del curso
"IA: Representación y solución de problemas"

Juan Miguel Gutiérrez Vidal
Juan Camilo Llanos

16 de septiembre de 2020

1. Introducción

Las damas inglesas, o mas conocida como "chekers", es uno de los juegos más viejos que han sobrevivido al paso del tiempo. Este mismo posee cerca de 500 billones de billones de posibles posiciones (5×10^{20}), por lo que hacer el cálculo de todas estas es algo bastante demandante, profundizaremos sobre esto más adelante. Primeramente, hablemos un poco sobre como se clasifican las soluciones de un juego, existen tres maneras :

- **Súper-débilmente solucionado:** El valor de la victoria teórica del juego ha sido determinado (se sabe cual es el resultado de jugar). Por ejemplo, se conoce que se Pedro gano el juego, pero nadie sabe cual es la estrategia ganadora.
- **Débilmente solucionado:** El valor de la victoria teórica del juego ha sido determinada, y se conoce la estrategia ganadora desde el inicio del juego.
- **Solucionado Fuertemente:** para cualquier estado, una estrategia es conocida para determinar el valor de la victoria teórica de este para ambos participantes.

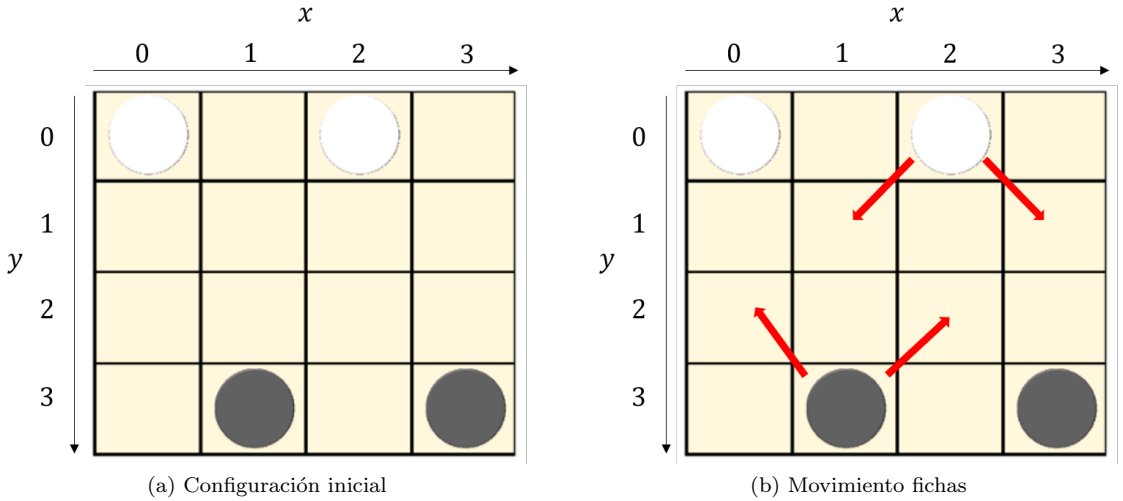
La tarea de "solucionar" el juego de las damas, determinando el final de este es bastante difícil , puesto que existen muchas combinaciones posibles para cada ficha. Este problema ha sido "solucionado" débilmente, y para esto fue necesario utilizar 50 computadores simultáneamente y durante mucho tiempo para obtener los resultados que se comentan en el paper [2] . Dada la complejidad computacional necesaria para solucionar el juego, decidimos convertirlo en un mini-damas de (4×4), con 4 fichas. Para solucionar un juego de 10 fichas, es necesario guardar las posiciones en 237 GB ,ya que para 10 fichas en un tablero de 8×8, existen 39, trillones de posiciones. De esta manera podremos saber si se están comportando los agentes inteligentemente y después será posible incrementar gradualmente el tablero.

Este juego ciertamente es complejo tanto en la toma de decisiones (la dificultad requerida para tomar las decisiones correctas) como en su espacio (el tamaño de espacio de búsqueda para los algoritmos). A continuación proponemos las reglas del juego y por las que se regirán los agentes artificiales.

1.1. Reglas

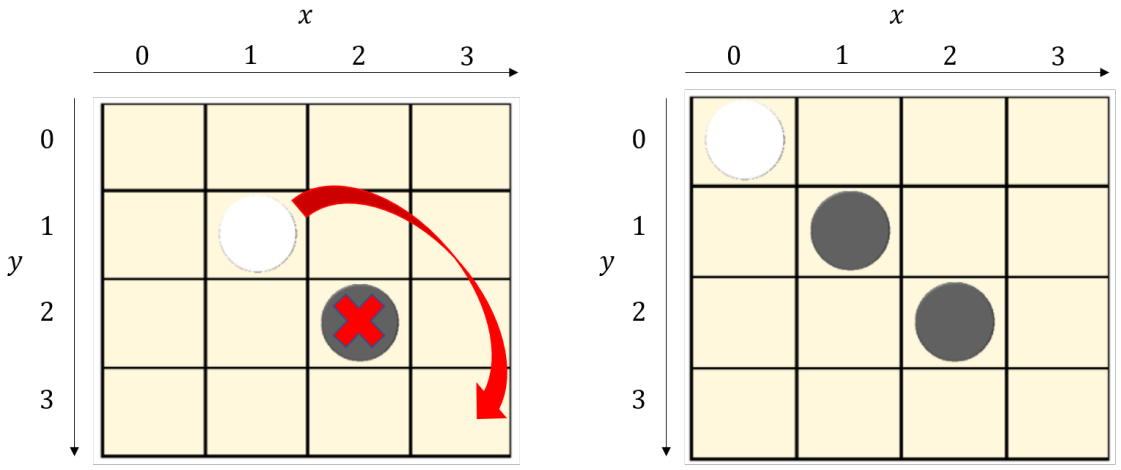
- **Objetivo:** Eliminar todas las fichas del oponente.
- **Numero de jugadores:** 2, uno juega con fichas blancas y otro con negras.
- **Configuración:** Cada jugador hace sus jugadas por turnos en un tablero 4×4. Cada uno inicia con dos fichas: Las blancas inician con las dos fichas en la posición $(0,0)$ y $(2,0)$. Las negras inician con las dos fichas en la posición $(1,3)$ y $(3,3)$ (Figura 1).
- **Movimientos**:
  - Los peones siempre se mueven diagonalmente hacia adelante, hacia el lado del oponente (Figura 1).
  - Para mover un peón en un espacio diagonal , es necesario que el espacio se encuentre vacío. Si el espacio se encuentra ocupado por una ficha del oponente y se encuentra el espacio diagonal en la misma dirección vacía, se mueve a este espacio vacío y se captura la ficha del contrincante (Figura 2).
  - Si todos los cuadrados adyacentes (Diagonales) al peón, se encuentran ocupados, el peón no se podrá mover (Figura 2).
- **Reinas:** Si las fichas llegan a la primera fila del bando el oponente, se convierten en reinas, estas pueden moverse y comer diagonalmente hacia atrás y adelante.
- **Ganador:** El primer jugador que capture todas las damas del oponente , gana !
- **Empate:**: Se considera empate si después de 10 movidas ninguna ficha se come a otra.(En las damas 8×8 son entre 40 y 50 movimientos [1])

El juego de las mini-damas involucra dos jugadores, quienes hacen sus jugadas por turnos en un tablero 4×4. Cada jugador inicia con dos fichas: Las blancas inician con las dos fichas en la posición $(0,0)$ y $(2,0)$. Las negras inician con las dos fichas en la posición $(1,3)$ y $(3,3)$.



(a) Configuración inicial (b) Movimiento fichas

Figura 1: a) Las blancas inician con las dos fichas en la posición $(0,0)$ y $(2,0)$. Las negras inician con las dos fichas en la posición $(1,3)$ y $(3,3)$. b) las fichas comen en diagonal hacia el lado del oponente.



(a) Captura en diagonal

(b) Bloqueo ficha blanca

Figura 2: Representación situaciones

Nuestro problema consiste en un implementar un agente que sea competente para jugar damas, así se ponen a competir dos agentes artificiales y se llega a un Equilibro de Nash, donde teóricamente esta demostrado que si se ponen a jugar dos maquinas que conocen los resultados finales para todos los movimientos, llegaran a un empate [2].

2. Métodos

La solución al problema requiere una definición formal previa a la implementación en python, que es la siguiente:

Estado Inicial: Situación del entorno desde el cual comienza el juego. En el caso de las damas, el estado inicial es el tablero con las 2 fichas de cada bando en sus posiciones respectivas.

Turno-Jugador(s): Define cuál jugador tiene el turno en el estado s, el cual puede ser *blancas* o *negras*. (Realmente es un propiedad del objeto damas, no una función).

Llenar(s): Llena el tablero con la configuración inicial del juego. Se debe correr este método al inicializar el objeto damas (El juego).

Posibles acciones(s): Descripción de las posibles acciones del Jugador(s), dado un estado s. En este caso, para cada ficha del jugador los posibles movimientos que puede realizar en un movimiento.

Funciones de transiciones (s, f, m): Descripción del entorno que resulta de la ejecución de mover la ficha f utilizando el movimiento m, por el Jugador(s) en el estado s. Junto con el estado inicial y las posibles acciones, la función de transiciones define el espacio de estados del juego.

Prueba de objetivo (s): Permite determinar si el juego se termina cuando se obtiene el estado s. Esto sucede si algún jugador, no tiene fichas con las que jugar, o han pasado mas de 10 jugadas sin comerse una ficha.

Función de utilidad (s): Definida sólo cuando el juego se termina en el estado s, y especifica la utilidad asociada al estado s. En el caso de las damas asumiremos que si el ganador son las **blancas**, la utilidad es 1; si el ganador son las *negras*, la utilidad es -1; en caso de empate, la utilidad es 0.

Una vez implementado el problema así definido, se procedió a implementar el algoritmo `minimax` mediante la función `minimax-decision` (Ver algoritmo 1). Este algoritmo crea un árbol de estados,

partiendo desde un estado dado s, atribuyéndole un valor `minimax` a cada estado con base en la función de utilidad arriba descrita. Los pagos para el jugador de las *blancas* es positivo, y los del jugador de las *negras* es negativos, así que el primero buscará estados que maximicen la utilidad, mientras que el segundo buscará estados que la minimicen. Estas funciones definen el back-end de la aplicación.

Algorithm 1 Min-max | s: Estado, f: ficha , m: movimiento

```
1:  function MINIMAX-DECISIÓN(estado)
2:      indice = arg máx_{f,m ∈ACCIONES(s)} VALOR-MIN(RESULTADO(estado,f,m),
3:      return f(indice), m(f(indice), indice)

4:  function VALOR-MAX(estado))
5:      if TEST-OBJETIVO(estado) then
6:          return UTILIDAD(estado)
7:      v ← −∞
8:      for f ∈ ACCIONES(estado) do
9:          for m ∈ f do
10:             v ← MAX(v,VALOR-MIN(RESULTADO(s, f, m)))
11:     return v

12: function VALOR-MIN(estado))
13:     if TEST-OBJETIVO(estado) then
14:         return UTILIDAD(estado)
15:     v ← −∞
16:     for f ∈ ACCIONES(estado) do
17:         for m ∈ f do
18:             v ← MIN(v,VALOR-MAX(RESULTADO(s, f, m)))
19:     return v
```

Además de implementar el algoritmo previo, implementamos el algoritmo $\alpha - \beta$ mediante la función `ab-decisión` (Algoritmo 2), este algoritmo funciona de manera similar al algoritmo de `mini-max`, solo que es mas eficiente, ya que no mira todo los posibles hojas del árbol de estados, si no que si digamos en un proceso de minimización la primera hoja es -1, y la segunda hoja que le sigue es 0, sabemos que elige -1, y no busca más. El resultado del algoritmo es el mismo obtenido por el de `mini-max`, es decir un empate, débilmente solucionado, por que no estamos recorriendo todos los posibles movimientos.

3. Resultados

El programa de agente se implementó para jugar con *blancas* o *negras*, de tal manera que el jugador humano, con el color contrario, hace una jugada, el computador corre el algoritmo `mini-max` o `alpha-beta` (Figura) sobre el tablero resultante y juega. Al comienzo del juego existen muchas opciones, por lo que el computador debe generar un árbol relativamente grande y debe esperar algún tiempo para obtener respuesta del jugador humano. Dado que el agente genera el árbol de estados completo, previendo todos los caminos posibles y sus acciones óptimas. Si el humano no juega bien, el computador sabrá encontrar cual jugada lleva a la victoria, en caso contrario si el jugador humano sabe cual es la acción óptima, entonces al final habrá un empate. De la misma manera si se ponen a competir dos agentes, terminarán en empate.

Por otra parte no resulta altamente costoso al computador calcular todos los posibles movimientos en este mini-juego.

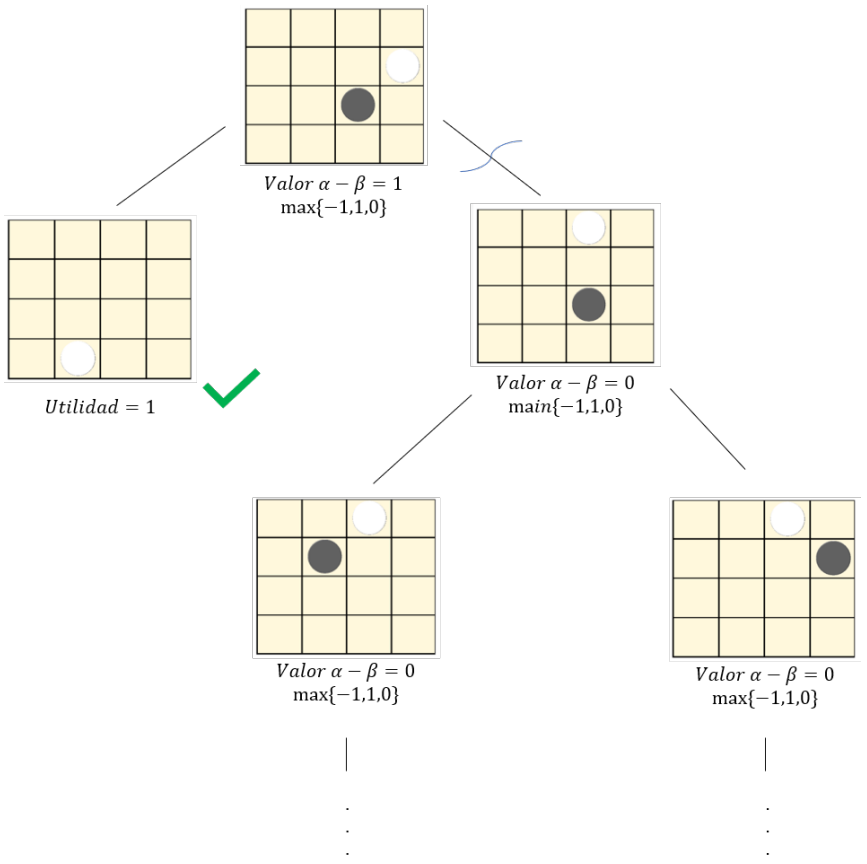


Figura 3: a) El algoritmo del alfa-beta, ya encontró una solución en la hoja izquierda, luego ya no revisa las demás. b) El algoritmo minmax,revisa tanto el lado izquierdo como derecho

Algorithm 2 ALPHA-BETA | s: Estado, f: ficha , m: movimiento

function MINIMAX-DECISIÓN(*estado*)
    indice = $\arg\max_{f,m \in ACCIONES(s)}$ VALOR-MIN(RESULTADO(*estado*,f,m, $-\infty, \infty$),
    return $f(indice), m(f(indice), indice)$

function VALOR-MAX(*estado*))
    if TEST-OBJETIVO(*estado*) **then**
        return UTILIDAD(*estado*)
    $v \leftarrow -\infty$
    for $f \in$ ACCIONES(*estado*) **do**
        for $m \in f$ **do**
            $v \leftarrow$ MAX(v,VALOR-MIN(RESULTADO(s, f, m)))
    if $v \geq \beta$ **then**
        $\alpha \leftarrow$MAX(α, v)
    return v

function VALOR-MIN(*estado*))
    if TEST-OBJETIVO(*estado*) **then**
        return UTILIDAD(*estado*)
    $v \leftarrow -\infty$
    for $f \in$ ACCIONES(*estado*) **do**
        for $m \in f$ **do**
            $v \leftarrow$ MIN(v,VALOR-MAX(RESULTADO(s, f, m)))
    if $v \leq \alpha$ **then**
        $\beta \leftarrow$MIN(α, v)
    return v

4. Discusión

Los dos algoritmos que implementa el programa de agente son muy poderosos, permitiendo que este nunca pierda y en el peor de los casos empate. Sin embargo la creación completa de decisión es ineficiente, el algoritmo alfa-beta disminuye un poco este costo y es más eficiente en tiempo. Falta aumentar el tablero a 6×6 y reconocer si es computacionalmente es viable encontrar una solución en un tiempo considerable. Seria útil además implementar la función heurística realizada en el articulo de "Chekers solved", [2] o plantear una mejor heurística que disminuya el costo computacional considerablemente.

5. Conclusiones

- Se implementó un programa de agente para el juego de las mini-damas que permite hacer simulación.
- Es posible notar que a medida que aumenta la complejidad de decisión y espacio, es necesario a recurrir algoritmos más eficientes o heurísticas mejores.
- los computadores tiene mas poder de predicción que los humanos en general, si son entrenados para ello. Esto es mediante la capacidad que tienen de "mirar al futuro" en el sentido de predecir cualquier movimiento del adversario.

Referencias

[1] Richard Pask. (2001). Starting out in chekers. Everyman Publishers plc,London.

[2] Schaeffer, Jonathan & Björnsson, Yngvi & Kishimoto, Akihiro & Müller, Martin & Lake, Robert & Lu, Paul & Sutphen, Steve. (2007). Checkers Is Solved. Science. 317. 1518-1522. 10.1126/science.1144079.