

¿Cómo juega un computador al ajedrez y por qué no puedo ganarle?

Pablo Sánchez

Dpto. Ingeniería Informática y Electrónica
Universidad de Cantabria
Santander (Cantabria, España)
p.sanchez@unican.es



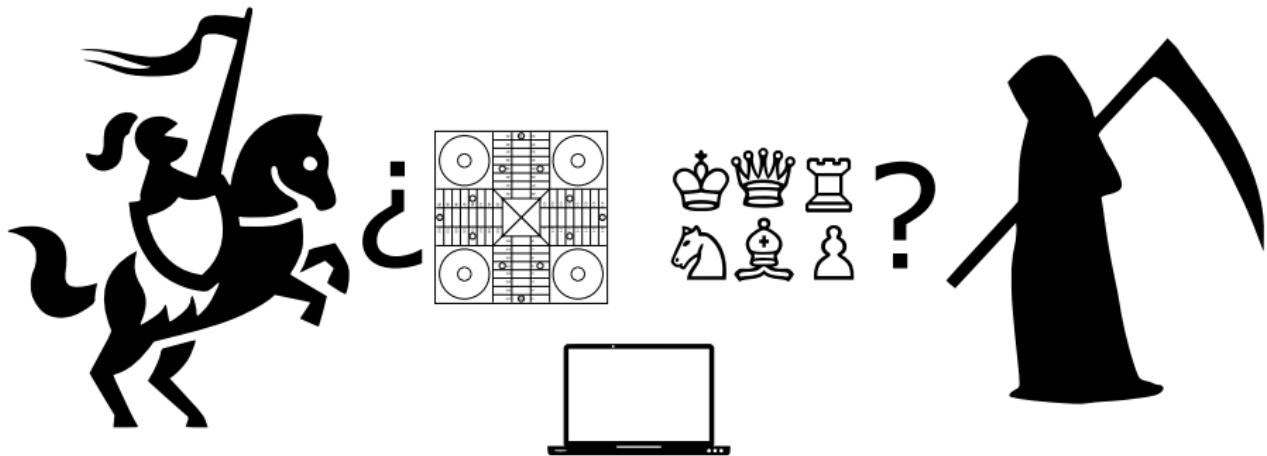
Índice

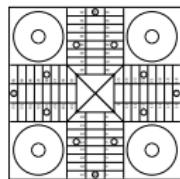
- 1 Introducción
- 2 Juegos Básicos: Algoritmo Minimax
- 3 Backtracking y Ramificación y Poda
- 4 Ajedrez por Computador
- 5 Parchís por Computador: Redes Neuronales
- 6 Conclusiones

Índice

- 1 Introducción
 - Motivación
 - ¿Qué Sabe Hacer un Computador?
- 2 Juegos Básicos: Algoritmo Minimax
- 3 Backtracking y Ramificación y Poda
- 4 Ajedrez por Computador
- 5 Parchís por Computador: Redes Neuronales
- 6 Conclusiones







Índice

- 1 Introducción
 - Motivación
 - *¿Qué Sabe Hacer un Computador?*
- 2 Juegos Básicos: Algoritmo Minimax
- 3 Backtracking y Ramificación y Poda
- 4 Ajedrez por Computador
- 5 Parchís por Computador: Redes Neuronales
- 6 Conclusiones

¿Qué Sabe Hacer un Computador?

- ➊ Sumas y restas.
- ➋ (Multiplicaciones y Divisiones).
- ➌ Comparar números por igualdad.
- ➍ Comparar números por desigualdad.
- ➎ Decidir el siguiente paso a ejecutar en función de una comparación.
- ➏ Leer y escribir números en memoria.

¿Qué Sabe Hacer un Computador?

- ➊ Sumas y restas.
- ➋ (Multiplicaciones y Divisiones).
- ➌ Comparar números por igualdad.
- ➍ Comparar números por desigualdad.
- ➎ Decidir el siguiente paso a ejecutar en función de una comparación.
- ➏ Leer y escribir números en memoria.

¿Qué Sabe Hacer un Computador?

- ➊ Sumas y restas.
- ➋ (Multiplicaciones y Divisiones).
- ➌ Comparar números por igualdad.
- ➍ Comparar números por desigualdad.
- ➎ Decidir el siguiente paso a ejecutar en función de una comparación.
- ➏ Leer y escribir números en memoria.

¿Qué Sabe Hacer un Computador?

- ➊ Sumas y restas.
- ➋ (Multiplicaciones y Divisiones).
- ➌ Comparar números por igualdad.
- ➍ Comparar números por desigualdad.
- ➎ Decidir el siguiente paso a ejecutar en función de una comparación.
- ➏ Leer y escribir números en memoria.

¿Qué Sabe Hacer un Computador?

- ① Sumas y restas.
- ② (Multiplicaciones y Divisiones).
- ③ Comparar números por igualdad.
- ④ Comparar números por desigualdad.
- ⑤ Decidir el siguiente paso a ejecutar en función de una comparación.
- ⑥ Leer y escribir números en memoria.

¿Qué Sabe Hacer un Computador?

- ① Sumas y restas.
- ② (Multiplicaciones y Divisiones).
- ③ Comparar números por igualdad.
- ④ Comparar números por desigualdad.
- ⑤ Decidir el siguiente paso a ejecutar en función de una comparación.
- ⑥ Leer y escribir números en memoria.

¿Qué Sabe Hacer un Computador?



Alan Turing

¿Qué Sabe Hacer un Computador?

- ➊ Un computador sólo hace operaciones básicas.
- ➋ Pero hace un billón (europeo) de operaciones básicas por segundo.
- ➌ A un computador se le da bien *buscar una aguja en un pajar*.
- ➍ Un computador no siente cansancio o frustración.
- ➎ Un computador consume energía y produce CO₂.

¿Qué Sabe Hacer un Computador?

- ➊ Un computador sólo hace operaciones básicas.
- ➋ Pero hace un billón (europeo) de operaciones básicas por segundo.
- ➌ A un computador se le da bien *buscar una aguja en un pajar*.
- ➍ Un computador no siente cansancio o frustración.
- ➎ Un computador consume energía y produce CO₂.

¿Qué Sabe Hacer un Computador?

- ① Un computador sólo hace operaciones básicas.
- ② Pero hace un billón (europeo) de operaciones básicas por segundo.
- ③ A un computador se le da bien *buscar una aguja en un pajar*.
- ④ Un computador no siente cansancio o frustración.
- ⑤ Un computador consume energía y produce CO₂.

¿Qué Sabe Hacer un Computador?

- ➊ Un computador sólo hace operaciones básicas.
- ➋ Pero hace un billón (europeo) de operaciones básicas por segundo.
- ➌ A un computador se le da bien *buscar una aguja en un pajar*.
- ➍ Un computador no siente cansancio o frustración.
- ➎ Un computador consume energía y produce CO₂.

¿Qué Sabe Hacer un Computador?

- ① Un computador sólo hace operaciones básicas.
- ② Pero hace un billón (europeo) de operaciones básicas por segundo.
- ③ A un computador se le da bien *buscar una aguja en un pajar*.
- ④ Un computador no siente cansancio o frustración.
- ⑤ Un computador consume energía y produce CO₂.

Índice

- 1 Introducción
- 2 Juegos Básicos: Algoritmo Minimax
- 3 Backtracking y Ramificación y Poda
- 4 Ajedrez por Computador
- 5 Parchís por Computador: Redes Neuronales
- 6 Conclusiones

Algoritmo Minimax

Algoritmo Minimax

Algoritmo, o procedimiento sistemático, para decidir cuál es el movimiento más óptimo en un juego con información perfecta, es decir, donde todos los datos del juego o partida son conocidos por todos los jugadores.

Algoritmo Minimax

- ① Represento el juego como un conjunto de números.
- ② Especifico cómo realizar movimientos válidos como una serie de operaciones básicas.
- ③ Especifico cómo decidir si el juego ha terminado.
- ④ Especifico cómo decidir cuán buena es la victoria (si hay diferencias).

Algoritmo Minimax

- ① Represento el juego como un conjunto de números.
- ② Especifico cómo realizar movimientos válidos como una serie de operaciones básicas.
- ③ Especifico cómo decidir si el juego ha terminado.
- ④ Especifico cómo decidir cuán buena es la victoria (si hay diferencias).

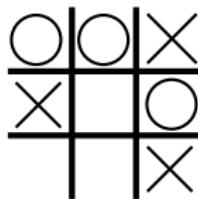
Algoritmo Minimax

- ① Represento el juego como un conjunto de números.
- ② Especifico cómo realizar movimientos válidos como una serie de operaciones básicas.
- ③ Especifico cómo decidir si el juego ha terminado.
- ④ Especifico cómo decidir cuán buena es la victoria (si hay diferencias).

Algoritmo Minimax

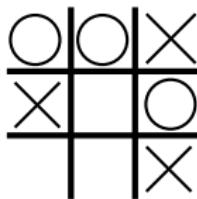
- ① Represento el juego como un conjunto de números.
- ② Especifico cómo realizar movimientos válidos como una serie de operaciones básicas.
- ③ Especifico cómo decidir si el juego ha terminado.
- ④ Especifico cómo decidir cuán buena es la victoria (si hay diferencias).

Algoritmo Minimax



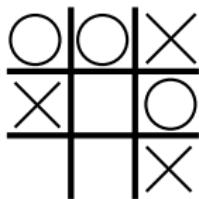
- ➊ El tablero es una tabla de números: -1/libre, 0/círculo, 1/cruz (pc).
 - ➋ Movimientos válidos: si hay hueco (-1), pongo la ficha que toque.
 - ➌ Final de la partida:
- Si el tablero es un vector de 9 posiciones, el resultado de la partida es:
- Si el vector es [-1, -1, -1, -1, -1, -1, -1, -1, -1] → 0 (ganador O)
 - Si el vector es [1, 1, 1, 1, 1, 1, 1, 1, 1] → 1 (ganador X)
 - Si el vector es [0, 0, 0, 0, 0, 0, 0, 0, 0] → 0 (empate)
- ➍ Valor de la partida:

Algoritmo Minimax



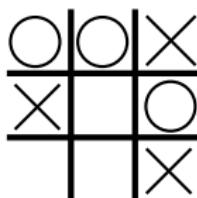
- ① El tablero es una tabla de números: -1/libre, 0/círculo, 1/cruz (pc).
- ② Movimientos válidos: si hay hueco (-1), pongo la ficha que toque.
- ③ Final de la partida:
 - Si gano el humano, el valor es 1.
 - Si gana el pc, el valor es 0.
 - Si empata, el valor es -1.
- ④ Valor de la partida:

Algoritmo Minimax



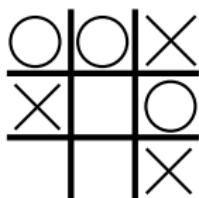
- ① El tablero es una tabla de números: -1/libre, 0/círculo, 1/cruz (pc).
 - ② Movimientos válidos: si hay hueco (-1), pongo la ficha que toque.
 - ③ Final de la partida:
 - ▶ Miro filas, columnas y diagonales por si hay tres fichas iguales.
 - ▶ No hay huecos (-1) libres.
- 💡 Valor de la partida:

Algoritmo Minimax



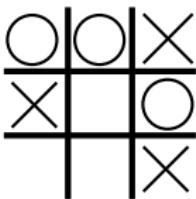
- ① El tablero es una tabla de números: -1/libre, 0/círculo, 1/cruz (pc).
- ② Movimientos válidos: si hay hueco (-1), pongo la ficha que toque.
- ③ Final de la partida:
 - ▶ Miro filas, columnas y diagonales por si hay tres fichas iguales.
 - ▶ No hay huecos (-1) libres.
- ④ Valor de la partida:

Algoritmo Minimax



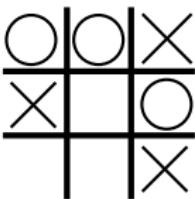
- ① El tablero es una tabla de números: -1/libre, 0/círculo, 1/cruz (pc).
- ② Movimientos válidos: si hay hueco (-1), pongo la ficha que toque.
- ③ Final de la partida:
 - ▶ Miro filas, columnas y diagonales por si hay tres fichas iguales.
 - ▶ No hay huecos (-1) libres.
- ④ Valor de la partida:
 - ▶ Tres circulos (O) en raya, pongo valor 1

Algoritmo Minimax



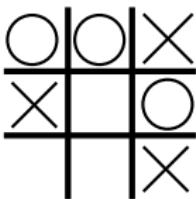
- ➊ El tablero es una tabla de números: -1/libre, 0/círculo, 1/cruz (pc).
- ➋ Movimientos válidos: si hay hueco (-1), pongo la ficha que toque.
- ➌ Final de la partida:
 - ▶ Miro filas, columnas y diagonales por si hay tres fichas iguales.
 - ▶ No hay huecos (-1) libres.
- ➍ Valor de la partida:
 - ▶ Tres círculos (0) en raya, pierdo, valor 0.
 - ▶ Tres cruces (1) en raya, gano, valor 2.
 - ▶ No hay huecos libres (-1) y ningún tres en raya, empate, valor 1.

Algoritmo Minimax



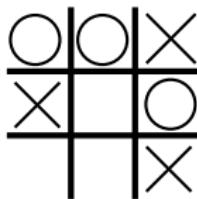
- ① El tablero es una tabla de números: -1/libre, 0/círculo, 1/cruz (pc).
- ② Movimientos válidos: si hay hueco (-1), pongo la ficha que toque.
- ③ Final de la partida:
 - ▶ Miro filas, columnas y diagonales por si hay tres fichas iguales.
 - ▶ No hay huecos (-1) libres.
- ④ Valor de la partida:
 - ▶ Tres círculos (0) en raya, pierdo, valor 0.
 - ▶ Tres cruces (1) en raya, gano, valor 2.
 - ▶ No hay huecos libres (-1) y ningún tres en raya, empate, valor 1.

Algoritmo Minimax



- ① El tablero es una tabla de números: -1/libre, 0/círculo, 1/cruz (pc).
- ② Movimientos válidos: si hay hueco (-1), pongo la ficha que toque.
- ③ Final de la partida:
 - ▶ Miro filas, columnas y diagonales por si hay tres fichas iguales.
 - ▶ No hay huecos (-1) libres.
- ④ Valor de la partida:
 - ▶ Tres círculos (0) en raya, pierdo, valor 0.
 - ▶ Tres cruces (1) en raya, gano, valor 2.
 - ▶ No hay huecos libres (-1) y ningún tres en raya, empate, valor 1.

Algoritmo Minimax



- ➊ El tablero es una tabla de números: -1/libre, 0/círculo, 1/cruz (pc).
- ➋ Movimientos válidos: si hay hueco (-1), pongo la ficha que toque.
- ➌ Final de la partida:
 - ▶ Miro filas, columnas y diagonales por si hay tres fichas iguales.
 - ▶ No hay huecos (-1) libres.
- ➍ Valor de la partida:
 - ▶ Tres círculos (0) en raya, pierdo, valor 0.
 - ▶ Tres cruces (1) en raya, gano, valor 2.
 - ▶ No hay huecos libres (-1) y ningún tres en raya, empate, valor 1.

Algoritmo Minimax

- ① Por turnos, voy calculando todos los movimientos posibles.
- ② Por cada partida finalizada, calculo su valor.
- ③ Tras generar todas las partidas posibles, propago los valores:

Algoritmo Minimax

- ① Por turnos, voy calculando todos los movimientos posibles.
- ② Por cada partida finalizada, calculo su valor.
- ③ Tras generar todas las partidas posibles, propago los valores:
 - Si soy el maximo, propago el movimiento más favorable para mí (mayor).
 - Si soy el minimo, propago el movimiento más desfavorable para mí (menor).

Algoritmo Minimax

- ① Por turnos, voy calculando todos los movimientos posibles.
- ② Por cada partida finalizada, calculo su valor.
- ③ Tras generar todas las partidas posibles, propago los valores:
 - ① Si muevo yo, escojo el movimiento más favorable para mí (*max*).
 - ② Si mueve el contrincante, escoge el movimiento menos favorable para mí (*min*).

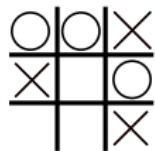
Algoritmo Minimax

- ① Por turnos, voy calculando todos los movimientos posibles.
- ② Por cada partida finalizada, calculo su valor.
- ③ Tras generar todas las partidas posibles, propago los valores:
 - ① Si muevo yo, escojo el movimiento más favorable para mí (*max*).
 - ② Si mueve el contrincante, escoge el movimiento menos favorable para mí (*min*).

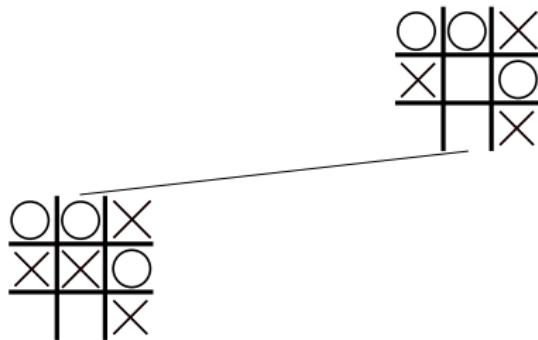
Algoritmo Minimax

- ① Por turnos, voy calculando todos los movimientos posibles.
- ② Por cada partida finalizada, calculo su valor.
- ③ Tras generar todas las partidas posibles, propago los valores:
 - ① Si muevo yo, escojo el movimiento más favorable para mí (*max*).
 - ② Si mueve el contrincante, escoge el movimiento menos favorable para mí (*min*).

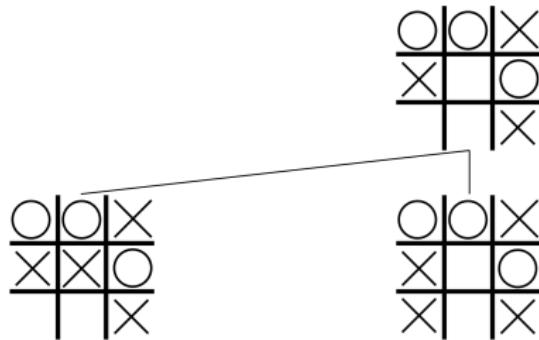
Algoritmo Minimax



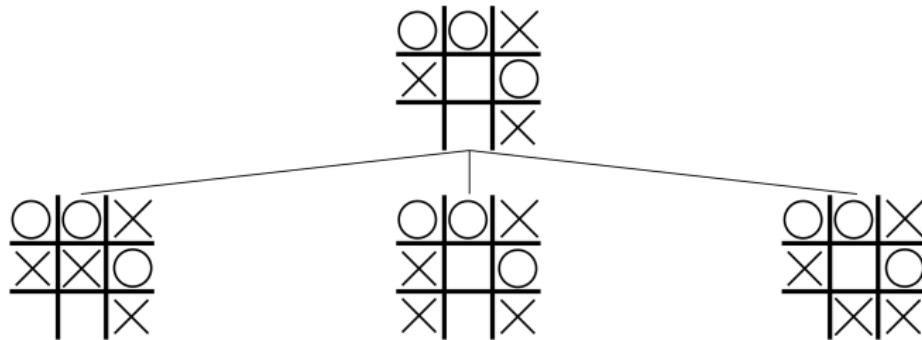
Algoritmo Minimax



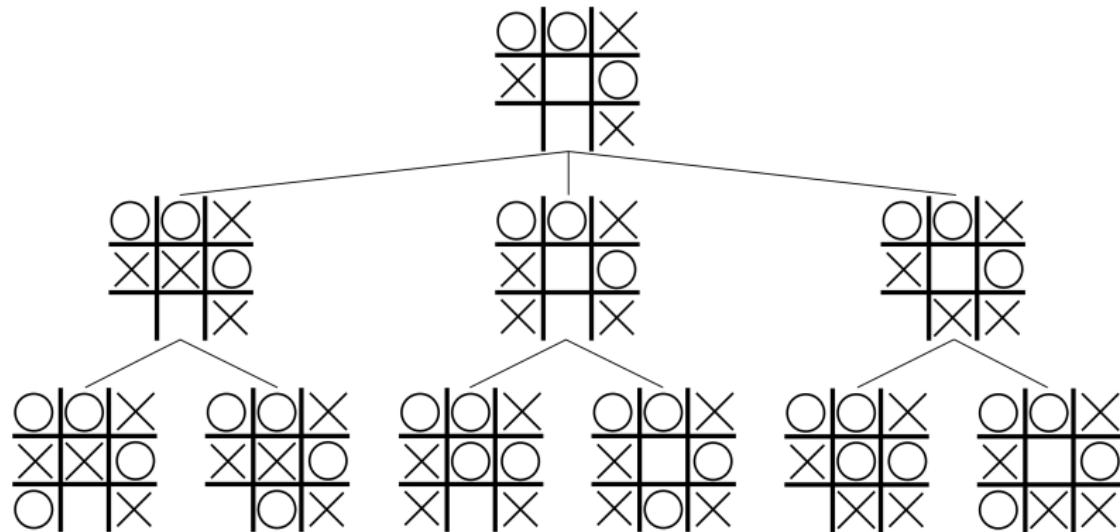
Algoritmo Minimax



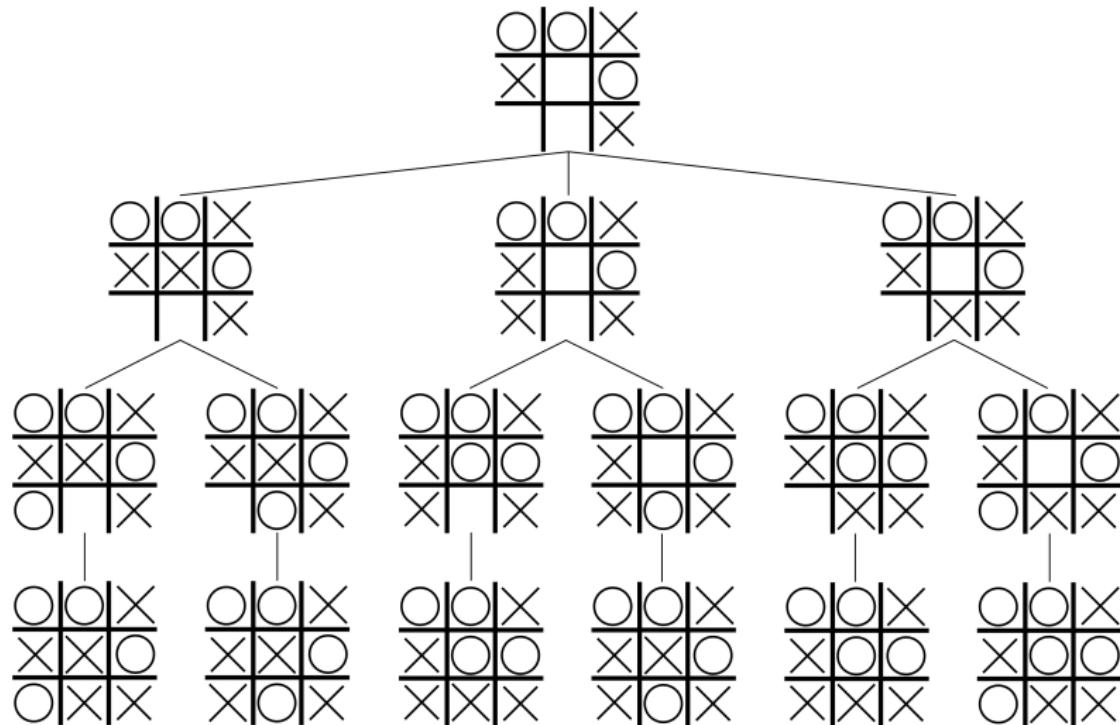
Algoritmo Minimax



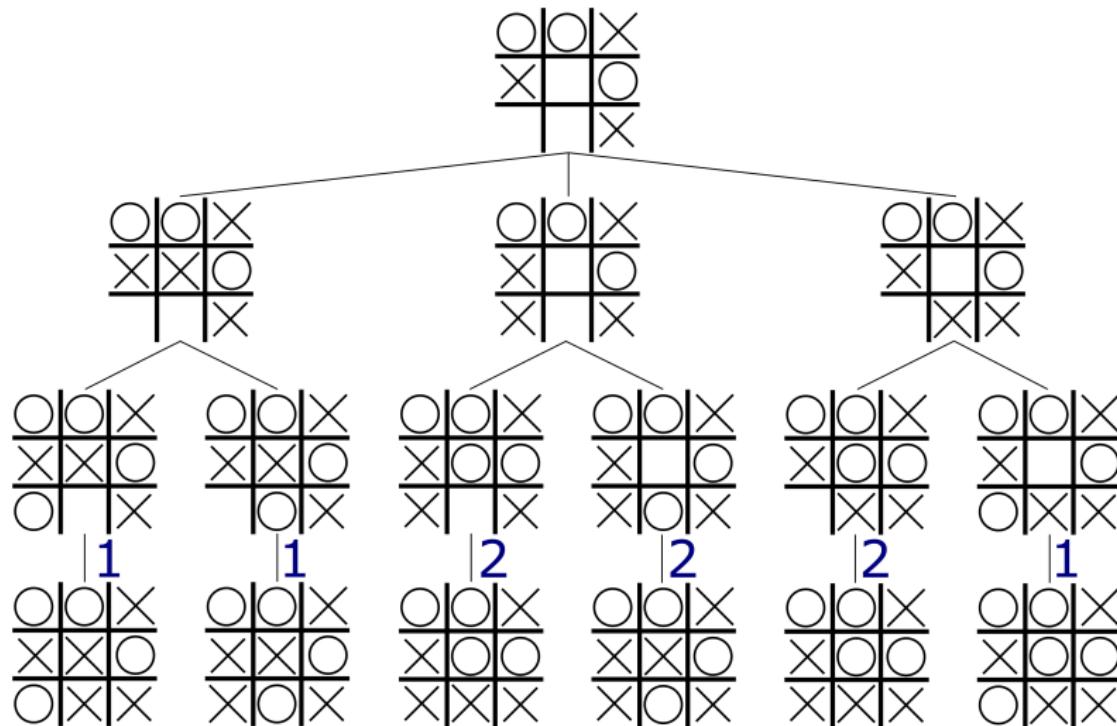
Algoritmo Minimax



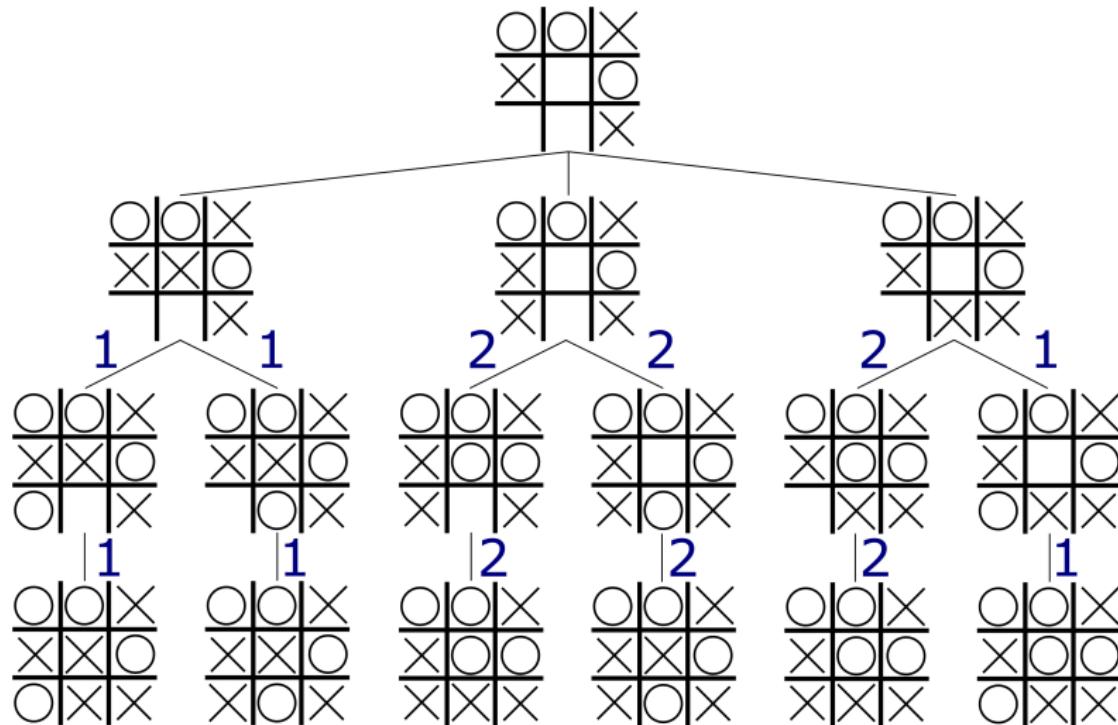
Algoritmo Minimax



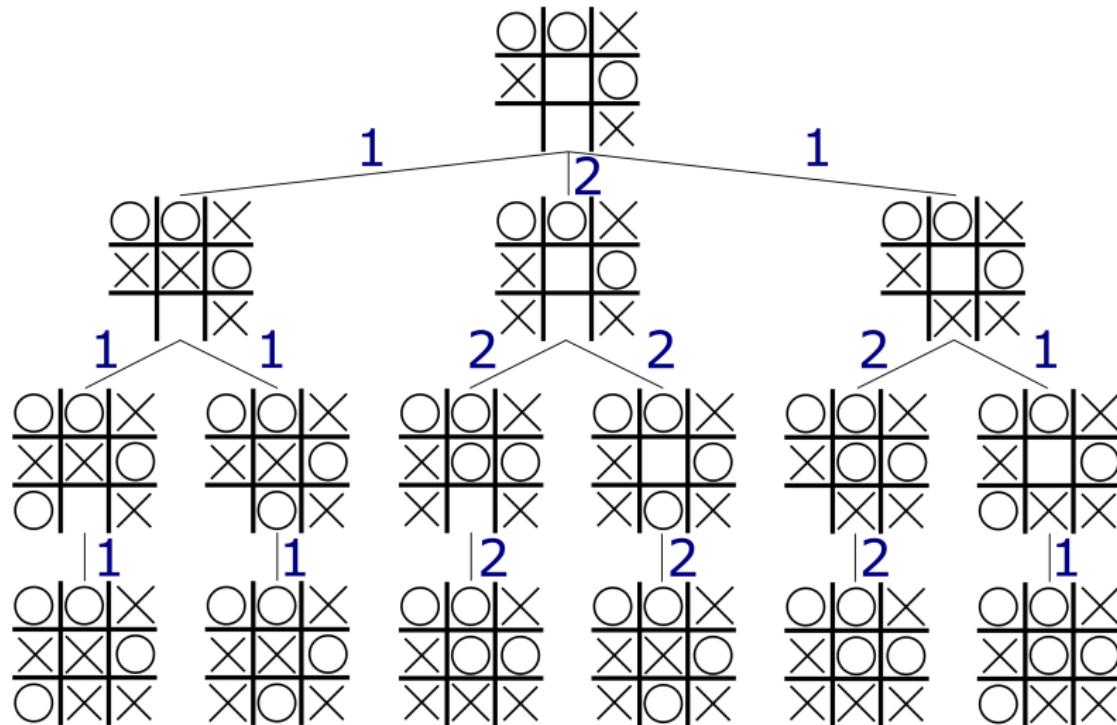
Algoritmo Minimax



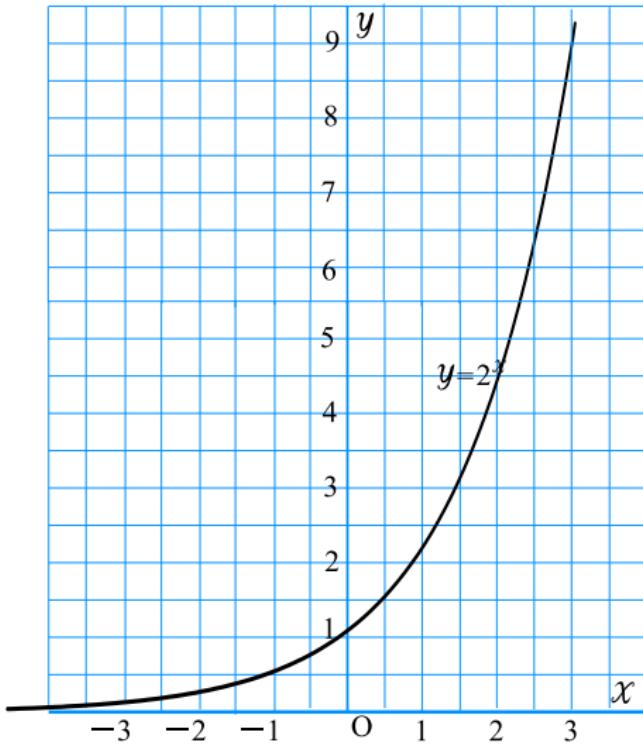
Algoritmo Minimax



Algoritmo Minimax



Problema Algoritmo Minimax



Índice

- 1 Introducción
- 2 Juegos Básicos: Algoritmo Minimax
- 3 Backtracking y Ramificación y Poda
- 4 Ajedrez por Computador
- 5 Parchís por Computador: Redes Neuronales
- 6 Conclusiones

Índice

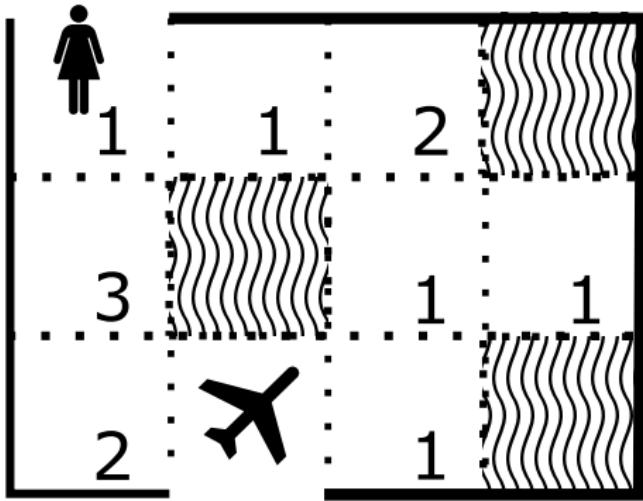
- 1 Introducción
- 2 Juegos Básicos: Algoritmo Minimax
- 3 Backtracking y Ramificación y Poda
 - Backtracking
 - Ramificación y Poda
- 4 Ajedrez por Computador
- 5 Parchís por Computador: Redes Neuronales
- 6 Conclusiones

Algoritmo *Backtracking*

Algoritmo *Backtracking* (Vuelta atrás)

Algoritmo para resolver problemas de optimización que deben satisfacer ciertas restricciones.

Algoritmo *Backtracking*



Algoritmo *Backtracking*

- ➊ Represento el problema como un conjunto de números.
- ➋ Específico cómo realizar pasos válidos hacia la solución.
- ➌ Específico cómo decidir si he encontrado una solución.
- ➍ Específico cómo decidir cuán buena es la solución (si hay diferencias).

Algoritmo *Backtracking*

- ① Represento el problema como un conjunto de números.
- ② Especifico cómo realizar pasos válidos hacia la solución.
- ③ Especifico cómo decidir si he encontrado una solución.
- ④ Especifico cómo decidir cuán buena es la solución (si hay diferencias).

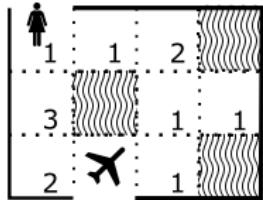
Algoritmo *Backtracking*

- ① Represento el problema como un conjunto de números.
- ② Especifico cómo realizar pasos válidos hacia la solución.
- ③ Especifico cómo decidir si he encontrado una solución.
- ④ Especifico cómo decidir cuán buena es la solución (si hay diferencias).

Algoritmo *Backtracking*

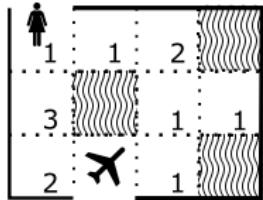
- ① Represento el problema como un conjunto de números.
- ② Especifico cómo realizar pasos válidos hacia la solución.
- ③ Especifico cómo decidir si he encontrado una solución.
- ④ Especifico cómo decidir cuán buena es la solución (si hay diferencias).

Algoritmo *Backtracking*



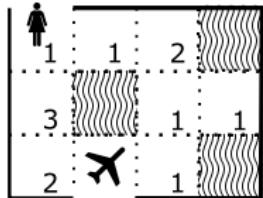
- ➊ Problema: El laberinto es una tabla de números.
 - ▶ -1 significa obstáculo.
 - ▶ Un valor positivo representa el coste de atravesar la casilla.
 - ▶ 0 indica que ya he pasado.
- ➋ Pasos válidos: Moverse derecha, abajo, izquierda y arriba si:
 - ➌ Solución: Estoy en la casilla de salida.
 - ➍ Coste: Suma de las casillas por las que he pasado.

Algoritmo *Backtracking*



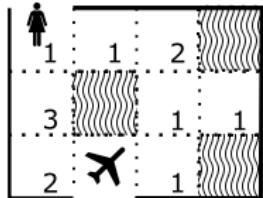
- ➊ Problema: El laberinto es una tabla de números.
 - ▶ -1 significa obstáculo.
 - ▶ Un valor positivo representa el coste de atravesar la casilla.
 - ▶ 0 indica que ya he pasado.
- ➋ Pasos válidos: Moverse derecha, abajo, izquierda y arriba si:
 - ➌ Solución: Estoy en la casilla de salida.
 - ➍ Coste: Suma de las casillas por las que he pasado.

Algoritmo *Backtracking*



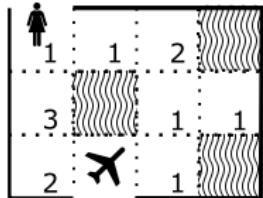
- ➊ Problema: El laberinto es una tabla de números.
 - ▶ -1 significa obstáculo.
 - ▶ Un valor positivo representa el coste de atravesar la casilla.
 - ▶ 0 indica que ya he pasado.
- ➋ Pasos válidos: Moverse derecha, abajo, izquierda y arriba si:
 - ▶ La casilla no es un obstáculo.
 - ▶ No he pasado por la casilla.
 - ▶ La casilla tiene un valor menor o igual que el anterior.
- ➌ Solución: Estoy en la casilla de salida.
- ➍ Coste: Suma de las casillas por las que he pasado.

Algoritmo *Backtracking*



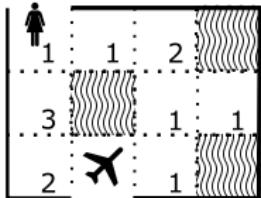
- ➊ Problema: El laberinto es una tabla de números.
 - ▶ -1 significa obstáculo.
 - ▶ Un valor positivo representa el coste de atravesar la casilla.
 - ▶ 0 indica que ya he pasado.
- ➋ Pasos válidos: Moverse derecha, abajo, izquierda y arriba si:
 - Sigo dentro del laberinto.
 - No estoy en un obstáculo.
 - No estoy pasando por la misma casilla más de una vez.
- ➌ Solución: Estoy en la casilla de salida.
- ➍ Coste: Suma de las casillas por las que he pasado.

Algoritmo *Backtracking*



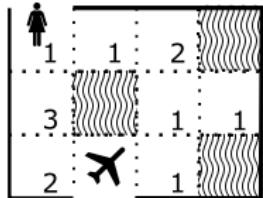
- ➊ Problema: El laberinto es una tabla de números.
 - ▶ -1 significa obstáculo.
 - ▶ Un valor positivo representa el coste de atravesar la casilla.
 - ▶ 0 indica que ya he pasado.
- ➋ Pasos válidos: Moverse derecha, abajo, izquierda y arriba si:
 - ▶ Sigo dentro del laberinto.
 - ▶ No me muevo a un obstáculo.
 - ▶ No es una posición explorada.
- ➌ Solución: Estoy en la casilla de salida.
- ➍ Coste: Suma de las casillas por las que he pasado.

Algoritmo *Backtracking*



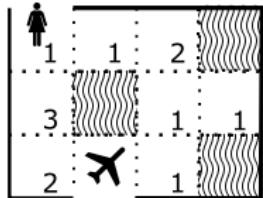
- ➊ Problema: El laberinto es una tabla de números.
 - ▶ -1 significa obstáculo.
 - ▶ Un valor positivo representa el coste de atravesar la casilla.
 - ▶ 0 indica que ya he pasado.
- ➋ Pasos válidos: Moverse derecha, abajo, izquierda y arriba si:
 - ▶ Sigo dentro del laberinto.
 - ▶ No me muevo a un obstáculo.
 - ▶ No es una posición explorada.
- ➌ Solución: Estoy en la casilla de salida.
- ➍ Coste: Suma de las casillas por las que he pasado.

Algoritmo *Backtracking*



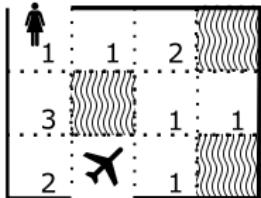
- ➊ Problema: El laberinto es una tabla de números.
 - ▶ -1 significa obstáculo.
 - ▶ Un valor positivo representa el coste de atravesar la casilla.
 - ▶ 0 indica que ya he pasado.
- ➋ Pasos válidos: Moverse derecha, abajo, izquierda y arriba si:
 - ▶ Sigo dentro del laberinto.
 - ▶ No me muevo a un obstáculo.
 - ▶ No es una posición explorada.
- ➌ Solución: Estoy en la casilla de salida.
- ➍ Coste: Suma de las casillas por las que he pasado.

Algoritmo *Backtracking*



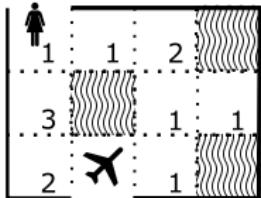
- ➊ Problema: El laberinto es una tabla de números.
 - ▶ -1 significa obstáculo.
 - ▶ Un valor positivo representa el coste de atravesar la casilla.
 - ▶ 0 indica que ya he pasado.
- ➋ Pasos válidos: Moverse derecha, abajo, izquierda y arriba si:
 - ▶ Sigo dentro del laberinto.
 - ▶ No me muevo a un obstáculo.
 - ▶ No es una posición explorada.
- ➌ Solución: Estoy en la casilla de salida.
- ➍ Coste: Suma de las casillas por las que he pasado.

Algoritmo *Backtracking*



- ➊ Problema: El laberinto es una tabla de números.
 - ▶ -1 significa obstáculo.
 - ▶ Un valor positivo representa el coste de atravesar la casilla.
 - ▶ 0 indica que ya he pasado.
- ➋ Pasos válidos: Moverse derecha, abajo, izquierda y arriba si:
 - ▶ Sigo dentro del laberinto.
 - ▶ No me muevo a un obstáculo.
 - ▶ No es una posición explorada.
- ➌ Solución: Estoy en la casilla de salida.
- ➍ Coste: Suma de las casillas por las que he pasado.

Algoritmo *Backtracking*



- ➊ Problema: El laberinto es una tabla de números.
 - ▶ -1 significa obstáculo.
 - ▶ Un valor positivo representa el coste de atravesar la casilla.
 - ▶ 0 indica que ya he pasado.
- ➋ Pasos válidos: Moverse derecha, abajo, izquierda y arriba si:
 - ▶ Sigo dentro del laberinto.
 - ▶ No me muevo a un obstáculo.
 - ▶ No es una posición explorada.
- ➌ Solución: Estoy en la casilla de salida.
- ➍ Coste: Suma de las casillas por las que he pasado.

Algoritmo *Backtracking*

- ➊ Si soy solución, calculo el coste y lo propago.
- ➋ Si no, si hay algún movimiento libre, lo genero y lo exploro.

Si el movimiento es válido, lo agrego a la lista de soluciones y genero un movimiento como solución.

Si el movimiento es inválido, lo elimino de la lista de soluciones y genero otro movimiento como solución.

- ➌ Si no hay más movimientos libres, *vuelvo a atrás* y propago el valor de la solución.

Algoritmo *Backtracking*

- ➊ Si soy solución, calculo el coste y lo propago.
- ➋ Si no, si hay algún movimiento libre, lo genero y lo exploro.
 - ▶ Si el movimiento lleva a una solución y no había solución, anoto el movimiento como solución.
 - ▶ Si el movimiento lleva a una solución, había solución y la mejora, anoto el movimiento como solución.
 - ▶ Si el movimiento lleva a una solución, había solución y no la mejora, descarto el movimiento.
- ➌ Si no hay más movimientos libres, *vuelvo a atrás* y propago el valor de la solución.

Algoritmo *Backtracking*

- ➊ Si soy solución, calculo el coste y lo propago.
- ➋ Si no, si hay algún movimiento libre, lo genero y lo exploro.
 - ▶ Si el movimiento lleva a una solución y no había solución, anoto el movimiento como solución.
 - ▶ Si el movimiento lleva a una solución, había solución y la mejora, anoto el movimiento como solución.
 - ▶ Si el movimiento lleva a una solución, había solución y no la mejora, descarto el movimiento.
- ➌ Si no hay más movimientos libres, *vuelvo a atrás* y propago el valor de la solución.

Algoritmo *Backtracking*

- ➊ Si soy solución, calculo el coste y lo propago.
- ➋ Si no, si hay algún movimiento libre, lo genero y lo exploro.
 - ▶ Si el movimiento lleva a una solución y no había solución, anoto el movimiento como solución.
 - ▶ Si el movimiento lleva a una solución, había solución y la mejora, anoto el movimiento como solución.
 - ▶ Si el movimiento lleva a una solución, había solución y no la mejora, descarto el movimiento.
- ➌ Si no hay más movimientos libres, *vuelvo a atrás* y propago el valor de la solución.

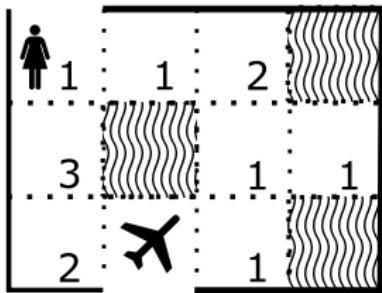
Algoritmo *Backtracking*

- ➊ Si soy solución, calculo el coste y lo propago.
- ➋ Si no, si hay algún movimiento libre, lo genero y lo exploro.
 - ▶ Si el movimiento lleva a una solución y no había solución, anoto el movimiento como solución.
 - ▶ Si el movimiento lleva a una solución, había solución y la mejora, anoto el movimiento como solución.
 - ▶ Si el movimiento lleva a una solución, había solución y no la mejora, descarto el movimiento.
- ➌ Si no hay más movimientos libres, *vuelvo a atrás* y propago el valor de la solución.

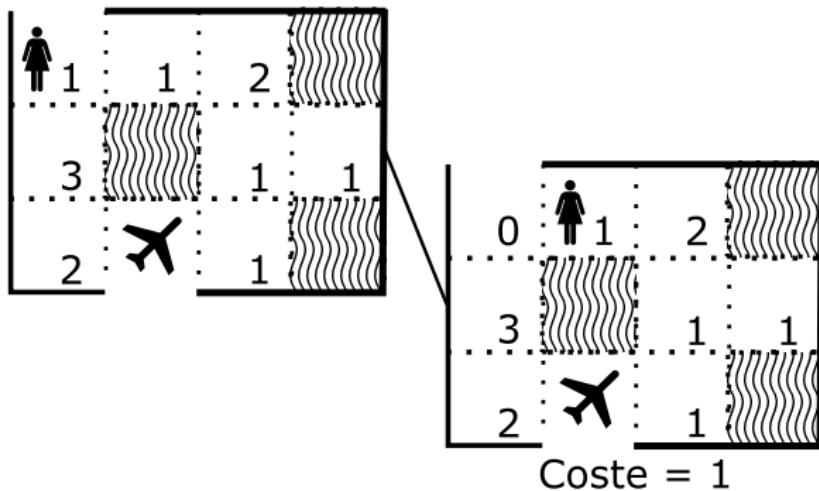
Algoritmo *Backtracking*

- ➊ Si soy solución, calculo el coste y lo propago.
- ➋ Si no, si hay algún movimiento libre, lo genero y lo exploro.
 - ▶ Si el movimiento lleva a una solución y no había solución, anoto el movimiento como solución.
 - ▶ Si el movimiento lleva a una solución, había solución y la mejora, anoto el movimiento como solución.
 - ▶ Si el movimiento lleva a una solución, había solución y no la mejora, descarto el movimiento.
- ➌ Si no hay más movimientos libres, *vuelvo a atrás* y propago el valor de la solución.

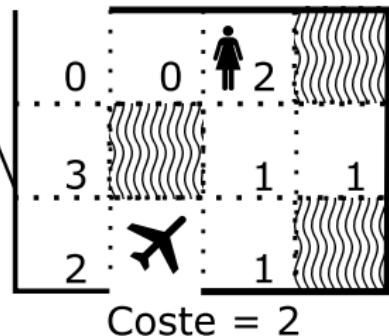
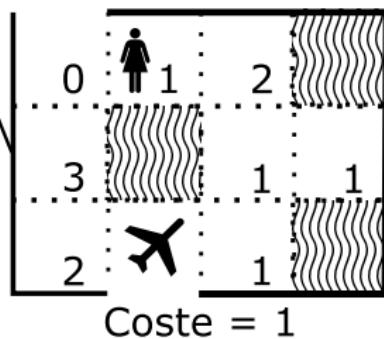
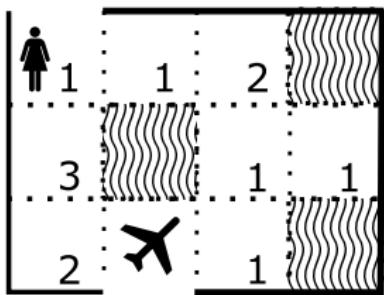
Algoritmo *Backtracking*



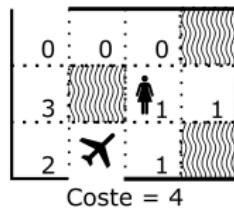
Algoritmo *Backtracking*



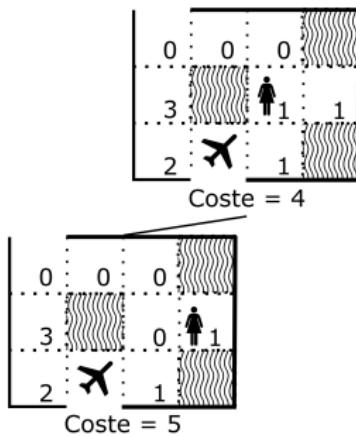
Algoritmo *Backtracking*



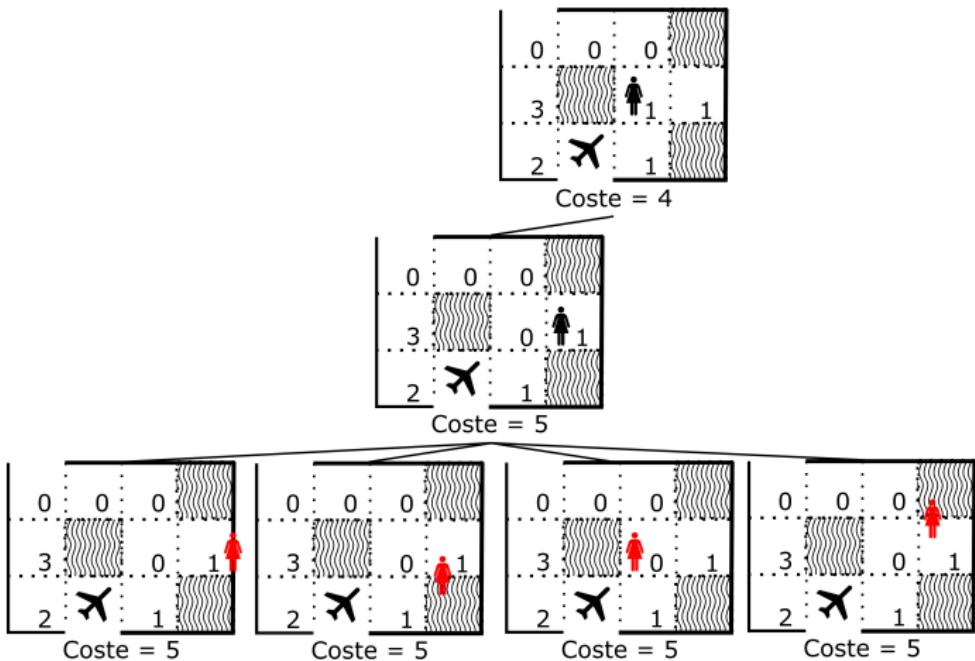
Algoritmo *Backtracking*



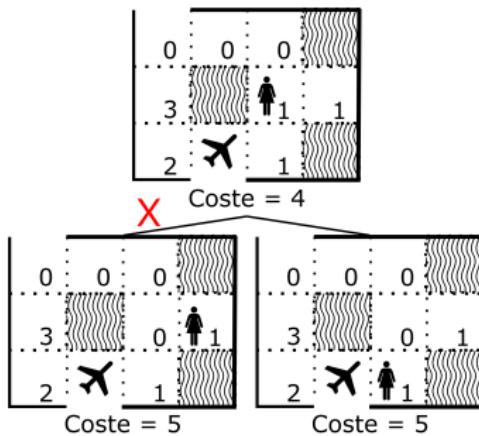
Algoritmo *Backtracking*



Algoritmo *Backtracking*



Algoritmo *Backtracking*



Índice

- 1 Introducción
- 2 Juegos Básicos: Algoritmo Minimax
- 3 Backtracking y Ramificación y Poda
 - Backtracking
 - Ramificación y Poda
- 4 Ajedrez por Computador
- 5 Parchís por Computador: Redes Neuronales
- 6 Conclusiones

Algoritmo Ramificación y Poda

Algoritmo Ramificación y Poda

① Versión de *backtracking* donde:

- Expando todos los movimientos válidos de un nodo.
- Evalúo lo prometedor de cada nodo.
- Exploro el nodo más prometedor.
- No exploro los movimientos que no pueden alcanzar una solución óptima.

② Para evaluar lo prometedor, utilizo *heurísticas*.

Algoritmo Ramificación y Poda

Algoritmo Ramificación y Poda

① Versión de *backtracking* donde:

- ▶ Expando todos los movimientos válidos de un nodo.
 - Evalúo lo prometedor de cada nodo.
 - Exploro el nodo más prometedor.
- No exploro los movimientos que no pueden alcanzar una solución óptima.

② Para evaluar lo prometedor, utilizo *heurísticas*.

Algoritmo Ramificación y Poda

Algoritmo Ramificación y Poda

① Versión de *backtracking* donde:

- ▶ Expando todos los movimientos válidos de un nodo.
- ▶ Evalúo lo prometedor de cada nodo.
- ▶ Exploro el nodo más prometedor.
- ▶ No exploro los movimientos que no pueden alcanzar una solución óptima.

② Para evaluar lo prometedor, utilizo *heurísticas*.

Algoritmo Ramificación y Poda

Algoritmo Ramificación y Poda

① Versión de *backtracking* donde:

- ▶ Expando todos los movimientos válidos de un nodo.
- ▶ Evalúo lo prometedor de cada nodo.
- ▶ Exploro el nodo más prometedor.
- ▶ No exploro los movimientos que no pueden alcanzar una solución óptima.

② Para evaluar lo prometedor, utilizo *heurísticas*.

Algoritmo Ramificación y Poda

Algoritmo Ramificación y Poda

① Versión de *backtracking* donde:

- ▶ Expando todos los movimientos válidos de un nodo.
- ▶ Evalúo lo prometedor de cada nodo.
- ▶ Exploro el nodo más prometedor.
- ▶ No exploro los movimientos que no pueden alcanzar una solución óptima.

② Para evaluar lo prometedor, utilizo *heurísticas*.

Algoritmo Ramificación y Poda

Algoritmo Ramificación y Poda

① Versión de *backtracking* donde:

- ▶ Expando todos los movimientos válidos de un nodo.
- ▶ Evalúo lo prometedor de cada nodo.
- ▶ Exploro el nodo más prometedor.
- ▶ No exploro los movimientos que no pueden alcanzar una solución óptima.

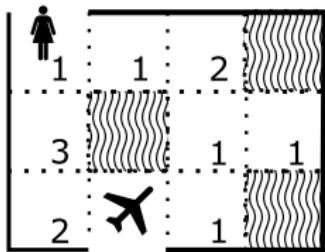
② Para evaluar lo prometedor, utilizo *heurísticas*.

Heurísticas

Heurística Ramificación y Poda

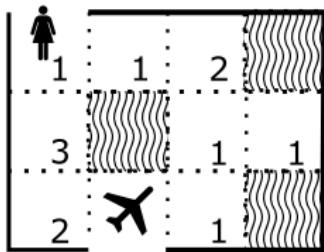
Función que devuelve una cota inferior y optimista del coste que me queda para alcanzar la solución.

Ejemplo de Heurística



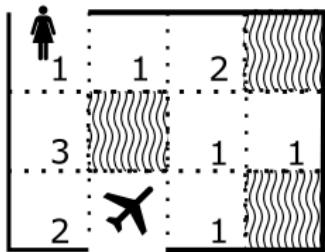
- ➊ Tengo que salir de la casilla actual.
- ➋ Me puedo mover hasta la salida en movimientos descendentes y laterales sin obstáculos.
- ➌ Todas las casillas por las que paso tienen valor 1.

Ejemplo de Heurística



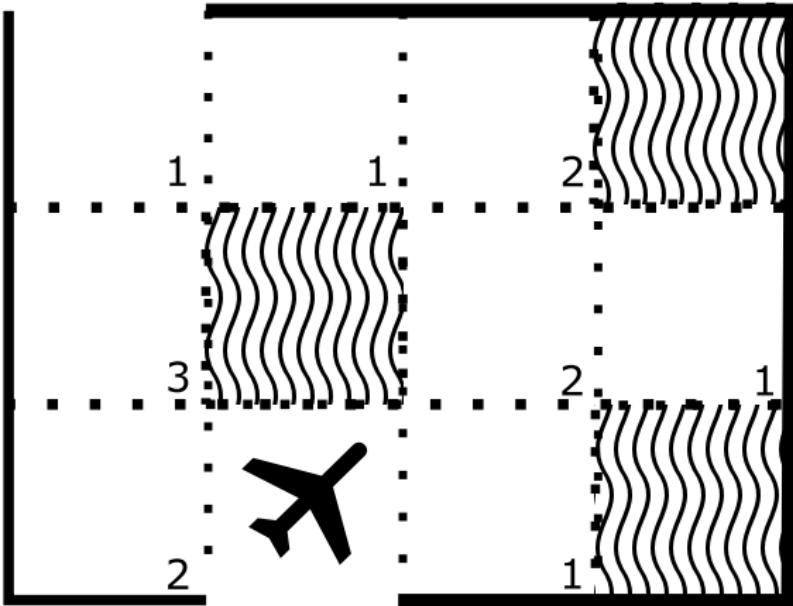
- ① Tengo que salir de la casilla actual.
- ② Me puedo mover hasta la salida en movimientos descendentes y laterales sin obstáculos.
- ③ Todas las casillas por las que paso tienen valor 1.

Ejemplo de Heurística

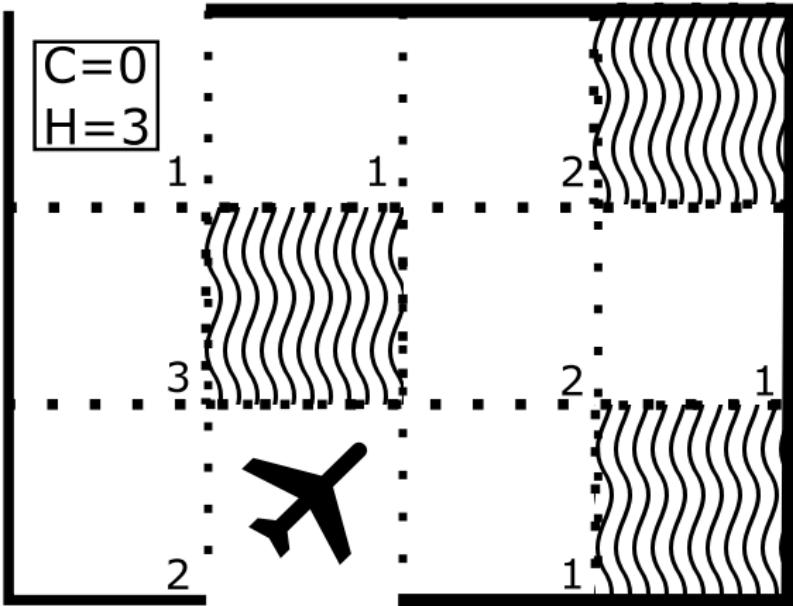


- ➊ Tengo que salir de la casilla actual.
- ➋ Me puedo mover hasta la salida en movimientos descendentes y laterales sin obstáculos.
- ➌ Todas las casillas por las que paso tienen valor 1.

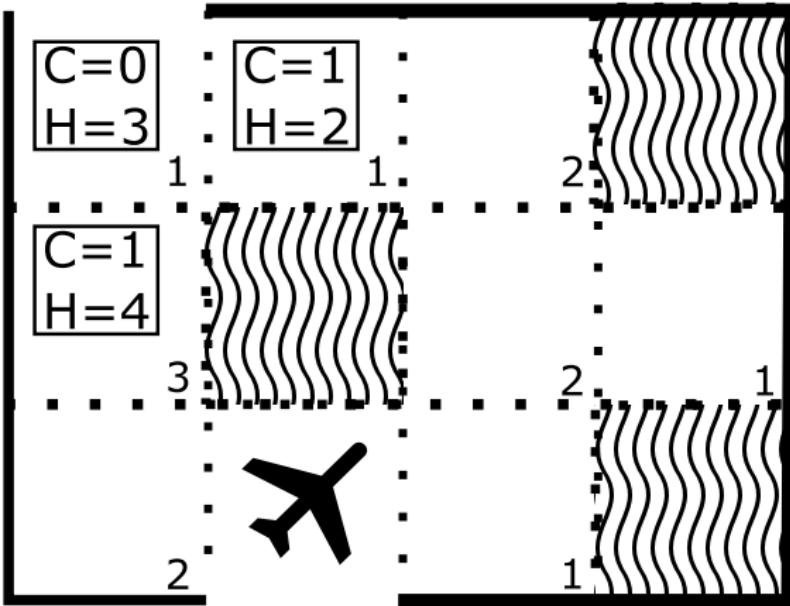
Algoritmo Ramificación y Poda



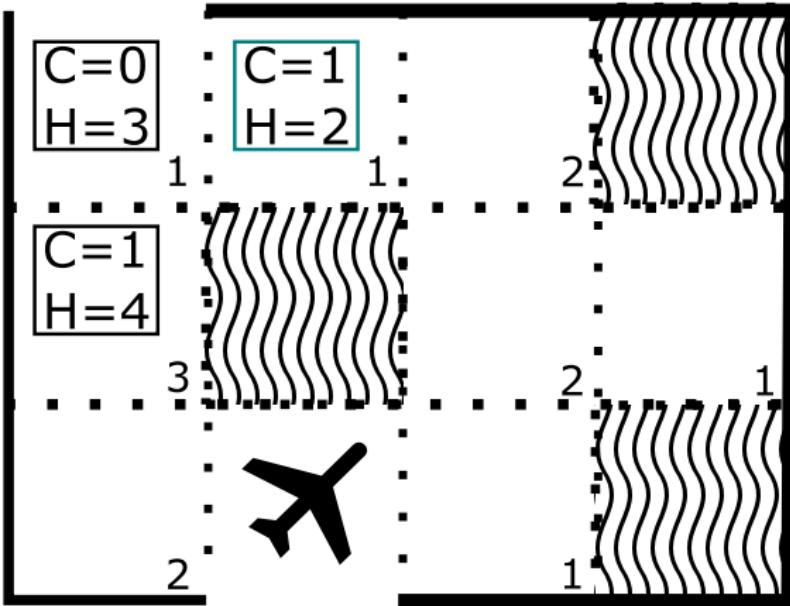
Algoritmo Ramificación y Poda



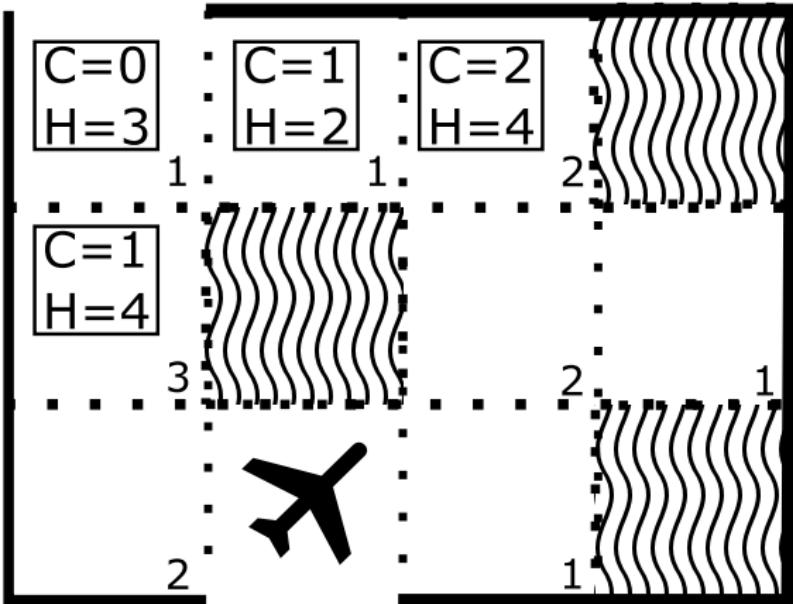
Algoritmo Ramificación y Poda



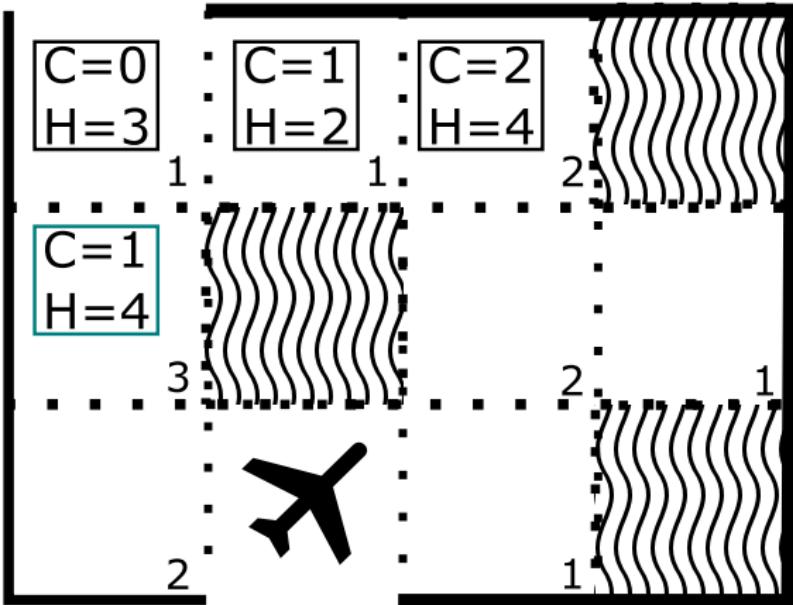
Algoritmo Ramificación y Poda



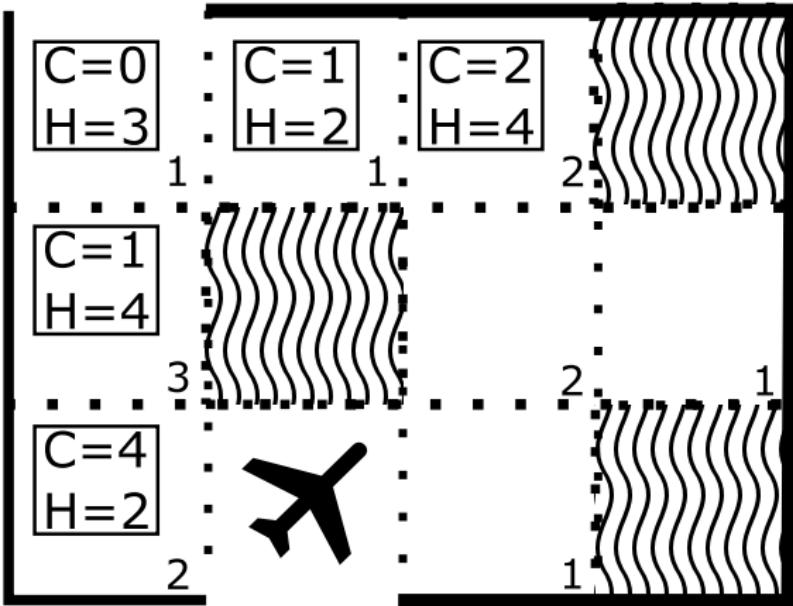
Algoritmo Ramificación y Poda



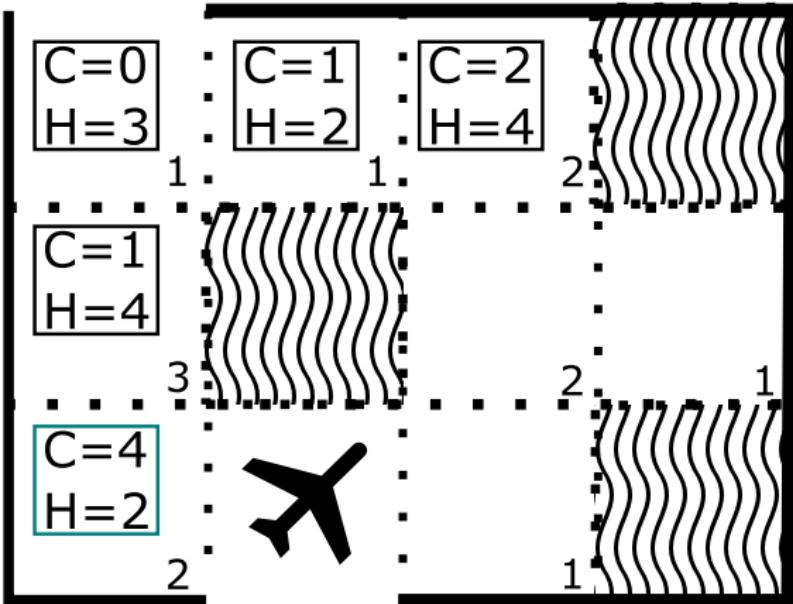
Algoritmo Ramificación y Poda



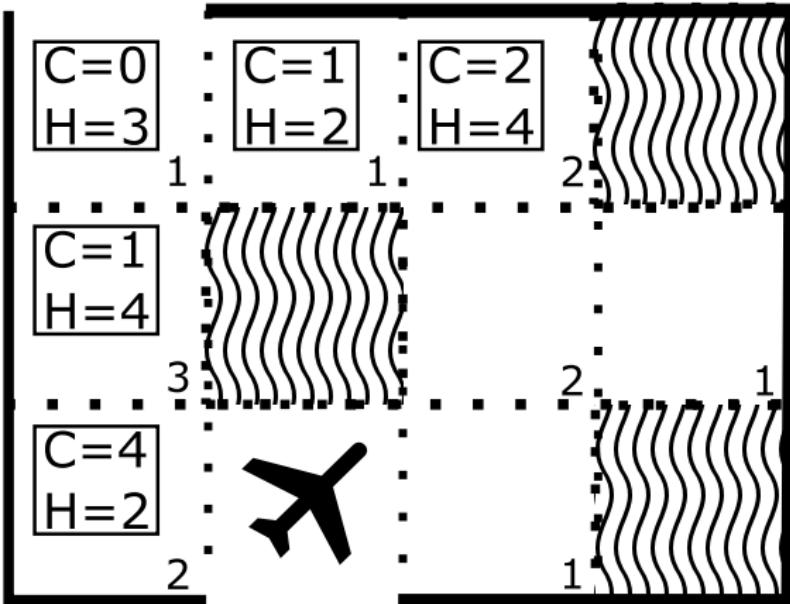
Algoritmo Ramificación y Poda



Algoritmo Ramificación y Poda

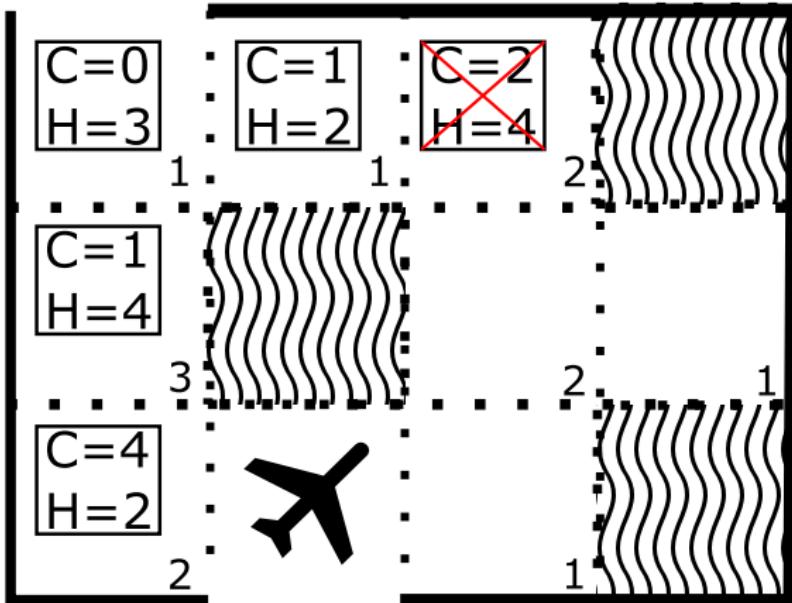


Algoritmo Ramificación y Poda



Solución = 6

Algoritmo Ramificación y Poda



Solución = 6

Índice

- 1 Introducción
- 2 Juegos Básicos: Algoritmo Minimax
- 3 Backtracking y Ramificación y Poda
- 4 Ajedrez por Computador
- 5 Parchís por Computador: Redes Neuronales
- 6 Conclusiones

Algoritmos para Jugar al Ajedrez

- ➊ Árbol de partidas de ajedrez tiene tamaño inabordable (actualmente).
- ➋ Se acotan los niveles de profundidad y tiempo de exploración.
- ➌ Se aplican heurísticas y se poda con cautela.
- ➍ Se generan 6 niveles o más de profundidad.
- ➎ Procesan entre 10.000-100.000 jugadas por segundo.
- ➏ Usan *endgame tablebases* y *opening books*.
- ➐ Software libre de ajedrez, como Stockfish, es capaz de batir a campeones mundiales.

Algoritmos para Jugar al Ajedrez

- ➊ Árbol de partidas de ajedrez tiene tamaño inabordable (actualmente).
- ➋ Se acotan los niveles de profundidad y tiempo de exploración.
- ➌ Se aplican heurísticas y se poda con cautela.
- ➍ Se generan 6 niveles o más de profundidad.
- ➎ Procesan entre 10.000-100.000 jugadas por segundo.
- ➏ Usan *endgame tablebases* y *opening books*.
- ➐ Software libre de ajedrez, como Stockfish, es capaz de batir a campeones mundiales.

Algoritmos para Jugar al Ajedrez

- ➊ Árbol de partidas de ajedrez tiene tamaño inabordable (actualmente).
- ➋ Se acotan los niveles de profundidad y tiempo de exploración.
- ➌ Se aplican heurísticas y se poda con cautela.
- ➍ Se generan 6 niveles o más de profundidad.
- ➎ Procesan entre 10.000-100.000 jugadas por segundo.
- ➏ Usan *endgame tablebases* y *opening books*.
- ➐ Software libre de ajedrez, como Stockfish, es capaz de batir a campeones mundiales.

Algoritmos para Jugar al Ajedrez

- ➊ Árbol de partidas de ajedrez tiene tamaño inabordable (actualmente).
- ➋ Se acotan los niveles de profundidad y tiempo de exploración.
- ➌ Se aplican heurísticas y se poda con cautela.
- ➍ Se generan 6 niveles o más de profundidad.
- ➎ Procesan entre 10.000-100.000 jugadas por segundo.
- ➏ Usan *endgame tablebases* y *opening books*.
- ➐ Software libre de ajedrez, como Stockfish, es capaz de batir a campeones mundiales.

Algoritmos para Jugar al Ajedrez

- ➊ Árbol de partidas de ajedrez tiene tamaño inabordable (actualmente).
- ➋ Se acotan los niveles de profundidad y tiempo de exploración.
- ➌ Se aplican heurísticas y se poda con cautela.
- ➍ Se generan 6 niveles o más de profundidad.
- ➎ Procesan entre 10.000-100.000 jugadas por segundo.
- ➏ Usan *endgame tablebases* y *opening books*.
- ➐ Software libre de ajedrez, como Stockfish, es capaz de batir a campeones mundiales.

Algoritmos para Jugar al Ajedrez

- ➊ Árbol de partidas de ajedrez tiene tamaño inabordable (actualmente).
- ➋ Se acotan los niveles de profundidad y tiempo de exploración.
- ➌ Se aplican heurísticas y se poda con cautela.
- ➍ Se generan 6 niveles o más de profundidad.
- ➎ Procesan entre 10.000-100.000 jugadas por segundo.
- ➏ Usan *endgame tablebases* y *opening books*.
- ➐ Software libre de ajedrez, como Stockfish, es capaz de batir a campeones mundiales.

Algoritmos para Jugar al Ajedrez

- ① Árbol de partidas de ajedrez tiene tamaño inabordable (actualmente).
- ② Se acotan los niveles de profundidad y tiempo de exploración.
- ③ Se aplican heurísticas y se poda con cautela.
- ④ Se generan 6 niveles o más de profundidad.
- ⑤ Procesan entre 10.000-100.000 jugadas por segundo.
- ⑥ Usan *endgame tablebases* y *opening books*.
- ⑦ Software libre de ajedrez, como Stockfish, es capaz de batir a campeones mundiales.

Historia de los Algoritmos para Jugar al Ajedrez

- ➊ 1957: Se inventa la *poda alfa-beta*.
- ➋ 1976: Un computador gana su primer torneo.
- ➌ 1989: *DeepThought* gana al gran maestro *Ben Larsen*.
- ➍ 1996: *DeepBlue* gana una partida a *Garry Kasparov*.
- ➎ 1997: *DeepBlue* derrota a *Garry Kasparov*, 3.5 a 2.5.
- ➏ 1998: *Rebel 10* derrota a *Viswanathan Anand*, 5 a 3.
- ➐ 2002: *Vladimir Kramnik vs Deep Fritz*: tácticas anticomputador.
- ➑ 2006: *Vladimir Kramnik vs Deep Fritz*: *the science is done*.
- ➒ 2009: Un teléfono móvil gana un torneo de sexta categoría.
- ➓ 2017: *AlphaZero* derrota a *Stockfish*.

Historia de los Algoritmos para Jugar al Ajedrez

- ➊ 1957: Se inventa la *poda alfa-beta*.
- ➋ 1976: Un computador gana su primer torneo.
- ➌ 1989: *DeepThought* gana al gran maestro *Ben Larsen*.
- ➍ 1996: *DeepBlue* gana una partida a *Garry Kasparov*.
- ➎ 1997: *DeepBlue* derrota a *Garry Kasparov*, 3.5 a 2.5.
- ➏ 1998: *Rebel 10* derrota a *Viswanathan Anand*, 5 a 3.
- ➐ 2002: *Vladimir Kramnik vs Deep Fritz*: tácticas anticomputador.
- ➑ 2006: *Vladimir Kramnik vs Deep Fritz*: *the science is done*.
- ➒ 2009: Un teléfono móvil gana un torneo de sexta categoría.
- ➓ 2017: *AlphaZero* derrota a *Stockfish*.

Historia de los Algoritmos para Jugar al Ajedrez

- ➊ 1957: Se inventa la *poda alfa-beta*.
- ➋ 1976: Un computador gana su primer torneo.
- ➌ 1989: *DeepThought* gana al gran maestro *Ben Larsen*.
- ➍ 1996: *DeepBlue* gana una partida a *Garry Kasparov*.
- ➎ 1997: *DeepBlue* derrota a *Garry Kasparov*, 3.5 a 2.5.
- ➏ 1998: *Rebel 10* derrota a *Viswanathan Anand*, 5 a 3.
- ➐ 2002: *Vladimir Kramnik vs Deep Fritz*: tácticas anticomputador.
- ➑ 2006: *Vladimir Kramnik vs Deep Fritz*: *the science is done*.
- ➒ 2009: Un teléfono móvil gana un torneo de sexta categoría.
- ➓ 2017: *AlphaZero* derrota a *Stockfish*.

Historia de los Algoritmos para Jugar al Ajedrez

- ➊ 1957: Se inventa la *poda alfa-beta*.
- ➋ 1976: Un computador gana su primer torneo.
- ➌ 1989: *DeepThought* gana al gran maestro *Ben Larsen*.
- ➍ 1996: *DeepBlue* gana una partida a *Garry Kasparov*.
- ➎ 1997: *DeepBlue* derrota a *Garry Kasparov*, 3.5 a 2.5.
- ➏ 1998: *Rebel 10* derrota a *Viswanathan Anand*, 5 a 3.
- ➐ 2002: *Vladimir Kramnik vs Deep Fritz*: tácticas anticomputador.
- ➑ 2006: *Vladimir Kramnik vs Deep Fritz*: *the science is done*.
- ➒ 2009: Un teléfono móvil gana un torneo de sexta categoría.
- ➓ 2017: *AlphaZero* derrota a *Stockfish*.

Historia de los Algoritmos para Jugar al Ajedrez

- ➊ 1957: Se inventa la *poda alfa-beta*.
- ➋ 1976: Un computador gana su primer torneo.
- ➌ 1989: *DeepThought* gana al gran maestro *Ben Larsen*.
- ➍ 1996: *DeepBlue* gana una partida a *Garry Kasparov*.
- ➎ 1997: *DeepBlue* derrota a *Garry Kasparov*, 3.5 a 2.5.
- ➏ 1998: *Rebel 10* derrota a *Viswanathan Anand*, 5 a 3.
- ➐ 2002: *Vladimir Kramnik vs Deep Fritz*: tácticas anticomputador.
- ➑ 2006: *Vladimir Kramnik vs Deep Fritz*: *the science is done*.
- ➒ 2009: Un teléfono móvil gana un torneo de sexta categoría.
- ➓ 2017: *AlphaZero* derrota a *Stockfish*.

Historia de los Algoritmos para Jugar al Ajedrez

- ➊ 1957: Se inventa la *poda alfa-beta*.
- ➋ 1976: Un computador gana su primer torneo.
- ➌ 1989: *DeepThought* gana al gran maestro *Ben Larsen*.
- ➍ 1996: *DeepBlue* gana una partida a *Garry Kasparov*.
- ➎ 1997: *DeepBlue* derrota a *Garry Kasparov*, 3.5 a 2.5.
- ➏ 1998: *Rebel 10* derrota a *Viswanathan Anand*, 5 a 3.
- ➐ 2002: *Vladimir Kramnik vs Deep Fritz*: tácticas anticomputador.
- ➑ 2006: *Vladimir Kramnik vs Deep Fritz*: *the science is done*.
- ➒ 2009: Un teléfono móvil gana un torneo de sexta categoría.
- ➓ 2017: *AlphaZero* derrota a *Stockfish*.

Historia de los Algoritmos para Jugar al Ajedrez

- ① 1957: Se inventa la *poda alfa-beta*.
- ② 1976: Un computador gana su primer torneo.
- ③ 1989: *DeepThought* gana al gran maestro *Ben Larsen*.
- ④ 1996: *DeepBlue* gana una partida a *Garry Kasparov*.
- ⑤ 1997: *DeepBlue* derrota a *Garry Kasparov*, 3.5 a 2.5.
- ⑥ 1998: *Rebel 10* derrota a *Viswanathan Anand*, 5 a 3.
- ⑦ 2002: *Vladimir Kramnik vs Deep Fritz*: tácticas anticomputador.
- ⑧ 2006: *Vladimir Kramnik vs Deep Fritz*: *the science is done*.
- ⑨ 2009: Un teléfono móvil gana un torneo de sexta categoría.
- ⑩ 2017: *AlphaZero* derrota a *Stockfish*.

Historia de los Algoritmos para Jugar al Ajedrez

- ① 1957: Se inventa la *poda alfa-beta*.
- ② 1976: Un computador gana su primer torneo.
- ③ 1989: *DeepThought* gana al gran maestro *Ben Larsen*.
- ④ 1996: *DeepBlue* gana una partida a *Garry Kasparov*.
- ⑤ 1997: *DeepBlue* derrota a *Garry Kasparov*, 3.5 a 2.5.
- ⑥ 1998: *Rebel 10* derrota a *Viswanathan Anand*, 5 a 3.
- ⑦ 2002: *Vladimir Kramnik vs Deep Fritz*: tácticas anticomputador.
- ⑧ 2006: *Vladimir Kramnik vs Deep Fritz*: *the science is done*.
- ⑨ 2009: Un teléfono móvil gana un torneo de sexta categoría.
- ⑩ 2017: *AlphaZero* derrota a *Stockfish*.

Historia de los Algoritmos para Jugar al Ajedrez

- ① 1957: Se inventa la *poda alfa-beta*.
- ② 1976: Un computador gana su primer torneo.
- ③ 1989: *DeepThought* gana al gran maestro *Ben Larsen*.
- ④ 1996: *DeepBlue* gana una partida a *Garry Kasparov*.
- ⑤ 1997: *DeepBlue* derrota a *Garry Kasparov*, 3.5 a 2.5.
- ⑥ 1998: *Rebel 10* derrota a *Viswanathan Anand*, 5 a 3.
- ⑦ 2002: *Vladimir Kramnik vs Deep Fritz*: tácticas anticomputador.
- ⑧ 2006: *Vladimir Kramnik vs Deep Fritz*: *the science is done*.
- ⑨ 2009: Un teléfono móvil gana un torneo de sexta categoría.
- ⑩ 2017: *AlphaZero* derrota a *Stockfish*.

Historia de los Algoritmos para Jugar al Ajedrez

- ① 1957: Se inventa la *poda alfa-beta*.
- ② 1976: Un computador gana su primer torneo.
- ③ 1989: *DeepThought* gana al gran maestro *Ben Larsen*.
- ④ 1996: *DeepBlue* gana una partida a *Garry Kasparov*.
- ⑤ 1997: *DeepBlue* derrota a *Garry Kasparov*, 3.5 a 2.5.
- ⑥ 1998: *Rebel 10* derrota a *Viswanathan Anand*, 5 a 3.
- ⑦ 2002: *Vladimir Kramnik vs Deep Fritz*: tácticas anticomputador.
- ⑧ 2006: *Vladimir Kramnik vs Deep Fritz*: *the science is done*.
- ⑨ 2009: Un teléfono móvil gana un torneo de sexta categoría.
- ⑩ 2017: *AlphaZero* derrota a *Stockfish*.

Índice

- 1 Introducción
- 2 Juegos Básicos: Algoritmo Minimax
- 3 Backtracking y Ramificación y Poda
- 4 Ajedrez por Computador
- 5 Parchís por Computador: Redes Neuronales
- 6 Conclusiones

Índice

- 1 Introducción
- 2 Juegos Básicos: Algoritmo Minimax
- 3 Backtracking y Ramificación y Poda
- 4 Ajedrez por Computador
- 5 Parchís por Computador: Redes Neuronales
 - Aplicando minimax al parchís
 - Redes Neuronales
- 6 Conclusiones

Parchís con Minimax

① Árbol de profundidad infinita:

- ▶ Nadie sale nunca.
- ▶ Nadie entra nunca.
- ▶ Salgo, avanco un poco, tres seis y vuelta a casa.
- ▶ Nos comemos mutuamente de manera indefinida.
- ▶ Barrera que nunca se abre.

② El jugador contrario no escoge movimiento libremente, depende del dado.

③ Tres objetivos: avanzar, no ser comido y comer.

Parchís con Minimax

① Árbol de profundidad infinita:

- ▶ Nadie sale nunca.
- ▶ Nadie entra nunca.
- ▶ Salgo, avanco un poco, tres seis y vuelta a casa.
- ▶ Nos comemos mutuamente de manera indefinida.
- ▶ Barrera que nunca se abre.

② El jugador contrario no escoge movimiento libremente, depende del dado.

③ Tres objetivos: avanzar, no ser comido y comer.

Parchís con Minimax

① Árbol de profundidad infinita:

- ▶ Nadie sale nunca.
- ▶ Nadie entra nunca.
- ▶ Salgo, avanco un poco, tres seis y vuelta a casa.
- ▶ Nos comemos mutuamente de manera indefinida.
- ▶ Barrera que nunca se abre.

② El jugador contrario no escoge movimiento libremente, depende del dado.

③ Tres objetivos: avanzar, no ser comido y comer.

Parchís con Minimax

① Árbol de profundidad infinita:

- ▶ Nadie sale nunca.
- ▶ Nadie entra nunca.
- ▶ Salgo, avanco un poco, tres seis y vuelta a casa.
- ▶ Nos comemos mutuamente de manera indefinida.
- ▶ Barrera que nunca se abre.

② El jugador contrario no escoge movimiento libremente, depende del dado.

③ Tres objetivos: avanzar, no ser comido y comer.

Parchís con Minimax

① Árbol de profundidad infinita:

- ▶ Nadie sale nunca.
- ▶ Nadie entra nunca.
- ▶ Salgo, avanco un poco, tres seis y vuelta a casa.
- ▶ Nos comemos mutuamente de manera indefinida.
- ▶ Barrera que nunca se abre.

② El jugador contrario no escoge movimiento libremente, depende del dado.

③ Tres objetivos: avanzar, no ser comido y comer.

Parchís con Minimax

① Árbol de profundidad infinita:

- ▶ Nadie sale nunca.
- ▶ Nadie entra nunca.
- ▶ Salgo, avanco un poco, tres seis y vuelta a casa.
- ▶ Nos comemos mutuamente de manera indefinida.
- ▶ Barrera que nunca se abre.

② El jugador contrario no escoge movimiento libremente, depende del dado.

③ Tres objetivos: avanzar, no ser comido y comer.

Parchís con Minimax

1 Árbol de profundidad infinita:

- ▶ Nadie sale nunca.
- ▶ Nadie entra nunca.
- ▶ Salgo, avanco un poco, tres seis y vuelta a casa.
- ▶ Nos comemos mutuamente de manera indefinida.
- ▶ Barrera que nunca se abre.

2 El jugador contrario no escoge movimiento libremente, depende del dado.

3 Tres objetivos: avanzar, no ser comido y comer.

Parchís con Minimax

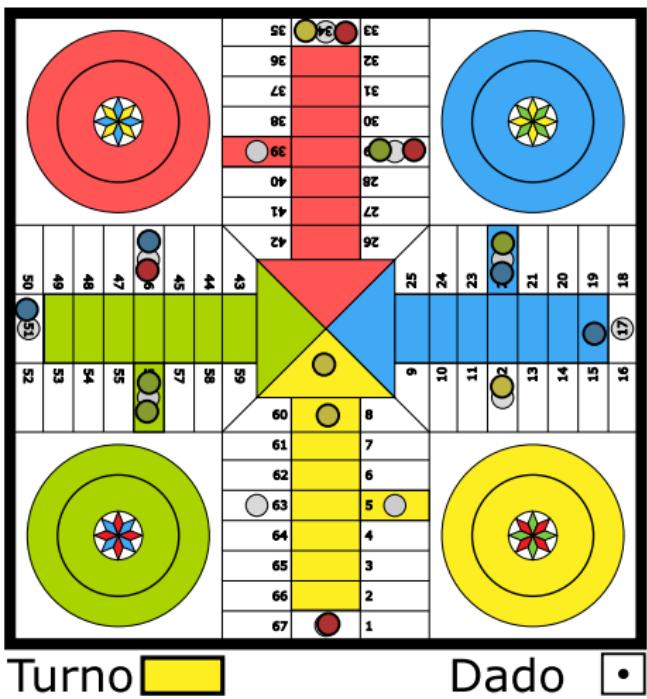
1 Árbol de profundidad infinita:

- ▶ Nadie sale nunca.
- ▶ Nadie entra nunca.
- ▶ Salgo, avanco un poco, tres seis y vuelta a casa.
- ▶ Nos comemos mutuamente de manera indefinida.
- ▶ Barrera que nunca se abre.

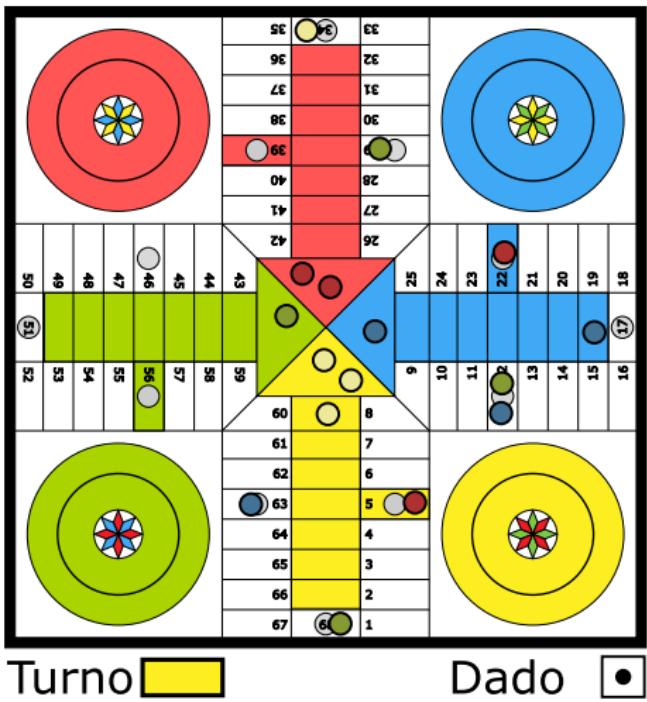
2 El jugador contrario no escoge movimiento libremente, depende del dado.

3 Tres objetivos: avanzar, no ser comido y comer.

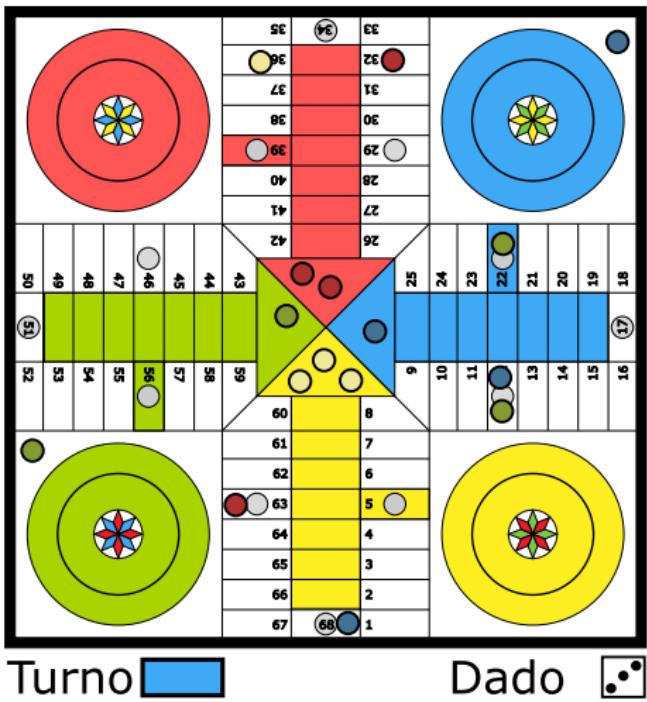
Jugadas Curiosas I: Ahorro de Saldo



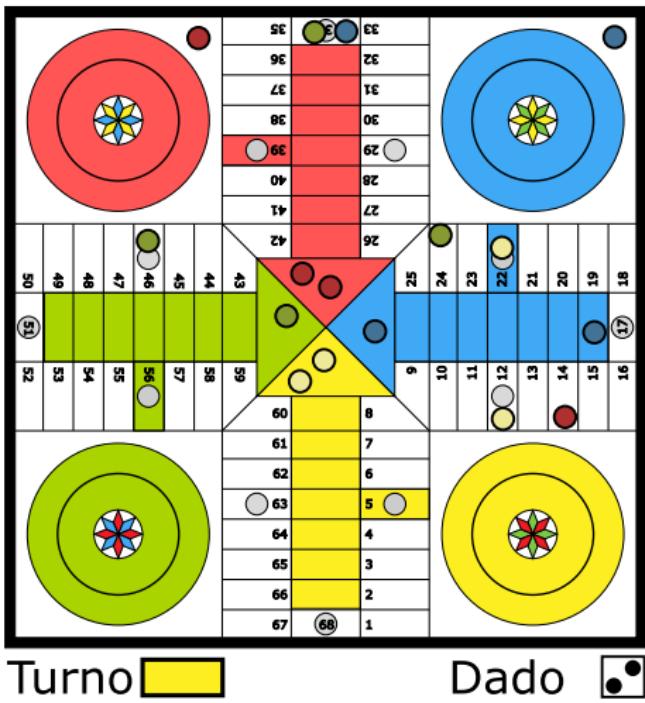
Jugadas Curiosas II: Evitar Ficha Única



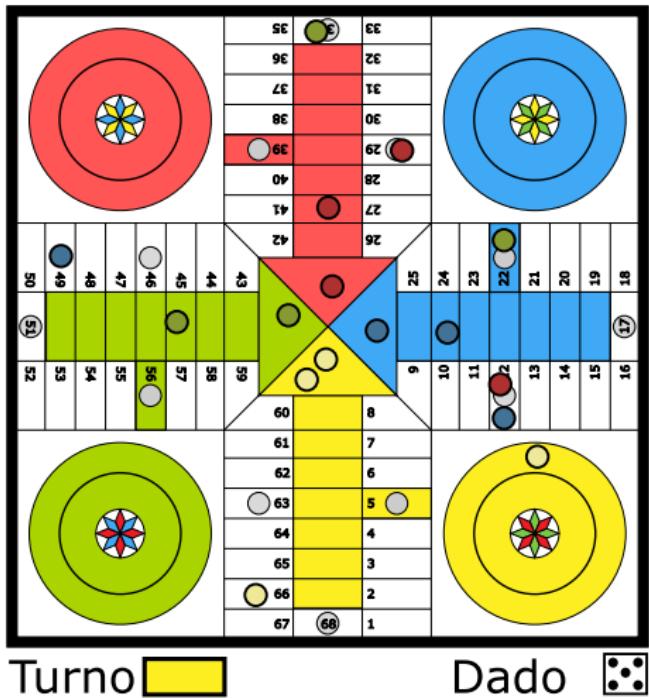
Jugadas Curiosas III: Bien Común



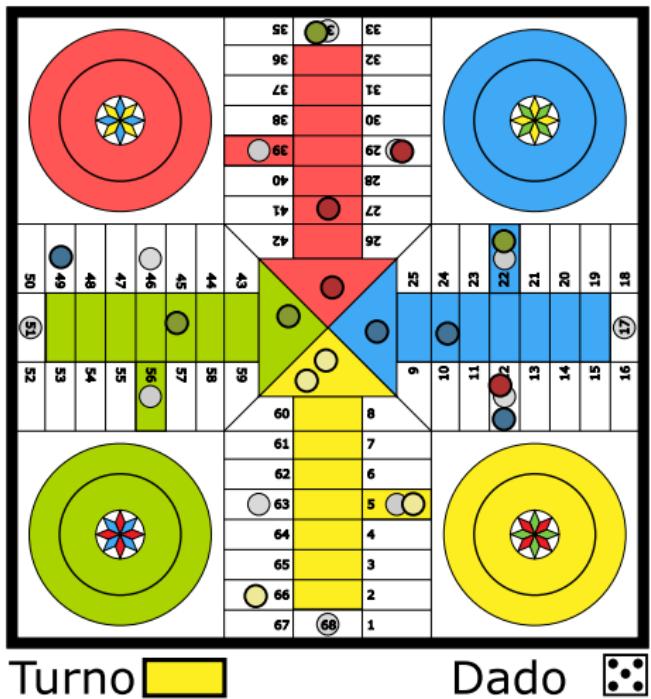
Jugadas Curiosas IV: Cantos de Sirenas



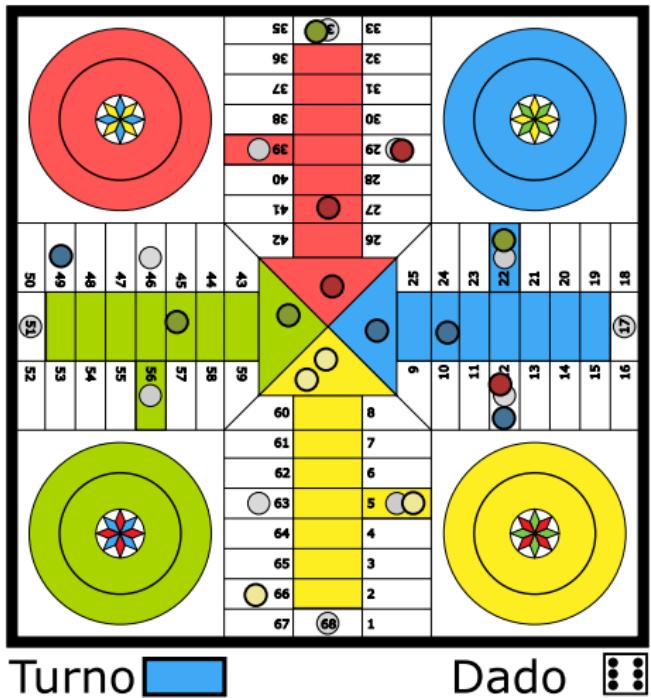
Jugadas Curiosas V: Retirada de Augusto



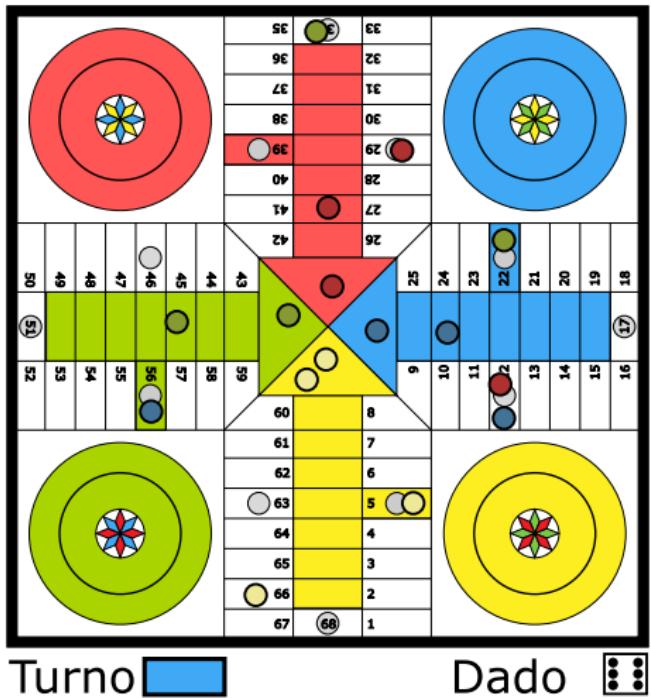
Jugadas Curiosas V: Retirada de Augusto



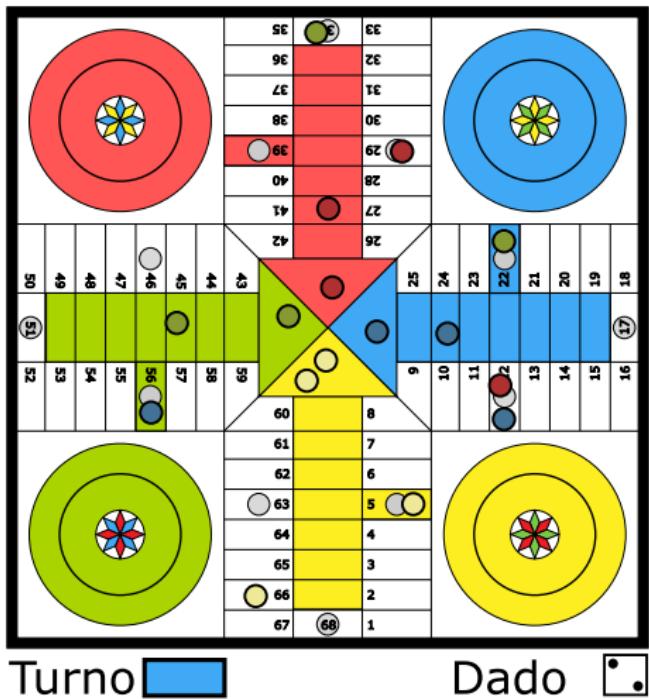
Jugadas Curiosas V: Retirada de Augusto



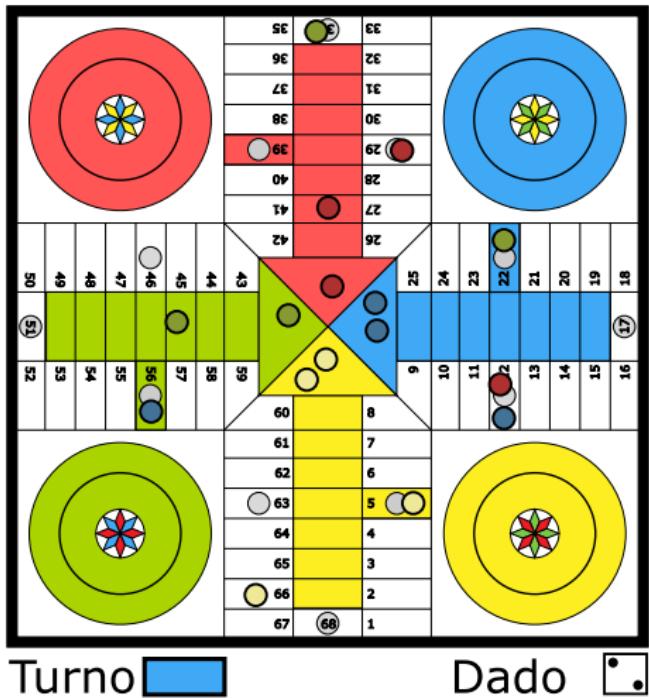
Jugadas Curiosas V: Retirada de Augusto



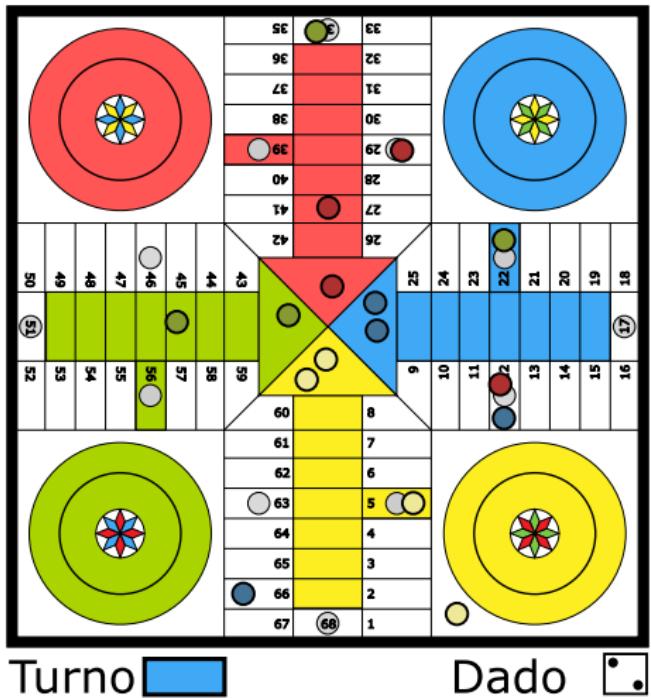
Jugadas Curiosas V: Retirada de Augusto



Jugadas Curiosas V: Retirada de Augusto



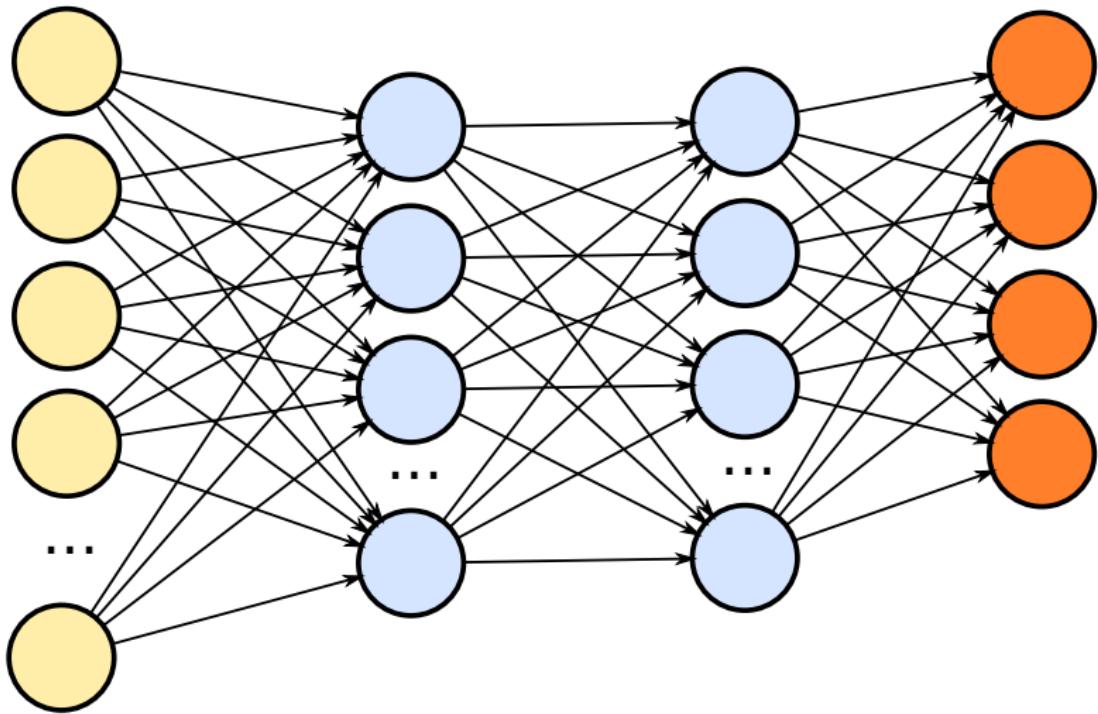
Jugadas Curiosas V: Retirada de Augusto



Índice

- 1 Introducción
- 2 Juegos Básicos: Algoritmo Minimax
- 3 Backtracking y Ramificación y Poda
- 4 Ajedrez por Computador
- 5 Parchís por Computador: Redes Neuronales
 - Aplicando minimax al parchís
 - **Redes Neuronales**
- 6 Conclusiones

Concepto de Red Neuronal



Concepto de Red Neuronal

- ➊ Dado una serie de estímulos de entradas (posiciones fichas), produce un valor de salida (ficha a mover).
- ➋ La decisión puede ser acertada o desacertada.
- ➌ Si es desacertada, la red es capaz de modificar el valor de sus conexiones para que la próxima vez sea más acertada (aprende).
- ➍ Tras un largo entrenamiento, la red ha acumulado experiencia y tiene una muy alta fiabilidad.

Concepto de Red Neuronal

- ➊ Dado una serie de estímulos de entradas (posiciones fichas), produce un valor de salida (ficha a mover).
- ➋ La decisión puede ser acertada o desacertada.
- ➌ Si es desacertada, la red es capaz de modificar el valor de sus conexiones para que la próxima vez sea más acertada (aprende).
- ➍ Tras un largo entrenamiento, la red ha acumulado experiencia y tiene una muy alta fiabilidad.

Concepto de Red Neuronal

- ① Dado una serie de estímulos de entradas (posiciones fichas), produce un valor de salida (ficha a mover).
- ② La decisión puede ser acertada o desacertada.
- ③ Si es desacertada, la red es capaz de modificar el valor de sus conexiones para que la próxima vez sea más acertada (aprende).
- ④ Tras un largo entrenamiento, la red ha acumulado experiencia y tiene una muy alta fiabilidad.

Concepto de Red Neuronal

- ① Dado una serie de estímulos de entradas (posiciones fichas), produce un valor de salida (ficha a mover).
- ② La decisión puede ser acertada o desacertada.
- ③ Si es desacertada, la red es capaz de modificar el valor de sus conexiones para que la próxima vez sea más acertada (aprende).
- ④ Tras un largo entrenamiento, la red ha acumulado experiencia y tiene una muy alta fiabilidad.

Índice

- 1 Introducción
- 2 Juegos Básicos: Algoritmo Minimax
- 3 Backtracking y Ramificación y Poda
- 4 Ajedrez por Computador
- 5 Parchís por Computador: Redes Neuronales
- 6 Conclusiones

Conclusiones

- ➊ Un computador calcula muchísimo más rápido que un humano.
- ➋ Un humano no puede ganarle a un computador en actividades basadas en cálculos, como el ajedrez.
- ➌ Ante un reto de vida o muerte, entre ajedrez y parchís contra un computador, elige parchís.
- ➍ ¿Realmente piensa un computador y es inteligente?
- ➎ Se puede decir que un computador es inteligente cuando realiza una tarea que al realizarla un humano diríamos que es inteligente.
- ➏ Los humanos tenemos una mayor eficiencia energética.

Conclusiones

- ① Un computador calcula muchísimo más rápido que un humano.
- ② Un humano no puede ganarle a un computador en actividades basadas en cálculos, como el ajedrez.
- ③ Ante un reto de vida o muerte, entre ajedrez y parchís contra un computador, elige parchís.
- ④ ¿Realmente piensa un computador y es inteligente?
- ⑤ Se puede decir que un computador es inteligente cuando realiza una tarea que al realizarla un humano diríamos que es inteligente.
- ⑥ Los humanos tenemos una mayor eficiencia energética.

Conclusiones

- ① Un computador calcula muchísimo más rápido que un humano.
- ② Un humano no puede ganarle a un computador en actividades basadas en cálculos, como el ajedrez.
- ③ Ante un reto de vida o muerte, entre ajedrez y parchís contra un computador, elige parchís.
- ④ ¿Realmente piensa un computador y es inteligente?
- ⑤ Se puede decir que un computador es inteligente cuando realiza una tarea que al realizarla un humano diríamos que es inteligente.
- ⑥ Los humanos tenemos una mayor eficiencia energética.

Conclusiones

- ① Un computador calcula muchísimo más rápido que un humano.
- ② Un humano no puede ganarle a un computador en actividades basadas en cálculos, como el ajedrez.
- ③ Ante un reto de vida o muerte, entre ajedrez y parchís contra un computador, elige parchís.
- ④ ¿Realmente piensa un computador y es inteligente?
- ⑤ Se puede decir que un computador es inteligente cuando realiza una tarea que al realizarla un humano diríamos que es inteligente.
- ⑥ Los humanos tenemos una mayor eficiencia energética.

Conclusiones

- ① Un computador calcula muchísimo más rápido que un humano.
- ② Un humano no puede ganarle a un computador en actividades basadas en cálculos, como el ajedrez.
- ③ Ante un reto de vida o muerte, entre ajedrez y parchís contra un computador, elige parchís.
- ④ ¿Realmente piensa un computador y es inteligente?
- ⑤ Se puede decir que un computador es inteligente cuando realiza una tarea que al realizarla un humano diríamos que es inteligente.
- ⑥ Los humanos tenemos una mayor eficiencia energética.

Conclusiones

- ① Un computador calcula muchísimo más rápido que un humano.
- ② Un humano no puede ganarle a un computador en actividades basadas en cálculos, como el ajedrez.
- ③ Ante un reto de vida o muerte, entre ajedrez y parchís contra un computador, elige parchís.
- ④ ¿Realmente piensa un computador y es inteligente?
- ⑤ Se puede decir que un computador es inteligente cuando realiza una tarea que al realizarla un humano diríamos que es inteligente.
- ⑥ Los humanos tenemos una mayor eficiencia energética.

Gracias por vuestra atención.

