Damas

1. Autores

Camila Montserrat Alderete González
Luis Alberto Cañete Baez
Oscar Leonardo Pedrozo González

2. Resumen

Damas es un juego muy simple de jugar en comparación al ajedrez, en forma breve el juego de damas es un juego donde las reglas son que Cada jugador controla las piezas de un color situadas al comienzo a cada lado del tablero. Empieza el juego las blancas. Los movimientos se hacen por turno, uno por jugador, en diagonal, una sola casilla y en sentido de avance, o sea, hacia el campo del oponente. Si un jugador consigue llevar una de sus fichas al lado contrario del tablero cambiará dicho peón por una dama o reina (dos fichas del mismo color una encima de otra).La dama se mueve también en diagonal, pero puede hacerlo hacia delante y hacia atrás. Según las opciones de mesa puede avanzar una casilla como el peón o recorrer cualquier número de casillas mientras estén libres. Nunca podrá saltar por encima de sus propias piezas o dos piezas contiguas.Una pieza puede capturar otra si puede saltar por encima de ella siempre en dirección de ataque y en diagonal y caer en la casilla inmediatamente detrás de aquella. Finaliza la partida cuando un jugador abandona, se queda sin fichas o estas no tienen posibilidad de movimiento (bloqueo o ahogada).Se llega a tablas o empate cuando ambos jugadores lo pactan.
En este artículo se aborda el problema que supone el juego de Damas desde el punto de vista de la computación y los algoritmos como la complejidad que supone estos, donde realizan pruebas de rendimiento para realizar comparaciones de rendimiento y posteriormente dar una conclusión.

3. Introducción

El juego de Damas se cree que surgió en algún lugar entre Francia y España entre los años 1000-1500 d.C. En sus comienzos el juego se llamó *«ferses»*, nombre que ostentaba la reina en el juego de ajedrez. El movimiento de las fichas del juego de damas fue tomado de los que hacía la reina del ajedrez de acuerdo a las reglas del juego en esa época. Años más tarde a las damas se le incorporaría un nuevo movimiento que le daba la ventaja de poder saltar sobre otras fichas para realizar su captura.
Arthur Lee Samuel fue uno de los pioneros en el campo de la inteligencia artificial y los videojuegos informáticos, quien acuñó el término "aprendizaje automático" en 1959. Creó con éxito el primer programa checkers basado en el aprendizaje automático, dando una demostración temprana de los conceptos fundamentales de la inteligencia artificial.
A pesar de que las damas tienen menos jugadas que ajedrez, también posee su complejidad de implementación. En el presente trabajo desarrollaremos el algoritmo de Minimax y de Reinforcement learning con sus comparaciones respectivas y las conclusiones a las que llegamos analizando los resultados.

4. Descripción del Problema

El juego de damas se ha implementado con las siguientes reglas.

- Se juega en un tablero de 8x8 con 64 posiciones, estas alternadas entre 32 posiciones de color oscuro y 32 de color claro.
- Las piezas de color claro se encuentran en la parte inferior a la derecha de cada jugador.

- Solo las celdas oscuras están ocupadas en el tablero.
- Las fichas poseen los colores blanco y negro para distinguir a los jugadores de manera sencilla.
- Los jugadores pueden mover una ficha a la vez, por turnos.
- Existen dos tipos de fichas, los peones y las fichas coronadas (damas).
- Los peones pueden moverse hacia adelante en diagonal, mientras que la dama puede moverse hacia atrás y adelante en diagonal. Para capturar una ficha, se debe encontrar frente (casilla en diagonal) a la ficha seleccionada del jugador.
- En caso de ser posible la captura, el jugador tiene la opción de capturar o mover otra ficha.
- Una captura puede ser seguida por otras de manera consecutiva, siempre que la jugada sea permitida.
- El peón se convertirá en dama al alcanzar la fila más cercana al oponente, terminando con la secuencia de captura si la hubiera.
- Luego de capturar la ficha, la dama queda en la siguiente posición de la ficha capturada. Es decir no puede ir más de una celda de captura.
- Aquel jugador que se quede primero sin fichas es el perdedor
- En caso de que transcurran 50 movimientos sin captura de piezas, el juego se declara empate.

5. Algoritmos Implementados

Los algoritmos que implementamos en el juego son el de Minimax, Minimax con poda alfa beta y Reinforcement learning.
El algoritmo minimax es uno de los más usados para resolver este tipo de problemas porque busca que el jugador obtenga la máxima ganancia, es decir, el mejor movimiento, y este algoritmo evalúa los posibles movimientos de tu jugador y del oponente.
La poda alfa beta busca optimizar el algoritmo de minimax, descartando las evaluaciones menos óptimas sin afectar el resultado de la evaluación. Reduciendo el tiempo de evaluación y facilitando así aumentar la profundidad de la búsqueda.
Finalmente, tenemos el algoritmo de Reinforcement learning, que a diferencia de Minimax este asocia los estados posibles con una probabilidad. De esta manera, se puede saber que tan favorable es ir a una casilla. El único inconveniente es que se debe entrenar exhaustivamente al agente para que aprenda las jugadas posibles más favorables.

5.1. Algoritmo Minimax

Minimax es un algoritmo de backtracking que se utiliza en la toma de decisiones y la teoría del juego para encontrar el movimiento óptimo para un jugador, asumiendo que su oponente también juega de manera óptima. Es ampliamente utilizado en juegos por turnos de dos jugadores.
En Minimax los dos jugadores se denominan maximizador y minimizador. El maximizador intenta obtener la puntuación más alta posible mientras que el minimizador intenta hacer que su oponente obtenga la menor puntuación posible. Por cada movimiento que haga el jugador maximizador , este debe simular el peor escenario con el que se pueda topar y elegir el que menos le perjudique. Cuando se cambia de turno, el adversario juega como maximizador y considera a su oponente como jugador minimizador.

```
function minimax(nodo, profundidad, maxPlayer)
       if depth ==0 or nodo es un nodo terminal then
       return static evaluacion del nodo

       if MaxPlayer then
              maxEva= -∞
              for each nodoHijo of nodo do
                     eva= minimax(nodoHijo, profundidad-1, false)
                     maxEva= max(maxEva,eva)
              return maxEva

       else
              minEva= +∞
              for each nodoHijo of nodo do
                     eva= minimax(nodoHijo, profundidad-1, true)
                     minEva= min(minEva, eva)
              return minEva
```

5.2. Algoritmo Minimax con poda alfa beta

La poda alfa-beta es una técnica de optimización para el algoritmo minimax. Reduce el tiempo de cálculo. Esto nos permite buscar mucho más rápido e incluso adentrarnos en niveles más profundos en el árbol del juego si tiene un límite de tiempo, ya que al podar ramas innecesarias se visitan menos nodos y se tiene la posibilidad de profundizar más, de modo que esta mejoraría su desempeño frente a un minimax sin poda. Se llama poda Alfa-Beta porque pasa 2 parámetros adicionales en la función minimax.
Alpha es el mejor valor que el maximizador puede garantizar actualmente en ese nivel o por encima.
Beta es el mejor valor que el minimizador puede garantizar actualmente en ese nivel o superior.

```
function minimax(node, depth, isMaximizingPlayer, alpha, beta):

    if node is a leaf node :
        return value of the node

    if isMaximizingPlayer :
        bestiVal = -∞
        for each child node :
            value = minimax(node, depth+1, false, alpha, beta)
            bestVal = max( bestVal, value)
            alpha = max( alpha, bestVal)
            if beta <= alpha:
                break
        return bestVal

    else :
        bestVal = +∞
        for each child node :
            value = minimax(node, depth+1, true, alpha, beta)
            bestVal = min( bestVal, value)
            beta = min( beta, bestVal)
            if beta <= alpha:
                break
        return bestVal
```

5.3. Reinforcement learning

El aprendizaje por refuerzo no necesita una función de evaluación de un estado no final para obtener la utilidad, a diferencia del otro algoritmo mencionado anteriormente. Este algoritmo se basa en la recompensa, es decir si es positiva o negativa. La recompensa se propaga a la jugada anterior del jugador, cada vez que ambos realicen una secuencia de movimientos. Esto también significa que el tiempo de aprendizaje aumenta al aumentar la cantidad de acciones posibles de cada estado o si aumenta la profundidad en la que se encuentran los estados finales. En este algoritmo alfa se representa como un valor entre 0 y 1, el es un parámetro que maneja la velocidad de propagación de la recompensa. Valores cerca de cero de alfa conllevan a realizar más iteraciones de entrenamiento,lo que puede llevar a un agente mejor entrenado. El valor alfa se actualiza siguiendo la siguiente fórmula

```
alpha = 0.5 - 0.49 * iteracion_actual/cantidad_total_iteraciones
```

El parámetro qRate controla la razón a la que se debería explotar los nuevos estados para tener una mayor robustez ante estados desconocidos o revisitar posibles estados que disminuyan nuestras posibilidades de estancarnos en un mínimo local.
Para este trabajo se usó para el entrenamiento un qRate de 0.8; de modo que al generar un número aleatorio y si este sale menor o igual al qRate, entonces se realiza una jugada elitista, en caso contrario se realiza una jugada aleatoria, lo cual es necesario para equilibrar la explotación y la diversificación durante el entrenamiento.
En la tabla de aprendizaje se almacenan la deseabilidad de los posibles estados. Si no se encuentra la deseabilidad para el estado, se le asigna uno que luego irá refinando a medida que el entrenamiento transcurra.
Para este trabajo se realizaron 10.000 iteraciones, en el cual se entrenó a las fichas negras contra el algoritmo de minimax profundidad 2.

```
function reinforcement-learning
	maxProb = - ∞
	q = random(0,1)
	if q <= qRate:
		for each tablerosPosibles of movimientosPosibles(tablero:
			prob = getprobabilidad(tablerosPosibles)
			if probabilidad > maxProb:
				maxProb = prob
				tableroCandidato = tablerosPosibles
		updateProbabilidad(ultimoTablero, maxProb)
	else:
		tableroCandidato = random(movimientosPosibles(tablero))
	ultimoTablero = tablero
	return tableroCandidato

function updateProbabilidad(tablero, newprob)
	prob = getprobabilidad(tablero)
	prob = prob + alfa * (probNueva - prob)
	saveProbabilidad(tablero, prob)
```

6. Resultados Experimentales

Se realizaron pruebas experimentales entre distintos algoritmos con varias profundidades, obteniendo como datos los nodos expandidos, tiempo de juego y resultado final del juego.

Aplicando el algoritmo de minimax en el caso de las negras y poda en las blancas, los nodos expandidos para 10 jugadas lanzaron los siguientes resultados.

#	**Negras: minimax(2)**	**Blancas : poda(4)**	**Ganador**
1	4099	32036	empate
2	1747	15795	blanco
3	3720	33274	blanco
4	10163	81703	blanco
5	1090	9404	blanco
6	4710	35165	blanco
7	6062	46896	empate
8	1503	10503	blanco
9	826	6367	negro
10	7825	45746	blanco

Las negras ganaron una sola vez mientras que las blancas ganaron 7 veces y hubo 2 empates.

#	**Negras: minimax(4)**	**Blancas : poda(2)**	**Ganador**
1	118203	993	negro
2	93496	780	negro
3	52214	384	negro
4	617982	4436	blanco
5	79750	752	negro
6	59989	361	negro
7	111485	1159	negro
8	37613	442	negro
9	58993	580	negro
10	131722	984	negro

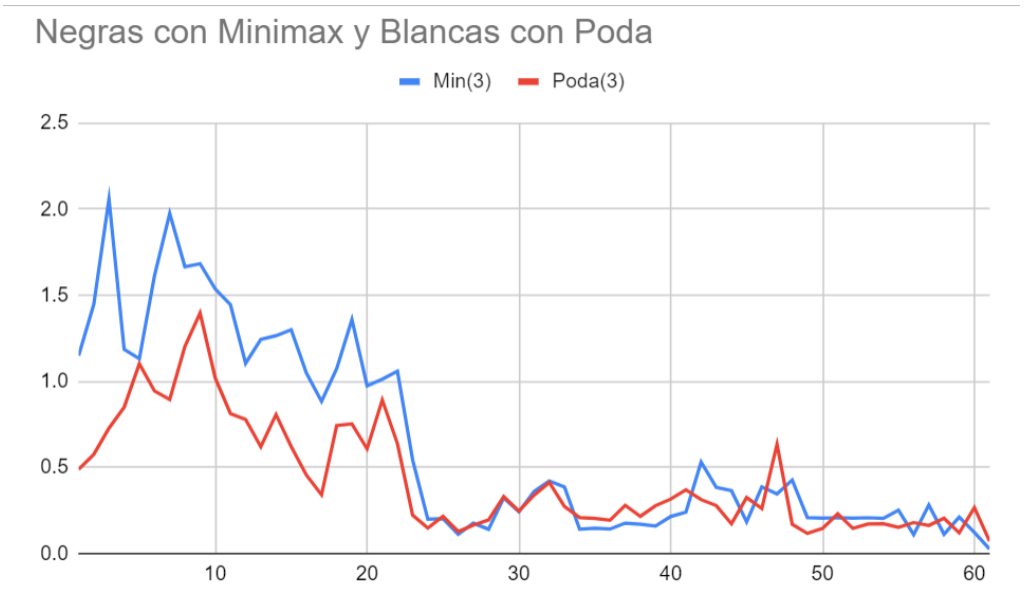
Las negras ganaron 9 veces mientras que las blancas ganaron solamente 1 vez.

#	Negras: minimax(3)	Blancas : poda(3)	Ganador
1	22096	21028	blanco
2	22706	11557	blanco
3	9682	1987	negro
4	23321	16388	blanco
5	15434	3361	negro
6	6503	2694	blanco
7	28068	11913	blanco
8	20212	8411	blanco
9	30939	7497	negro
10	26711	8074	negro

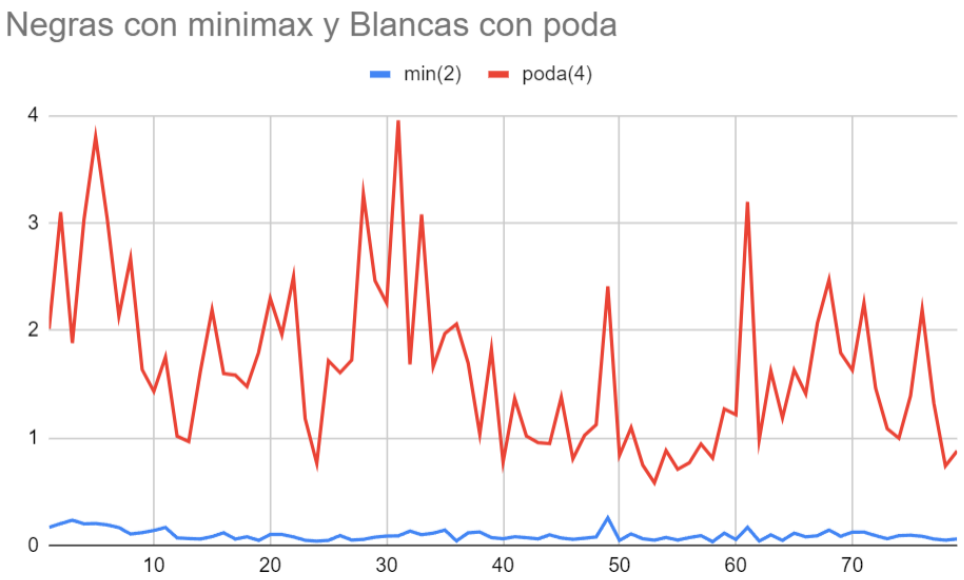
Las negras ganaron 4 veces mientras que las blancas ganaron 6 veces.

Examinando las tablas se puede apreciar cómo a mayor profundidad el algoritmo, ya sea minimax o minimax con poda, mejora la calidad del juego y ofrece más victorias frente a algoritmos con menor profundidad. Además se aprecia que el algoritmo de minimax con poda reduce la expansión de nodos innecesarios, volviéndo más eficiente y rápido frente a un minimax de igual profundidad.

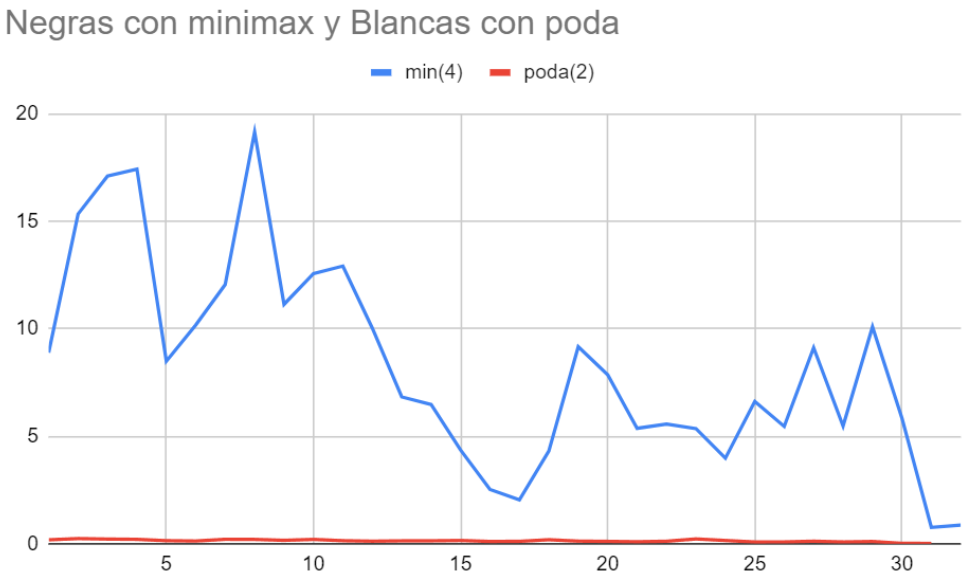
Posteriormente, analizamos los tiempos que tardaban los algoritmos en realizar las jugadas. El eje x representa la cantidad de jugadas que transcurren durante la partida y el eje y el tiempo de decisión de los algoritmos. En este documento mostraremos un gráfico por cada prueba, pero igualmente adjuntamos a este trabajo los resultados en formato de planilla con gráficos y también en formato de texto.



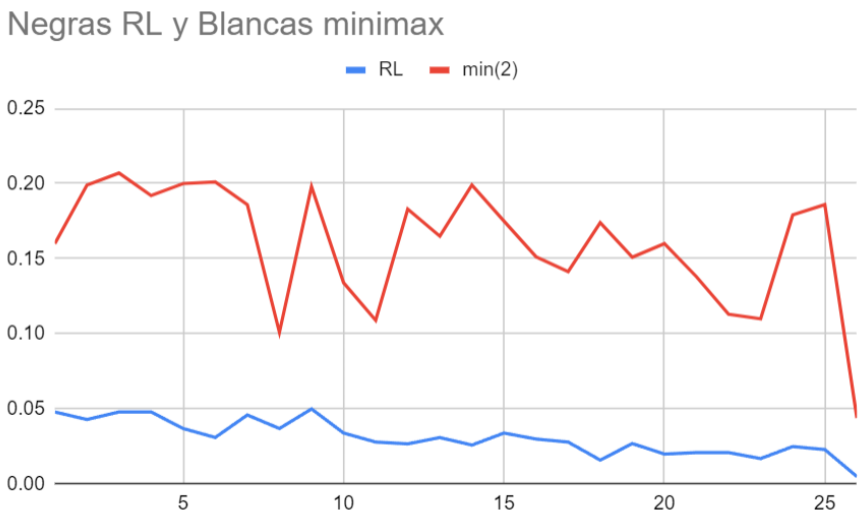
En este gráfico de Min(3) vs Poda(3) se puede apreciar como el minimax con poda y profundidad 3 emplea menos tiempo para tomar una decisión que el minimax tradicional con la misma profundidad.



Aquí en cambio se puede apreciar que el minimax con poda y profundidad 4 consume más tiempo para tomar una decisión frente al minimax de profundidad 2. Pero a cambio del tiempo, el minimax con poda a esa profundidad asegura su victoria frente a las blancas con minimax.



En este escenario se observa el tiempo que tarda el minimax con profundidad 4 en tomar decisiones ante el minimax con poda y profundidad 2. Se puede apreciar con respecto al gráfico anterior que es mejor emplear poda ante grandes profundidades ya que se ahorra tiempo, además no cambia afecta el resultado final del juego ya que la poda es solo optimización. Pese a esto, el algoritmo de minimax a esa profundidad asegura su victoria ante su adversario de menor profundidad.



Para el agente con aprendizaje reforzado se lo entrenó contra minimax con poda y profundidad 2, empleando unas 10000 iteraciones. Luego se lo enfrentó a diez juegos contra el mismo adversario pero en todos perdió. A pesar de ello se noto que el agente emplea menos tiempo en tomar decisiones quizás porque recurre directo a su tabla de aprendizaje y en caso de no encontrar el dato deseado, genera en el momento una deseabilidad para el estado siguiente.

Observando la tabla de aprendizaje del agente se verifica que de las 10000 iteraciones durante el entrenamiento, logró ganar en unas 7 veces.

Los algoritmos Minimax y Poda Alfa beta en profundidad de 2 tienen mayor probabilidad de ganar contra el aprendizaje por refuerzo, por lo que aquí se demuestra que el agente no fue lo suficientemente entrenado y quizá también no se implementó la estrategia adecuada.

Conclusión

Se compararon métodos minimax y minimax con poda alfa-beta y aprendizaje por refuerzo. Incluso con heurísticas muy simples, se han observado buenos resultados en juegos con métodos minimax. Sin embargo, los resultados del aprendizaje por refuerzo no fueron los esperados. Esto podría deberse a un error conceptual en la implementación, un mayor tiempo de entrenamiento o una elección incorrecta de los valores de los parámetros alfa, y la inspección del tablero y sus probabilidades asociadas reveló pequeñas inconsistencias al comparar los valores asignados con los intuitivos. disponible para el tablero, pero debido a la dificultad para rastrear la secuencia, también es difícil verificar la corrección del aprendizaje al producir tal resultado, ya que puede ser un valor perfectamente válido. Como oportunidad de mejora para el aprendizaje por refuerzo consideramos importante verificar de nuevo la implementación y contemplar aplicar otras estrategias, además considerar otras características del juego para dar una correcta valoración al estado, como por ejemplo considerar además de la cantidad de fichas de cada bando, las fichas coronadas, las piezas amenazadas, etc.

7. Referencias

1. McKinley, C.: Checkers rules. *Official game rules.* https://www.officialgamerules.org/checkers (2016). Accedido el 26 de octubre de 2022.
2. Idzham, K.: Study of Artificial Intelligence into Checkers Game using HTML and JavaScript. *IOP Conference Series: Materials Science and Engineering.* https://iopscience.iop.org/article/10.1088/1757-899X/864/1/012091/pdf (2020). Accedido el 30 de octubre de 2022.
3. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach. Prentice Hall,(1995).
4. Osinski, B., Budek, K.:What is reinforcement learning? The complete guide. *Deepsense.* (2018). Accedido el 30 octubre 2022.