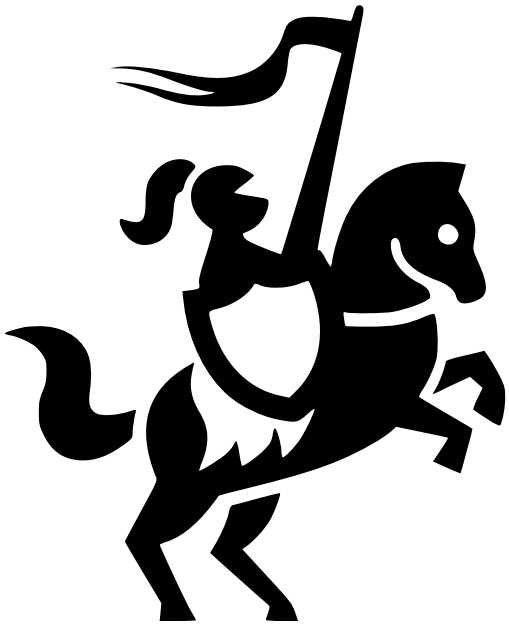
¿Cómo juega un computador al ajedrez y por qué no puedo ganarle?

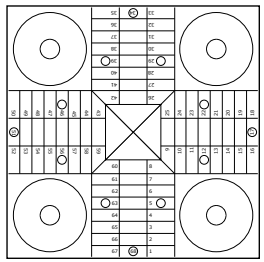
Pablo Sánchez

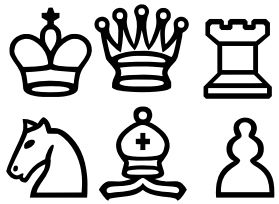
Dpto. Ingeniería Informática y Electrónica
Universidad de Cantabria
Santander (Cantabria, España)
p.sanchez@unican.es

Índice

Índice







Índice

¿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?

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 CO_2.

Índice

Algoritmo Minimax

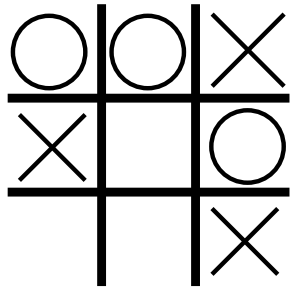
Algoritmo Minimax

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

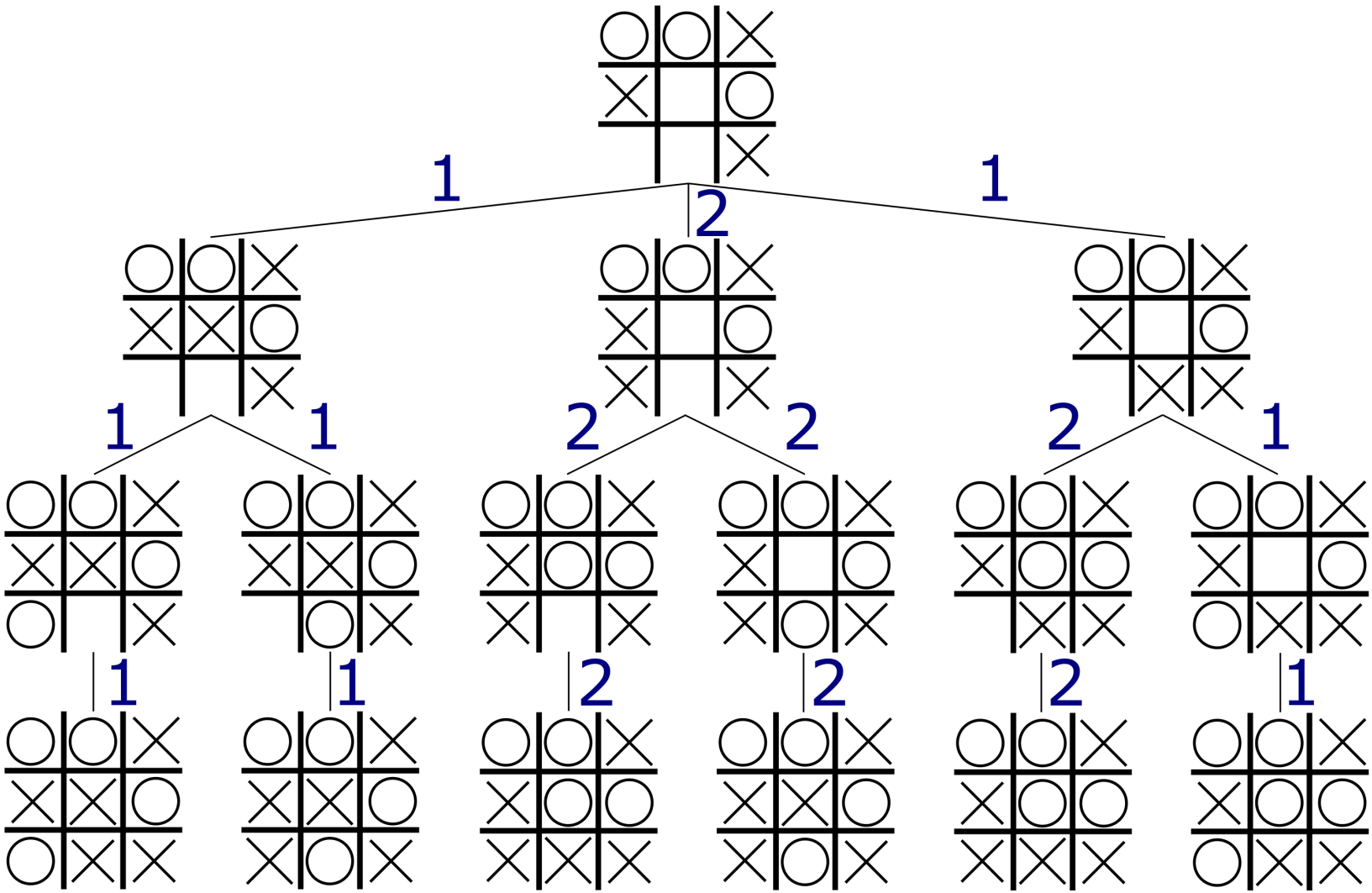


1. El tablero es una tabla de números: -1/libre, 0/círculo, 1/cruz (pc).
2. Movimientos válidos: si hay hueco (-1), pongo la ficha que toque.
3. Final de la partida:
   - Miro filas, columnas y diagonales por si hay tres fichas iguales.
   - No hay huecos (-1) libres.
4. Valor de la partida:
   - Tres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

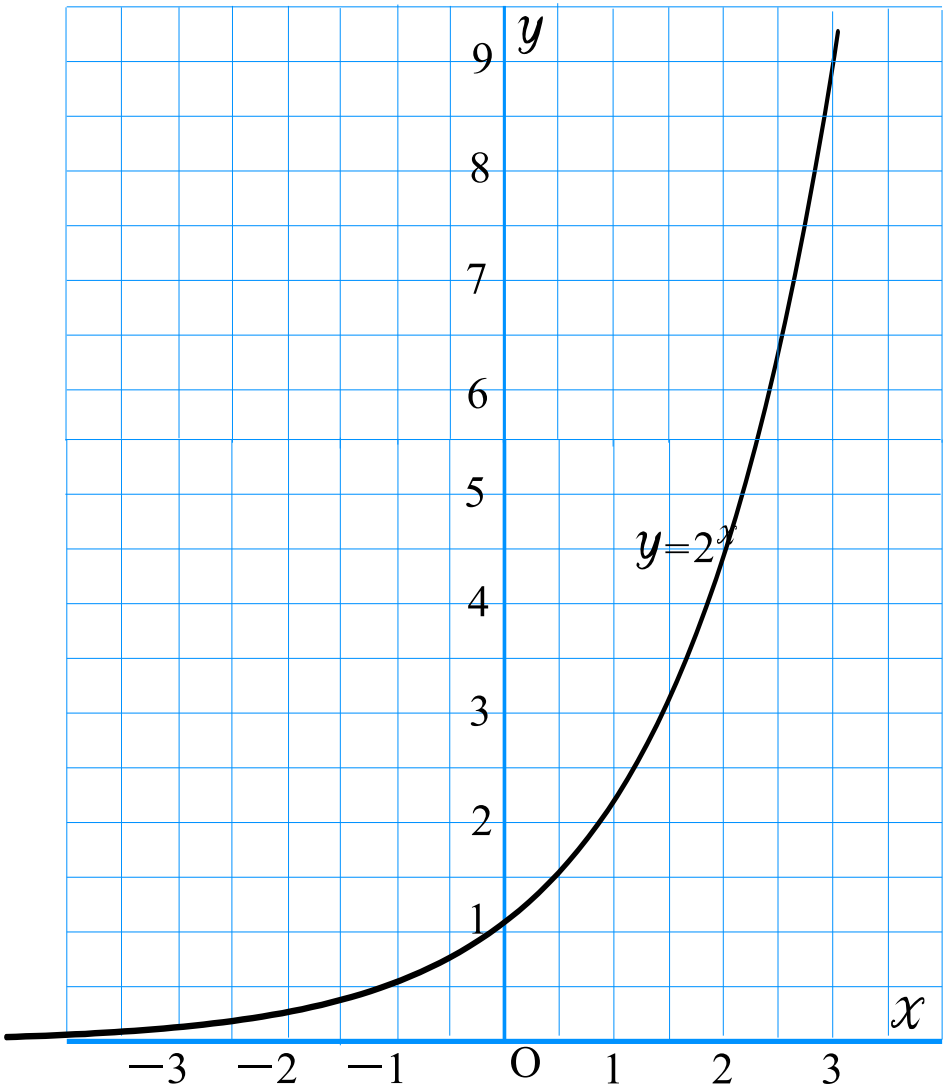
Algoritmo Minimax

1. 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



Problema Algoritmo Minimax



Índice

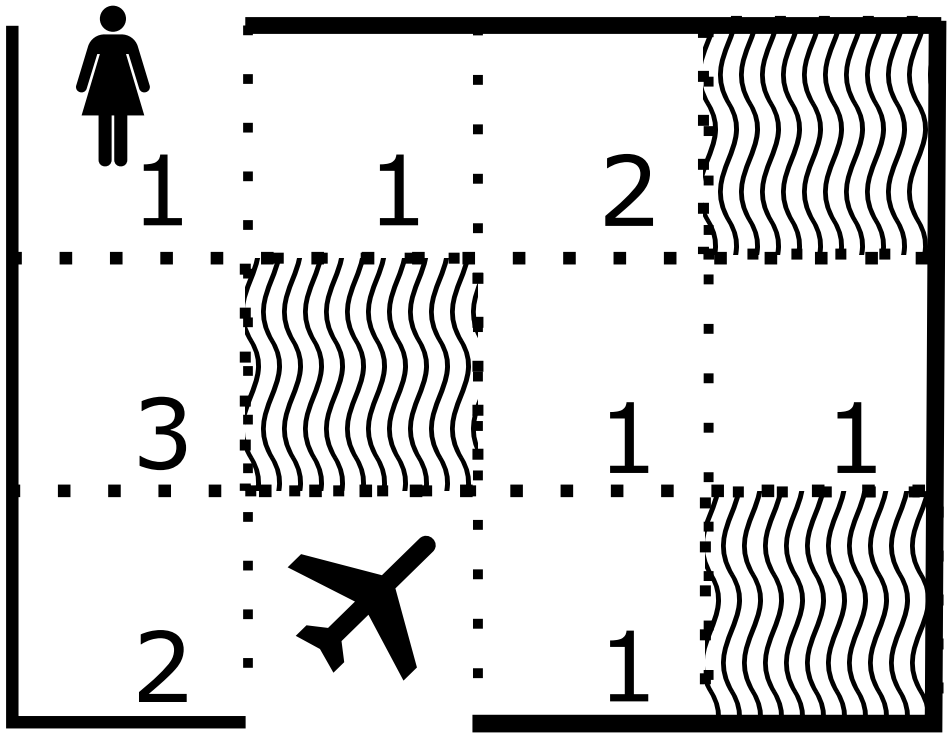
Índice

Algoritmo *Backtracking*

Algoritmo *Backtracking* (Vuelta atrás)

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

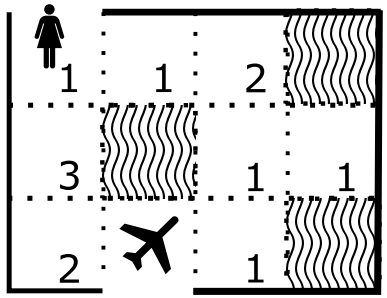
Algoritmo *Backtracking*



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*

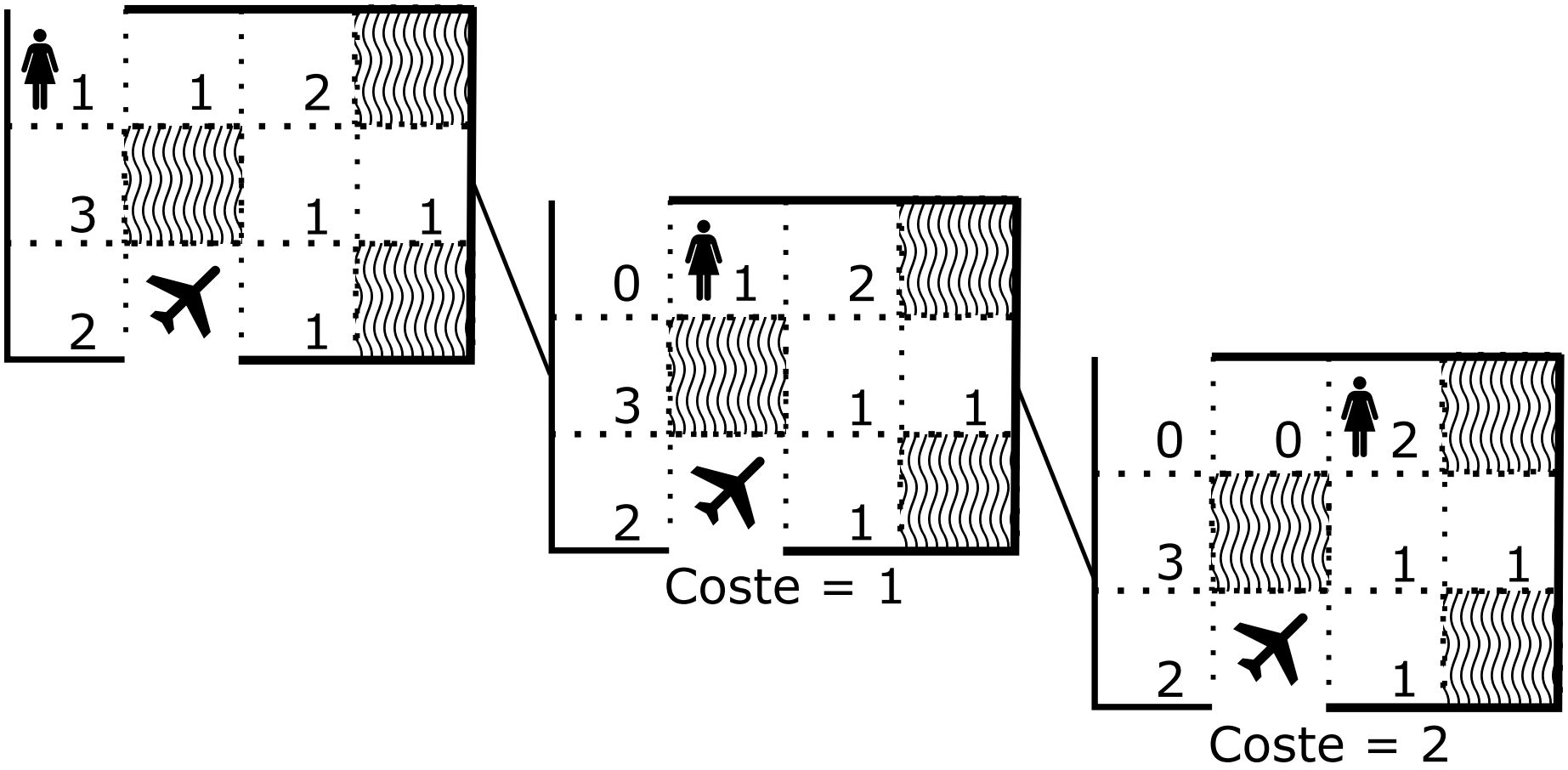


1. Problema: El laberinto es una tabla de números.
   - -1 significa obstáculo.
   - Un valor positivo representa el coste de atravesar la casilla.
   - 0 indica que ya he pasado.
2. Pasos válidos: Moverse derecha, abajo, izquierda y arriba si:
   - Sigo dentro del laberinto.
   - No me muevo a un obstáculo.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

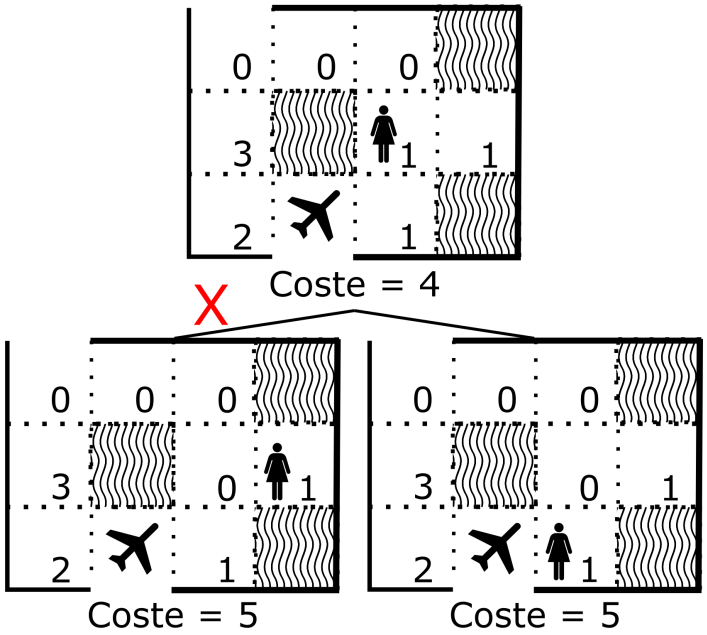
Algoritmo *Backtracking*

1. Si soy solución, calculo el coste y lo propago.
2. 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.
3. Si no hay más movimientos libres, *vuelvo a atrás* y propago el valor de la solución.

Algoritmo *Backtracking*



Algoritmo *Backtracking*



Índice

Algoritmo Ramificación y Poda

Algoritmo Ramificación y Poda

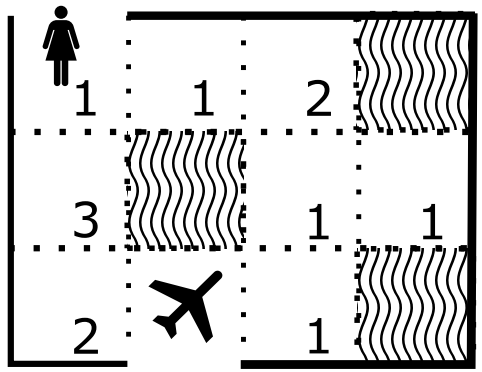
1. Versión de *backtracking* donde:
   - Expando todos los movimientos válidos de un nodo.
   - Evalúo lo prometedor de cada nodo.
   - Exploro el nodo más prometedor.
   - No exploro los movimientos que no pueden alcanzar una solución óptima.
2. Para evaluar lo prometedor, utilizo *heurísticas*.

Heurísticas

Heurística Ramificación y Poda

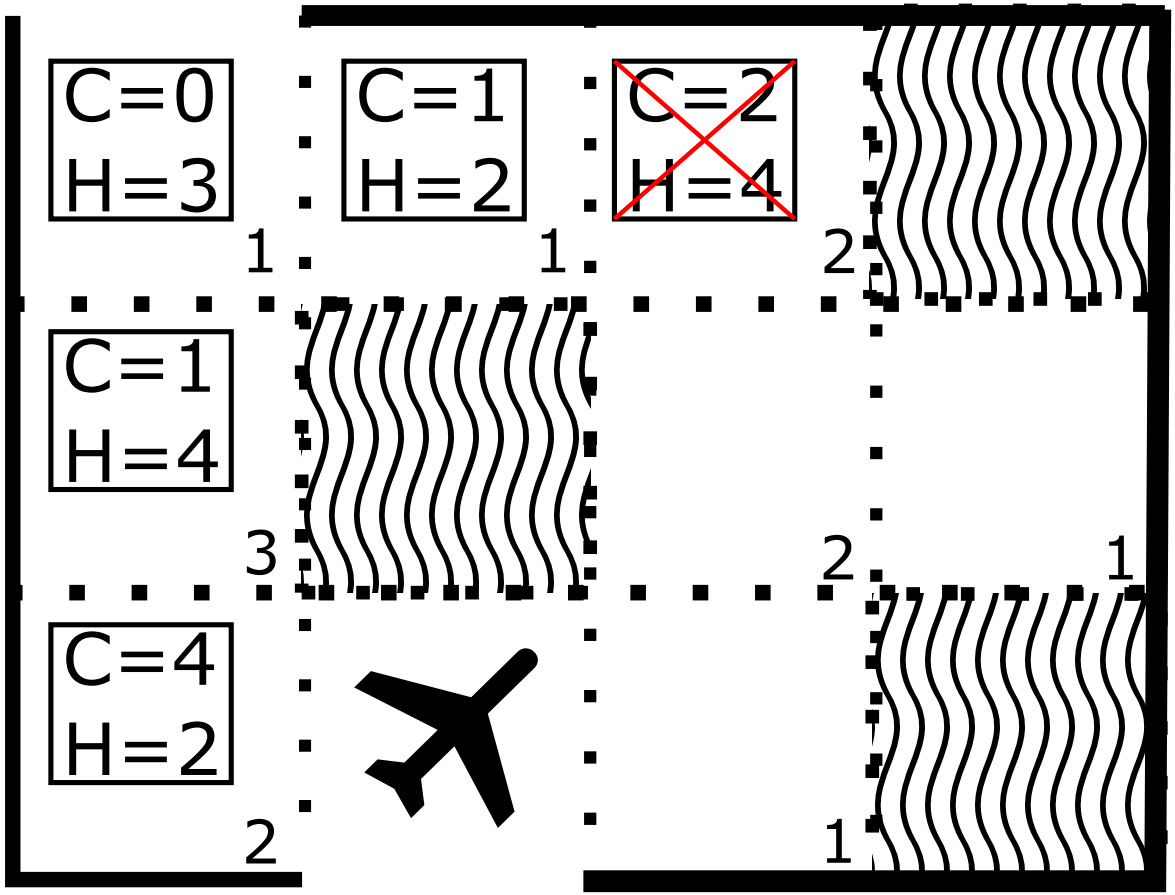
Función que devuelve una cota inferior y optimista del coste que me queda para alcanzar la solución.

Ejemplo de Heurística



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.

Algoritmo Ramificación y Poda



Solución = 6

Índice

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

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*.

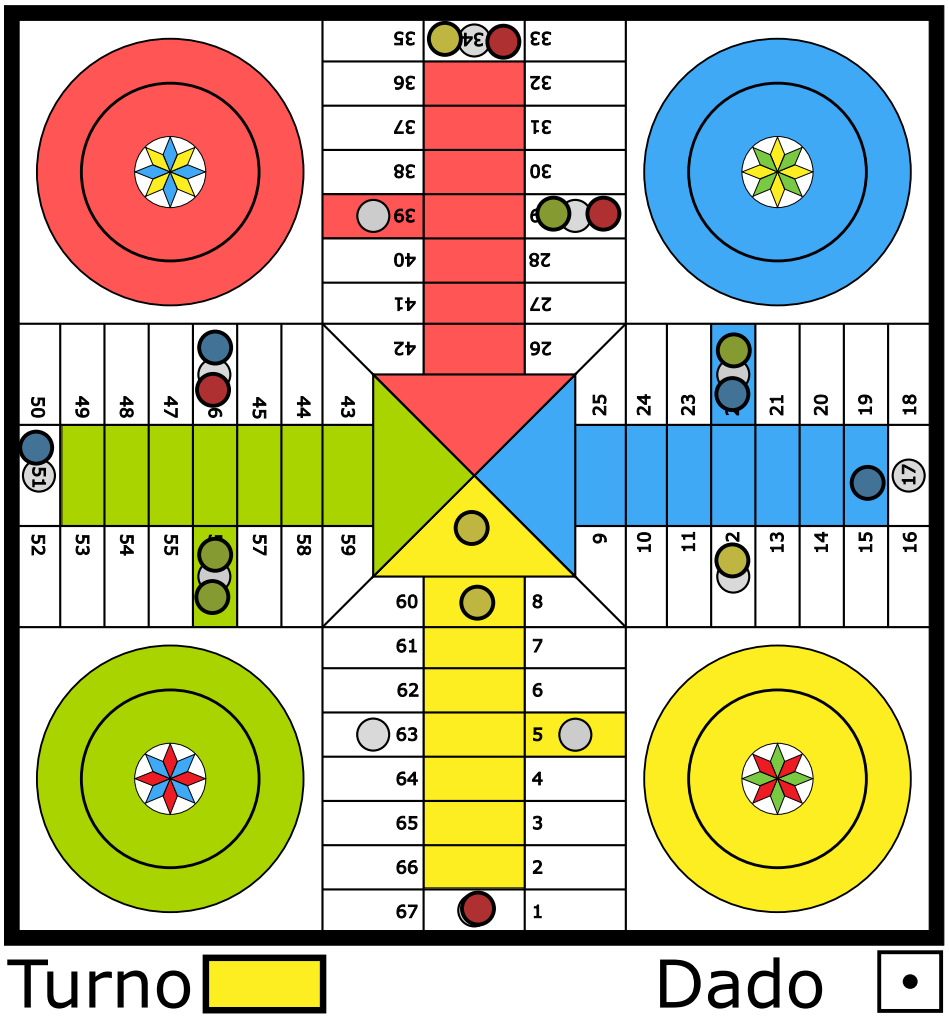
Índice

Índice

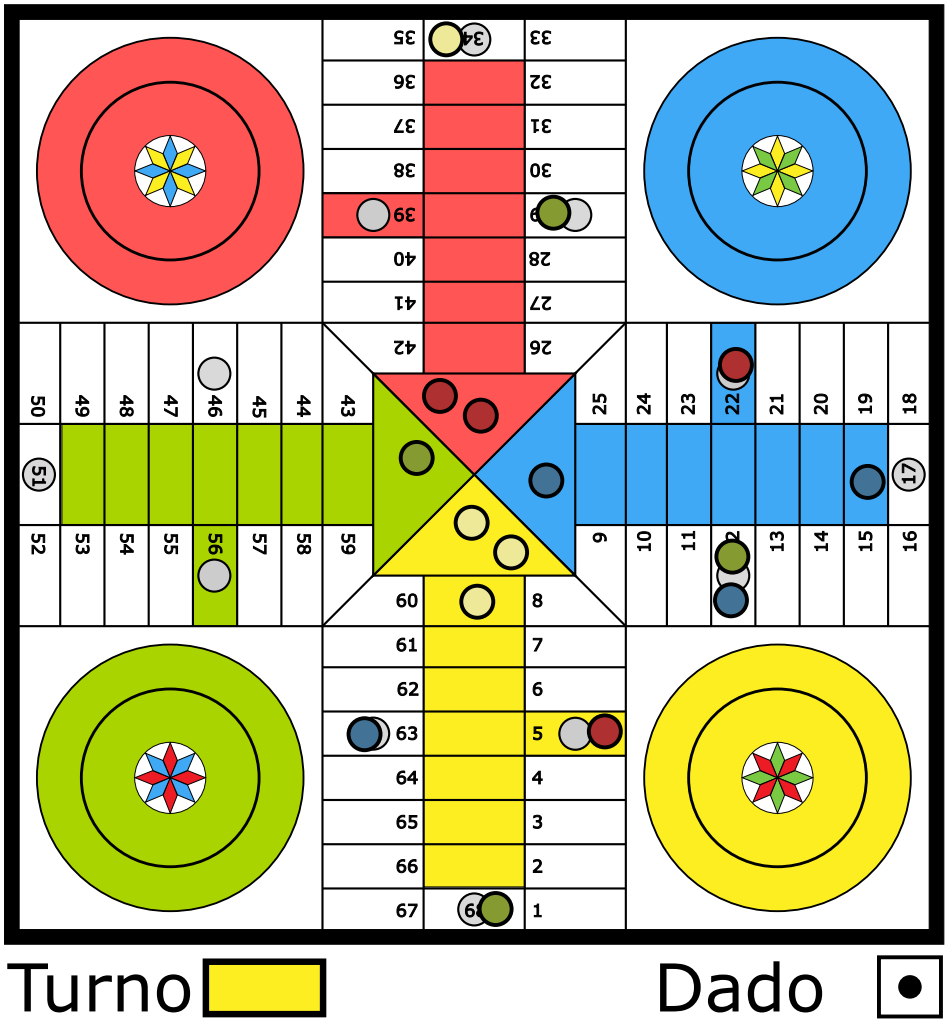
Parchís con Minimax

1. Árbol de profundidad infinita:
   - Nadie sale nunca.
   - Nadie entra nunca.
   - Salgo, avanzo 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.

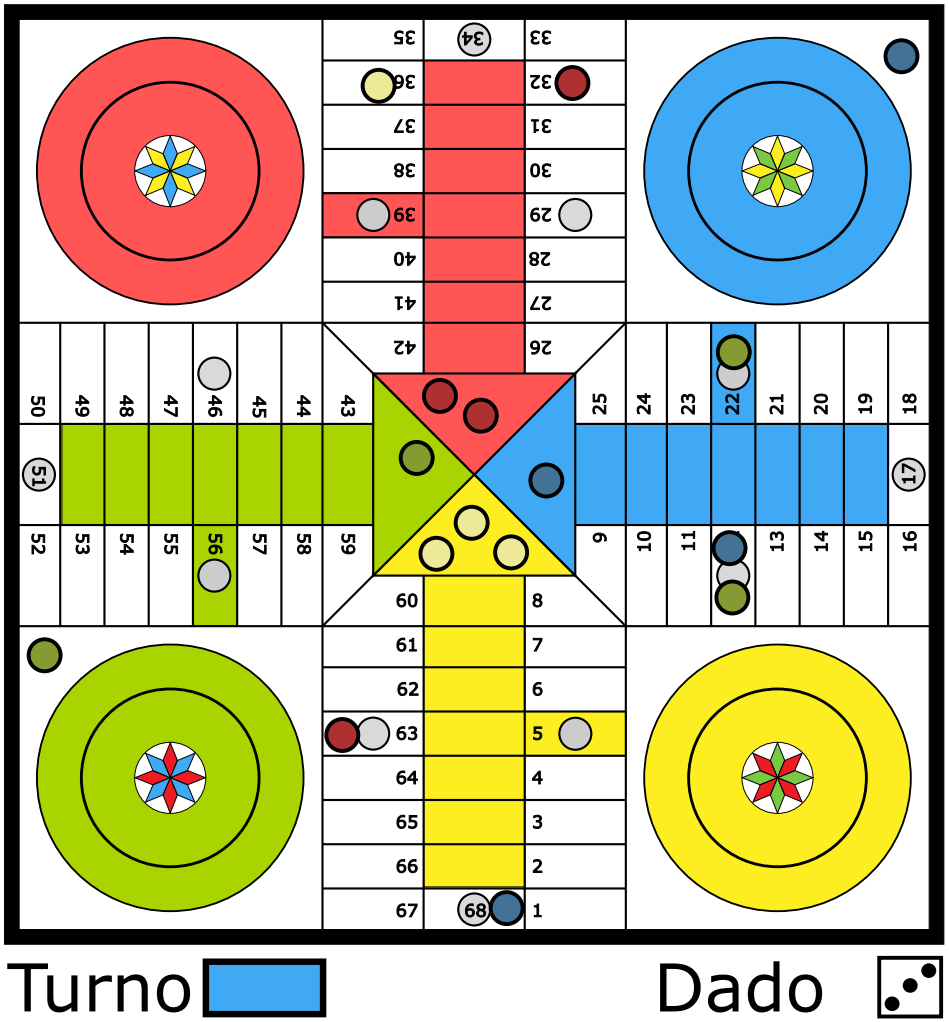
Jugadas Curiosas I: Ahorro de Saldo



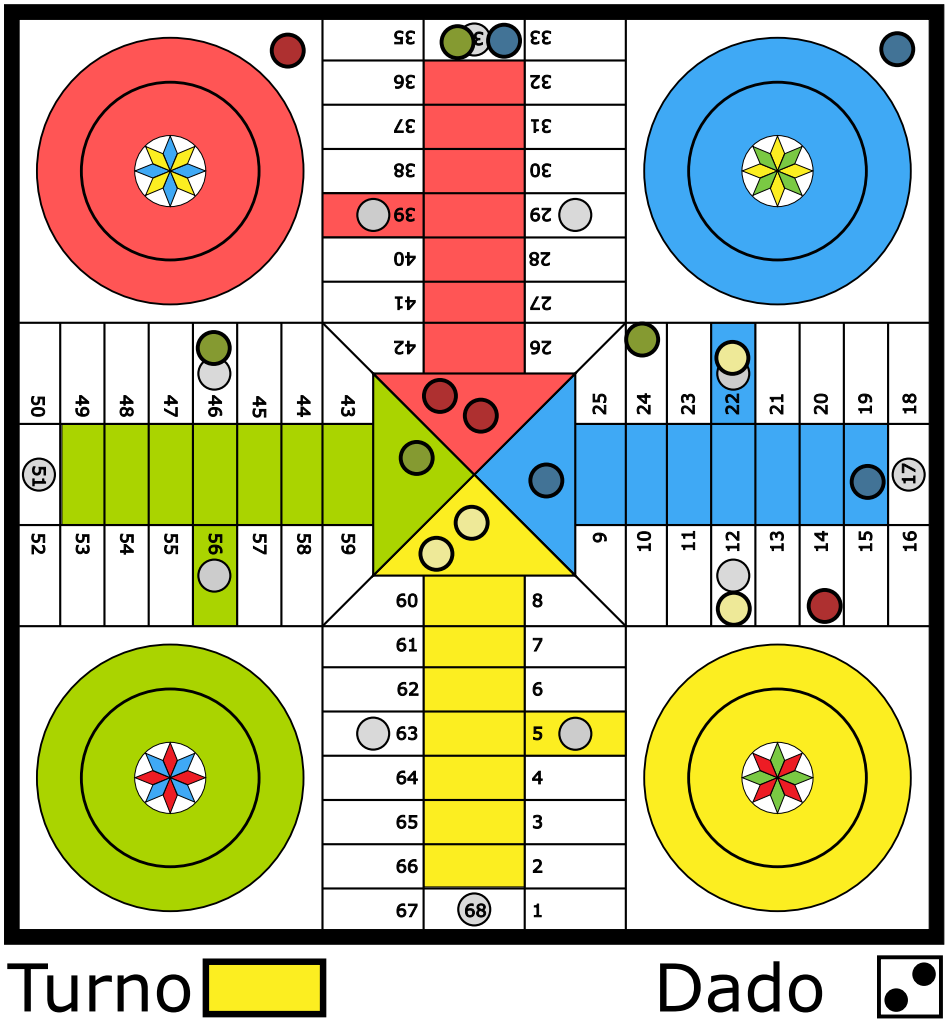
Jugadas Curiosas II: Evitar Ficha Única



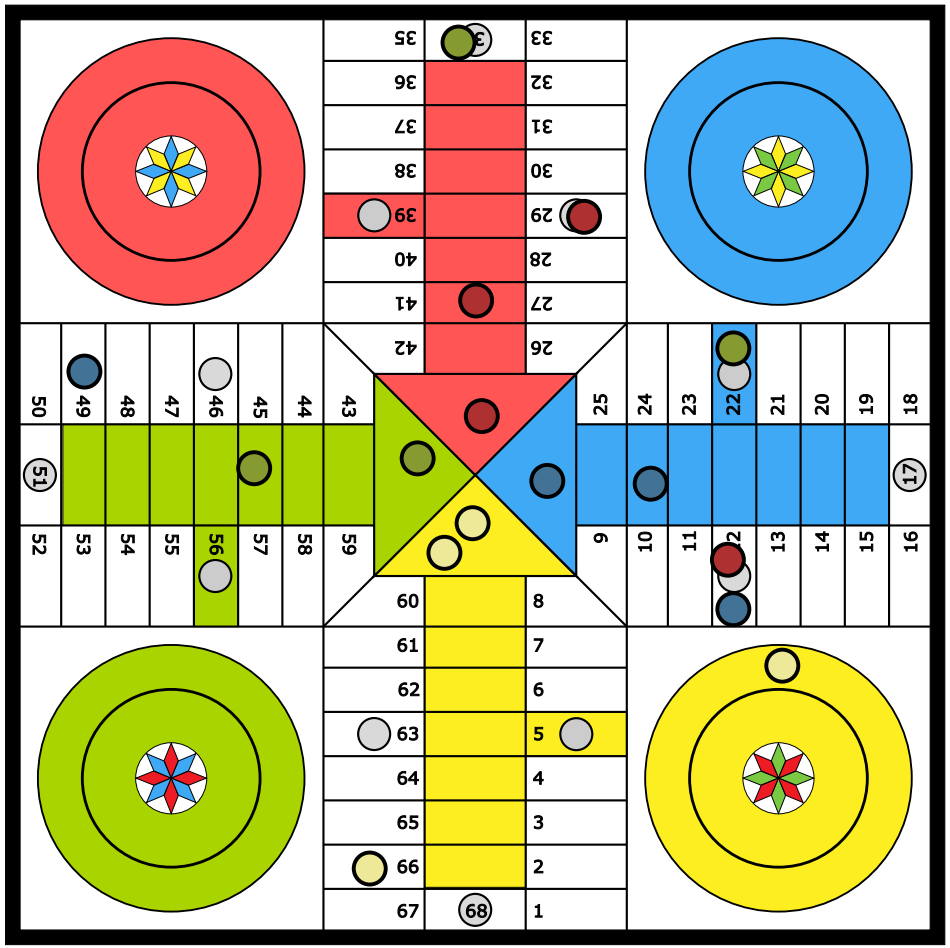
Jugadas Curiosas III: Bien Común



Jugadas Curiosas IV: Cantos de Sirenas

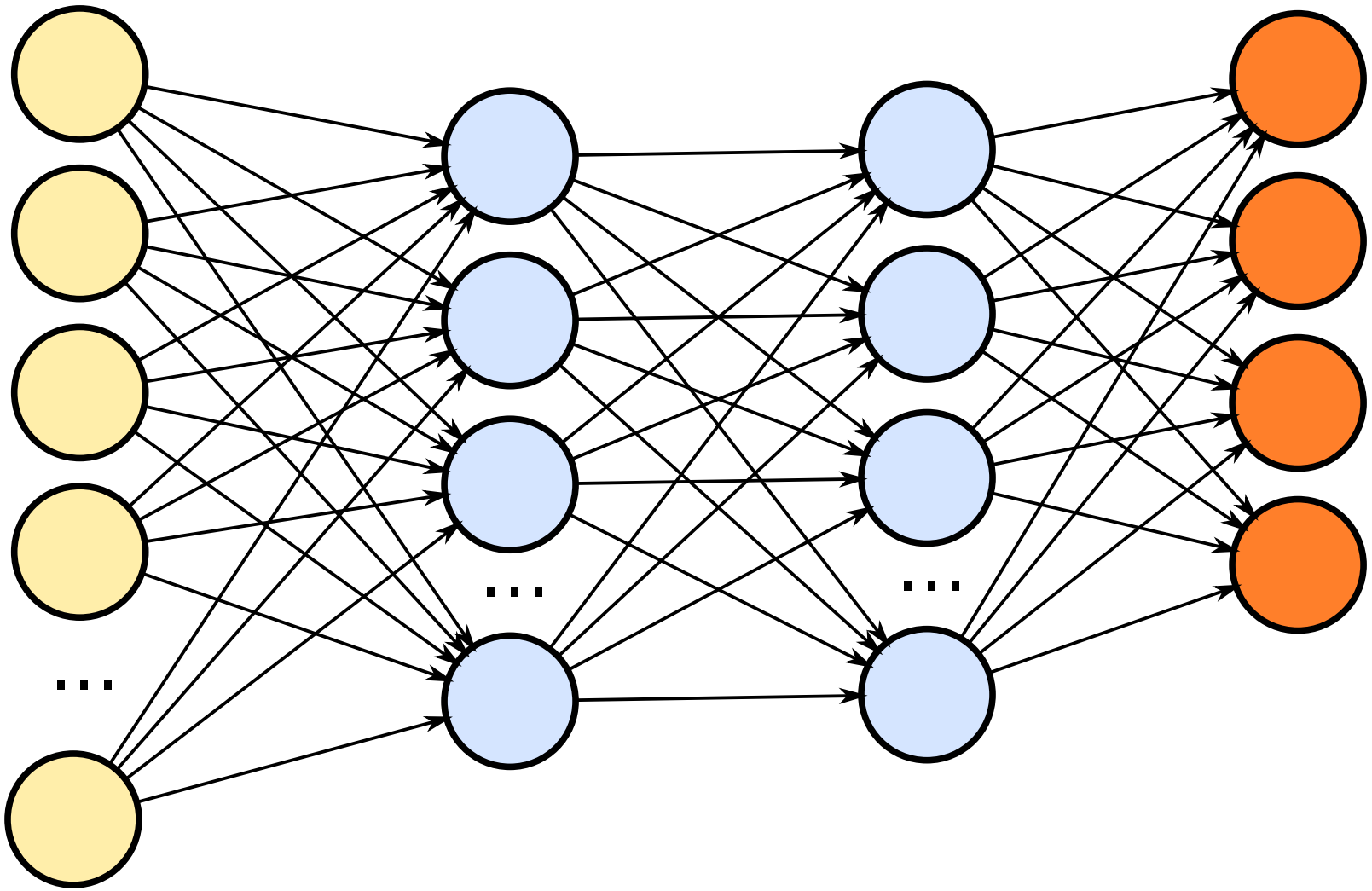


Jugadas Curiosas V: Retirada de Augusto



Índice

Concepto de Red Neuronal



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

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.