Simple Squares

“The Game About Squares”

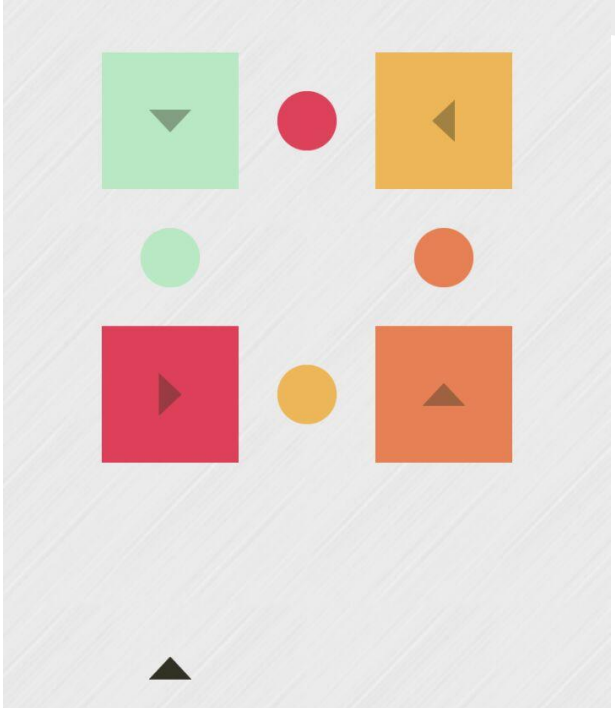
Grupo 7
Agustina Fainguersch
Nicolás Buchhalter
Francisco Depascuali

Introducción

- **Objetivo**: Llevar todos los bloques a sus respectivos targets.

- **Flechas negras:** rotan el sentido del bloque.

- Los bloques se pueden empujar entre sí.



Implementación

Bloques: posición, posición destino, direccion.

Flechas: posición, dirección.

Se dispone de dos listas:

```
List<Block> blocks = new ArrayList<Block>();
List<Arrow> arrows = new ArrayList<Arrow>();
```

Reglas: Arriba, Abajo, Derecha, Izquierda

Implementación

Función de costo: es constante de valor 1 ya que al aplicar una regla solo se puede dar un paso en el juego.

El costo en el estado final -> cantidad de movimientos para ganar.

Estado final: modificamos el motor original por un método que verifica si un estado es solución, fijándose si cada uno de los bloques está en su posición destino.

Estrategias no informadas

DFS

- Aplica las reglas en **profundidad**.
- Única solución → se pierde mucho tiempo explorando ramas sin encontrar soluciones.
- **Desventaja** → Lento
- Para evitar ramas infinitas → corte en 10x10
- Motor implementado → sin ciclos inf: si vuelve al mismo estado con mayor costo no lo considera.

BFS

- Aplica las reglas a lo **ancho del árbol**
- Mejor que DFS (si no hay múltiples soluciones)
- Profundidad → cantidad de movimientos (porque el costo es 1)

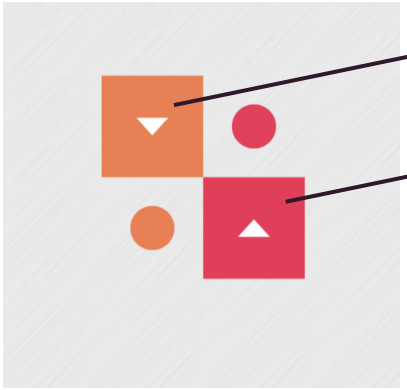
Iterative Deepening

- “Combinación” entre BFS y DFS.
- Explora en profundidad hasta cierto nivel.

Nivel 2

Ejemplo para ver el comportamiento del algoritmo

BFS

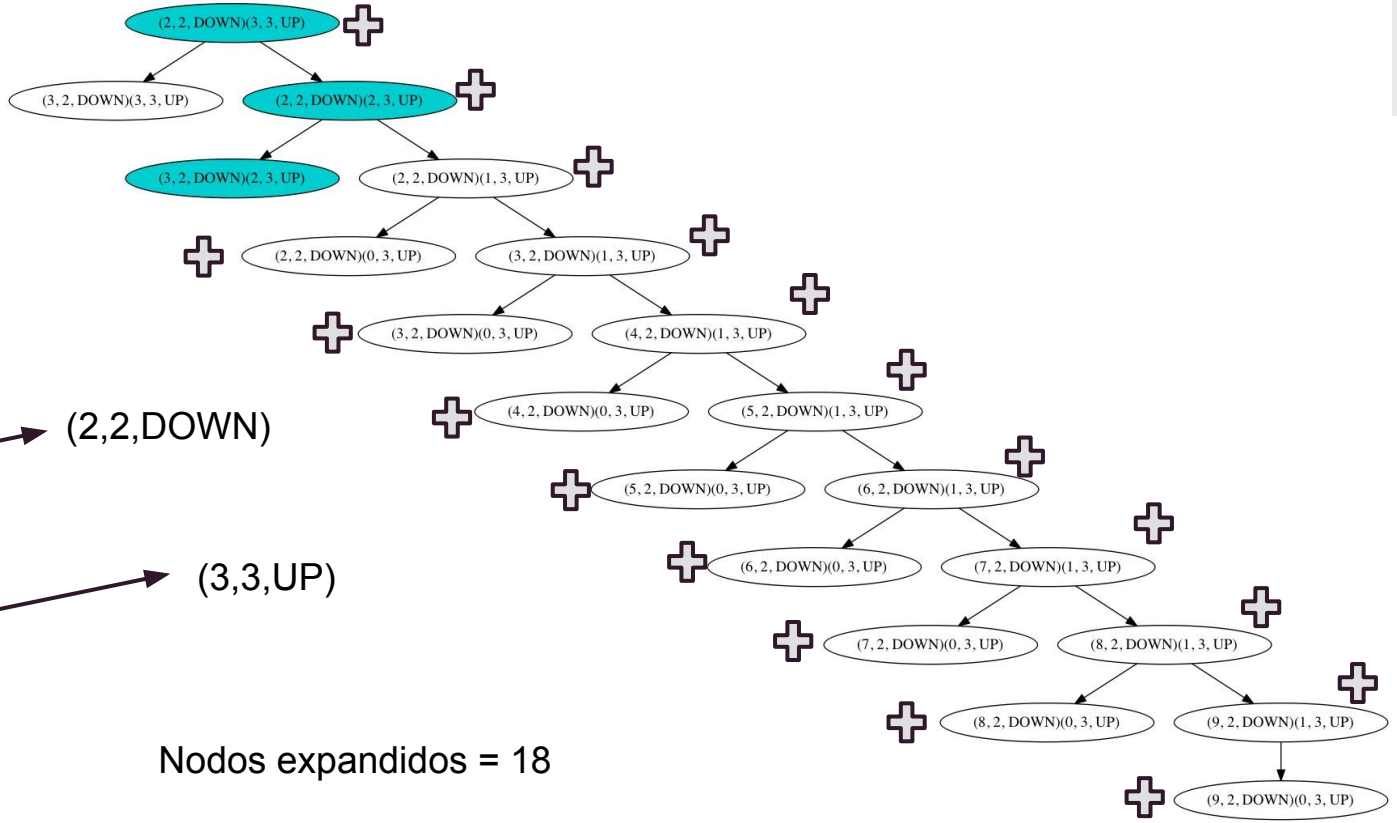


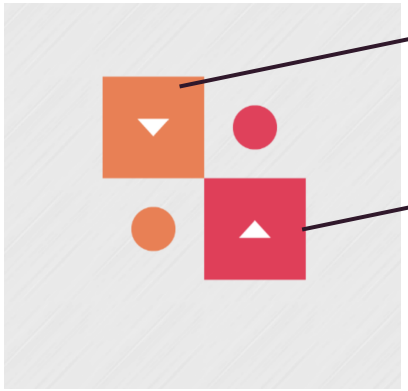
(2,2,DOWN)

(3,3,UP)

Nodos expandidos = 4

DFS



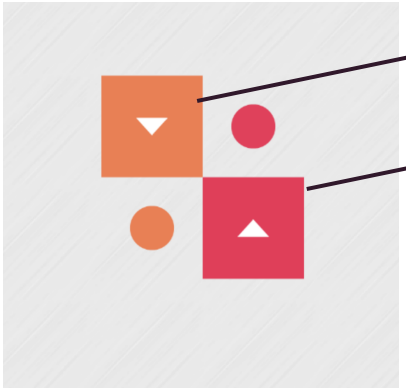


(2,2,DOWN)

(3,3,UP)

Nodos expandidos = 18

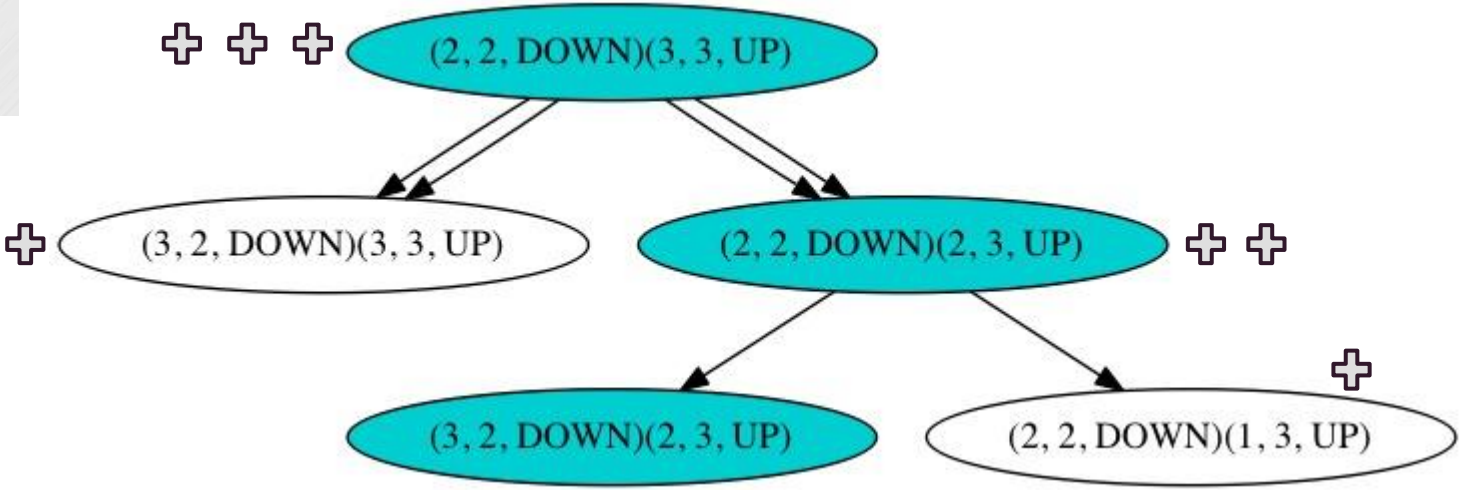
Iterative Deepening



(2,2,DOWN)

(3,3,UP)

Nodos expandidos = 7



Estrategias informadas

Greedy

- Toma el siguiente estado con **menor** h
- Analiza nodos **recién expandidos**
- **Desventaja**: lento (Se encuentra en otra rama)
- **Ventaja**: eficiente asignación de recursos

A*

- Toma el siguiente estado con **menor** h + costo

- Analiza todos los nodos frontera

- **Ventajas**:
  - Siempre toma el mejor camino → más rápido.
  - Si se le aplica un h admisible → encuentra camino óptimo.
- **Desventajas**:
  - Utiliza muchos recursos.

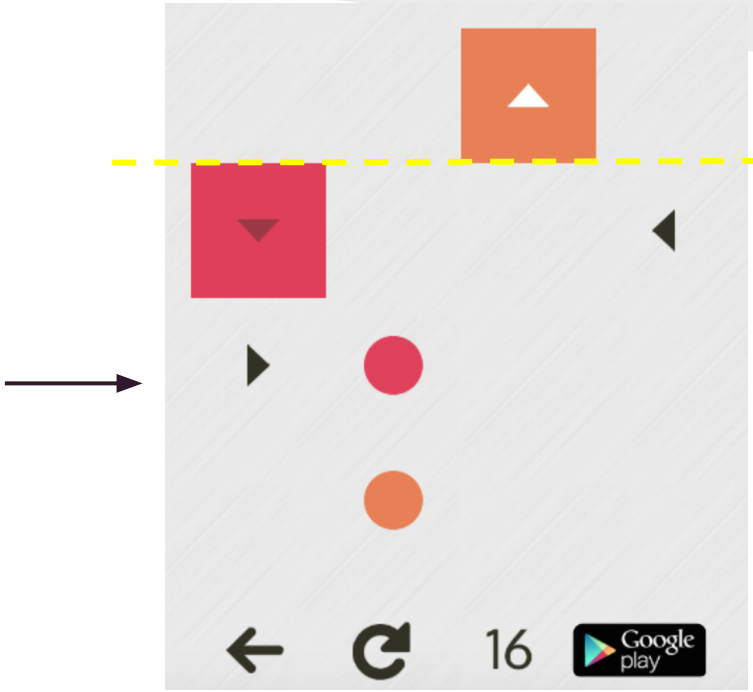
Heurísticas

Default

- Incluida en todas las otras heurísticas
- Se fija los casos en los que *ya no se puede ganar*
- Si el bloque sale de las dimensiones del tablero virtual (INITIAL_POSITION, MAX_POSITION) con una dirección que no es hacia adentro, devuelve +Inf

Ejemplo

Estado 1



Estado 2

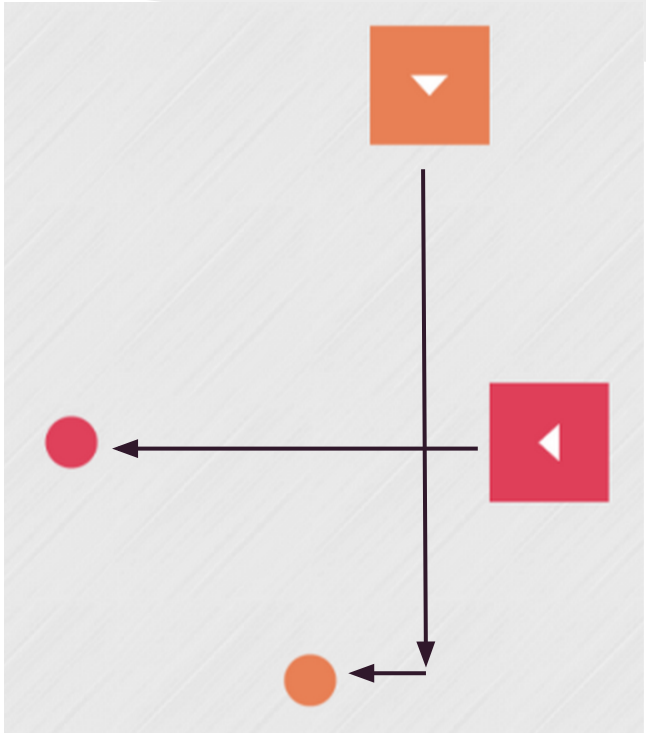
El valor heurístico del estado 2 es +Inf

Min distance 1

Suma → distancias de **Manhattan** de todos los bloques a su destino.

- **Ventaja**: tableros con caminos directos.
- **Desventaja**: no tiene en cuenta el empuje de un bloque a otro, ni las flechas que cambian el sentido del bloque.

¿Es admisible?



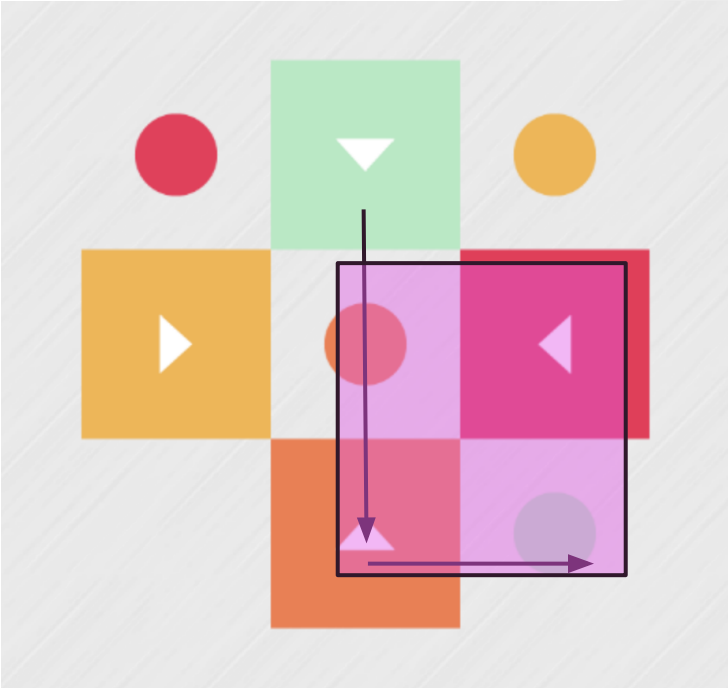
¿Admisible?

- h*(x) = 1 + 1 = 2 (naranja, luego amarillo)
- h(x) = 1 + 2 = 3

Como h(x) sobreestima h*(x) => h (x) no es admisible

Min distance 2

- Relaja el problema:
  Se divide → c/ dist de Manhattan por la cantidad de bloques en el área.
- ¿Es admisible?



¿Admisible?

- h*(x) = 7
- h(x) = 4 + 4 = 8

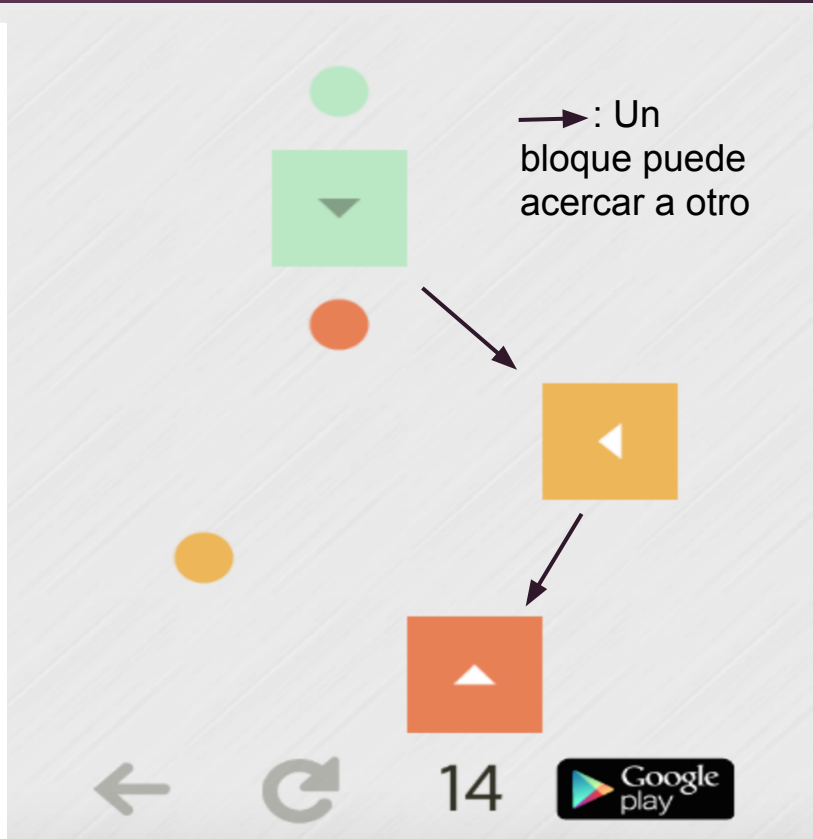
Como h(x) sobreestima h*(x) => h(x) no es admisible

Min distance 3

- Analiza para c/bloque si existe algún otro bloque que lo acerca a destino. Divide la distancia de manhattan por esa cantidad.

- **Desventaja**: No considera las flechas que pueden cambiar el sentido de los bloques.

- ¿Es admisible?



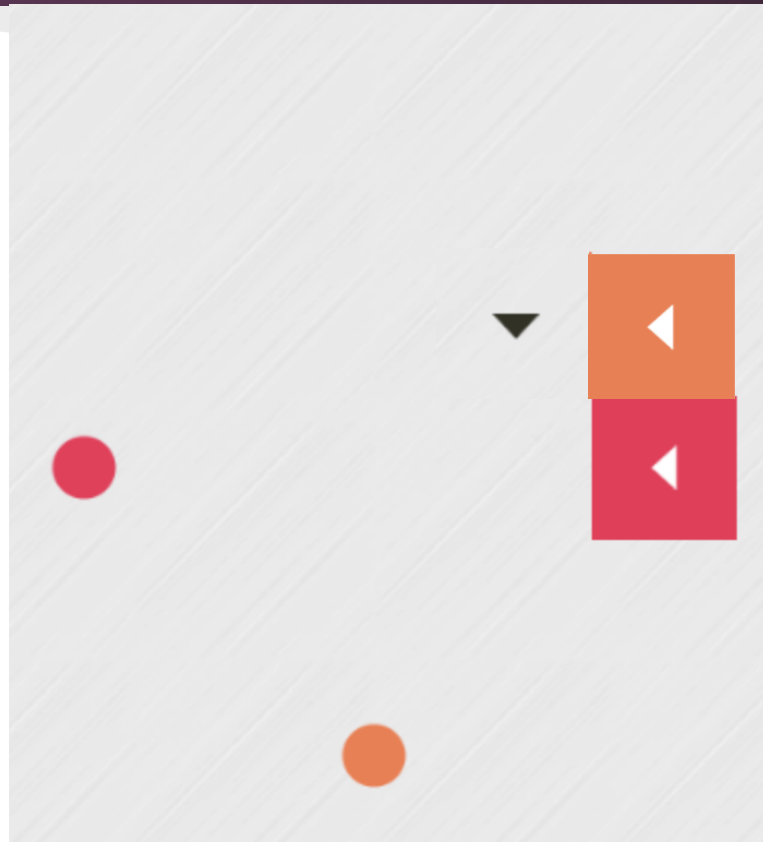
¿Admisible?

$h^*(x) = 8$

$h(x) = 5 + 4 = 9$

Como h(x) sobreestima h*(x) =>

h(x) no es admisible



Admissible min distance

- Toma el **máximo** de las distancias de Manhattan entre los bloques de un nivel.
- **Ventaja**: Se evita sobreestimar el costo.
- **Desventaja**: relajar problema → no tan buena heurística

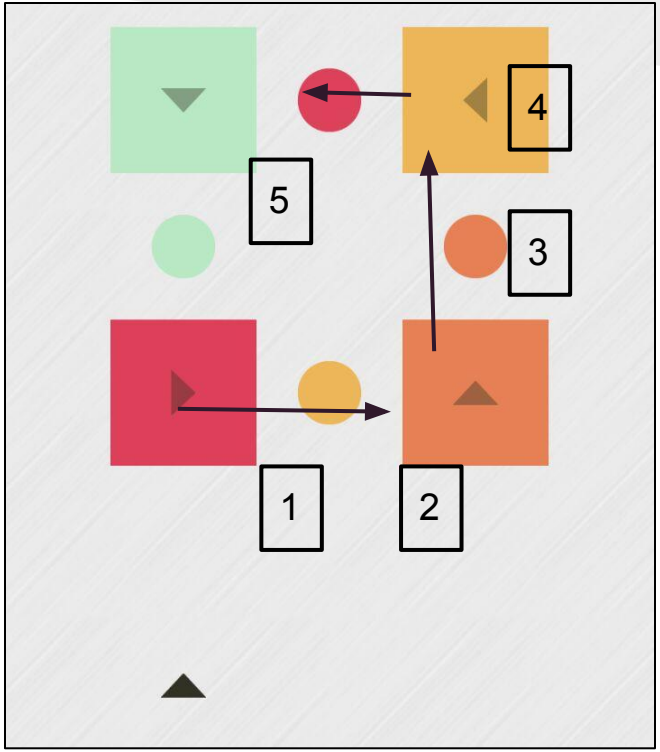
In Path

- Calcula el camino necesario para alcanzar el objetivo.

- Encaminado → # movimientos consec hasta destino

- No encaminado → dist. de Manhattan

- Devuelve→ suma para todos los bloques



Admissible In Path

- Toma el **máximo** de los recorridos In Path de todos los bloques de un nivel.
- **Ventaja**: Se evita sobreestimar el costo.
- **Desventaja**: relajar problema → no tan buena heurística

Not Admissible combination

Combina las dos mejores de las heurísticas no admisibles:

- Min Distance 1 (h1)
- In Path (h2)

No admisibles → Se debería tomar el mínimo.

Se toma el **máximo** → en la mayoría de los casos h1 y h2 < h*.

Admissible Combination

Para c/nodo → Toma el **máximo** entre:

Admissible Min Distance (h1)

Admissible In Path (h2)

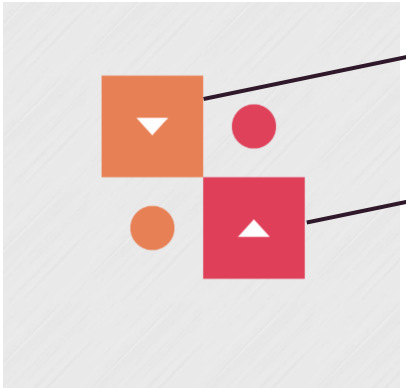
Ventaja: Se acerca lo más posible a h* para c/nodo.

Desventaja: Calcula ambas heurísticas, por lo que resulta más lenta.

A* mindistance1

Nodos rojos => h(x) = +inf

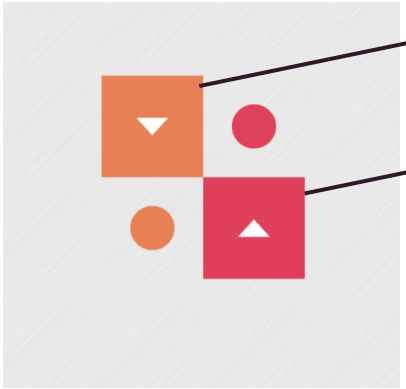
Nodos expandidos = 3



(2,2,DOWN)

(3,3,UP)

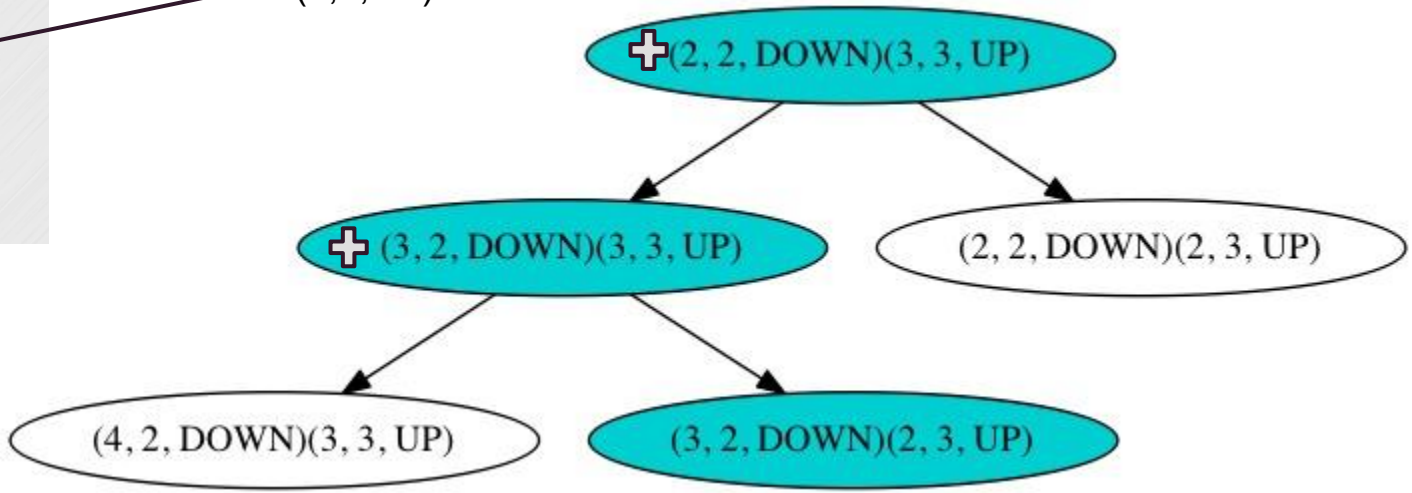
Greedy mindistance1



(2,2,DOWN)

(3,3,UP)

Nodos expandidos = 2



Preguntas

- ¿Por qué min distance 1 e inPath suelen dar mejores resultados que sus versiones admisibles?
- ¿Cuál heurística creen que fue mejor para el nivel 21?
- Si la cantidad de nodos expandidos es similar, ¿Por qué puede ser que utilizando inPath se tarde más tiempo que con minDistance?
- ¿Se les ocurre alguna otra heurística?