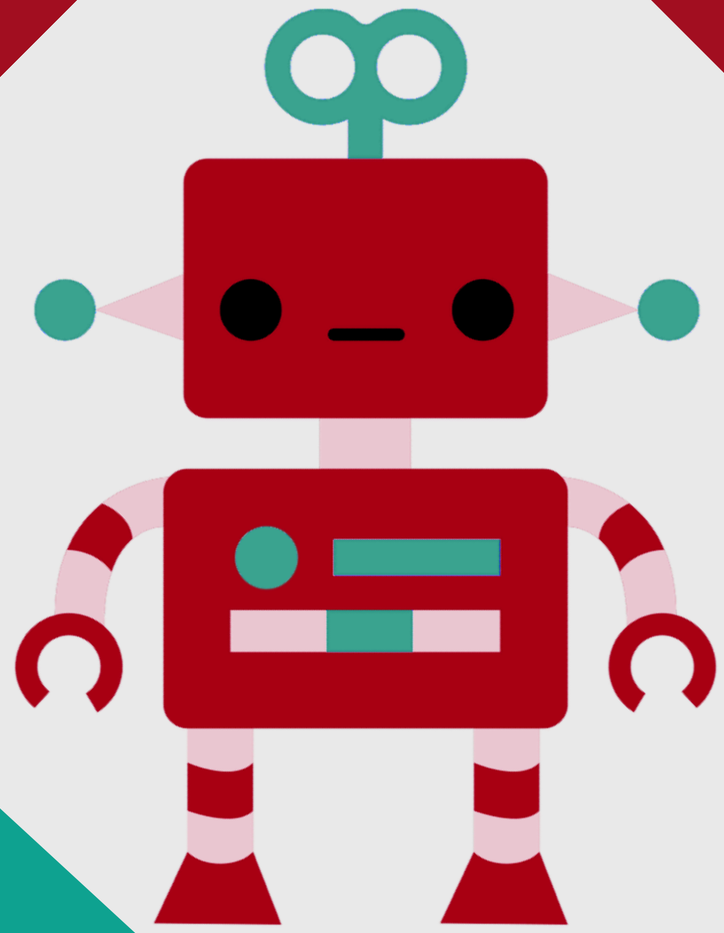
2023/2024

Planning & Reasoning

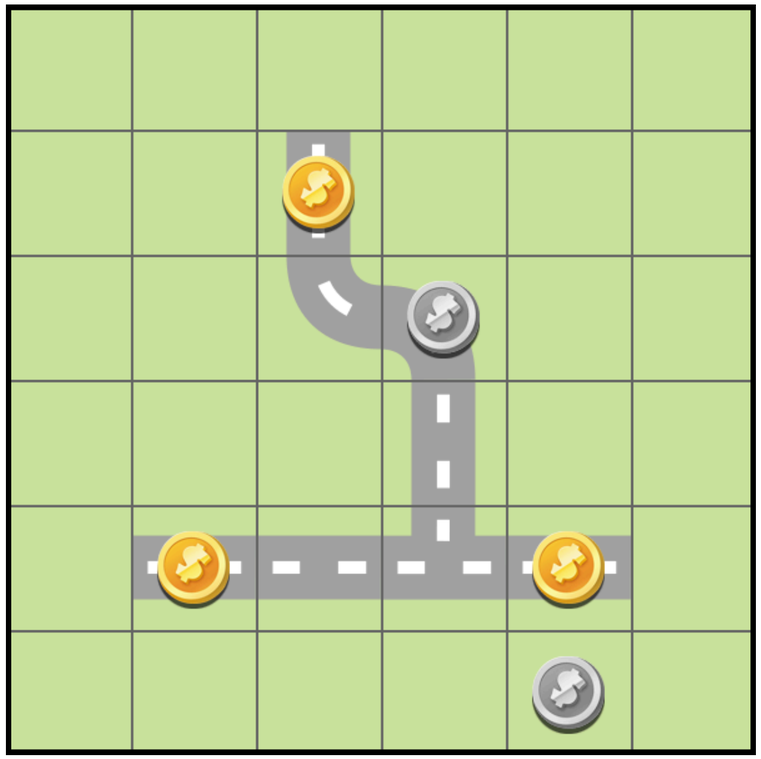
[Github]



Anna Carini
1771784

Alessandro Monteleone
1883922

First idea



Domain

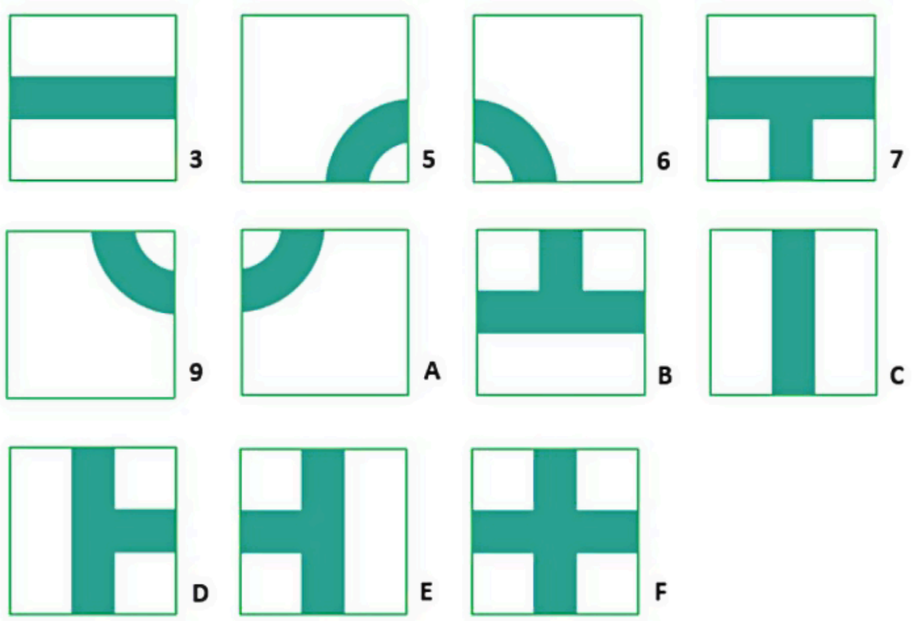
- Field divided in cells (grid)
- Pieces of gold and pieces of silver, placed in cells
- Tiles, used to build a path in the field

Goal

Build a path that **connects** all the pieces of **gold**, using the available tiles. **Maximize the value** obtained taking the pieces of **silver**.

First idea

Tiles



Actions & Costs

- **Place a tile**
  - Tile 3, Tile C → cost = 6
  - Tile 5, Tile 6, Tile 9, Tile A → cost = 2
  - Tile 7, Tile B, Tile D, Tile E → cost = 8
  - Tile F → cost = 15
- **Pick up silver** → gain = 100

First idea

Planning System

[Github Repository]

Planner & Heuristics

Planner:

- LM-CUT from IPC 2011

Heuristics:

- FF
- hmax
- Context-enhanced additive heuristic

Changes

Motivations:

Fast Downward **does not support numerical planning** or negative action costs

We can’t have a positive “gain” when picking up silvers

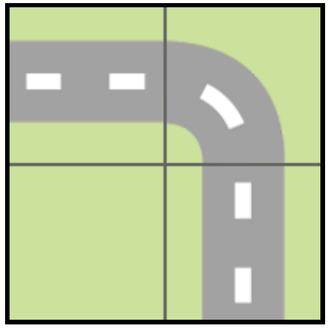
Changes:

We introduced **debts**: you start with one debt for each silver piece in the map, you can either pay it off with a (costly) action, or you can eliminate it for free by picking up a silver piece

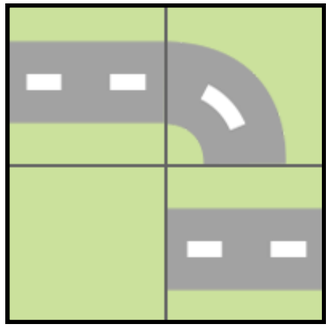
Our Approach

We needed to ensure that:

- adjacent tiles are connected properly to each other (see pictures on the right)
- all the cells with a gold piece have a tile
- all the cells with a gold piece are connected to each other



OK



Not OK

Our solution:

- only allow to place a tile if connects properly with at least one adjacent tile
- only allow to place a tile next to an already placed one (except for the first tile) $\rightarrow$ this ensures the path is never disconnected

Now the goal becomes: **having a tile on every cell with a gold piece**

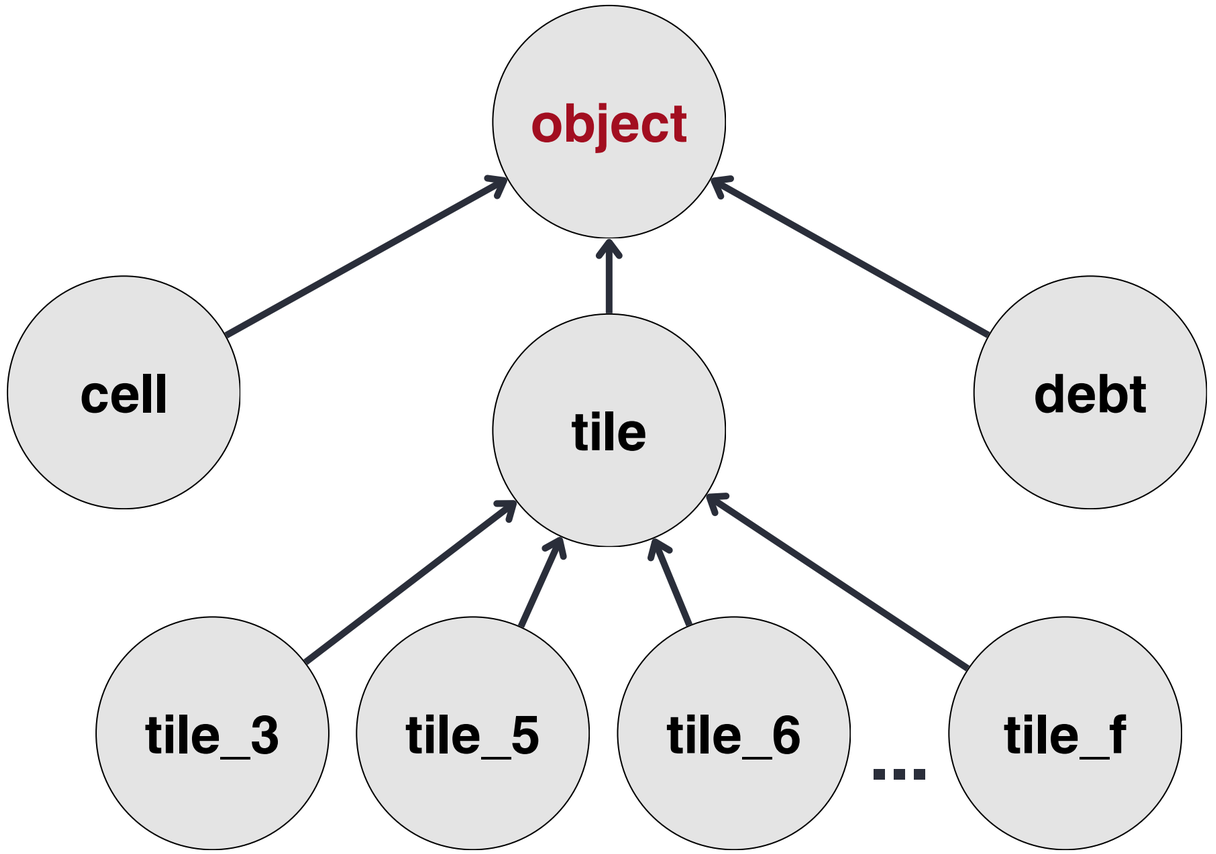
Implementation

Domain: Types

Types

The types *tile_3*, *tile_5*, etc are **sub-types** of the type *tile*.

Because of the way we implemented the domain and problems, we **don't need types to represent the gold and the silver** pieces.



Implementation

Domain: Predicates

Predicates (1)

(**is_above** ?c1 - cell ?c2 - cell)
(**is_right** ?c1 - cell ?c2 - cell)

→ To organize the cells into a grid. (The predicates is_below and is_left are superfluous.)

(**open_left** ?c - cell)
(**open_right** ?c - cell)
(**open_above** ?c - cell)
(**open_below** ?c - cell)

→ To ensure that the tiles build a **sensible path**. A cell is “open-left” if it has a tile that allows it to connect to the cell to its left, E.G. a tile of type 3.

Implementation

Domain: Predicates

Predicates (2)

(**has_tile** ?c - cell)	Indicates that this cell has a tile
(**has_silver** ?c - cell)	Indicates that this cell has a silver that hasn't been picked up yet
(**used** ?t - tile)	Indicates that this tile has been already used
(**first_tile_positioned**)	Indicates that at least 1 tile has been placed

Implementation

Domain: Predicates

Predicates (3)

(**has_debt** ?d - debt) → Indicates that there is a debt *d* that hasn't been paid yet

(**started_paying**) → Indicates that the agent has started paying the debts

The predicate *started_paying* must be false in order to place a tile. In this way we ensure the debts are only paid at the end, and thus we **reduce the number of reachable states** and improve the speed of the planner.

Implementation

Domain:
Action *place_tile_3*

Parameters

- **c - cell**
  where we want to place the tile
- **t - tile_3**
  tile to place

(?**c** - cell ?**t** - tile_3)

Preconditions

- cell c must not already have a tile
- tile t must not be used
- the agent must not have paid any debt yet
- either:
  - this is the first tile positioned
  - or there is an adjacent tile that connects properly to tile t

tile 3

Implementation

Domain:
Action *place_tile_3*

tile 3

Preconditions (PDDL)

```
(not (started_paying))
(not (used ?t))
(not (has_tile ?c))
```

```
(or
    (not (first_tile_positioned))
    (exists (?c2 - cell)
        (or
            (and (is_right ?c ?c2) (open_right ?c2) )
            (and (is_right ?c2 ?c) (open_left ?c2) )
        )
    )
)
```

Implementation

Domain:
Action *place_tile_3*

tile 3

Effect

- cell c has a tile
- tile t is used
- at least one tile has been placed
- cell c is “open” to connections toward its left and its right

- the total cost is increased by 6

Effect (PDDL)

(**has_tile** ?c)
(**used** ?t)
(**first_tile_positioned**)
(**open_right** ?c)
(**open_left** ?c)

(increase (total-cost) 6)

Implementation

Domain:
Action *pay_debt*

Parameters

- **d - debt**
  debt to pay

(?**d** - debt)

Preconditions

- the agent must have debt d

(**has_debt** ?d)

Effect

- the agent doesn't have debt d
- the agent has started paying the debts
- the total cost is increased by 15

(not (**has_debt** ?d))

(**started_paying**)

(increase (total-cost) 15)

Implementation

Domain:
Action *take_silver*

Parameters

- **c - cell**
  cell that contains the silver we're taking
- **d - debt**
  debt that'll be eliminated by taking the silver

(?**c** - cell ?**d** - debt)

Preconditions

- cell c must have a silver
- cell c must have a tile
- the agent must have debt d

(**has_silver** ?c)
(**has_tile** ?c)
(**has_debt** ?d)

Effect

- cell c doesn't have the silver anymore
- the agent doesn't have debt d anymore
- the agent has started paying the debts

(not (**has_silver** ?c))
(not (**has_debt** ?d))
(**started_paying**)

The total cost is not increased, in this way the agent eliminates the debt "for free"

Implementation

Problem: Objects & Init

Objects

- **cell** objects, one for each cell of the grid
- **tile** objects, one for each available tile
- **debt** objects, one for each silver

Initial state

- (**is_right** ?c1 ?c2) predicates, one for each pair of horizontally adjacent cells c1 and c2
- (**is_above** ?c1 ?c2) predicates, one for each pair of vertically adjacent cells c1 and c2
- (**has_silver** ?c) predicates, one for each cell c that has a silver piece
- (= (total-cost) 0)

Implementation

Problem: Goal & Metric

Goal

- (**has_tile** ?c) predicates, one for each cell c that has a gold piece
- (not (**has_debt** ?d)) predicates, one for each debt d

Metric

- **minimize** (total-cost)

As previously said, we must simply ensure that all the cells with a gold piece have a tile and that all the debts have been paid.

Optimizations

Although the solution we presented works, it's **very slow**.

To improve the speed, we:

- generated problems with at most 30 cells
- kept the amount of available tiles pretty low
- in the initial state, **we added a tile of type F in one of the cells containing a gold piece**
  - this allowed us to remove the *first_tile_positioned* predicate, and to reduce the number of reachable states, since the first tile now can't be placed wherever
  - the plans returned now miss an action that places a tile on one of the cells, and thus have a slightly smaller cost than the "real" one, but these effects are negligible

Tests

Adaptations:

- The 5 instances of the problem have changed from the ones in the proposal, since we figured out that they were too complex to solve in terms of time and memory, as you will see later the new ones still have millions of states.
- We also had to change the planner since LM-CUT from IPC 2011 is an optimal planner and it needed a lot of time to execute. We opted instead for **LAMA 2011** which has the advantage of finding sub-optimal solutions in less time and then trying to find better plans over time.
- Finally we have also changed one of the heuristics, in fact we adopted **lm-cut** instead of Context-enhanced additive heuristic, since we wanted to test the optimality of an admissible but not consistent heuristic using A* with reopening.

Tests - LAMA

	Size		Plan 1	Plan 2	Plan 3	Plan 4	Plan 5	Plan 6	Plan 7
1	• Grid: 5x5 • Tiles: 14 • Golds: 3 • Silvers: 3	**Time**	0.0029 s	0.0042 s	0.0152 s	0.08 s	7.1 s	33 s	213 s
		Plan cost	77	58	54	51	47	45	**44**
		States	14	22	162	1087	122.0 k	456.5 k	3.3 millions
2	• Grid: 4x7 • Tiles: 17 • Golds: 2 • Silvers: 3	**Time**	0.0020 s	0.0037 s	0.0173 s	0.33 s	15.7 s	1105 s	
		Plan cost	71	56	54	52	41	**36**	
		States	9	16	172	4125	214.9 k	16.8 millions	
3	• Grid: 4x5 • Tiles: 16 • Golds: 2 • Silvers: 3	**Time**	0.0022 s	0.0030 s	0.0096 s	0.015 s	0.35 s	401 s	
		Plan cost	50	46	41	37	35	**34**	
		States	10	47	223	353	8512	7.9 millions	
4	• Grid: 9x3 • Tiles: 17 • Golds: 3 • Silvers: 3	**Time**	0.0044 s	0.0055 s	0.0192 s	0.084 s	0.82 s		
		Plan cost	83	58	49	47	**38**		
		States	25	20	150	590	7009		
5	• Grid: 4x6 • Tiles: 19 • Golds: 3 • Silvers: 3	**Time**	0.0037 s	0.0060 s	0.0093 s	0.026 s	0.045 s	219 s	260 s
		Plan cost	77	64	62	58	49	47	**40**
		States	18	34	115	353	642	2.4 millions	3.0 millions

Tests - Other

We then tested the instances using **A* and Greedy as search algorithms**, and three different heuristics.

Heuristics:

FF → Neither consistent nor admissible, so it doesn't return an optimal plan

HMAX → Consistent and admissible, so with A* it returns an optimal plan

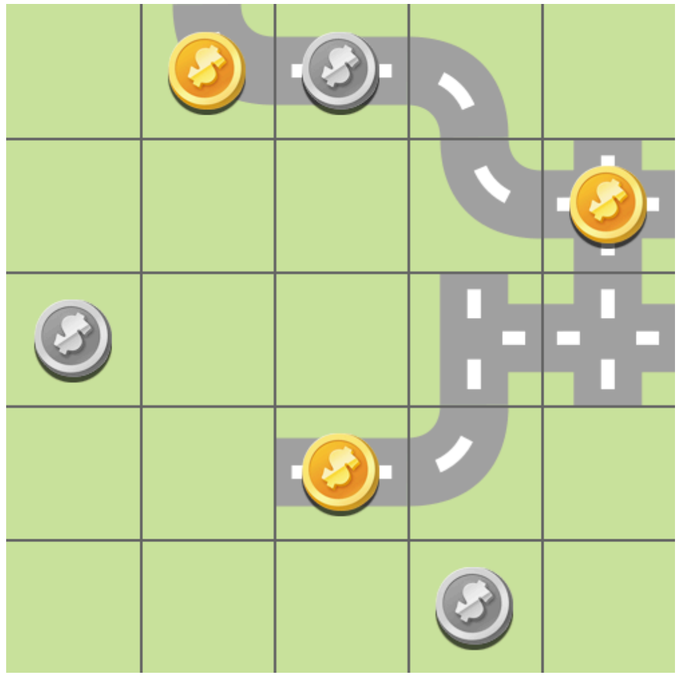
LM CUT → Admissible but not consistent, so it returns an optimal plan only if the A* algorithm reopens closed nodes

Tests - Other

	Size	ff, greedy (no reopening)			hmax, A* (no reopening)			lmcut, A* (reopening)		
		Time	Plan Cost	Memory	Time	Plan Cost	Memory	Time	Plan Cost	Memory
1	• Grid: 5x5 • Tiles: 14 • Golds: 3 • Silvers: 3	0.031s	73	11.59 MB 510 states	23m 20s	44	2.5 GB 14 millions states	12m 30s	44	1.0 GB 1.4 millions states
2	• Grid: 4x7 • Tiles: 17 • Golds: 2 • Silvers: 3	0.048s	51	11.75 MB 318 states	46m 32s	36	10.69 GB 60.9 millions states	50m 34s	36	9.97 GB 57.0 millions states
3	• Grid: 4x5 • Tiles: 16 • Golds: 2 • Silvers: 3	0.114s	50	11.59 MB 2494 states	30m 50s	34	4.4 GB 14 millions states	12m 50s	34	1.2 GB 2.5 millions states
4	• Grid: 9x3 • Tiles: 17 • Golds: 3 • Silvers: 3	0.045s	62	11.55 MB 692 states	7m 37s	38	1.81 GB 8.2 millions states	8m 45s	38	1.22 GB 5.5 millions states
5	• Grid: 4x6 • Tiles: 19 • Golds: 3 • Silvers: 3	0.029s	78	11.57 MB 520 states	27m 18s	40	6.17 GB 28.9 millions states	26m 17s	40	4.28 GB 20.0 millions states

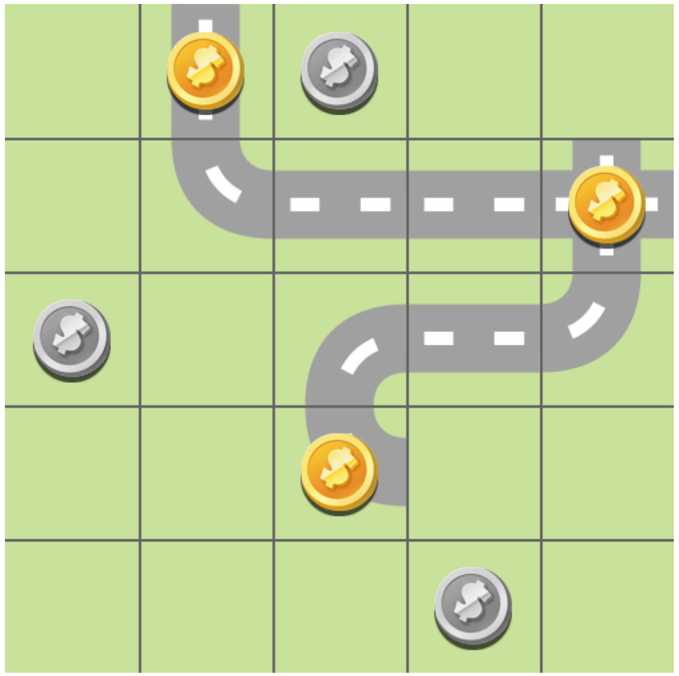
Tests

Greedy FF



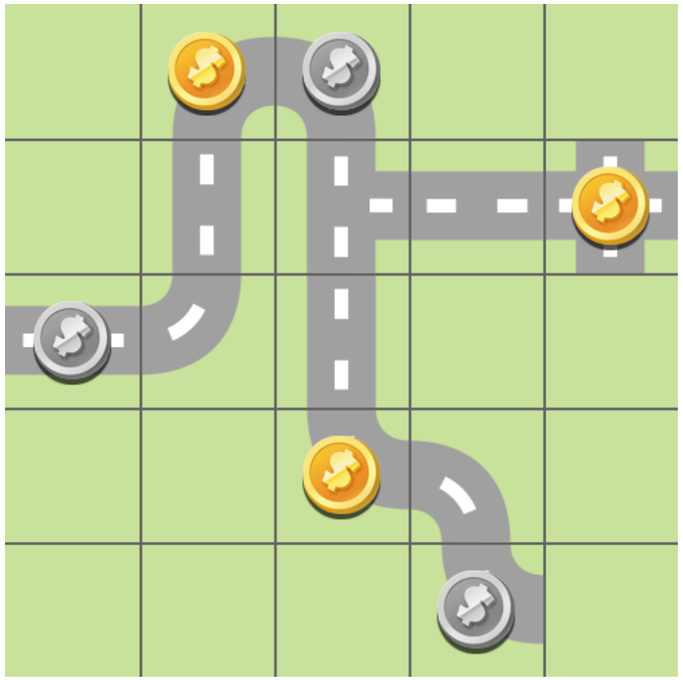
Cost: 73

LAMA 2011 (first plan)



Cost: 77

A* hmax, A* lm-cut,
LAMA 2011 (best plan)



Cost: 44

Thank you!