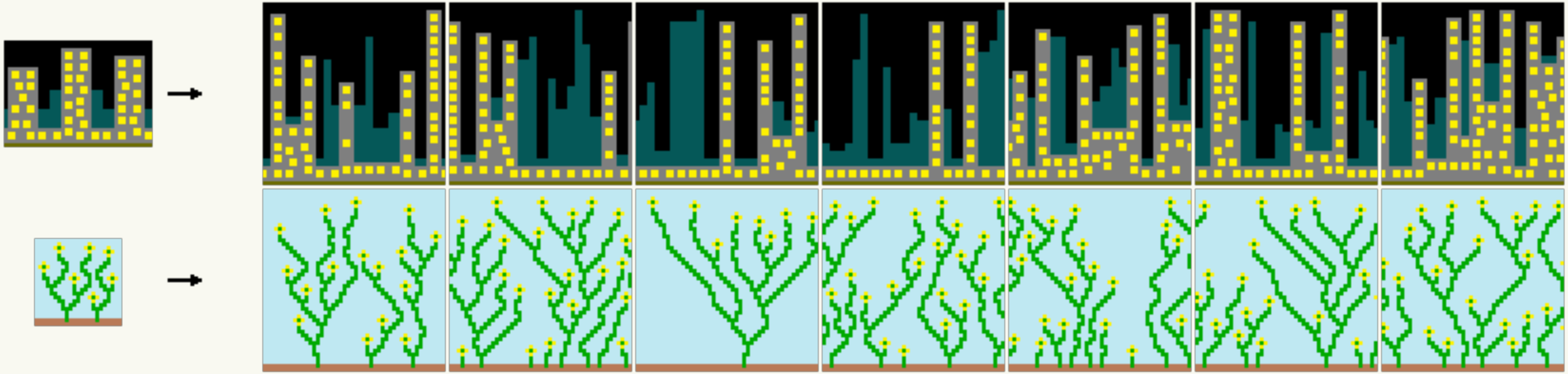
GDC
March 20-24, 2023
San Francisco, CA

Beyond WaveFunctionCollapse: Constraint-Based Tile Map Generation and Editing

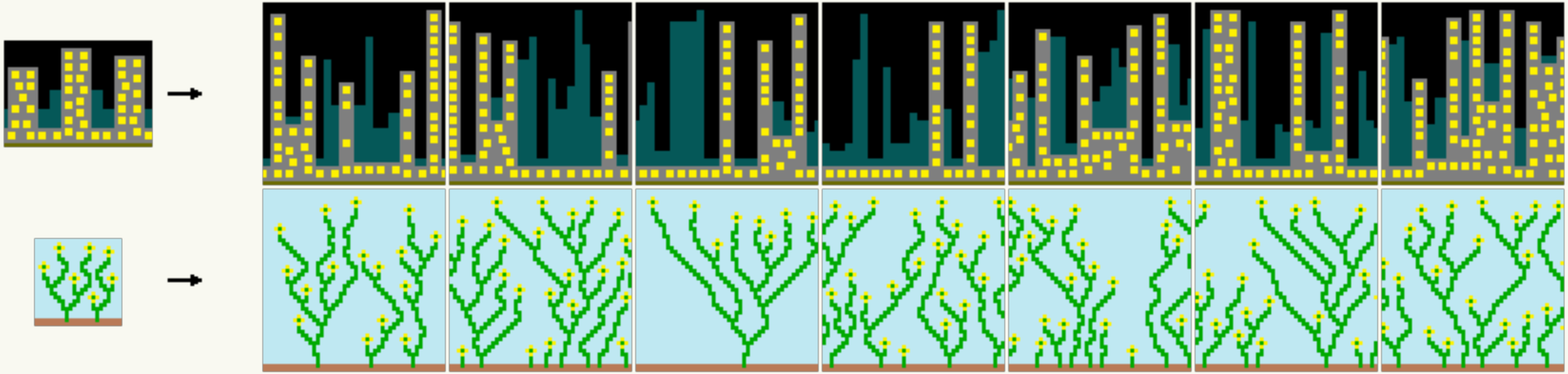
Seth Cooper
Northeastern University

WaveFunctionCollapse



[Maxim Gumin, https://github.com/mxgmn/WaveFunctionCollapse]

WaveFunctionCollapse



[Maxim Gumin, https://github.com/mxgmn/WaveFunctionCollapse]

Constraint-based Image (and level) generation

WaveFunctionCollapse

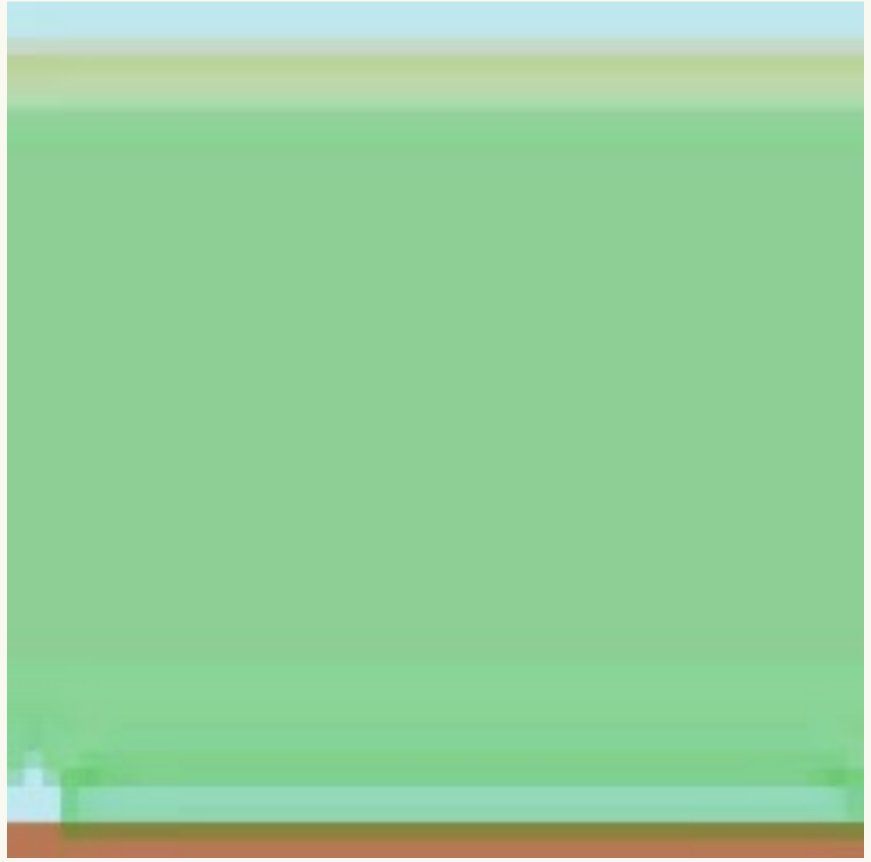
Constraints

- Generated images should only contain NxM (e.g. 3x3) patterns from example image (hard)
- The distribution of patterns in generated images should be similar to that in the input (soft)

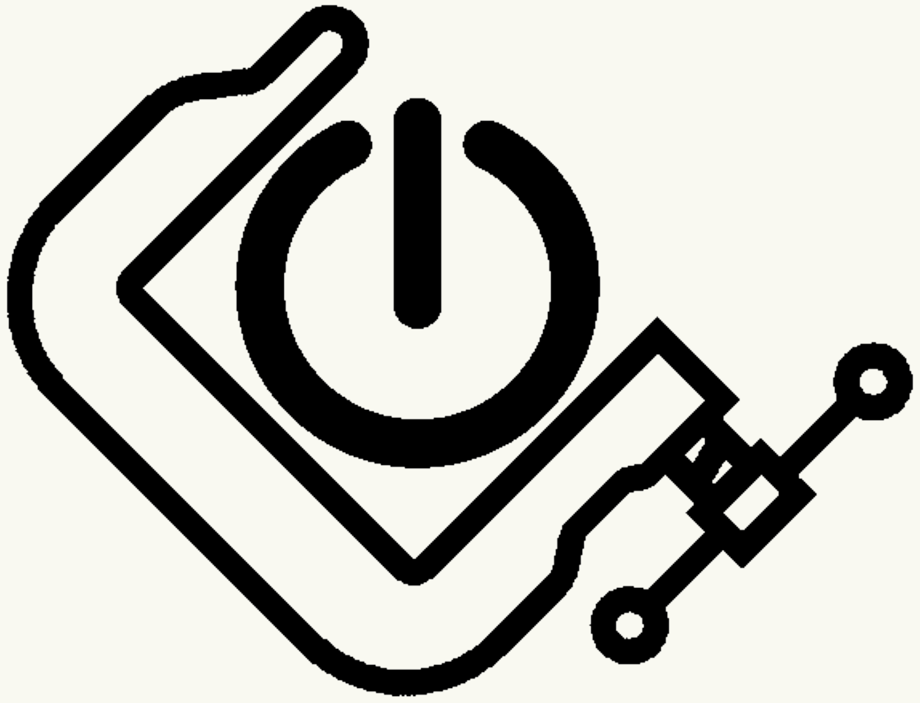
WaveFunctionCollapse

Solution Algorithm (roughly)

- Initialize grid so that all patterns can be at all locations
- Repeat:
  - Observation: pick possible pattern to go at a specific location
  - Propagation: update remaining possible patterns at other locations
- Until:
  - Every location has a pattern -> done
  - Some location has no possible patterns -> stuck

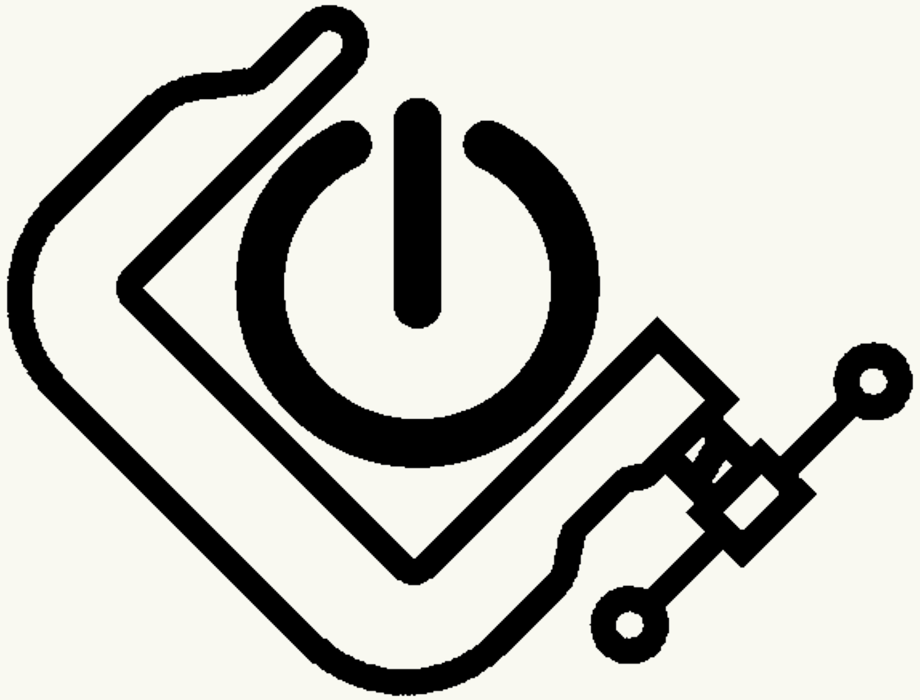


Constraint-Based Generation



- Express *what* should be in a level (maybe by a few examples) rather than *how* to generate it.
- Could decouple constraints and solver, “plug in” standard constraint solvers.
- “Modular” combination of constraints.

Constraint-Based Generation



- Sturgeon level generation system
- Example levels and applications
- More extensions

Constraint-Based Generation

Sturgeon

System for (generally 2D, tile-based) level generation and editing
via (Boolean) constraint solving

Constraint-Based Generation

Sturgeon

System for (generally 2D, tile-based) level generation and editing
via (Boolean) constraint solving

Set up generic (Boolean) constraint problem

Constraint-Based Generation

Sturgeon

System for (generally 2D, tile-based) level generation and editing
via (Boolean) constraint solving

Set up generic (Boolean) constraint problem

Give to low-level solver

Takes collection of of Boolean variables and constraints
Returns true/false assignment for variables that satisfies constraints
(all hard, as many soft as possible)

Constraint-Based Generation

Sturgeon

System for (generally 2D, tile-based) level generation and editing
via (Boolean) constraint solving

Set up generic (Boolean) constraint problem

Give to low-level solver

SAT-style [PySAT]; SMT; Answer Set; portfolio

Constraint-Based Generation

Sturgeon

System for (generally 2D, tile-based) level generation and editing
via (Boolean) constraint solving

Set up generic (Boolean) constraint problem

Give to low-level solver

SAT-style [PySAT]; SMT; Answer Set; portfolio

Process solution into a level

Constraint-Based Generation

Sturgeon
System for (generally 2D, tile-based) level generation and editing
via (Boolean) constraint solving

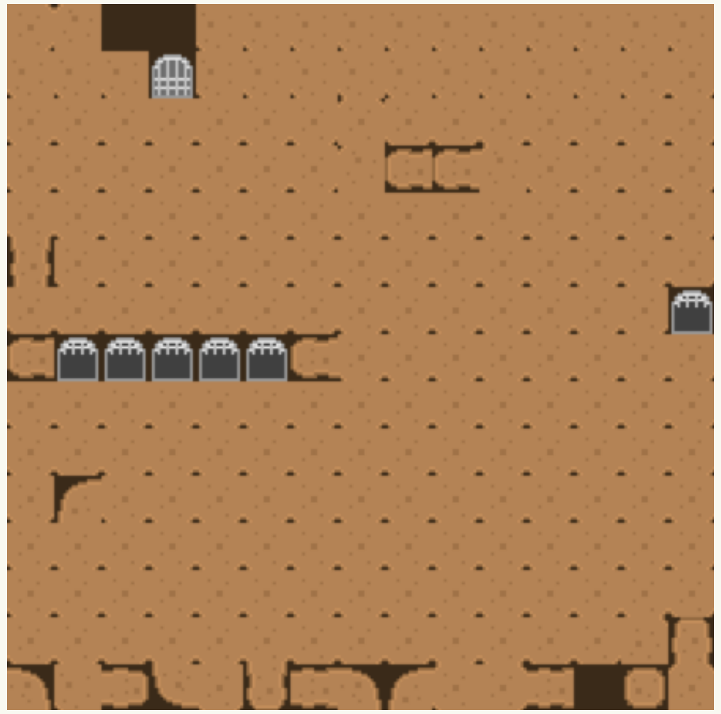
Set up generic (Boolean) constraint problem

Give to low-level solver
SAT-style [PySAT]; SMT; Answer Set; portfolio

Process solution into a level

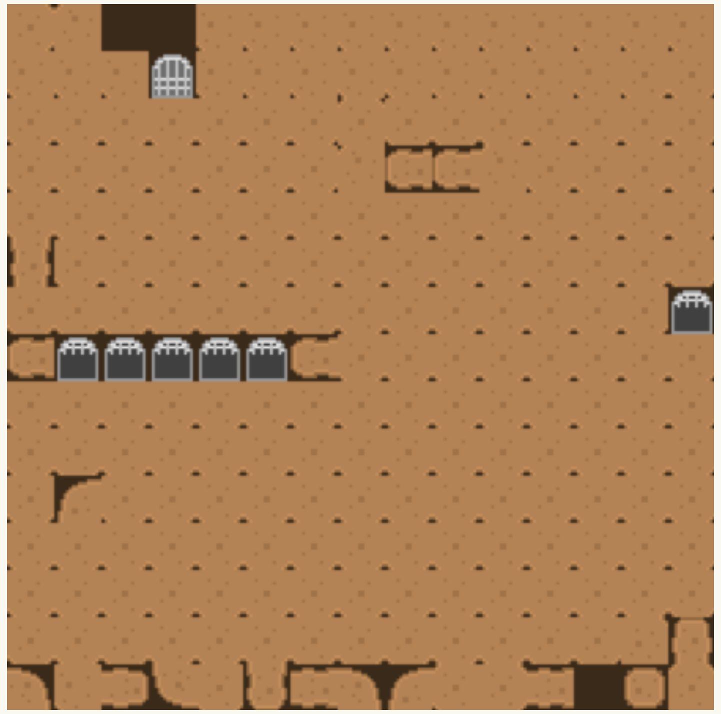
What constraints does Sturgeon use to generate a level?

Constraint-Based Generation



Tile

Constraint-Based Generation

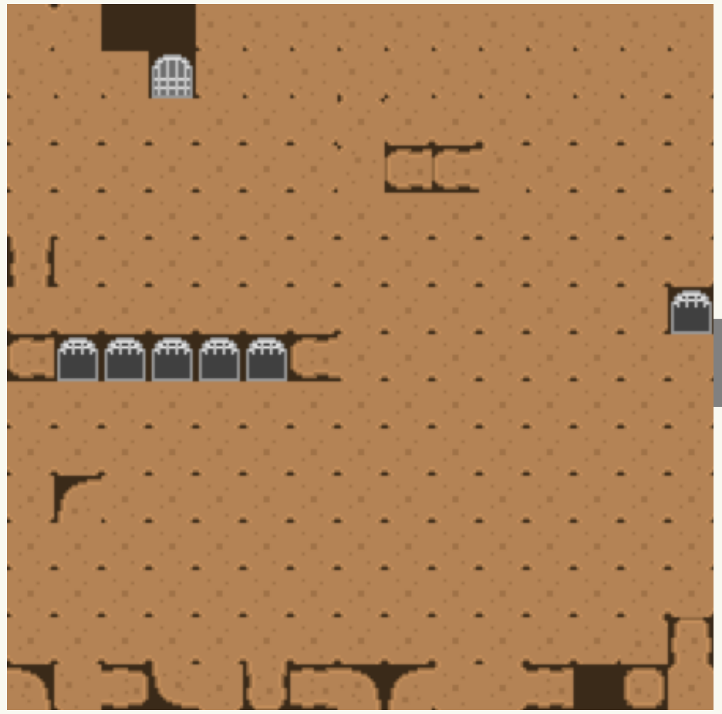


Tile

$\underbrace{\hspace{5cm}}$

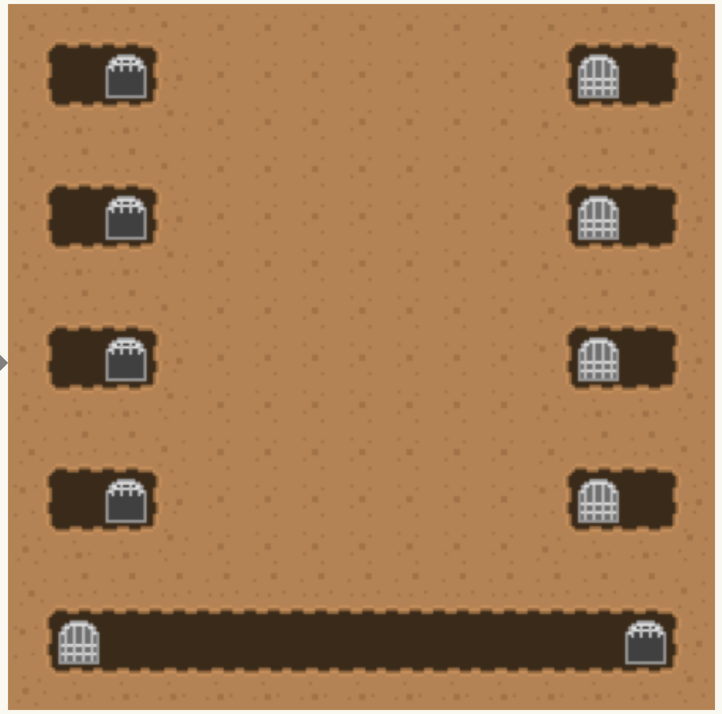
(like WFC solver)

Constraint-Based Generation



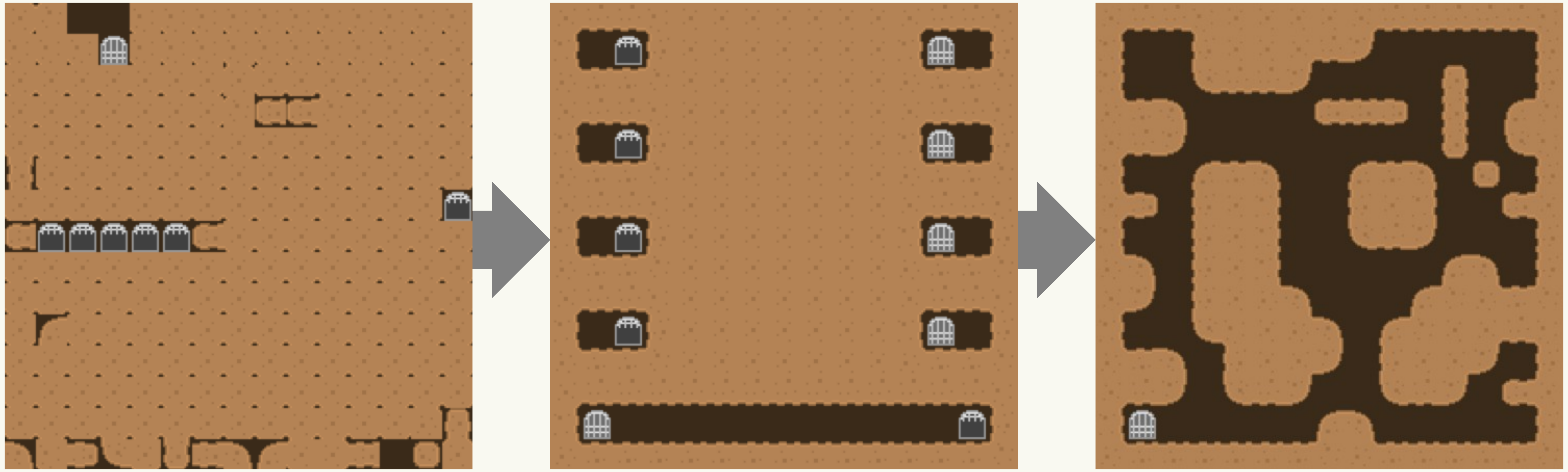
Tile

(like WFC solver)



Pattern

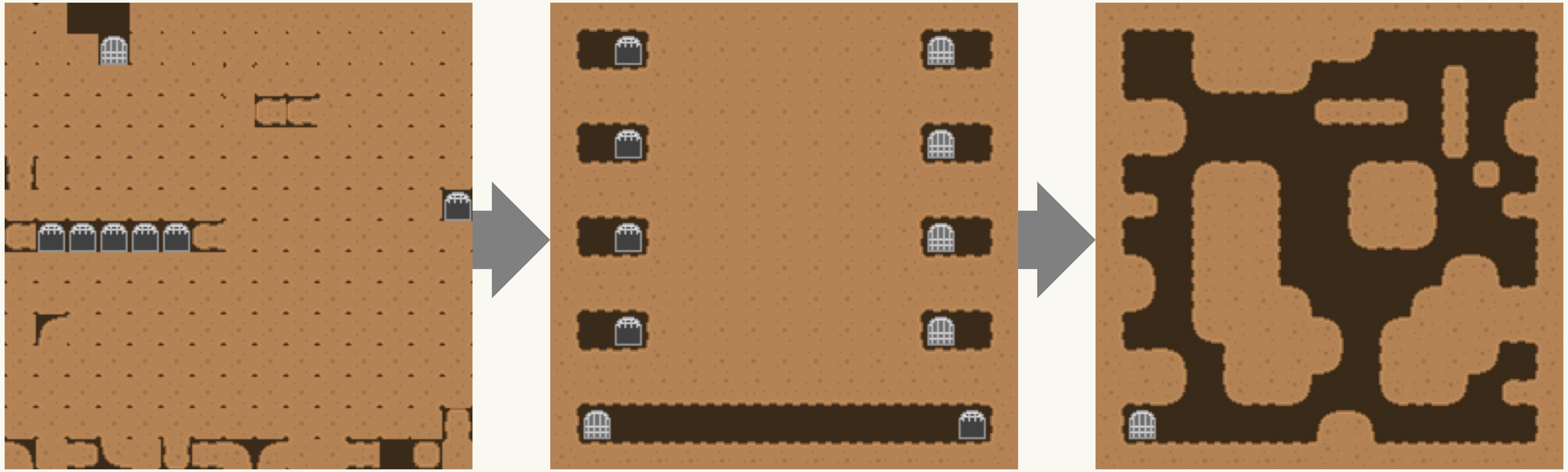
Constraint-Based Generation



Tile — Pattern — Distribution

(like WFC solver)

Constraint-Based Generation

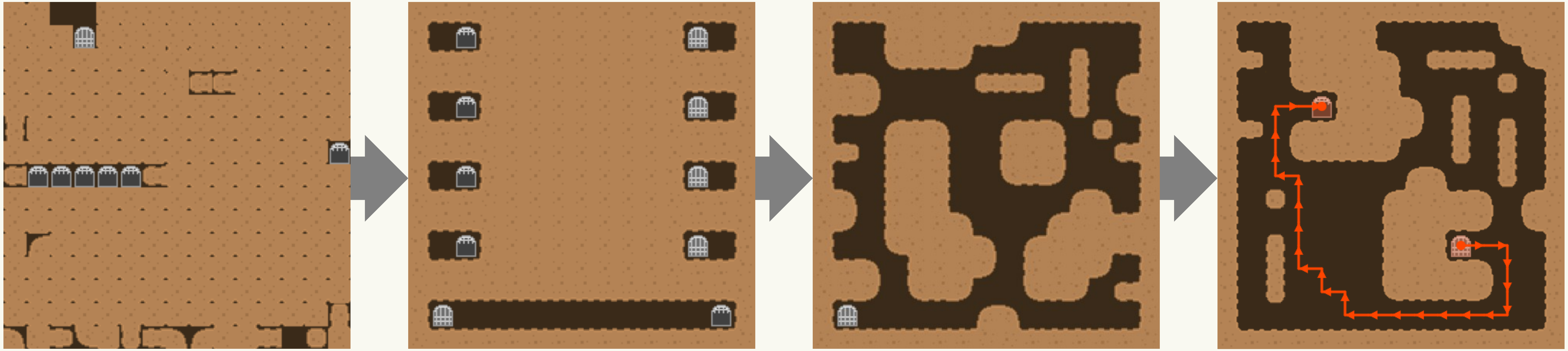


Tile | Pattern | Distribution

(like WFC solver)

(like WFC constraints)

Constraint-Based Generation



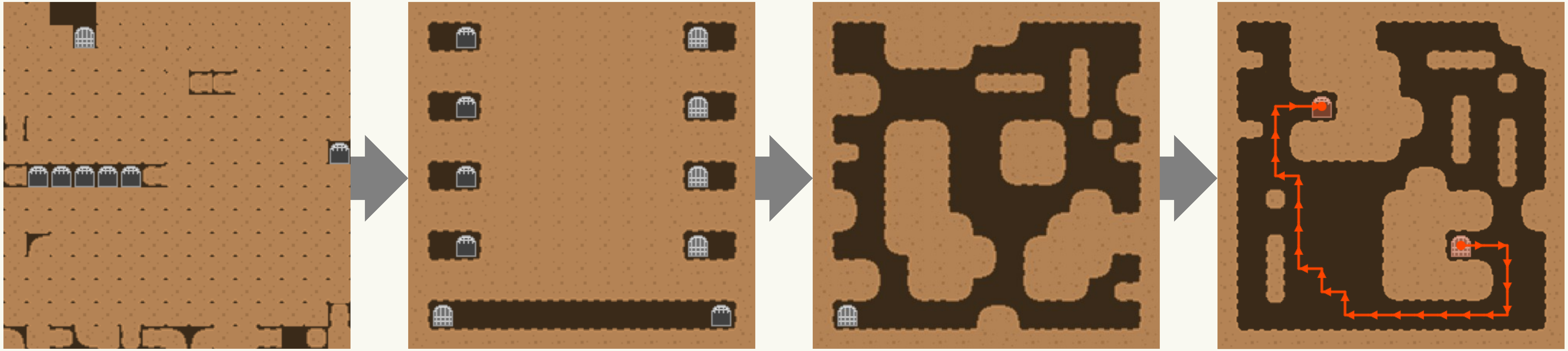
Tile | Pattern | Distribution | Reachability

(like WFC solver)

(like WFC constraints)

(beyond...)

Constraint-Based Generation



Tile

Pattern

Distribution

Reachability

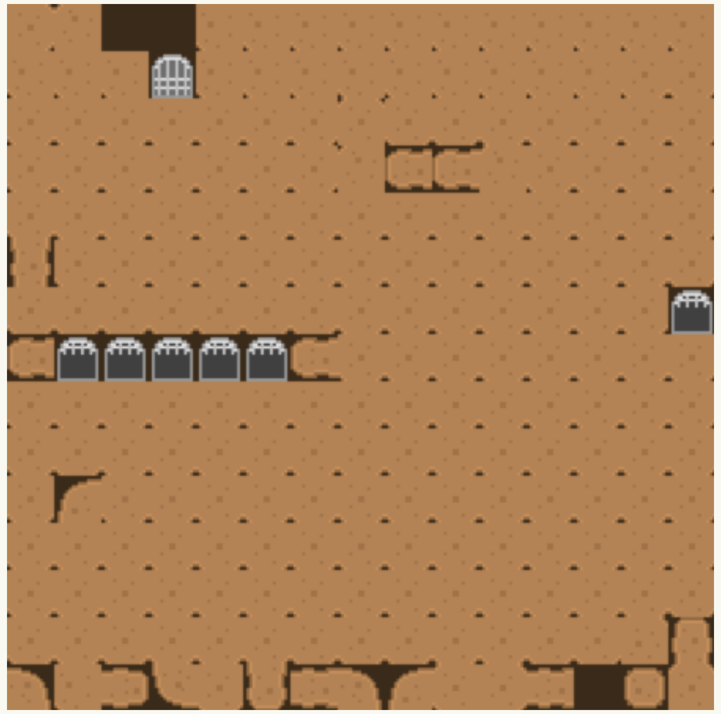
(like WFC solver)

(like WFC constraints)

(beyond...)

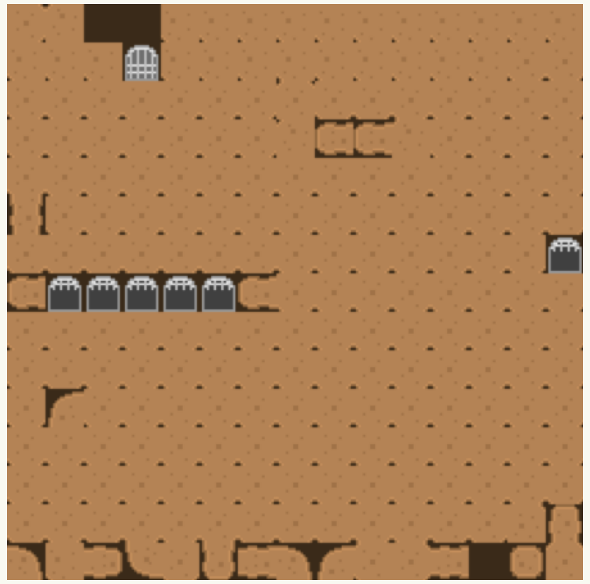
Take a closer look at how Sturgeon sets up and uses these constraints...

Tile Constraints



Tile

Tile Constraints

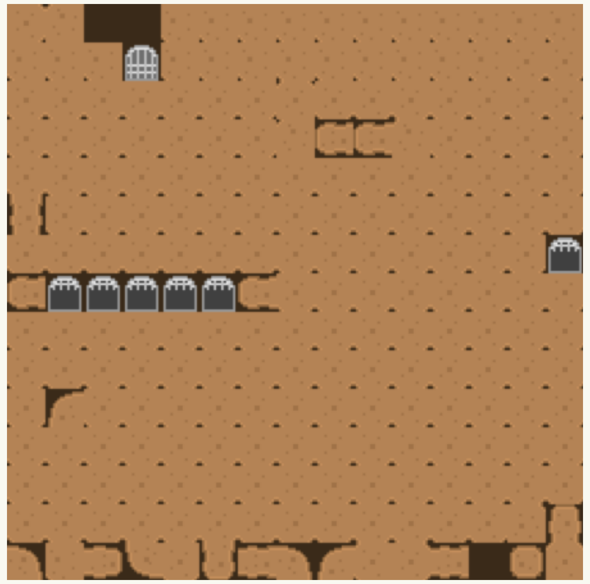


Interface

Outline

Setup: make a var at each location, for each possible tile there.

Tile Constraints



Outline

Setup: make a var at each location, for each possible tile there.

```
tile = MakeVar()
```

Interface

```
MakeVar()
```

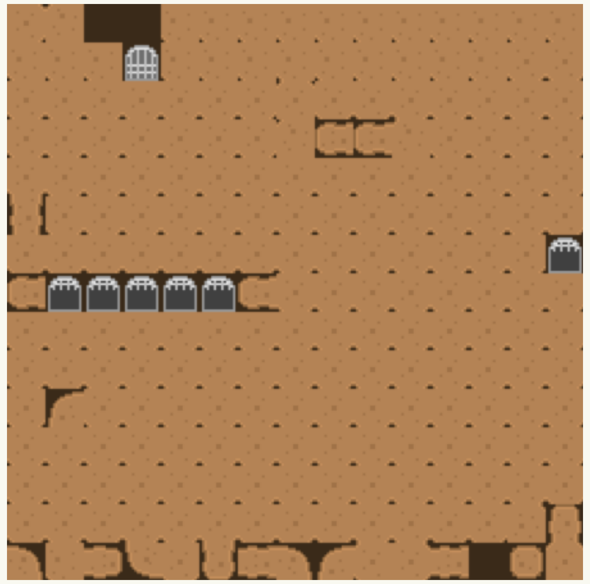
Tile Constraints

<u>Outline</u>

Setup: make a var at each location, for each possible tile there.

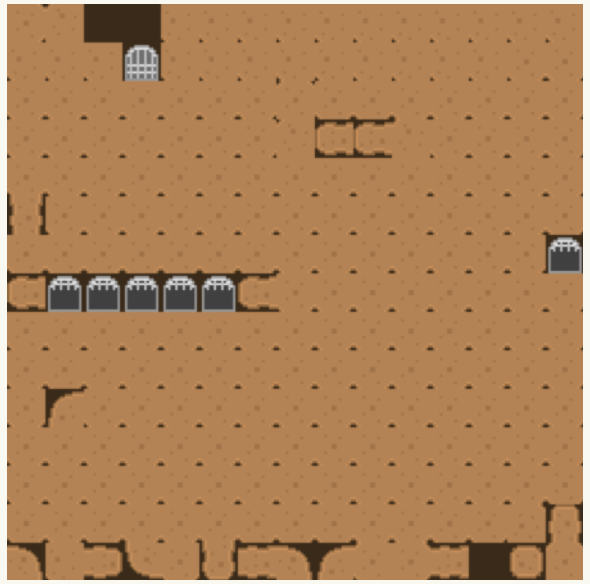
```
tile = MakeVar()
```

Setup: exactly 1 var can be true at each location.



<u>Interface</u>
`MakeVar()`

Tile Constraints



Outline

Setup: make a var at each location, for each possible tile there.

```
tile = MakeVar()
```

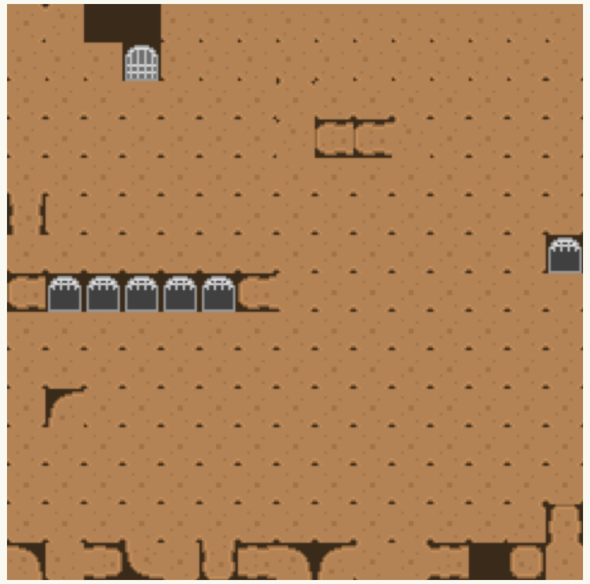
Setup: exactly 1 var can be true at each location.

```
CnstrCount(tileVarsAtLocation, 1, 1, HARD)
```

Interface

```
MakeVar()
CnstrCount(...)
```

Tile Constraints



Outline

Setup: make a var at each location, for each possible tile there.

```
tile = MakeVar()
```

Setup: exactly 1 var can be true at each location.

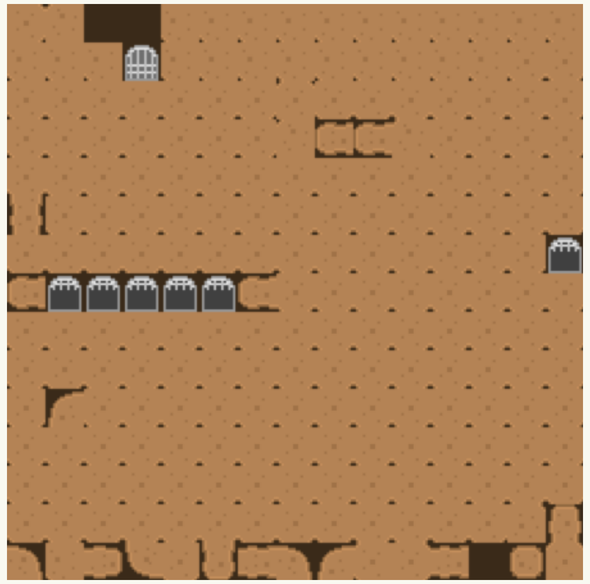
```
CnstrCount(tileVarsAtLocation, 1, 1, HARD)
```

Find a solution.

Interface

```
MakeVar()
CnstrCount(...)
```

Tile Constraints



Outline

Setup: make a var at each location, for each possible tile there.

```
tile = MakeVar()
```

Setup: exactly 1 var can be true at each location.

```
CnstrCount(tileVarsAtLocation, 1, 1, HARD)
```

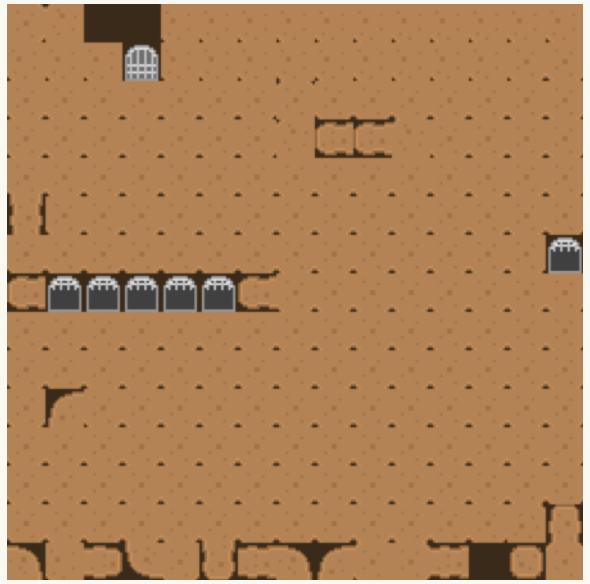
Find a solution.

```
Solve()
```

Interface

```
MakeVar()
CnstrCount(...)
Solve()
```

Tile Constraints



Outline

Setup: make a var at each location, for each possible tile there.

```
tile = MakeVar()
```

Setup: exactly 1 var can be true at each location.

```
CnstrCount(tileVarsAtLocation, 1, 1, HARD)
```

Find a solution.

```
Solve()
```

Process solution: at each location, find the var set to true.

Interface

```
MakeVar()
CnstrCount(...)
Solve()
```

Tile Constraints

<u>Outline</u>

Setup: make a var at each location, for each possible tile there.

```
tile = MakeVar()
```

Setup: exactly 1 var can be true at each location.

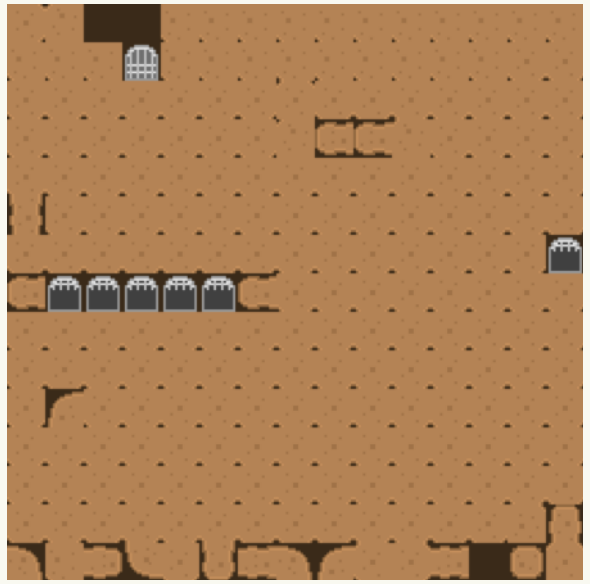
```
CnstrCount(tileVarsAtLocation, 1, 1, HARD)
```

Find a solution.

```
Solve()
```

Process solution: at each location, find the var set to true.

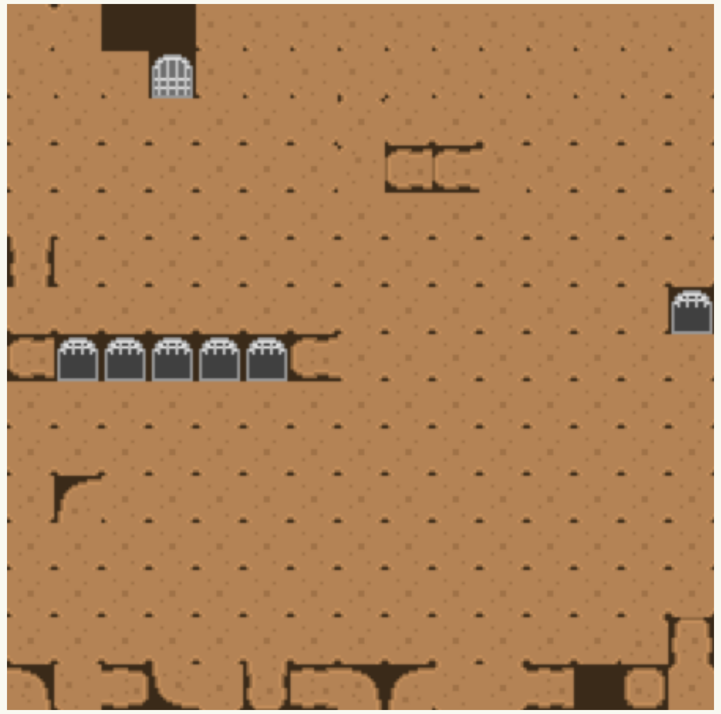
```
GetVar(tileVar)
```



<u>Interface</u>

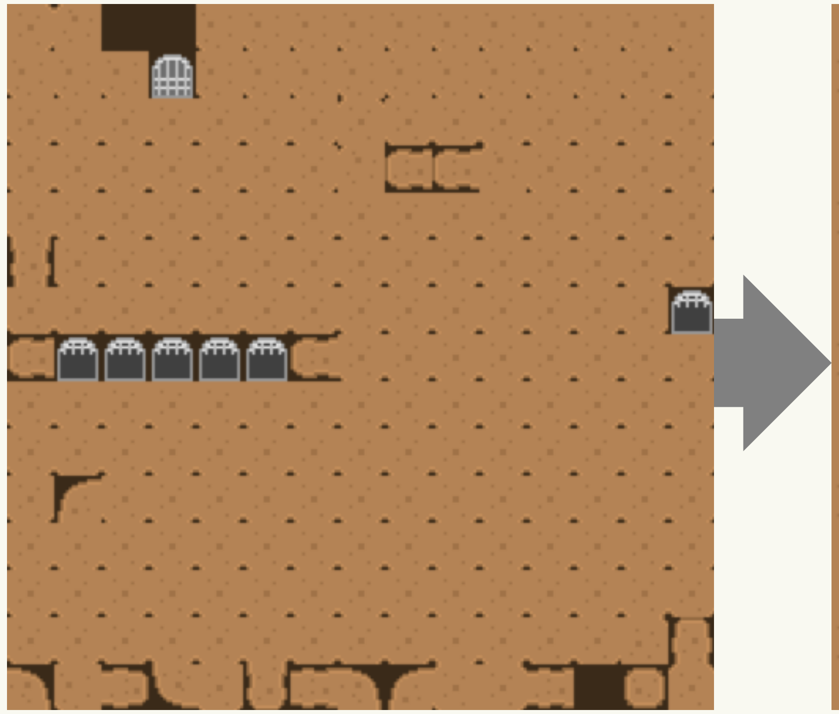
```
MakeVar()
CnstrCount(...)
Solve()
GetVar(...)
```

Pattern Constraints

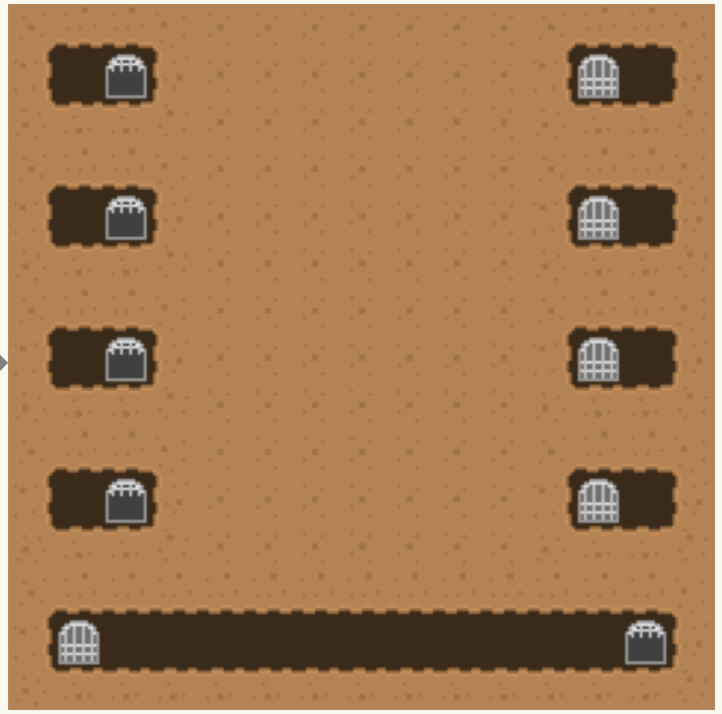


Tile

Pattern Constraints

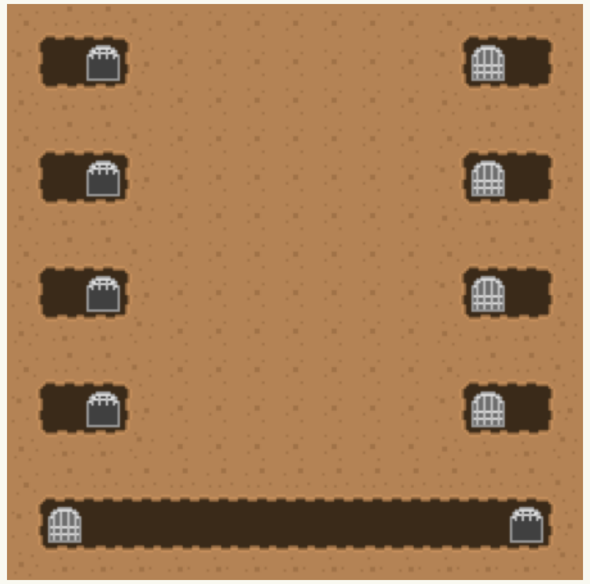


Tile



Pattern

Pattern Constraints

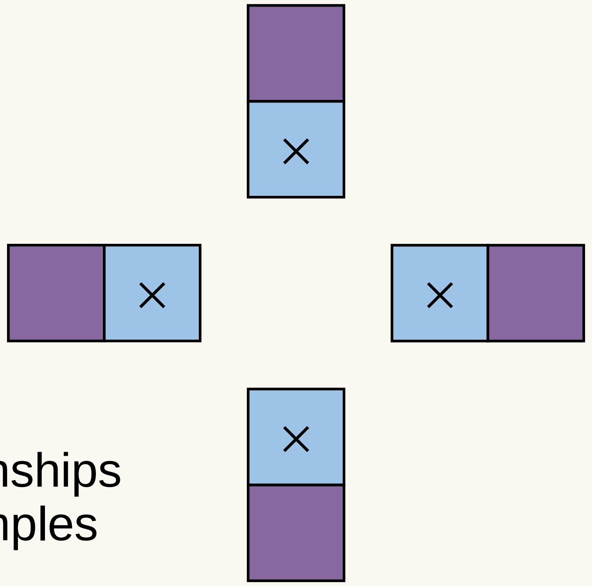


Outline

Setup tile constraints.

Setup: at each location, using an example level and pattern template,

Pattern template



What local relationships
to use from examples

Example level

Find and process solution.

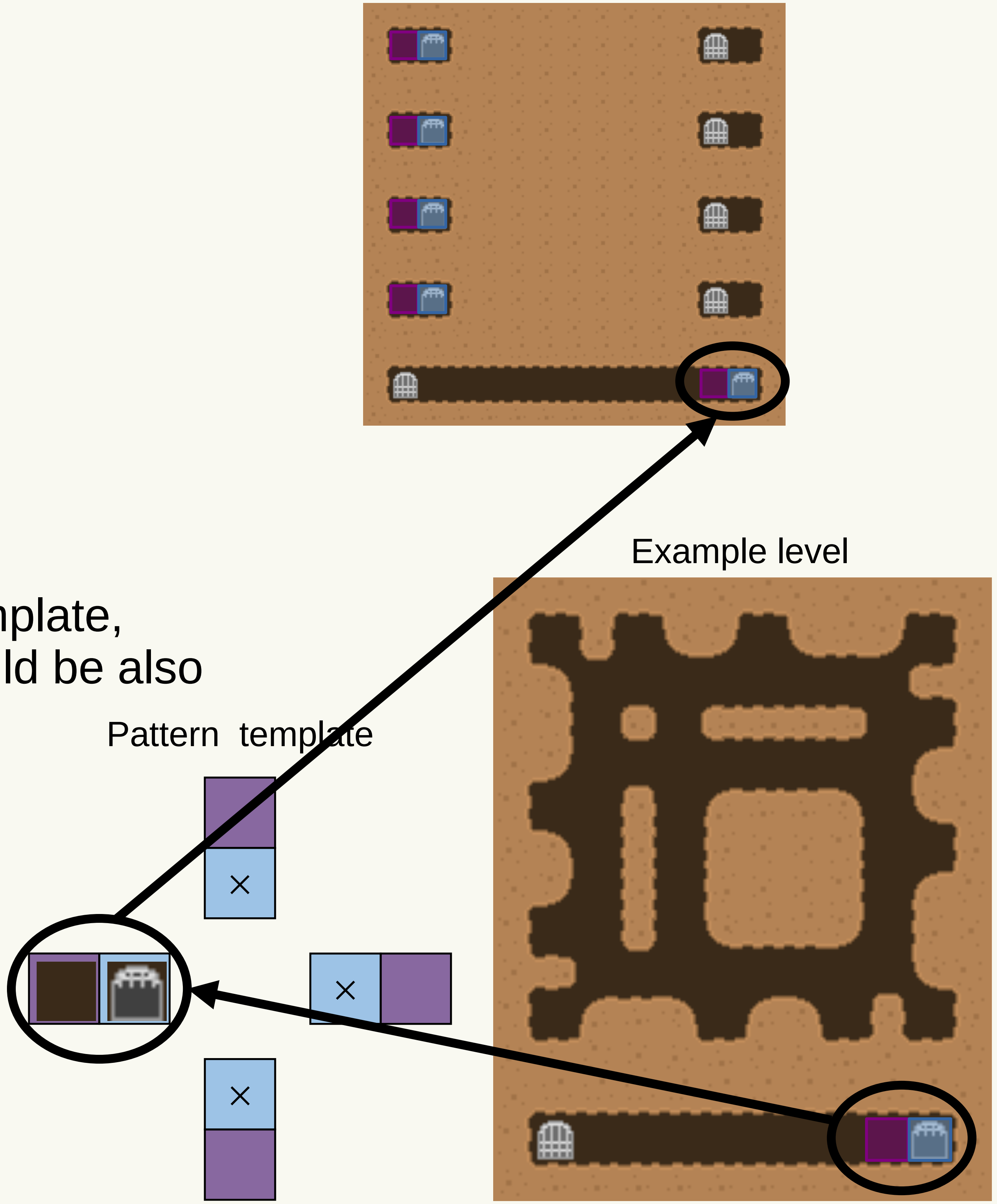
Pattern Constraints

<u>Outline</u>

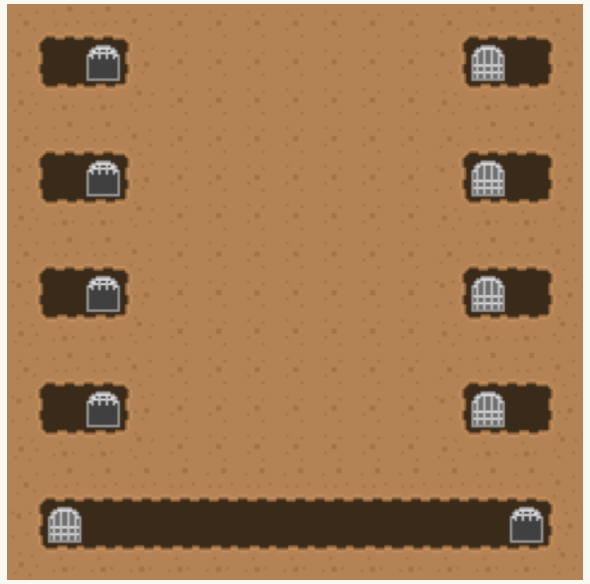
Setup tile constraints.

Setup: at each location, using an example level and pattern template,
an **input pattern** there means a relative **output pattern** should be also

Find and process solution.



Pattern Constraints



Interface
MakeVar()
CnstrCount(...)
Solve()
GetVar(...)
CnstrImpliesOr(...)

Outline

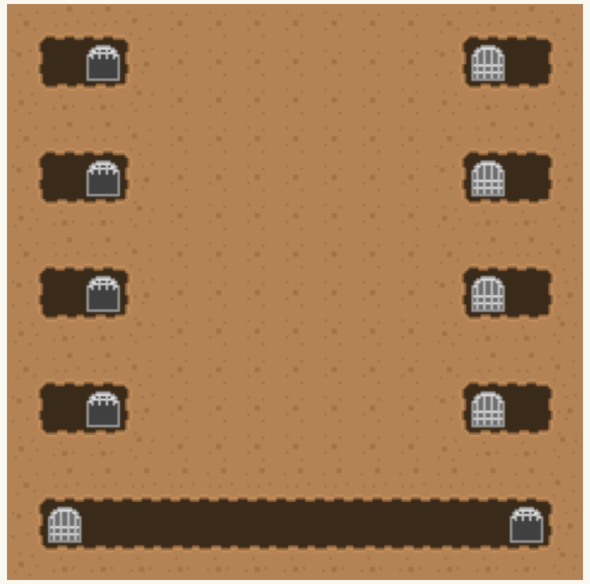
Setup tile constraints.

Setup: at each location, using an example level and pattern template,
an **input pattern** there means a relative **output pattern** should be also

```
CnstrImpliesOr(inPattern,
               outPatternsSeen, HARD)
```

Find and process solution.

Pattern Constraints



Outline

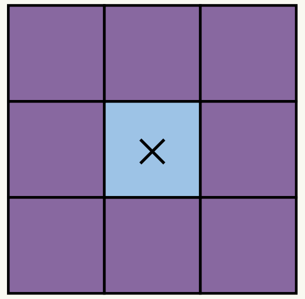
Setup tile constraints.

Setup: individual tile variables can be organized into *patterns* by *templates.*

Setup: at each location, using an example level and pattern template,
an **input pattern** there means a relative **output pattern** should be also

```
CnstrImpliesOr(inPattern,
               outPatternsSeen, HARD)
```

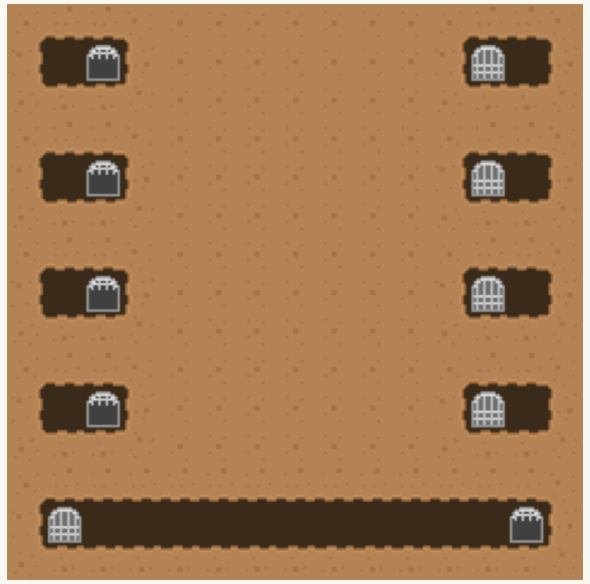
Pattern template



Find and process solution.

Interface
`MakeVar()`
`CnstrCount(...)`
`Solve()`
`GetVar(...)`
`CnstrImpliesOr(...)`

Pattern Constraints



Outline

Setup tile constraints.

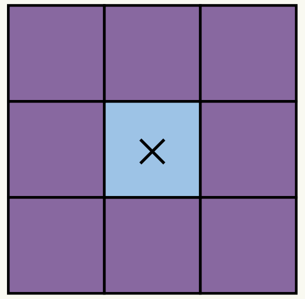
Setup: individual tile variables can be organized into *patterns* by *templates.*

```
pattern = MakeAnd(patternTileVars)
```

Setup: at each location, using an example level and pattern template,
an **input pattern** there means a relative **output pattern** should be also

```
CnstrImpliesOr(inPattern,
               outPatternsSeen, HARD)
```

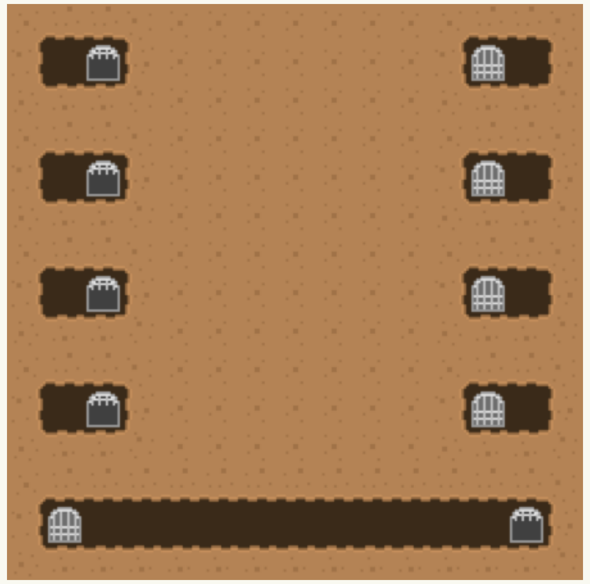
Pattern template



Find and process solution.

Interface
MakeVar()
CnstrCount(...)
Solve()
GetVar(...)
CnstrImpliesOr(...)
MakeAnd(...)

Pattern Constraints



Outline

Setup tile constraints.

Setup: individual tile variables can be organized into *patterns* by *templates.*

```
pattern = MakeAnd(patternTileVars)
```

Setup: at each location, using an example level and pattern template,
an **input pattern** there means a relative **output pattern** should be also

```
CnstrImpliesOr(inPattern,
               outPatternsSeen, HARD)
```

Setup: at each location, using templates, at least one **input pattern** must exist.

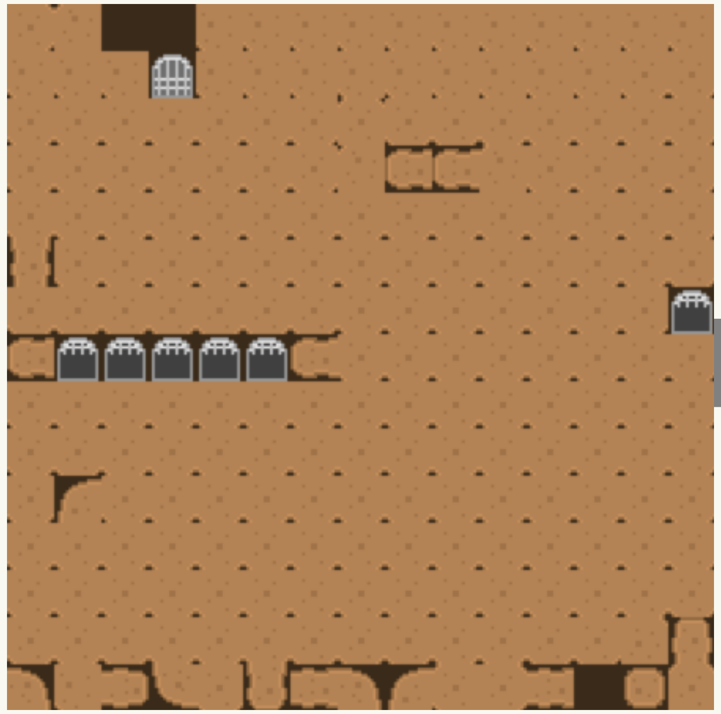
```
CnstrCount(inPatternsAtLocation, 1, inf, HARD)
```

Find and process solution.

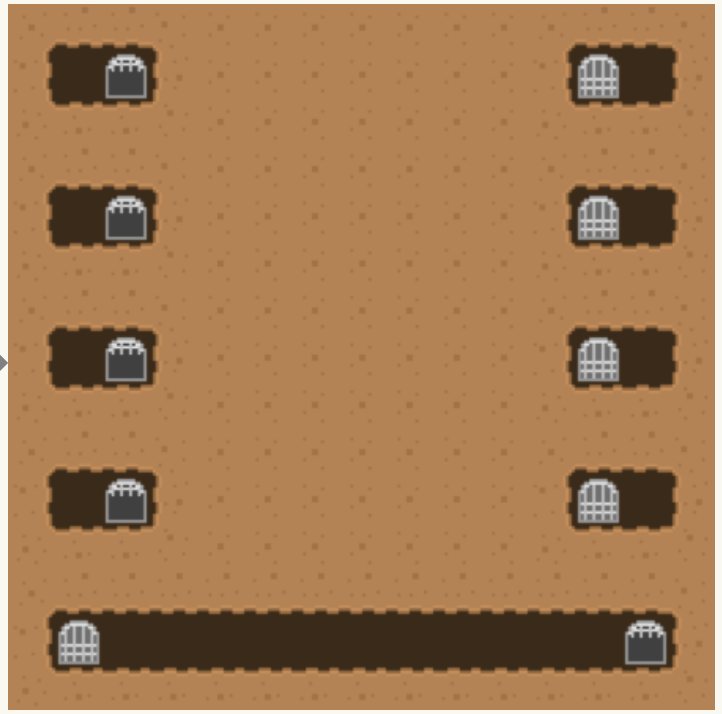
Interface

```
MakeVar()
CnstrCount(...)
Solve()
GetVar(...)
CnstrImpliesOr(...)
MakeAnd(...)
```

Distribution Constraints

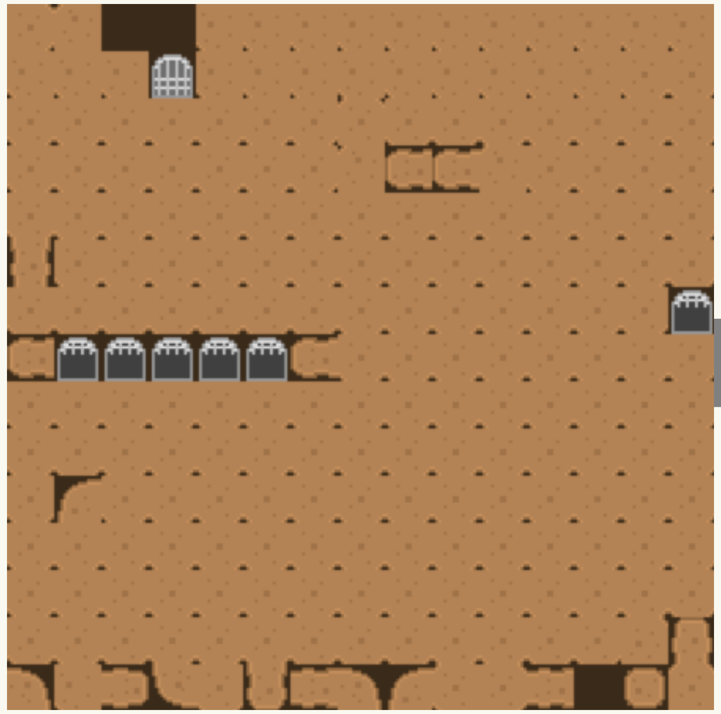


Tile

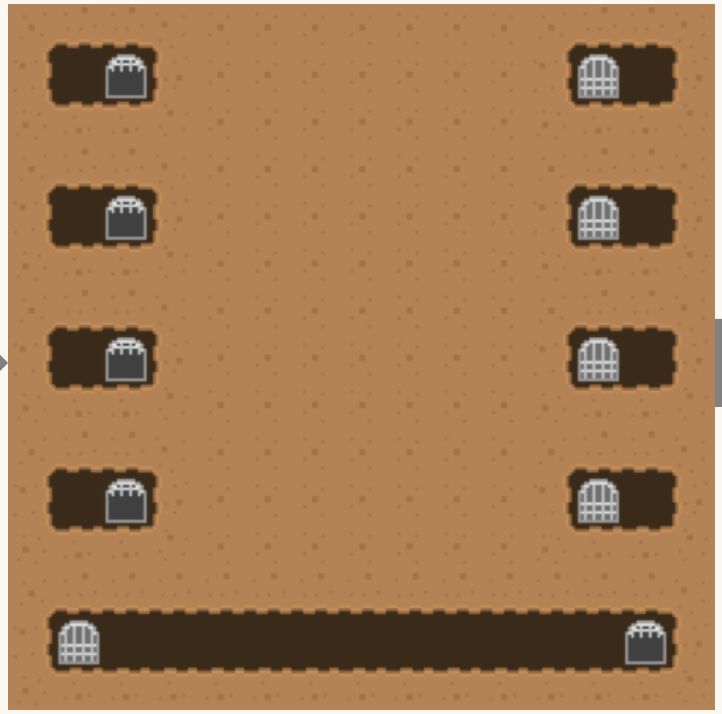


Pattern

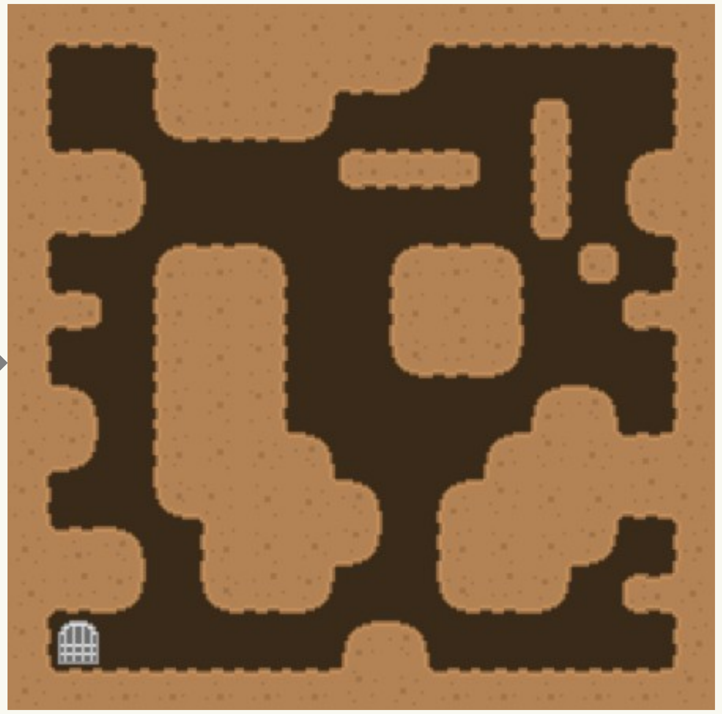
Distribution Constraints



Tile



Pattern



Distribution

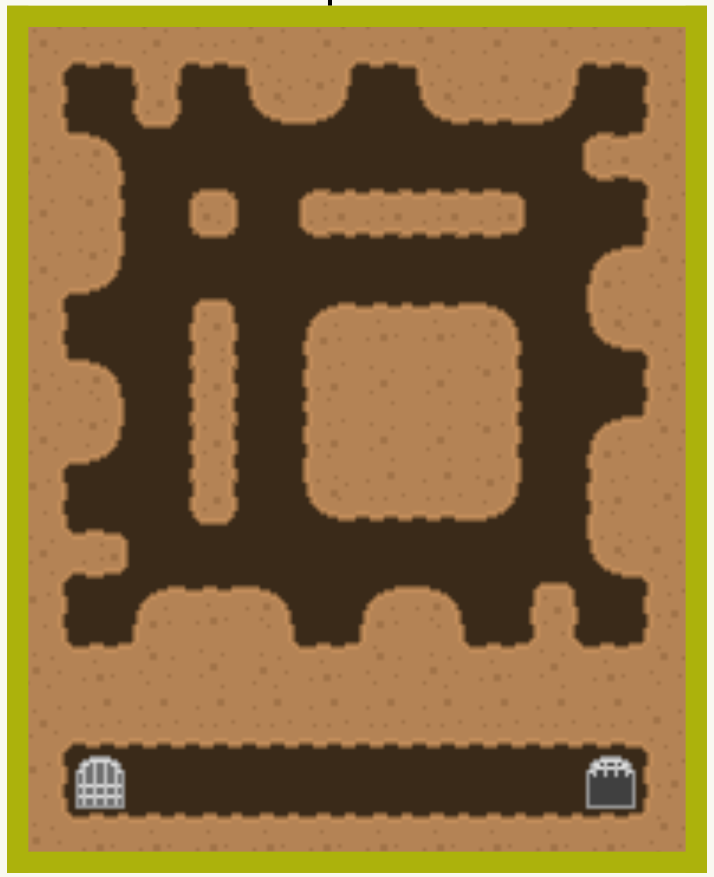
Distribution Constraints

Outline

Setup tile and pattern constraints.

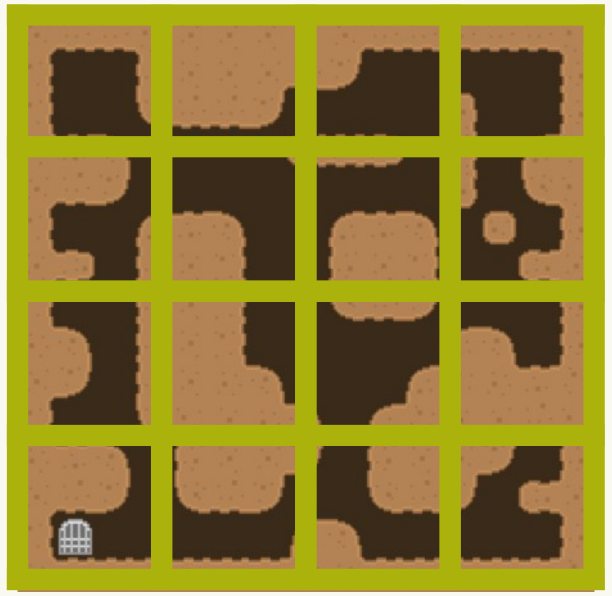
Setup: for each corresponding **region**, for each tile type,
the counts should be similar to the example.

Example level



Find and process solution.

Distribution Constraints



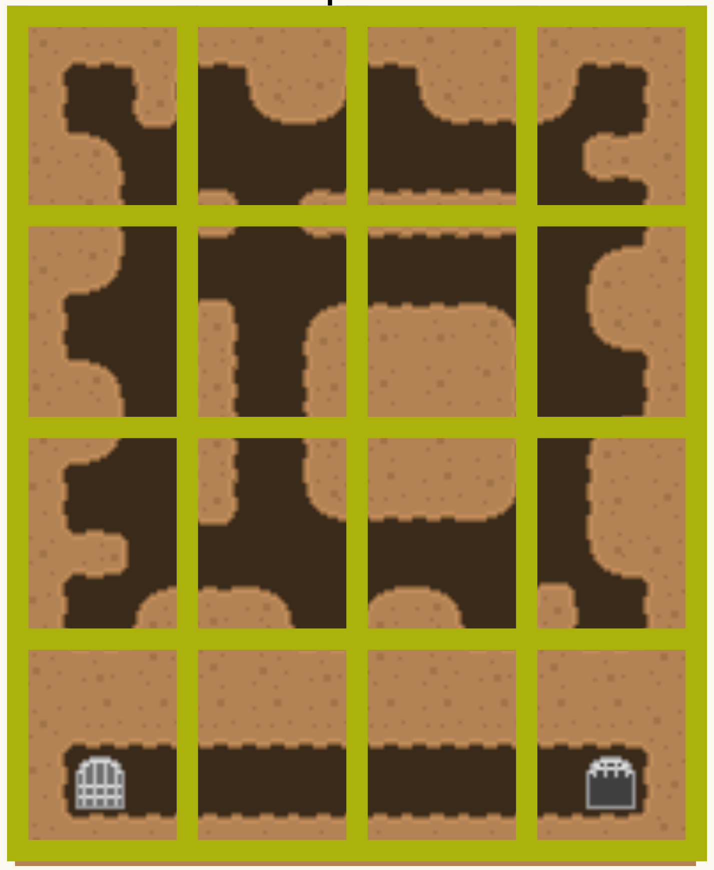
Outline

Setup tile and pattern constraints.

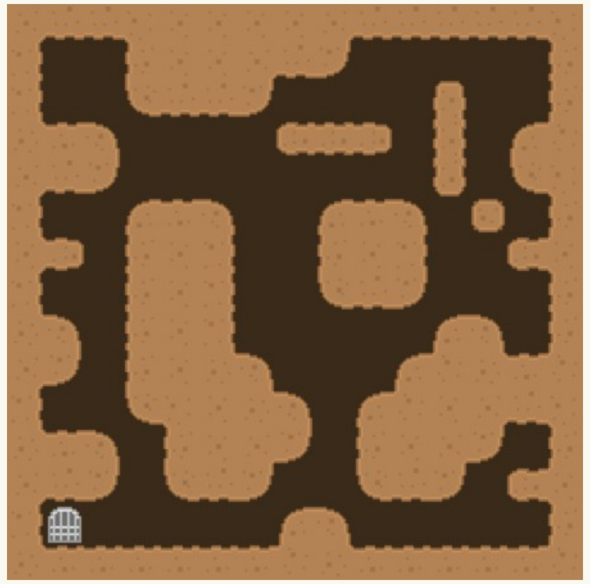
Setup: for each corresponding **region**, for each tile type, the counts should be similar to the example.

Find and process solution.

Example level



Distribution Constraints



Outline

Setup tile and pattern constraints.

Setup: for each corresponding **region**, for each tile type,
the counts should be similar to the example.

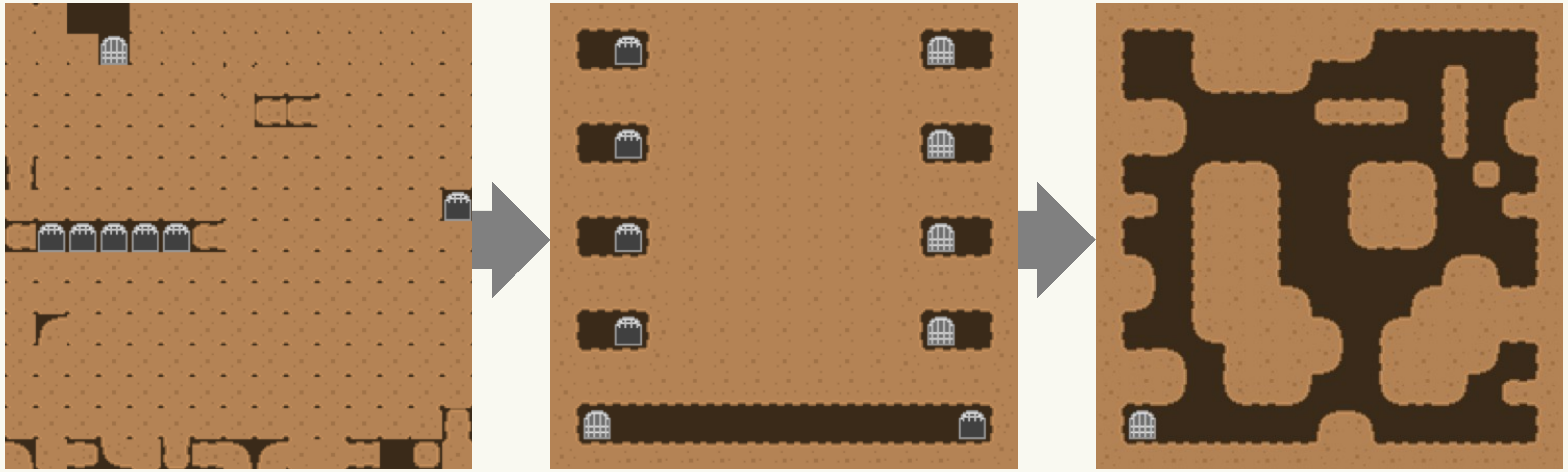
```
CnstrCount(tileVars, lo, hi, SOFT)
```

Find and process solution.

Interface

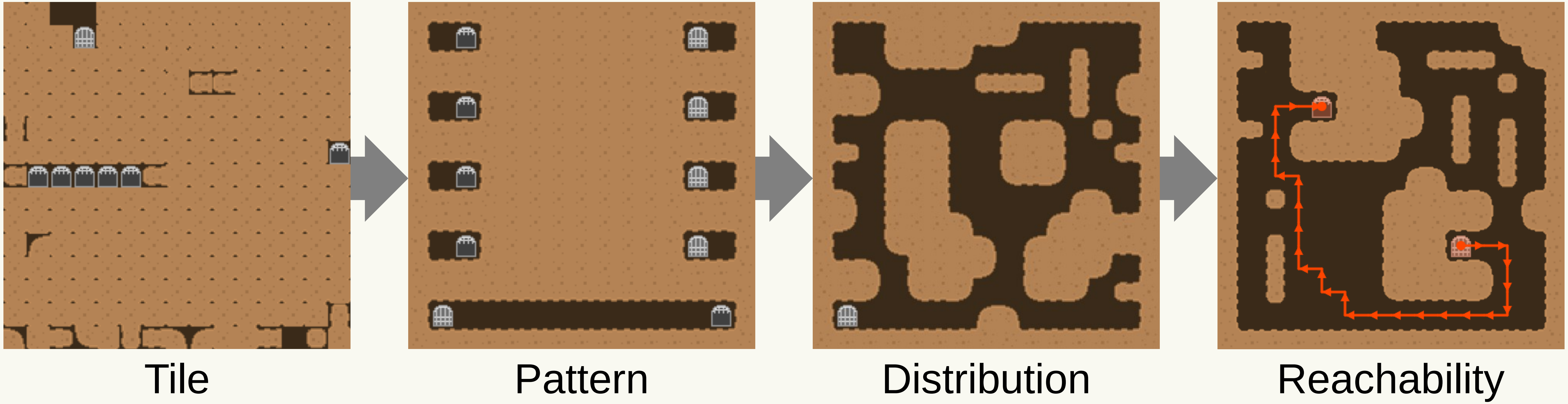
```
MakeVar()
CnstrCount(...)
Solve()
GetVar(...)
CnstrImpliesOr(...)
MakeAnd(...)
```

Constraint-Based Generation

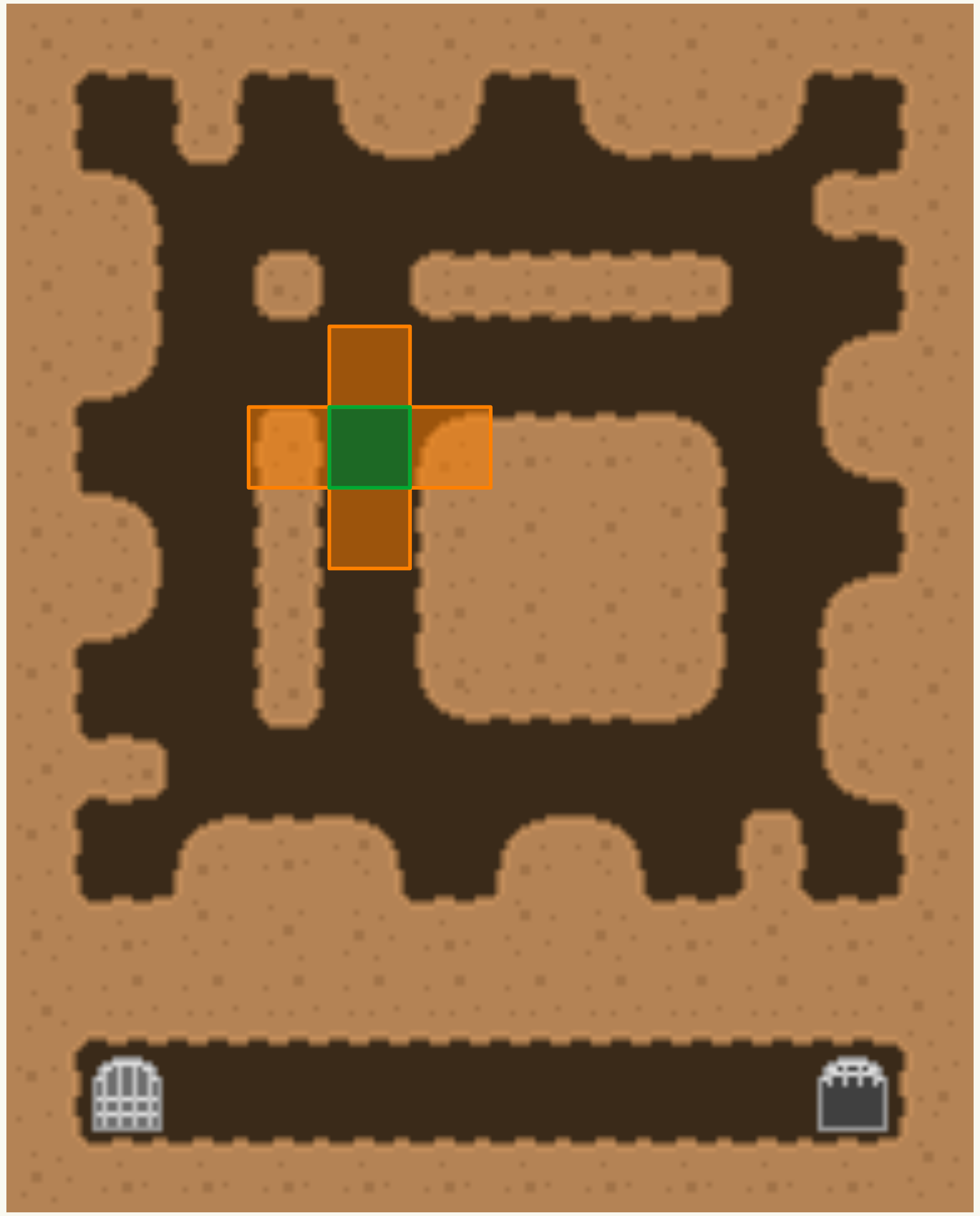


Tile | Pattern | Distribution

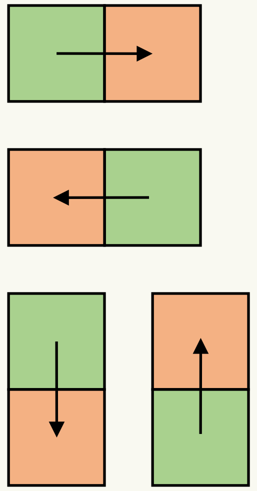
Constraint-Based Generation



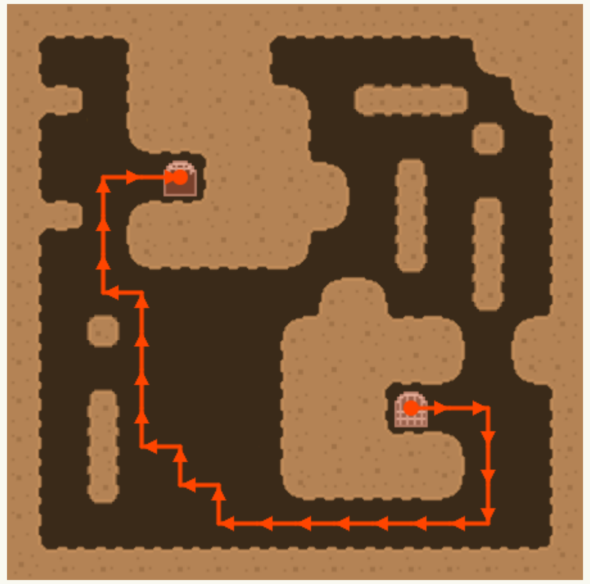
Reachability Constraints



Reachability template

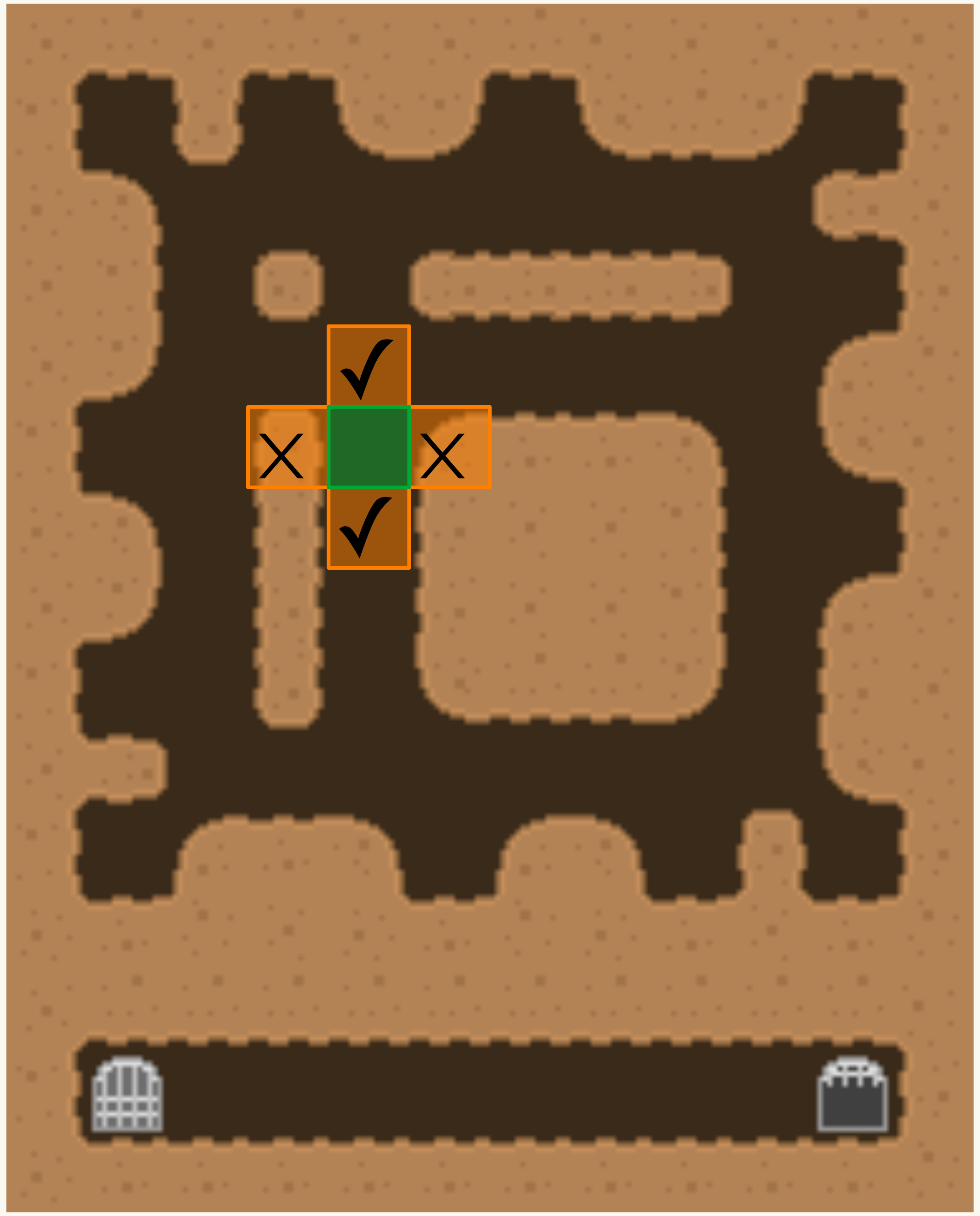


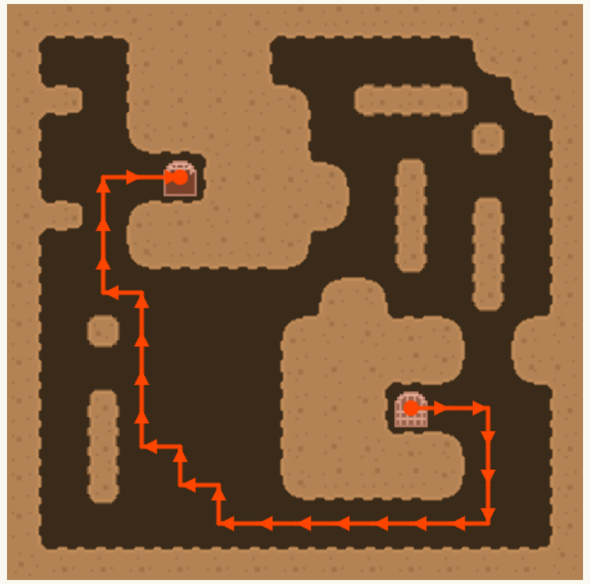
How player can move through level



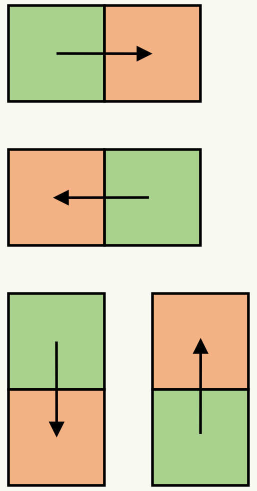
Interface
`MakeVar()`
`CnstrCount(...)`
`Solve()`
`GetVar(...)`
`CnstrImpliesOr(...)`
`MakeAnd(...)`

Reachability Constraints





Reachability template

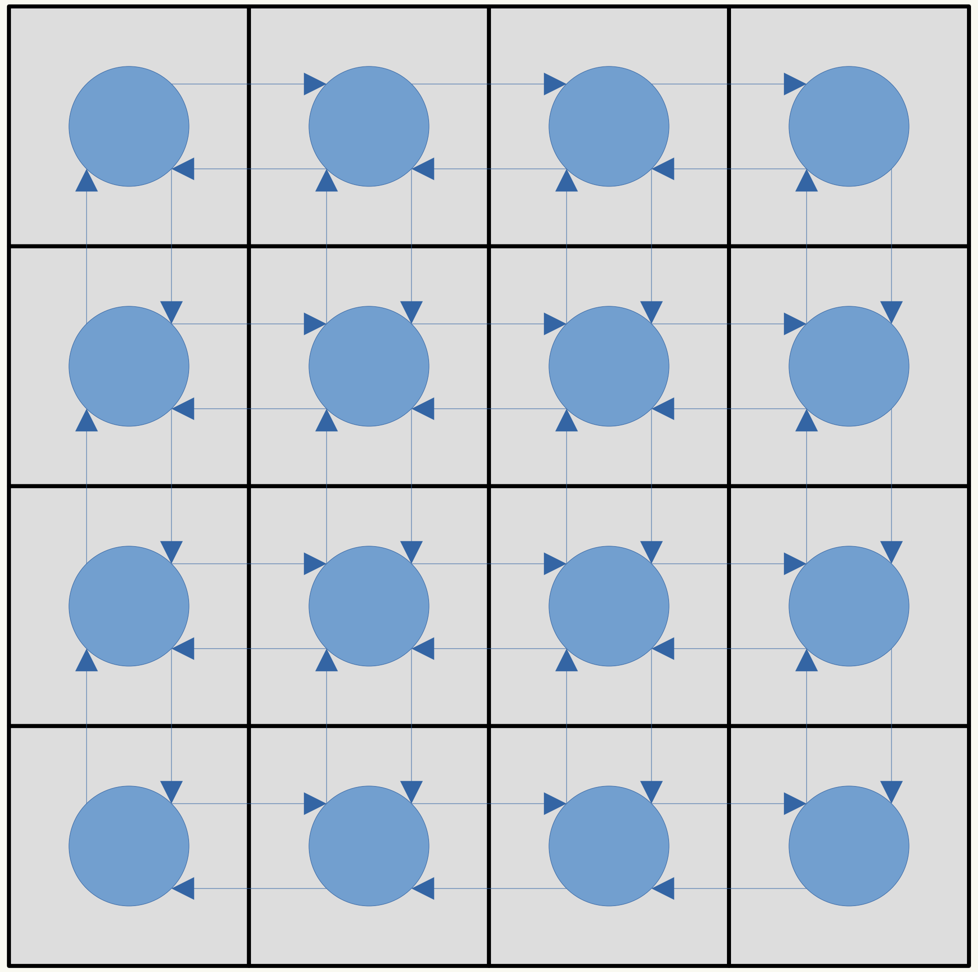


How player can move through level

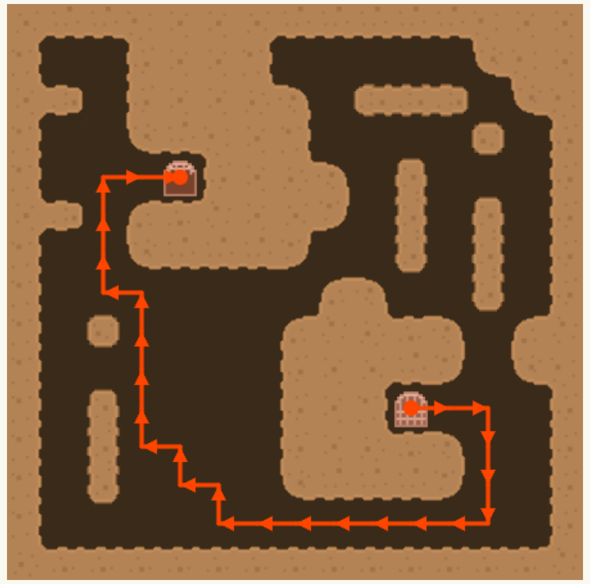
<u>Interface</u>

```
MakeVar()
CnstrCount(...)
Solve()
GetVar(...)
CnstrImpliesOr(...)
MakeAnd(...)
```

Reachability Constraints



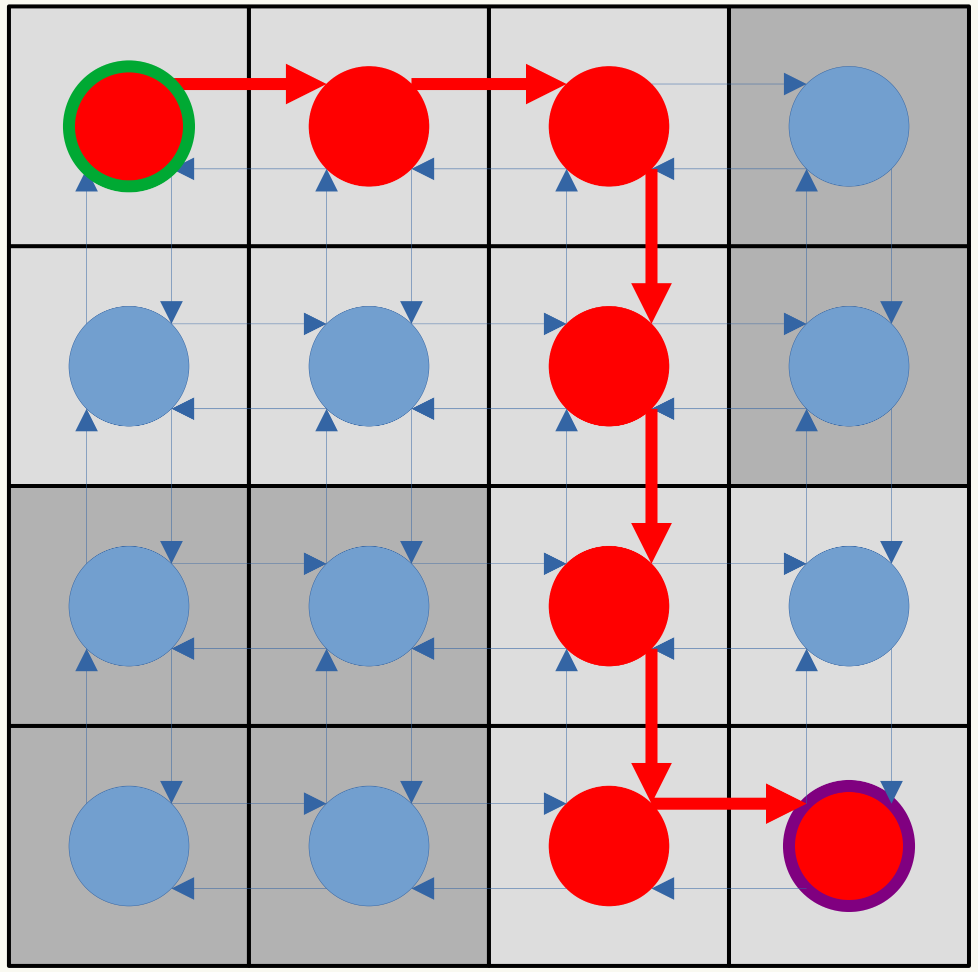
Convert to a graph of *possible* moves, adding variables for nodes and edges being part of the path, and constrain existence of a path in the graph.



<u>Interface</u>

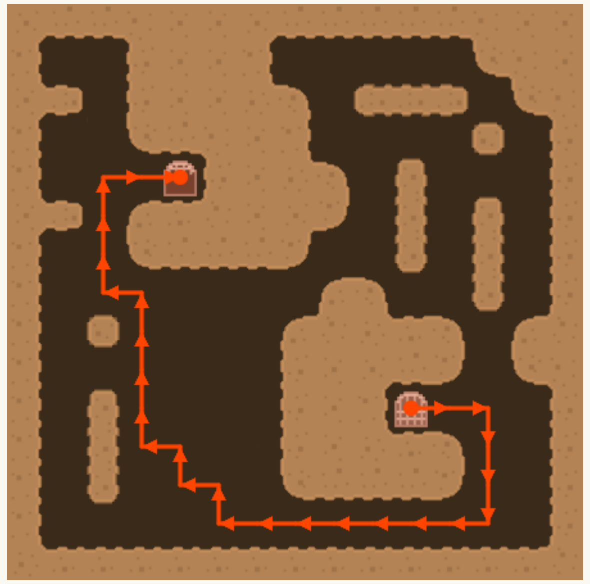
```
MakeVar()
CnstrCount(...)
Solve()
GetVar(...)
CnstrImpliesOr(...)
MakeAnd(...)
```

Reachability Constraints



Convert to a graph of *possible* moves, adding variables for nodes and edges being part of the path, and constrain existence of a path in the graph.

Only requires **a** path, not a short or direct one.



Interface

```
MakeVar()
CnstrCount(...)
Solve()
GetVar(...)
CnstrImpliesOr(...)
MakeAnd(...)
```

Level Generation

Outline

Setup tile constraints.

Setup pattern constraints.

Setup distribution constraints.

Setup reachability constraints.

Setup any additional custom constraints.

Find solution.

Process solution.

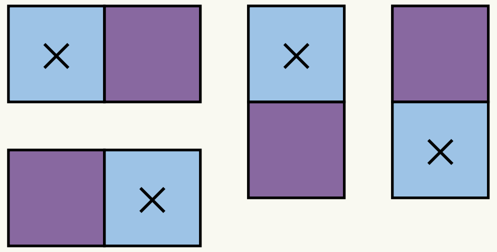
Interface

```
MakeVar()
MakeAnd(...)

CnstrCount(...)
CnstrImpliesOr(...)

Solve()

GetVar(...)
```

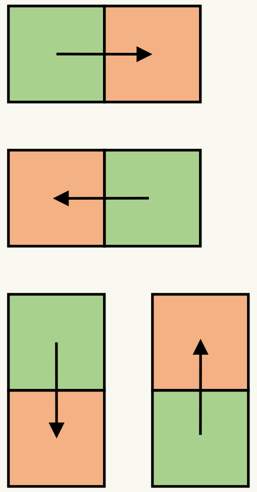
Level Generation

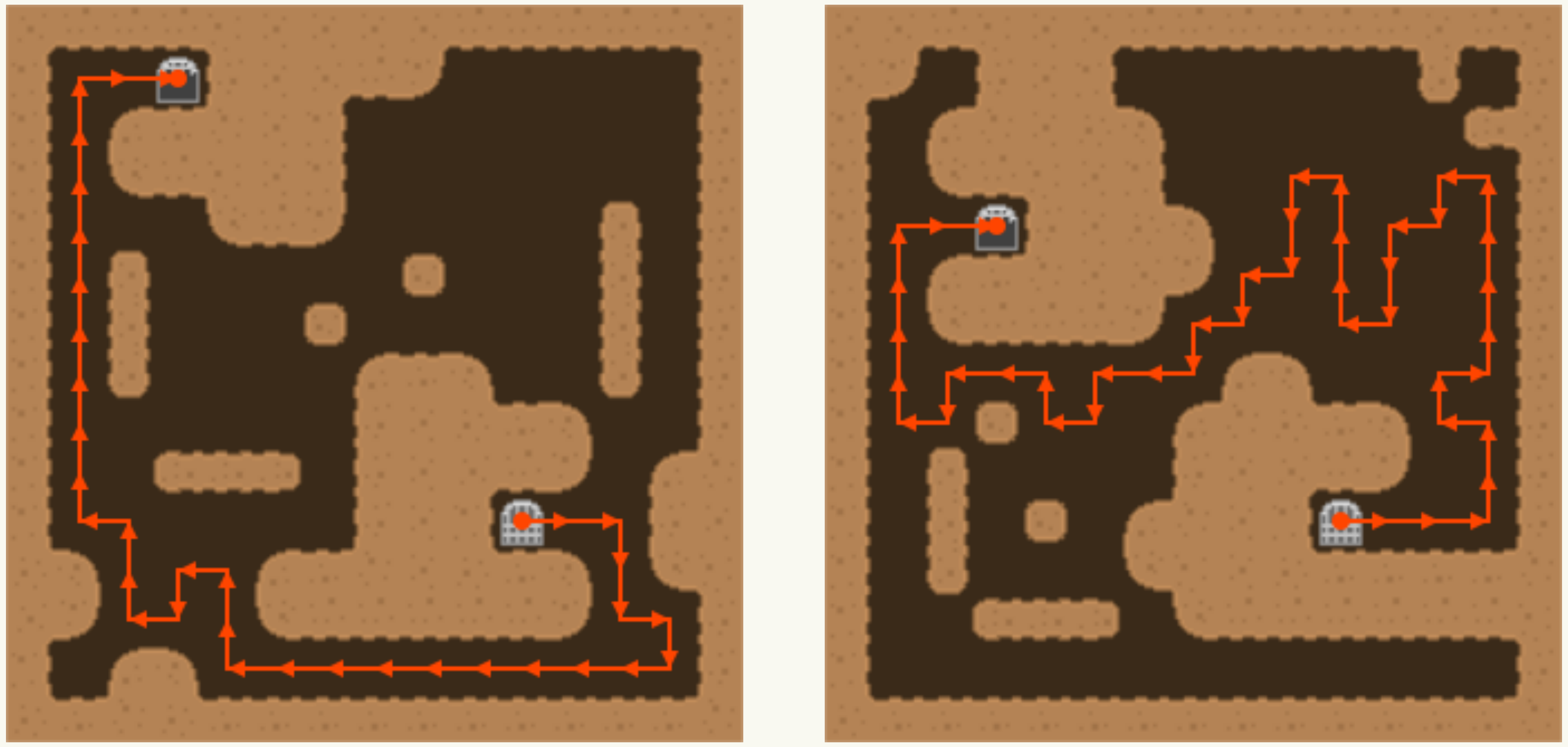
Cave

Pattern template



Reachability template

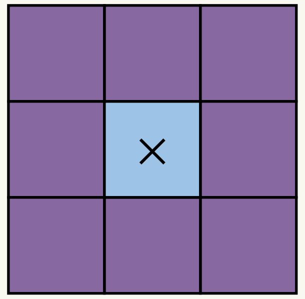




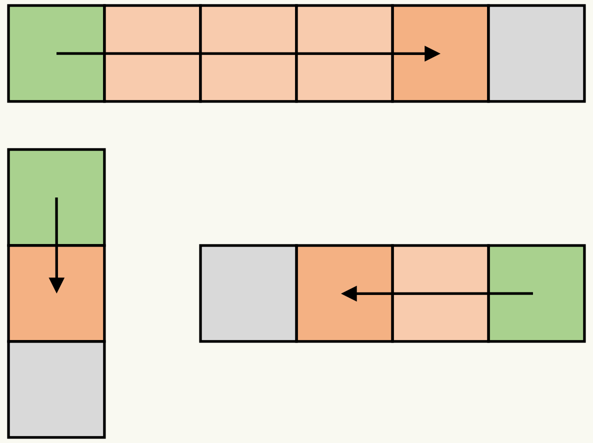
Level Generation

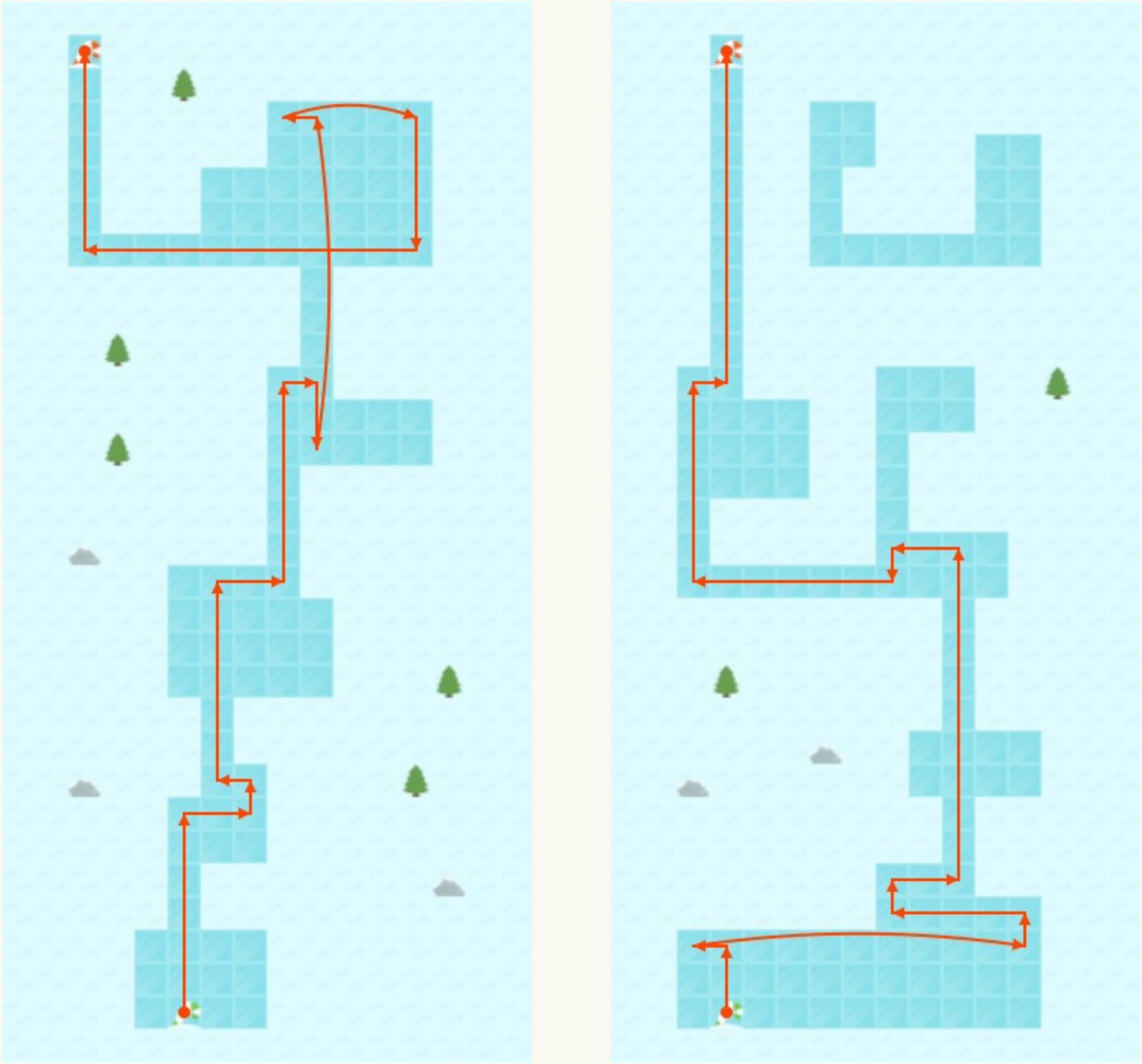
Sliding

Pattern template



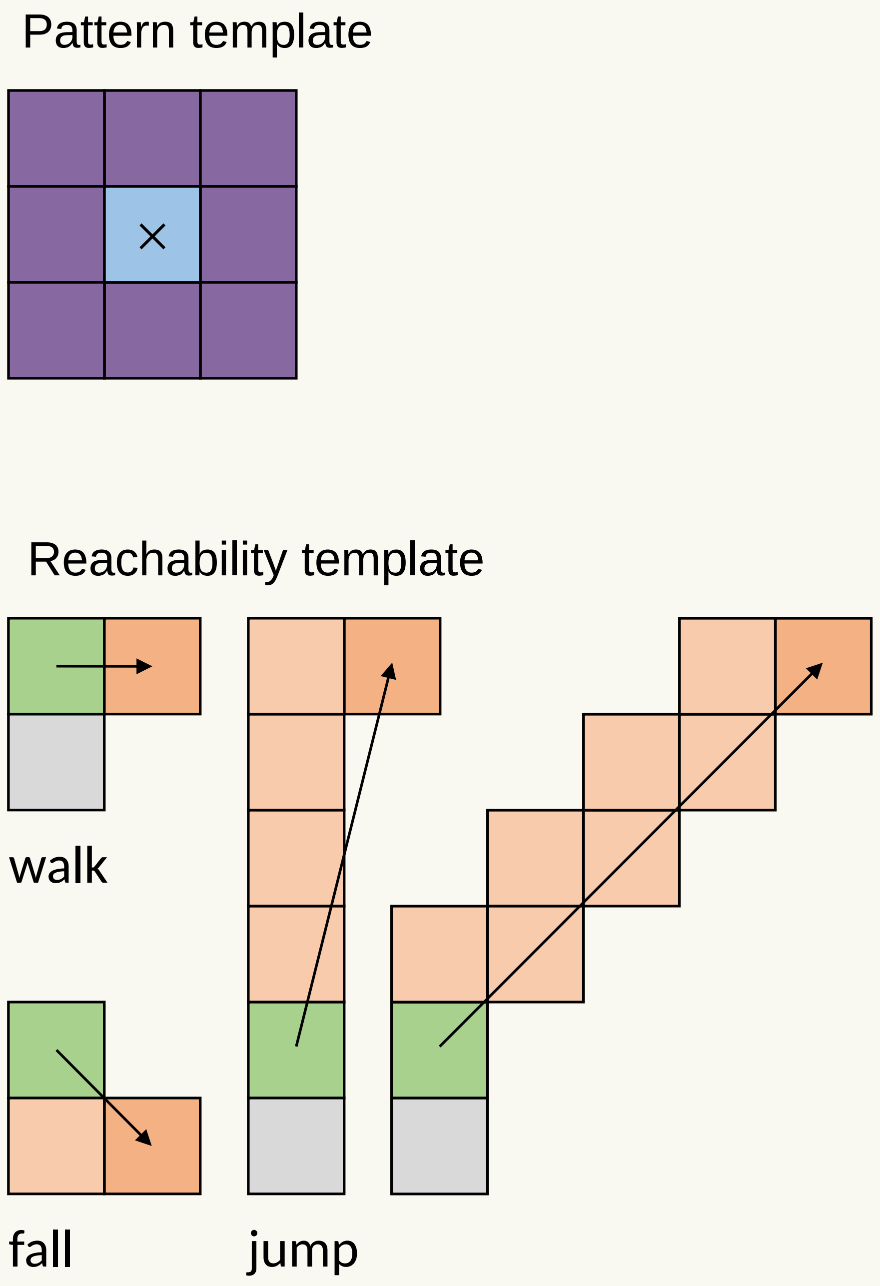
Reachability template

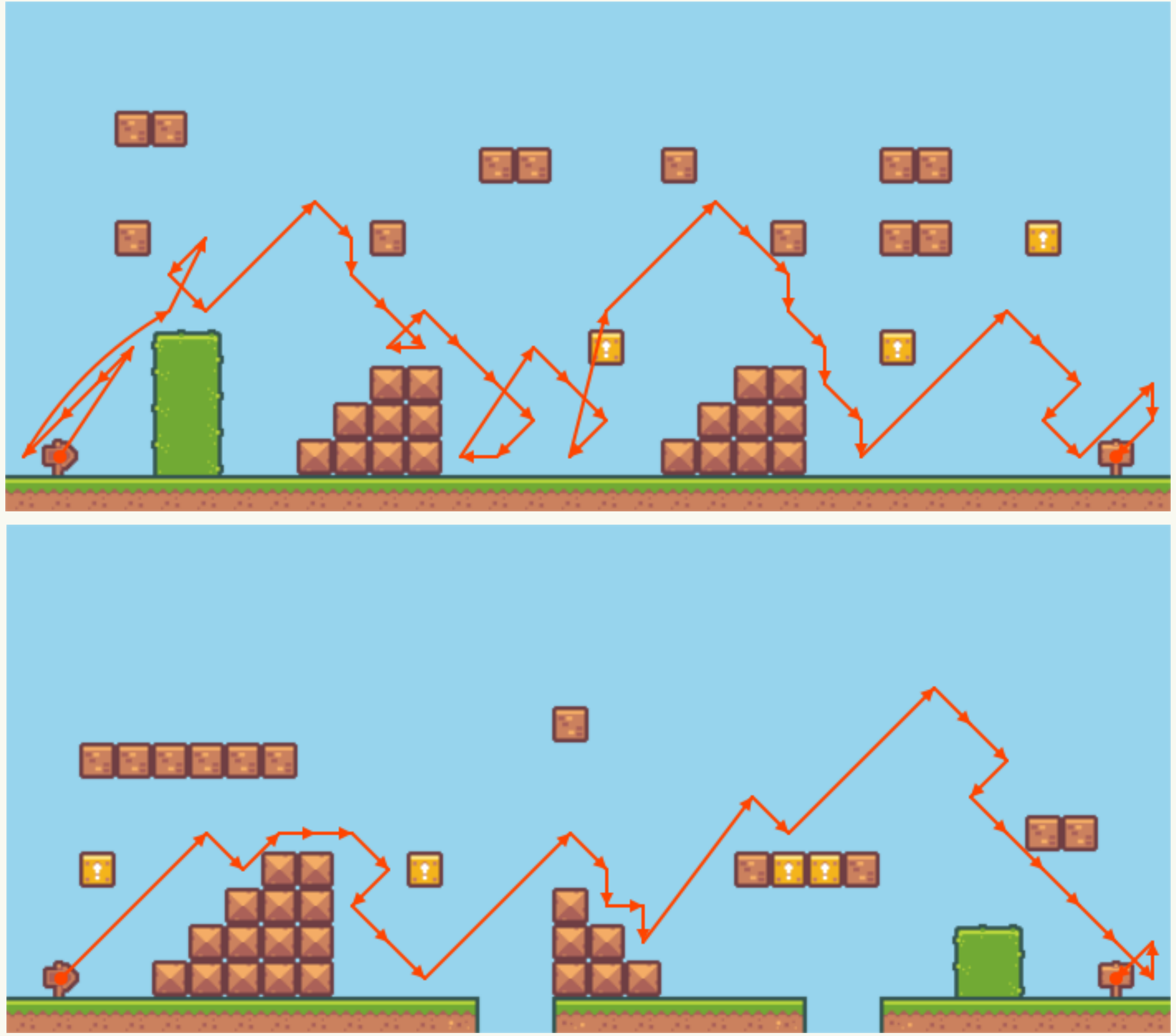




Level Generation

Platformer

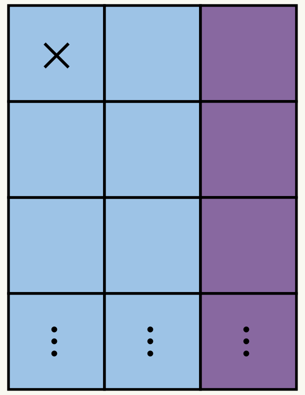




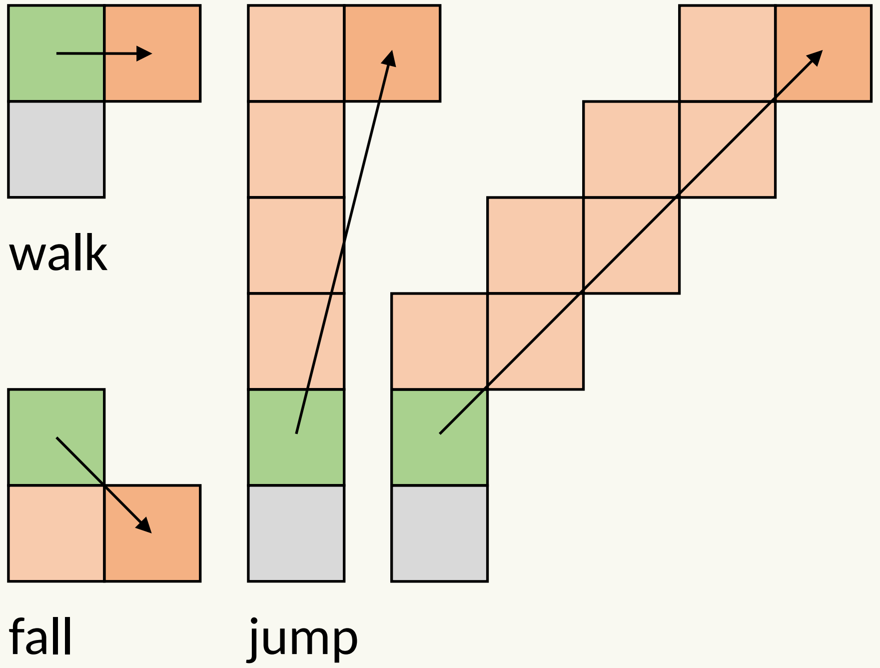
Level Generation

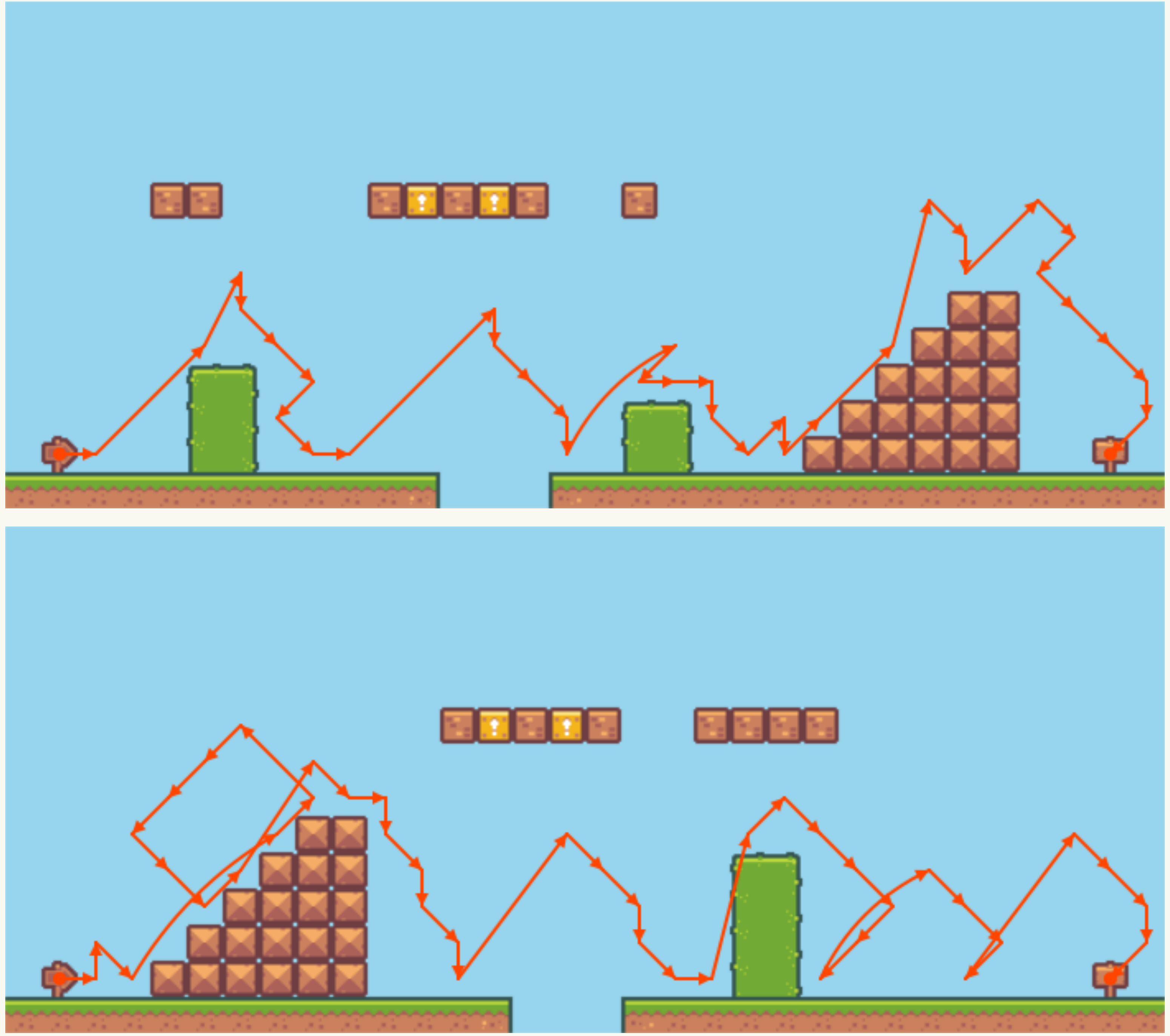
Platformer

Pattern template



Reachability template



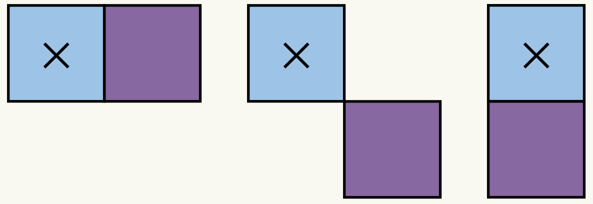


GDC

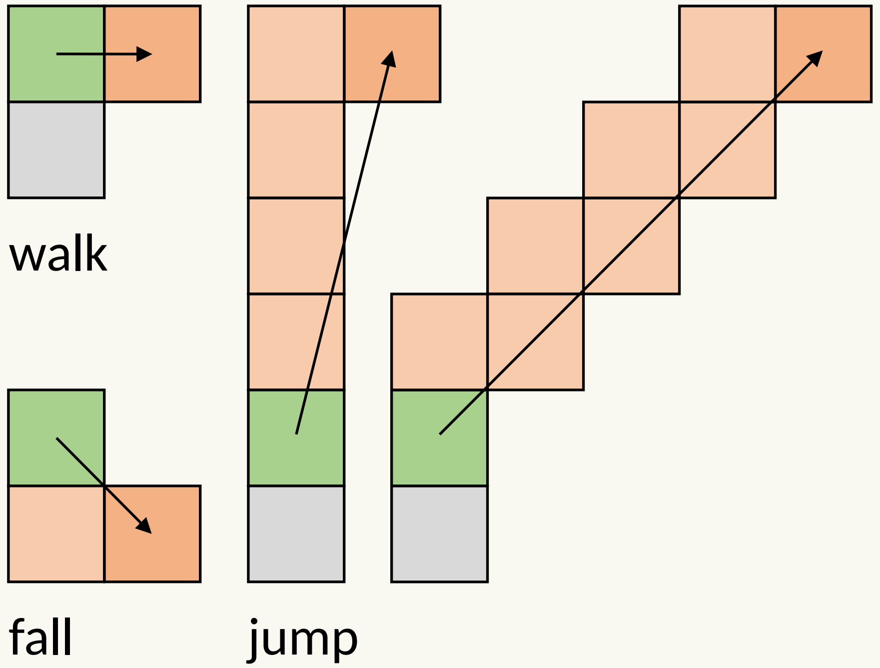
Level Generation

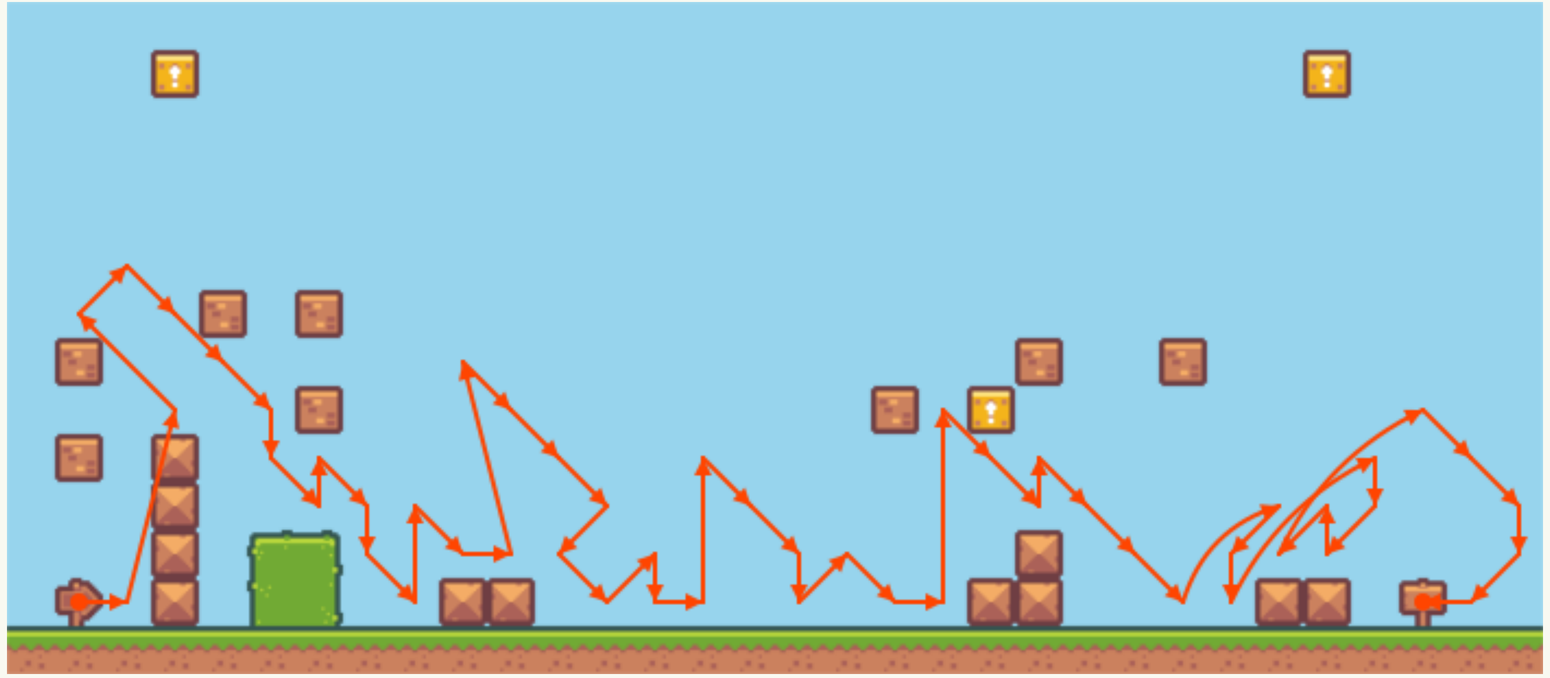
Platformer

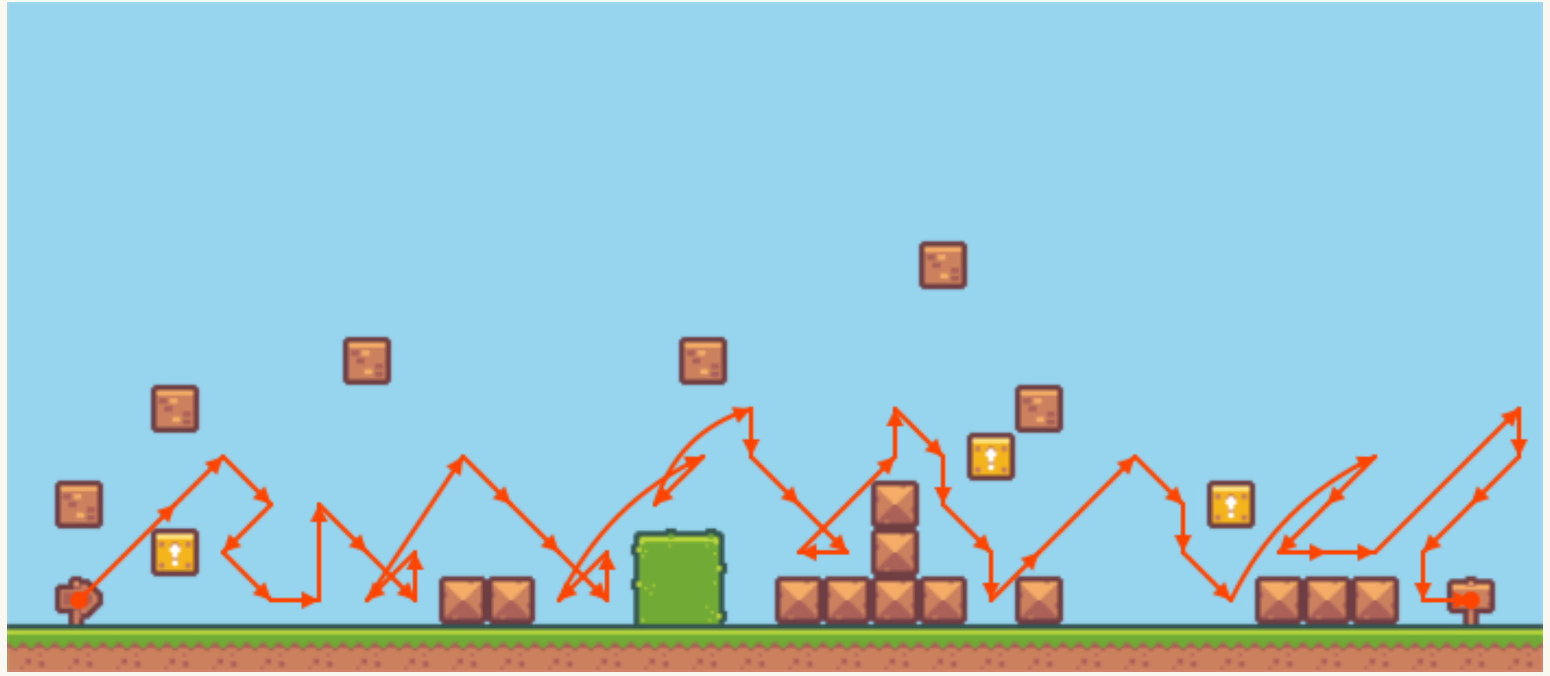
Pattern template



Reachability template



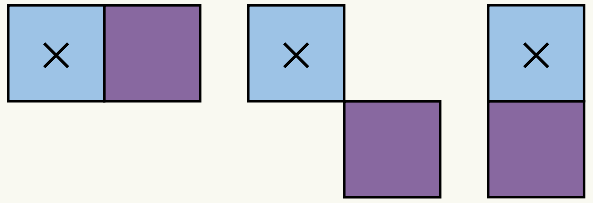




Level Generation

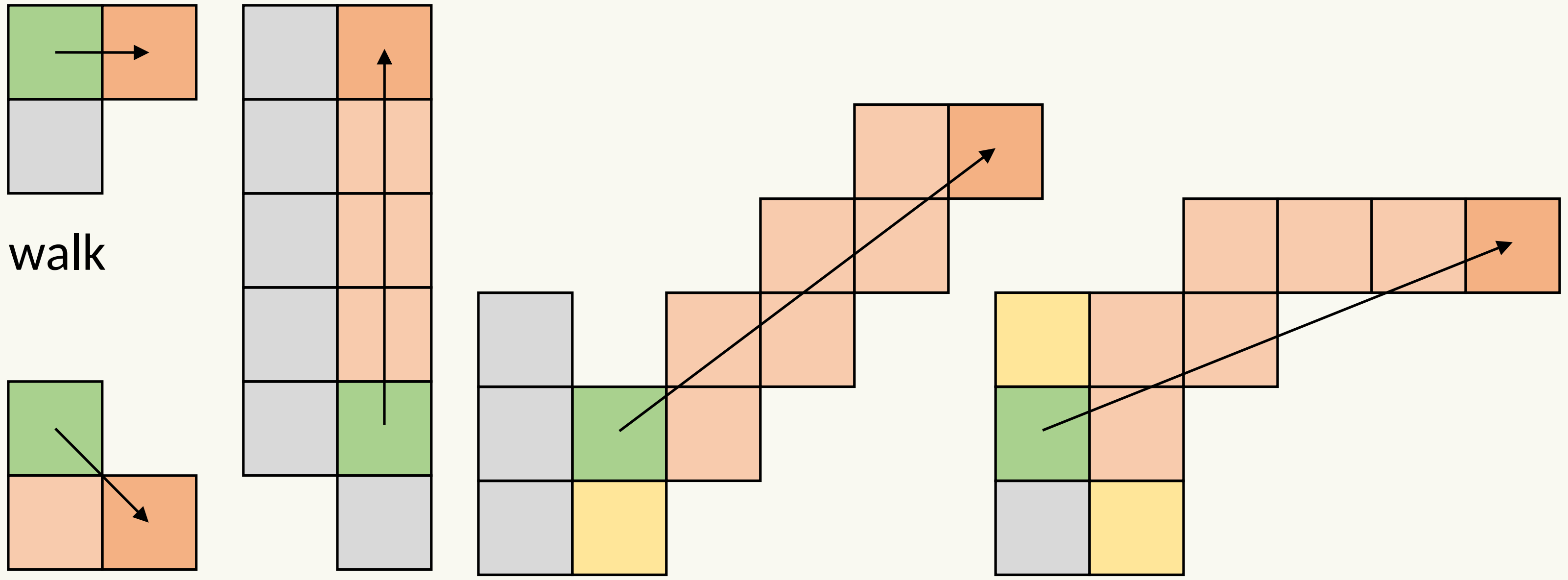
Vertical Platformer

Pattern template



counts regions vertically

Reachability template (e.g. Super Cat Tales)



walk

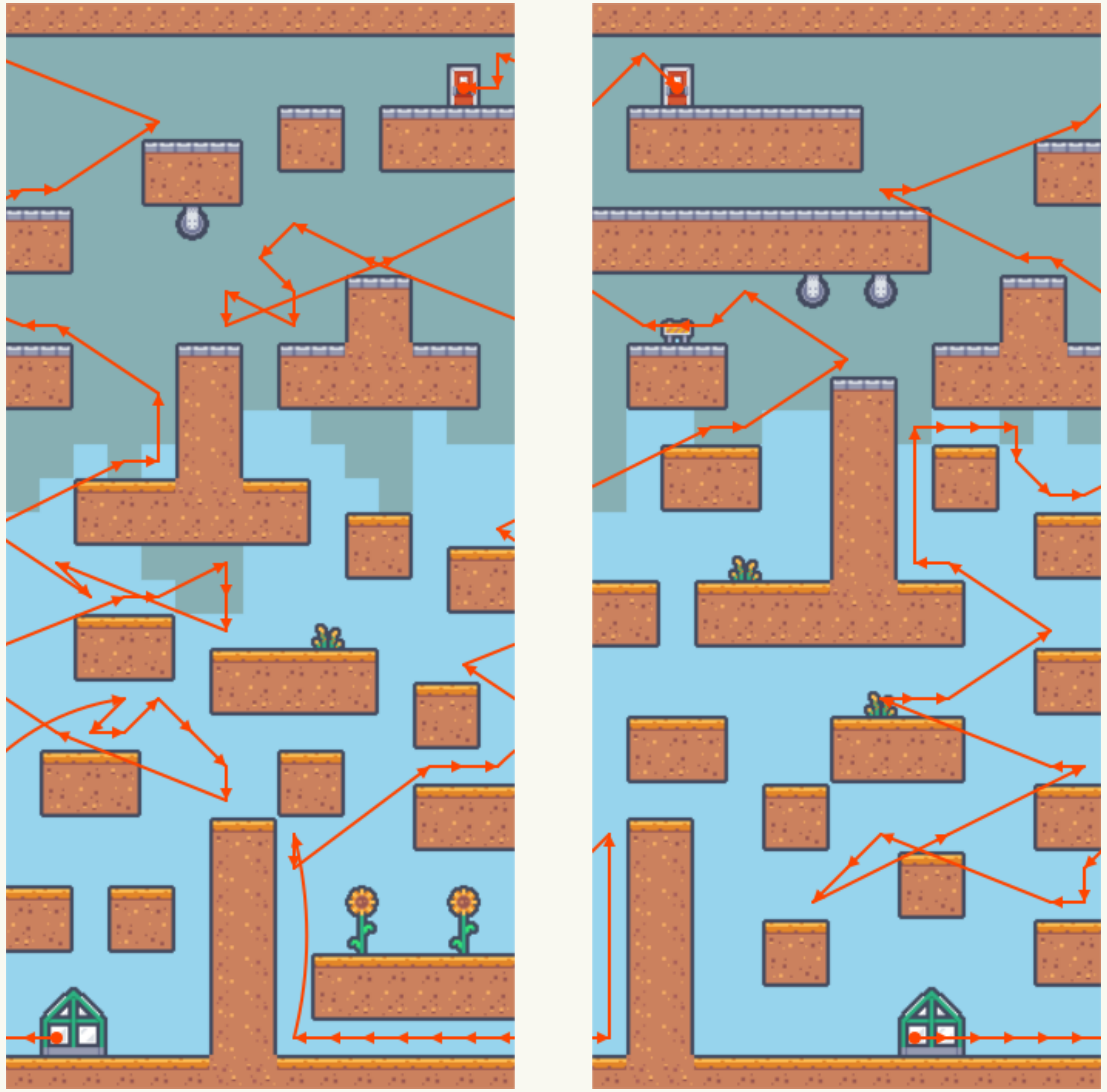
fall

wall climb

wall jump

ledge jump

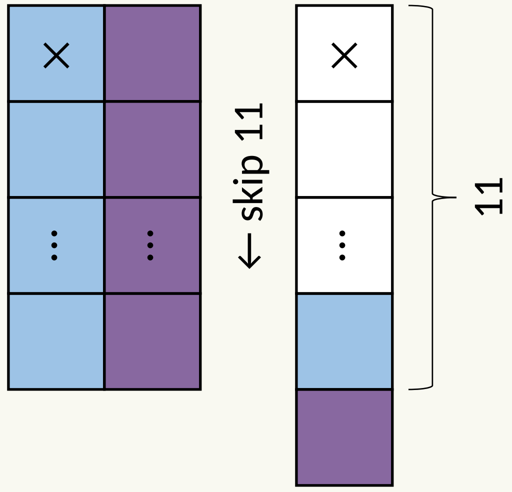
column-wrapping



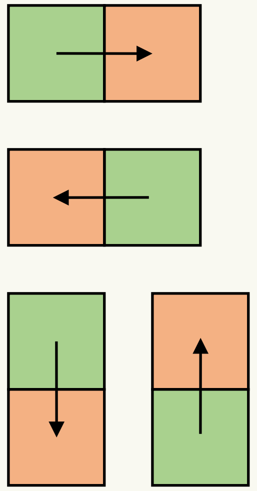
Level Generation

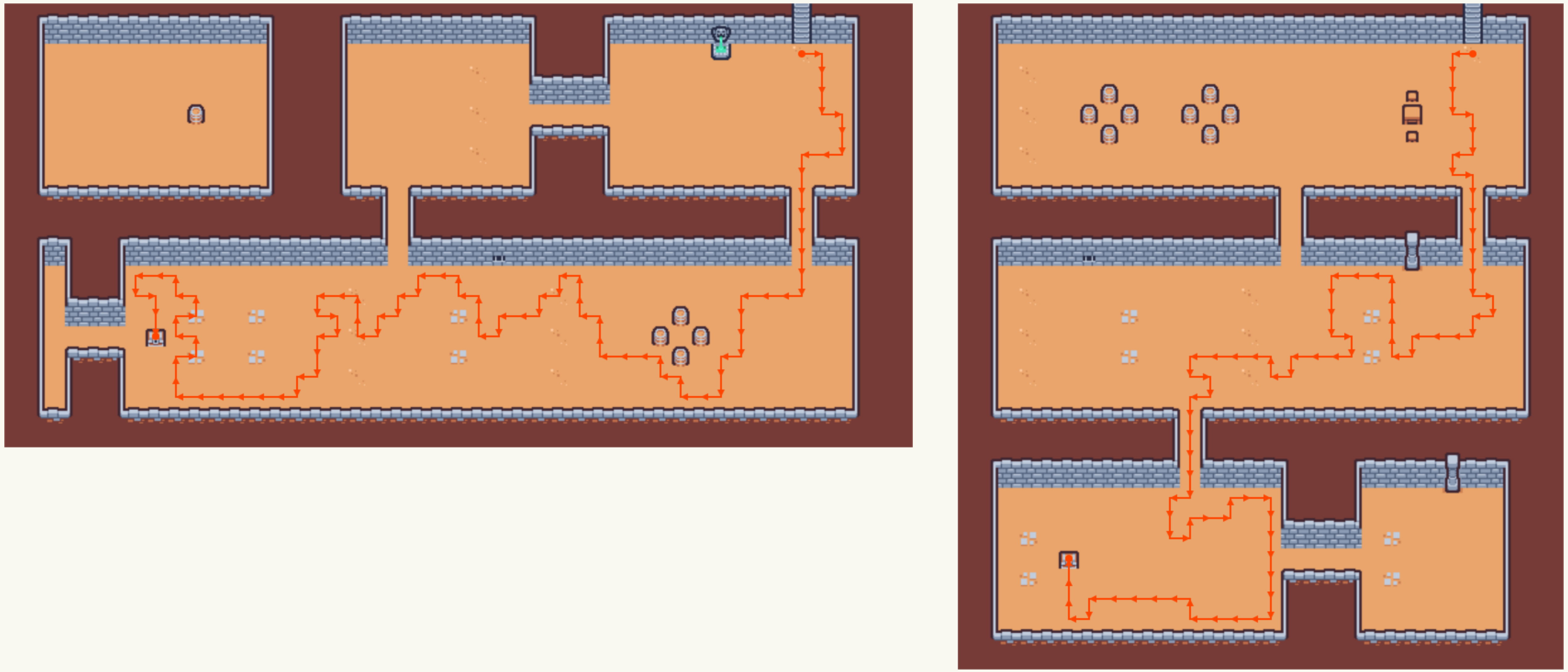
Dungeon

Pattern template



Reachability template





Applications

Setup tile constraints.

Setup pattern constraints.

Setup distribution constraints.

Setup reachability constraints.

Setup any additional custom constraints.

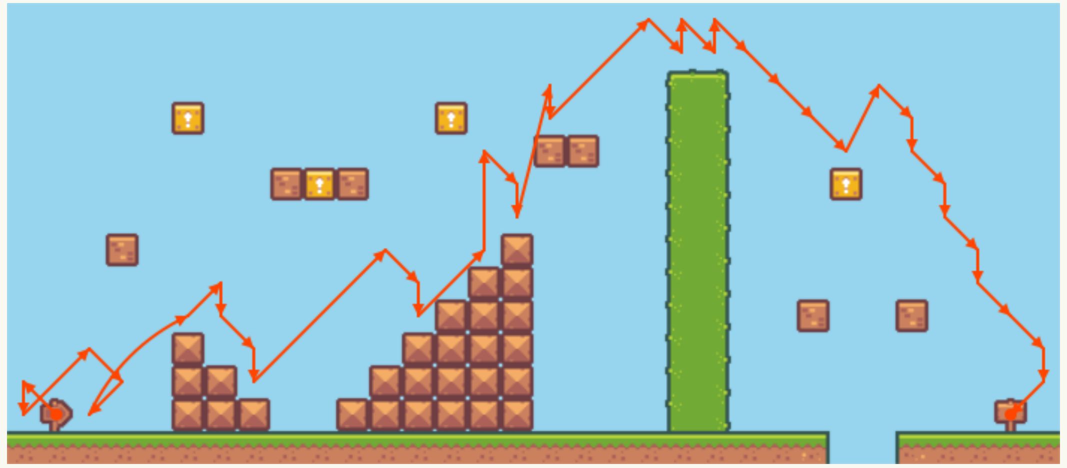
Find solution.

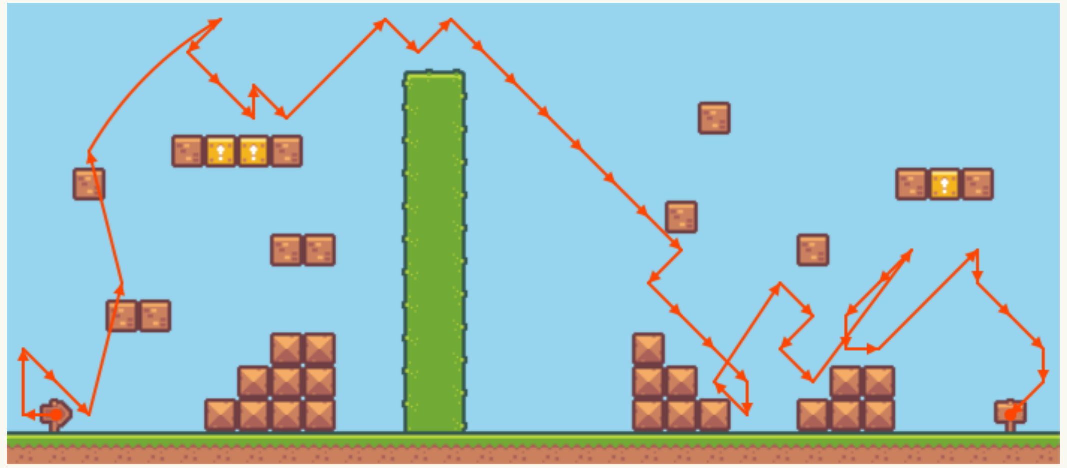
Process solution.

Applications

Tile Constraints

Exactly 1 in 3rd from top row



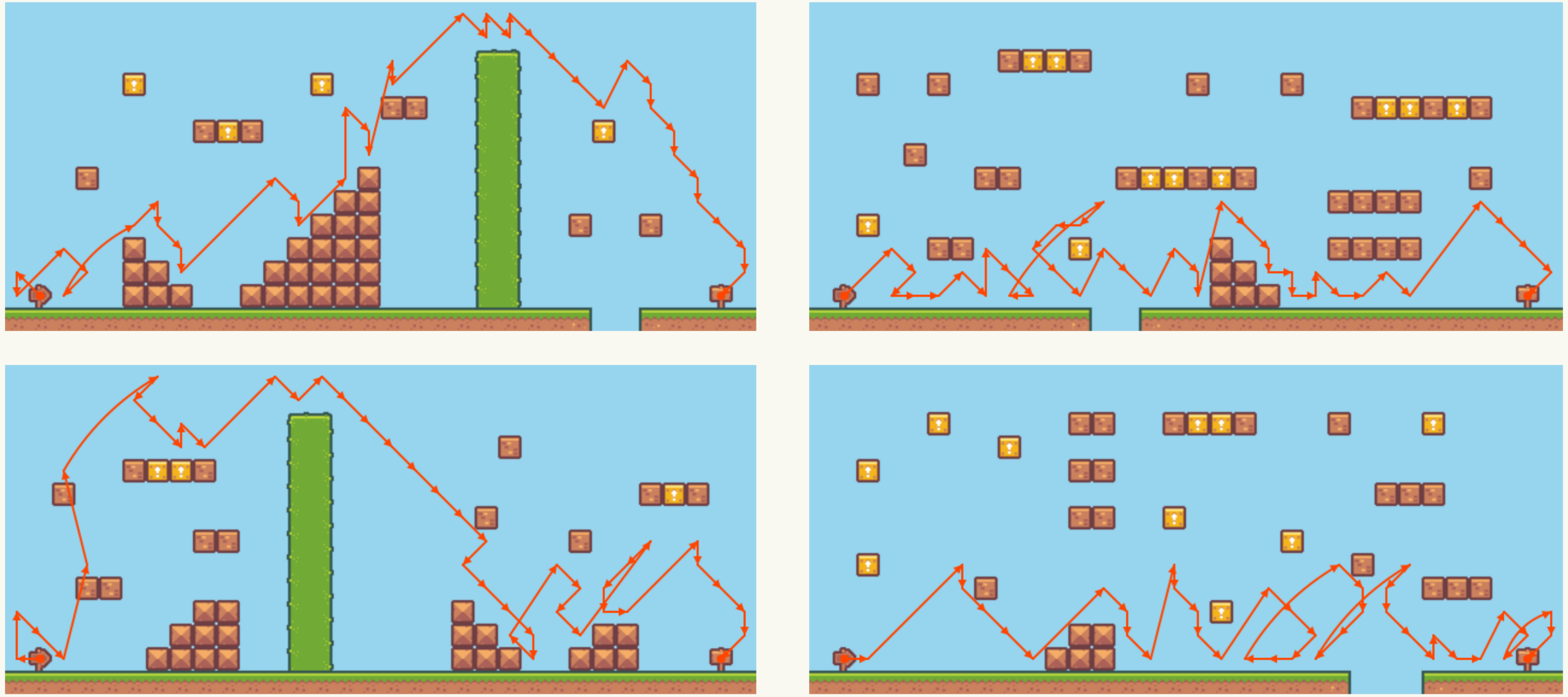


Applications

Tile Constraints

Exactly 1 in 3rd from top row

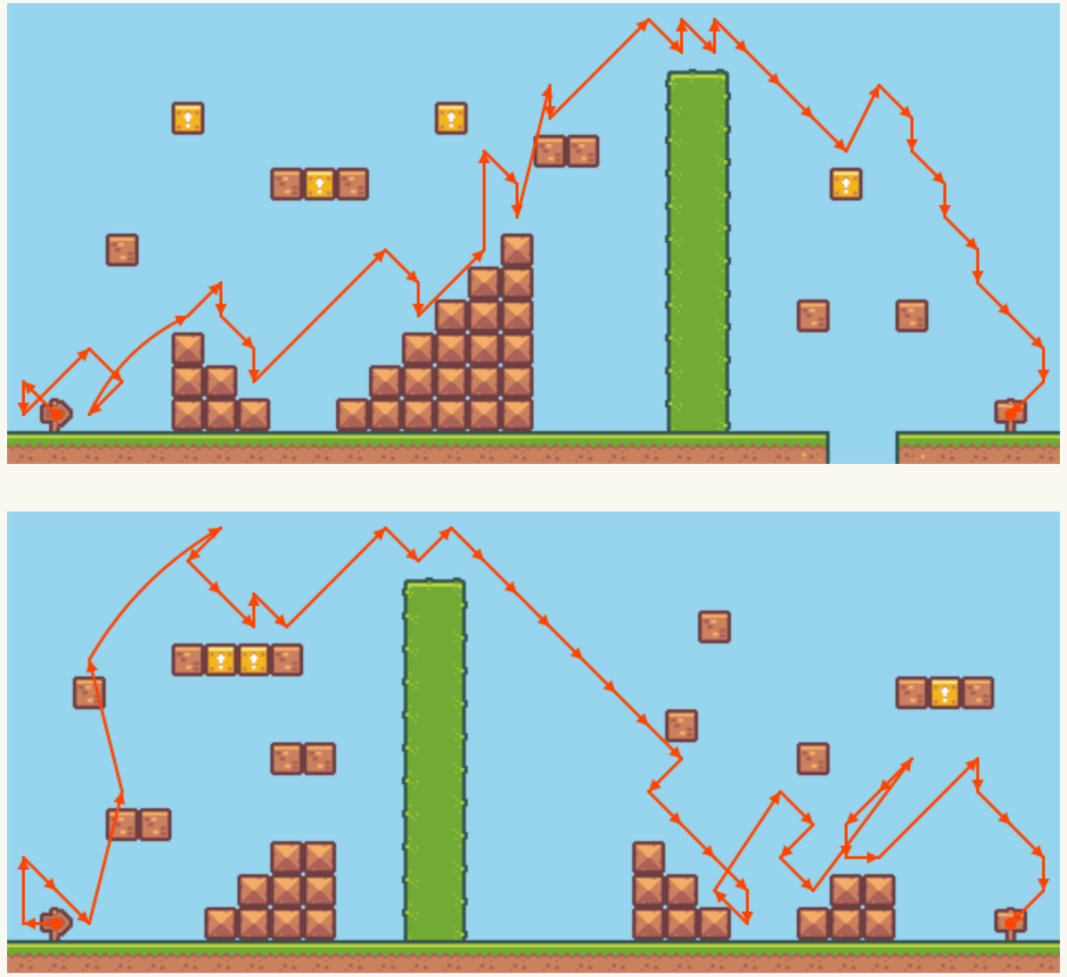
Exactly 10



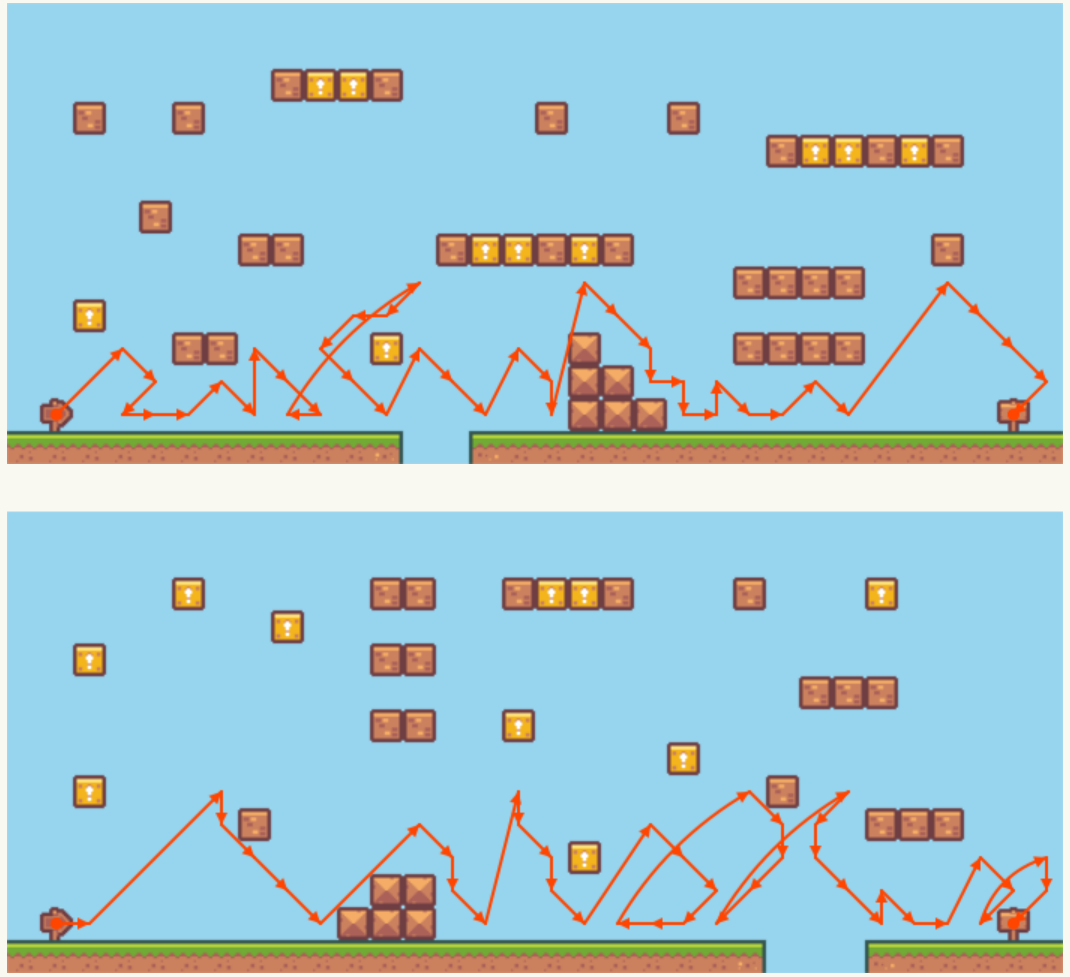
Applications

Tile Constraints

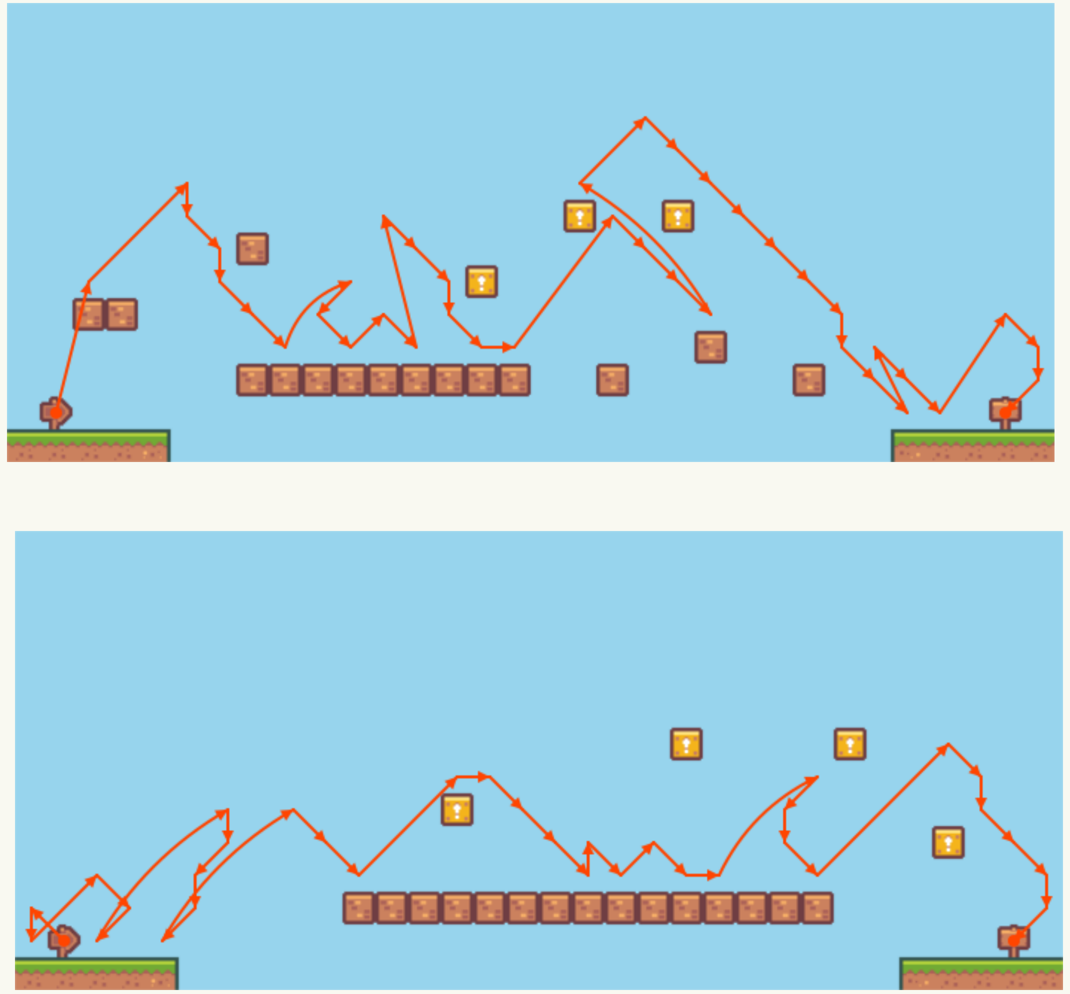
Exactly 1 in 3rd from top row



Exactly 10

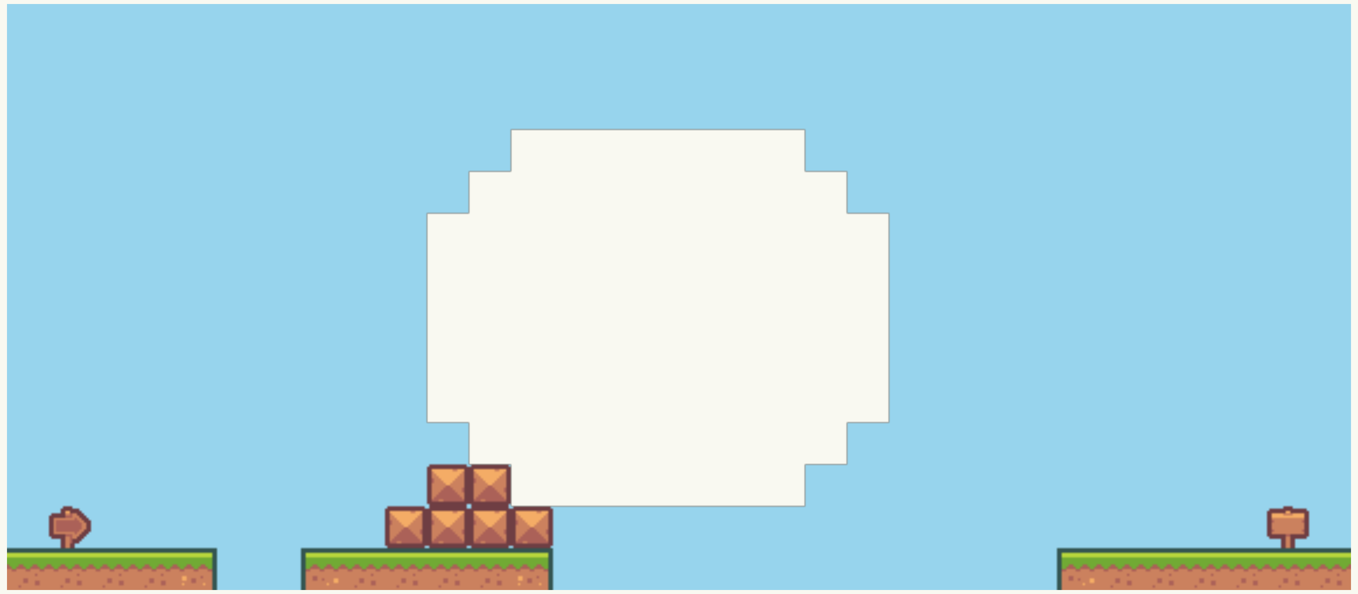


Maximize in bottom row (soft)



Applications

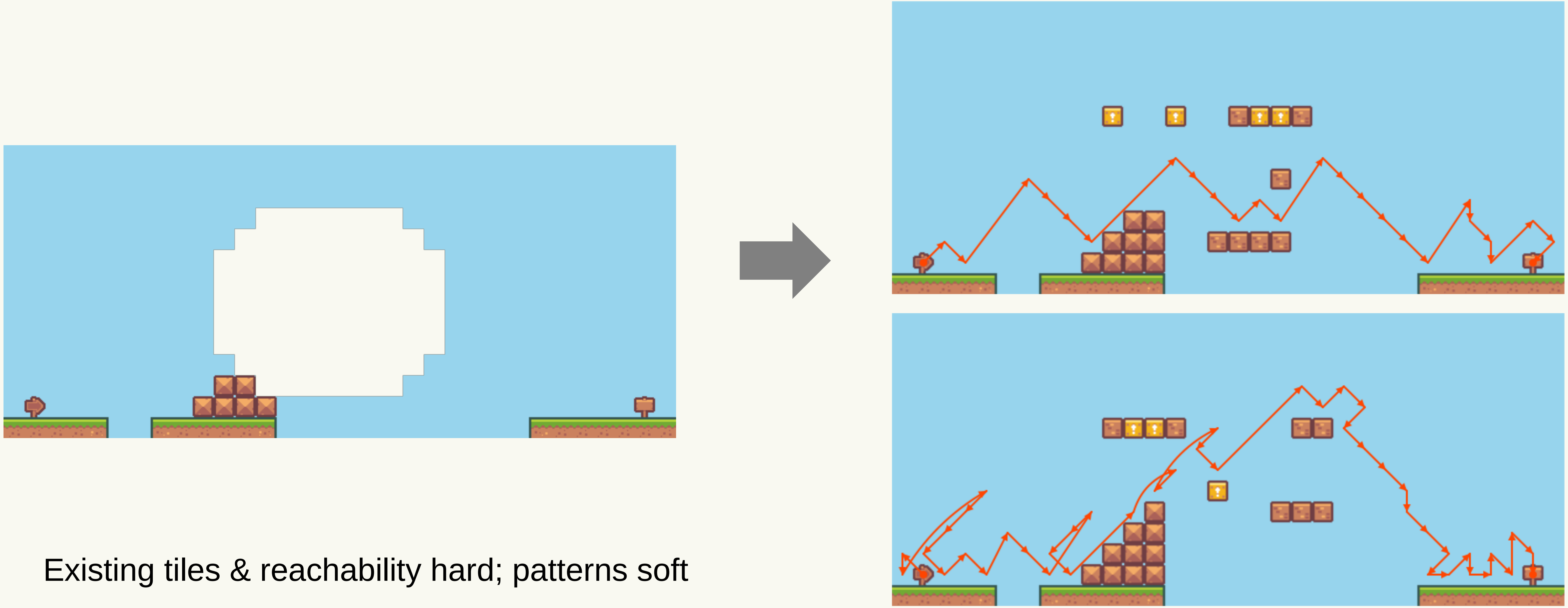
Infilling



Existing tiles & reachability hard; patterns soft

Applications

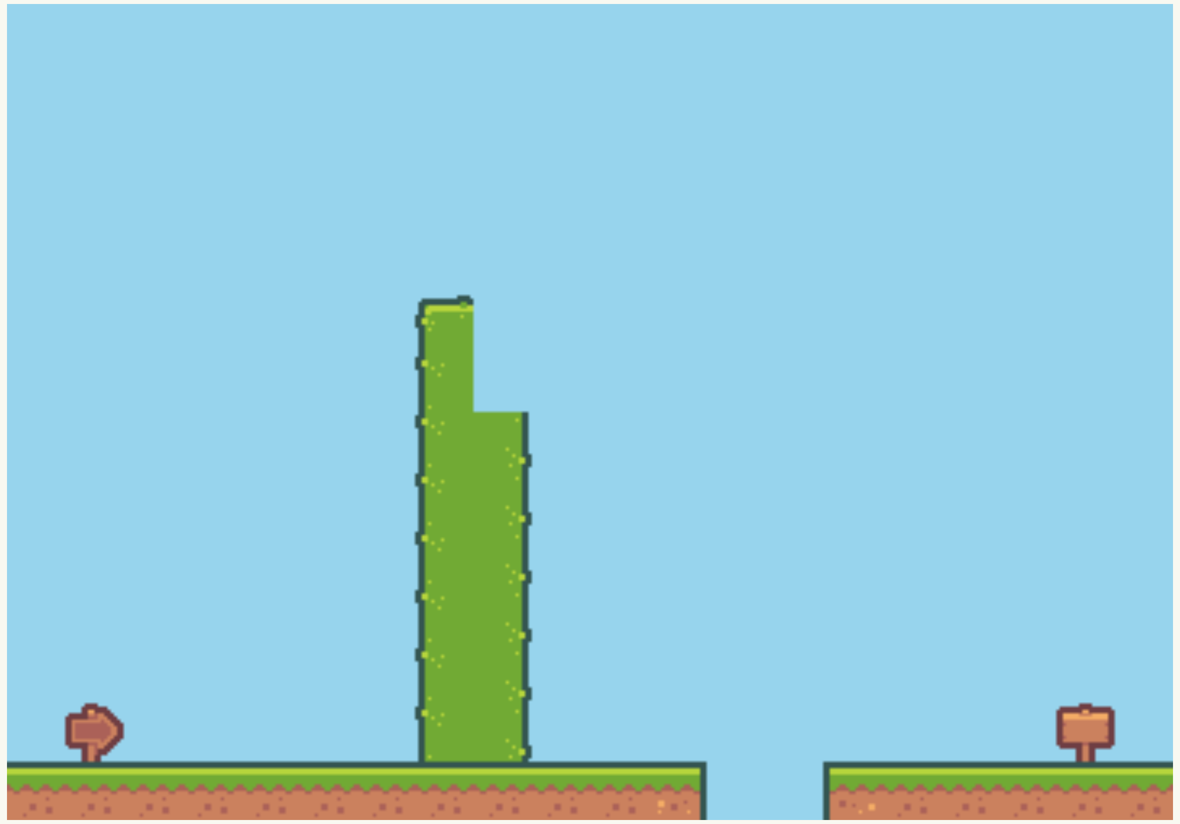
Infilling



Existing tiles & reachability hard; patterns soft

Applications

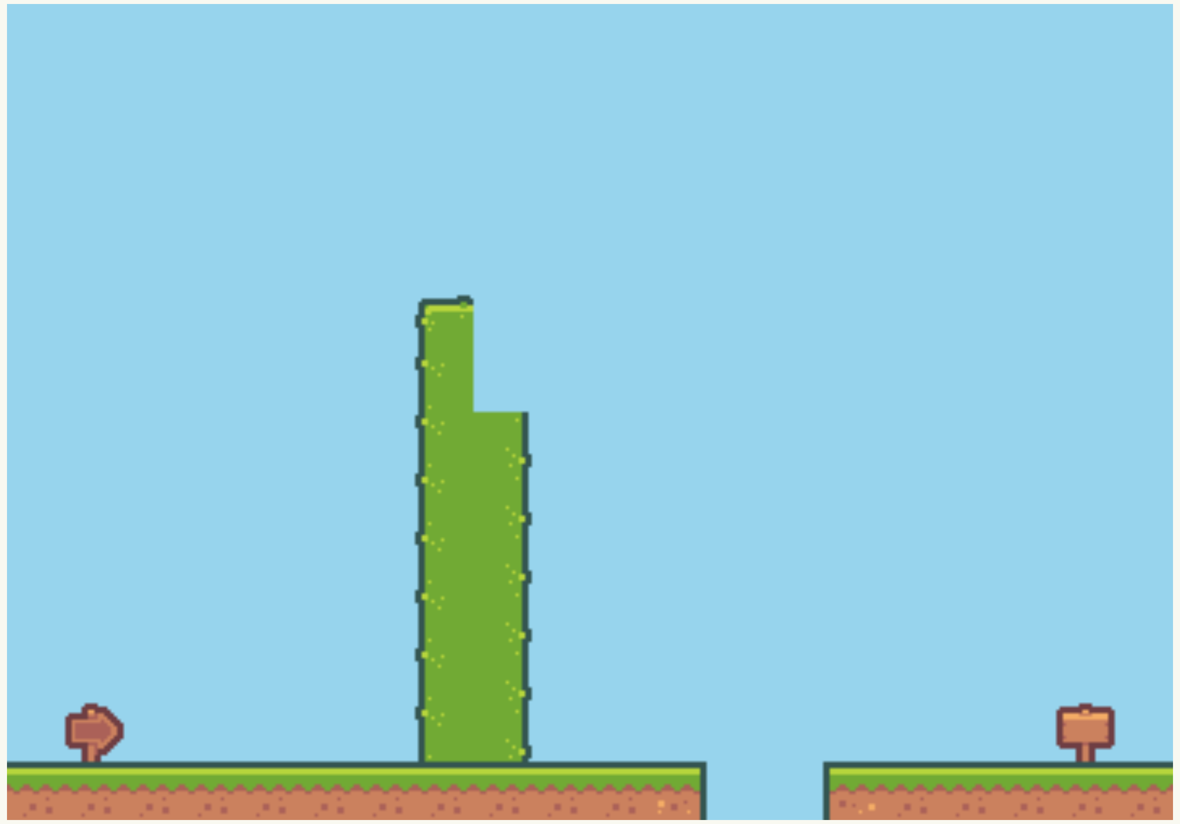
Repair

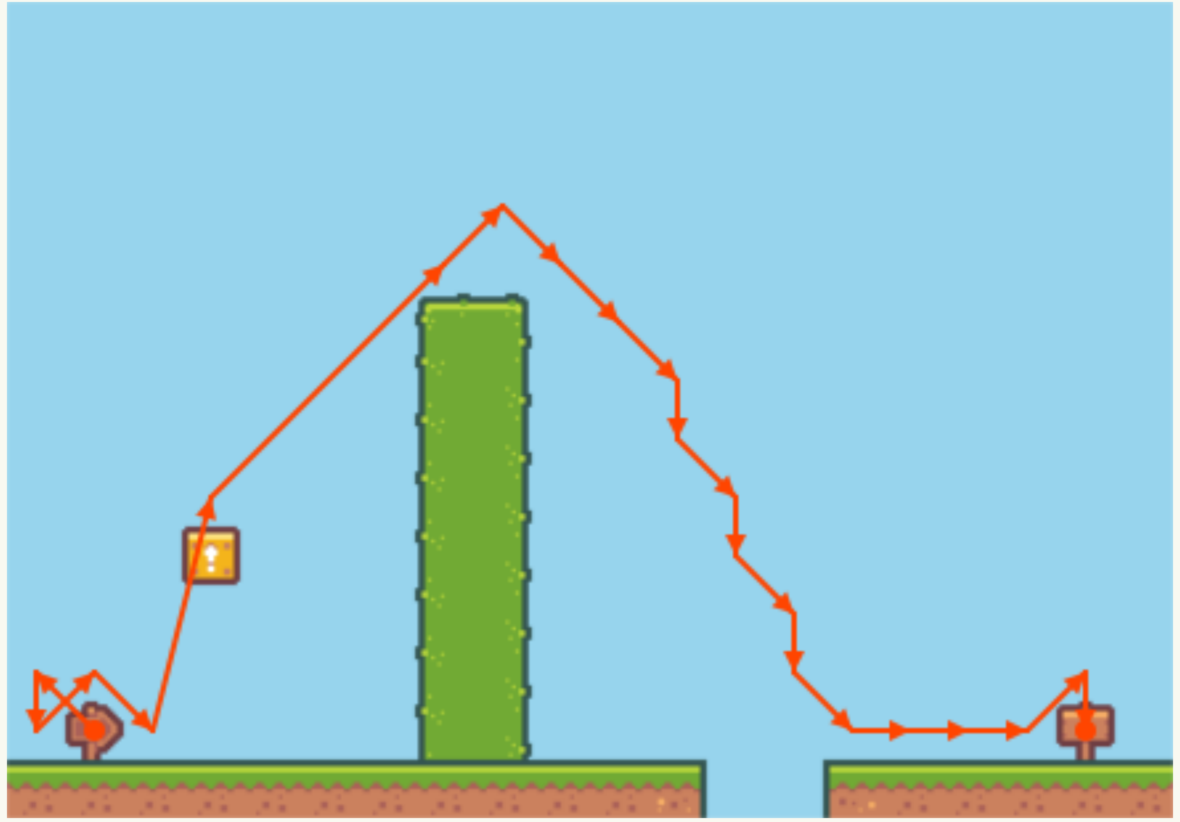


Patterns & reachability hard; existing tiles soft

Applications

Repair

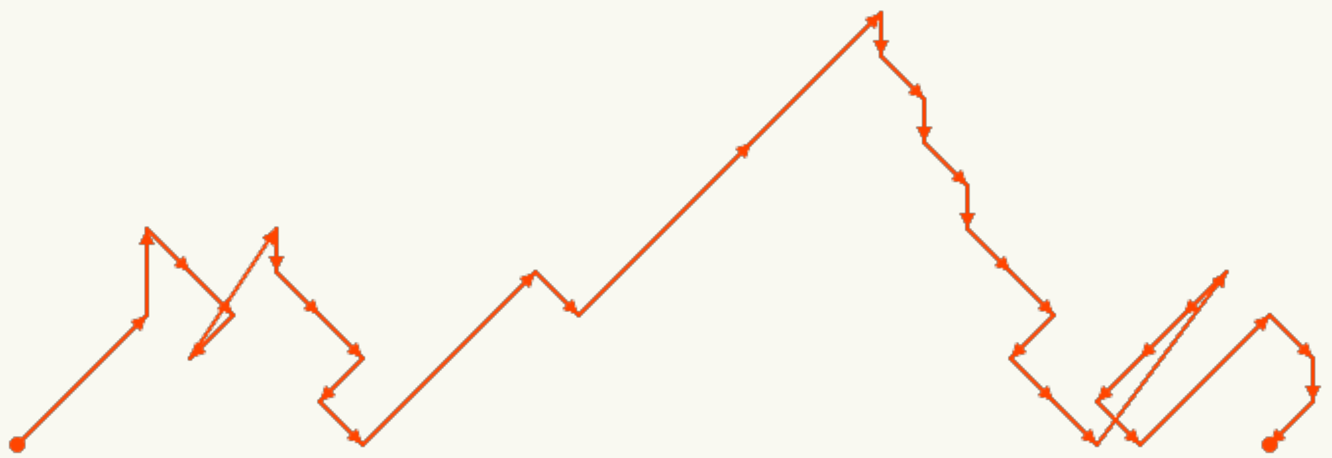




Patterns & reachability hard; existing tiles soft

Applications

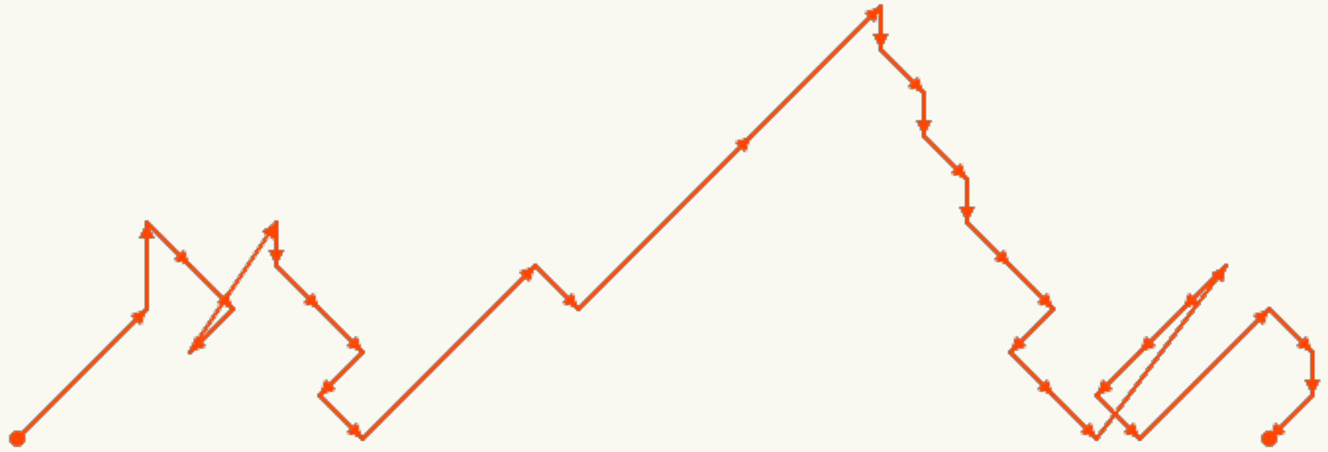
Path to Level

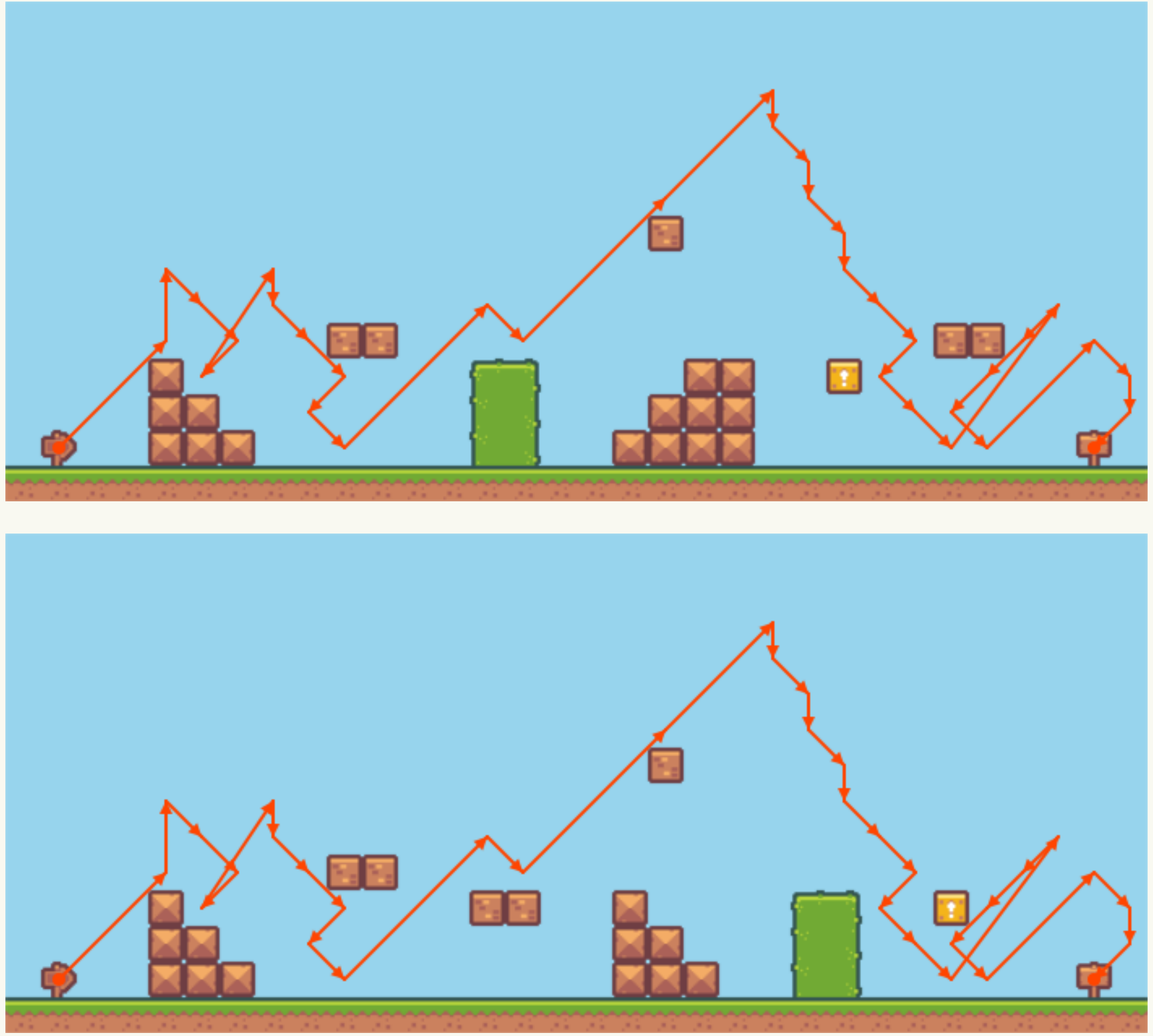


Generate level with desired “solution”

Applications

Path to Level

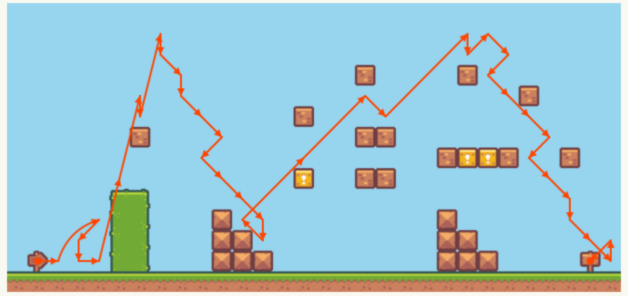




Generate level with desired “solution”

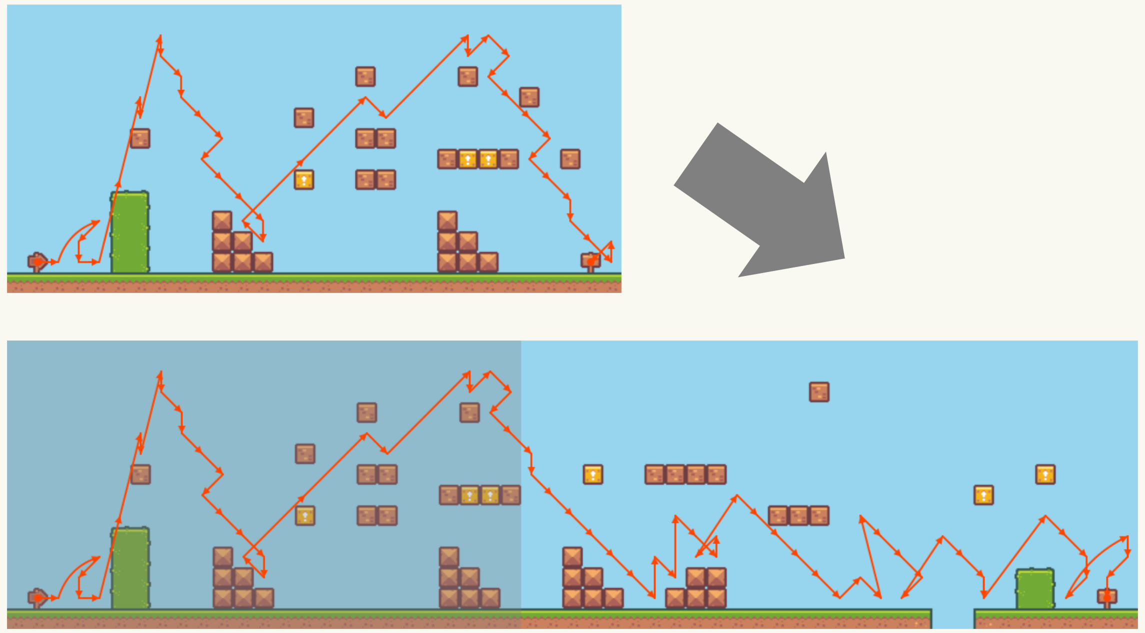
Applications

Extension



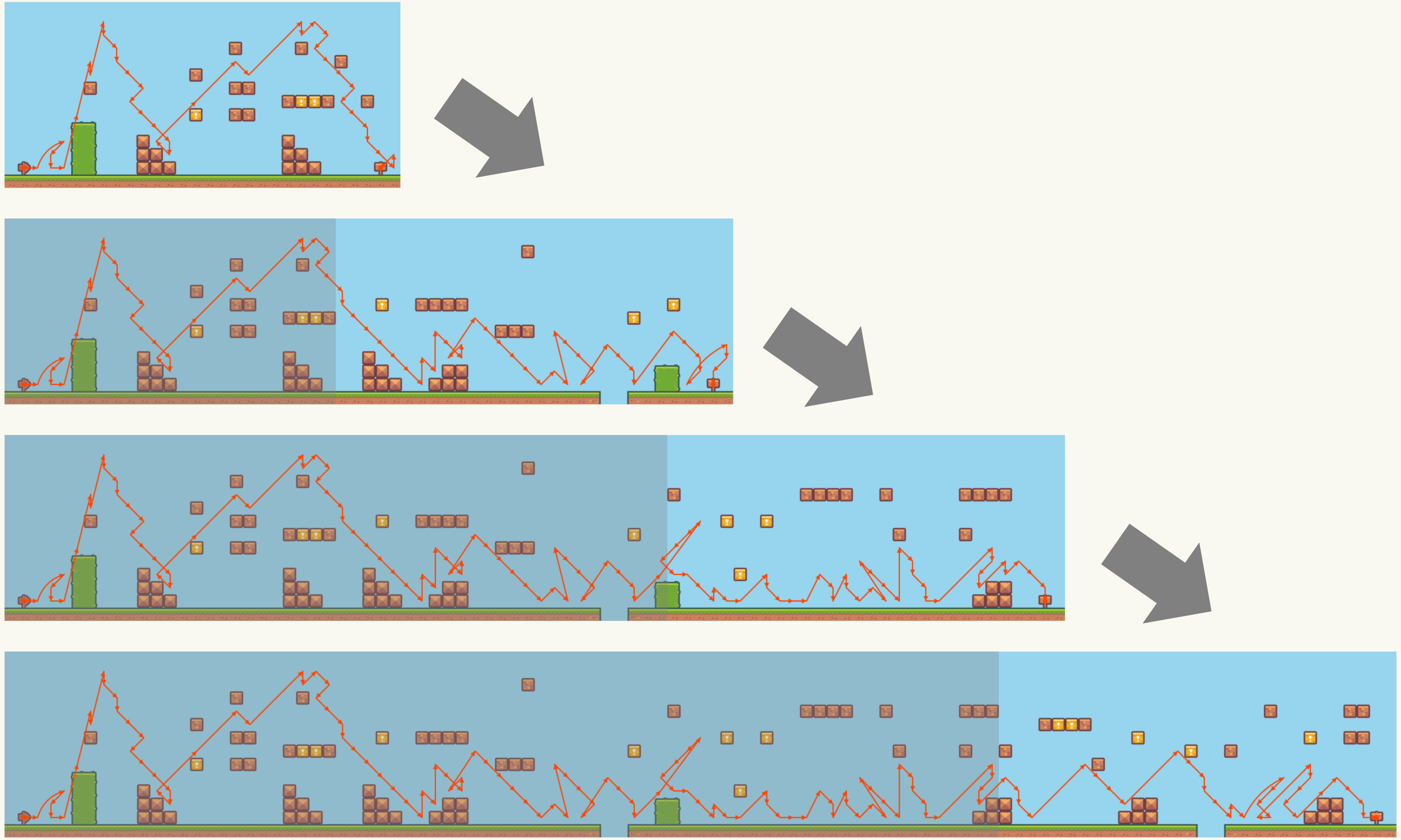
Applications

Extension



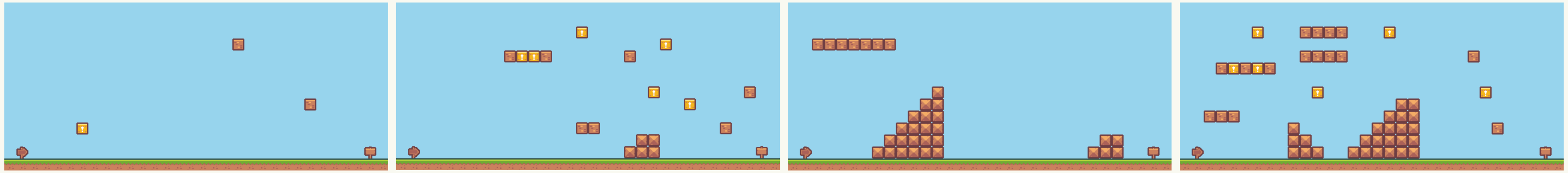
Applications

Extension



Applications

Range

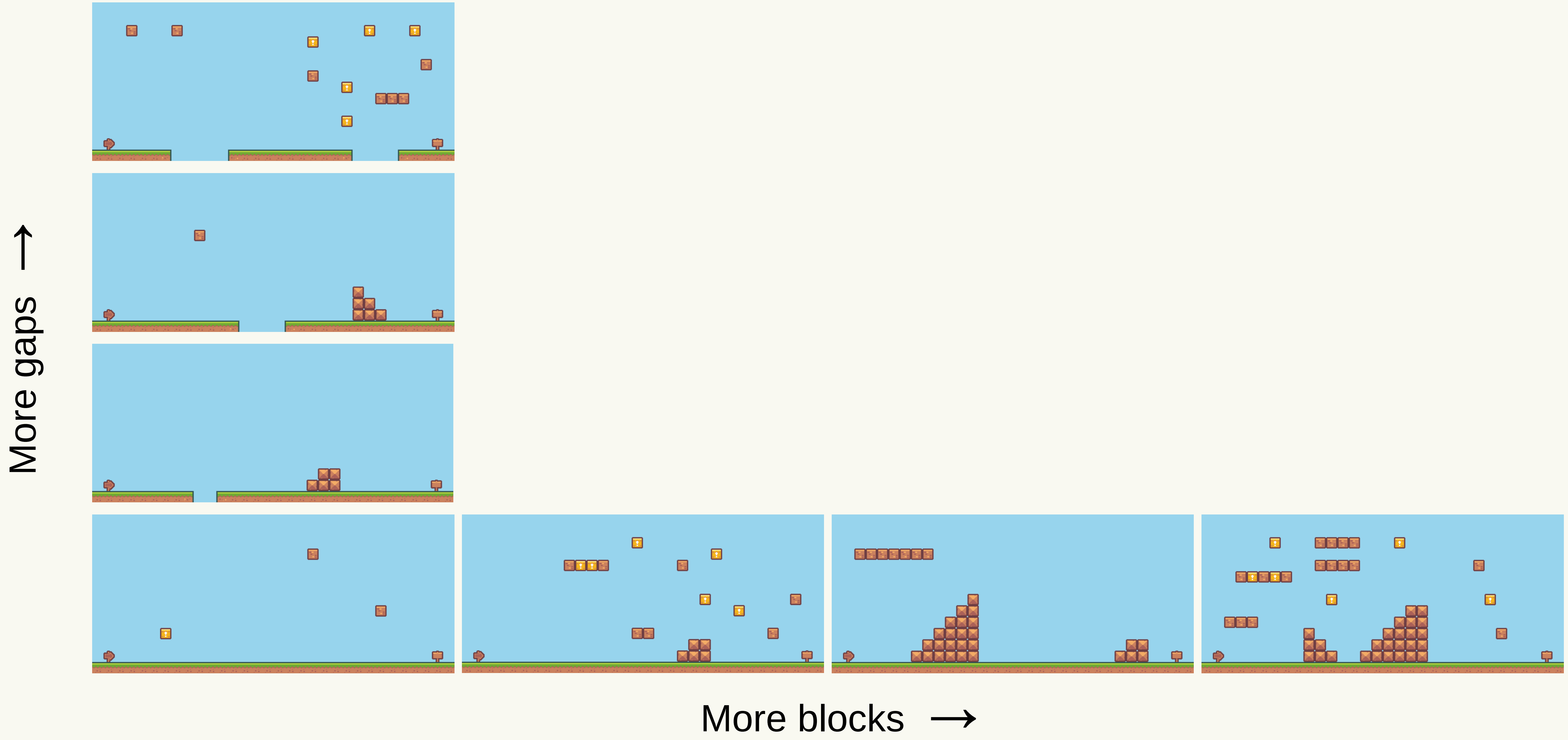


More gaps ⟶

More blocks ⟶

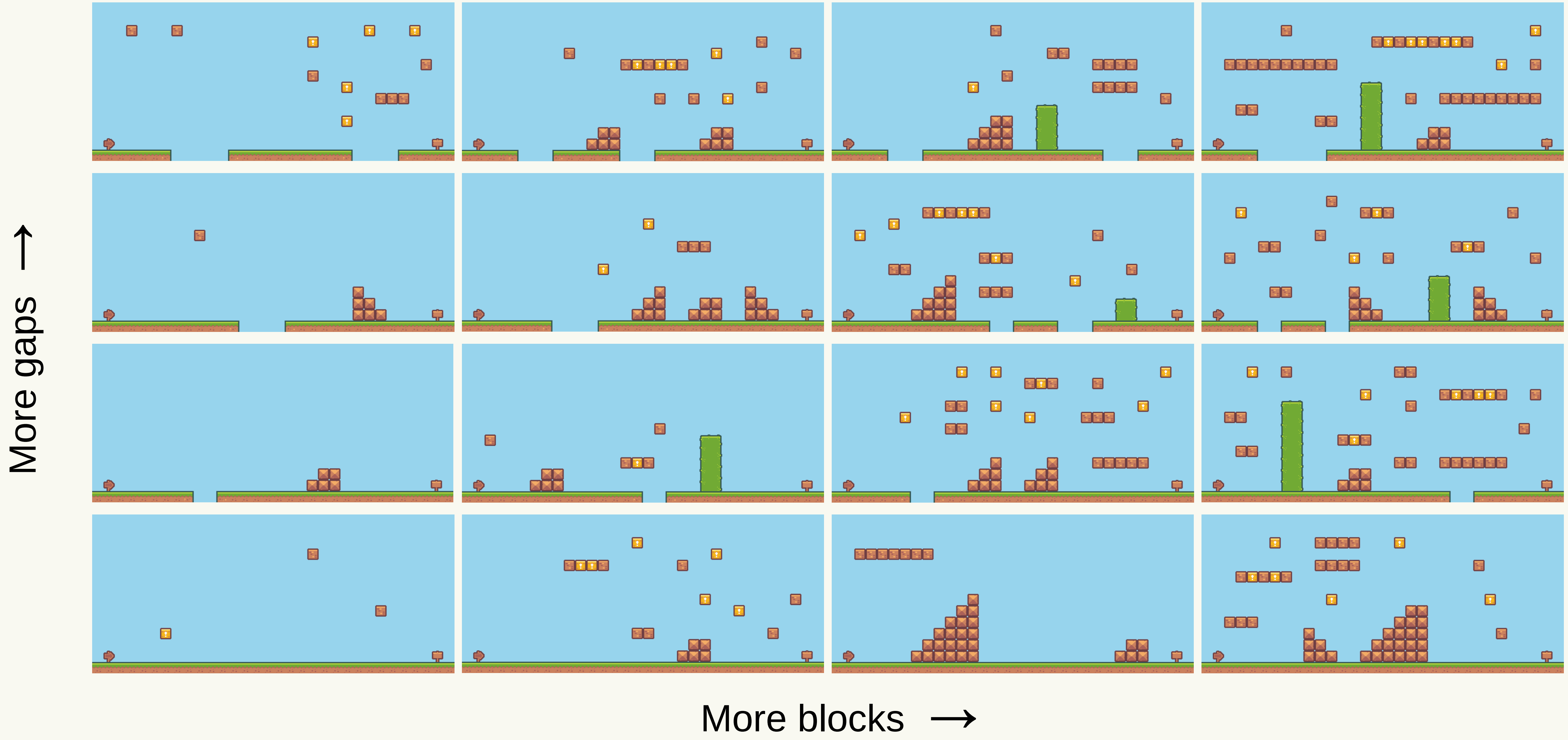
Applications

Range



Applications

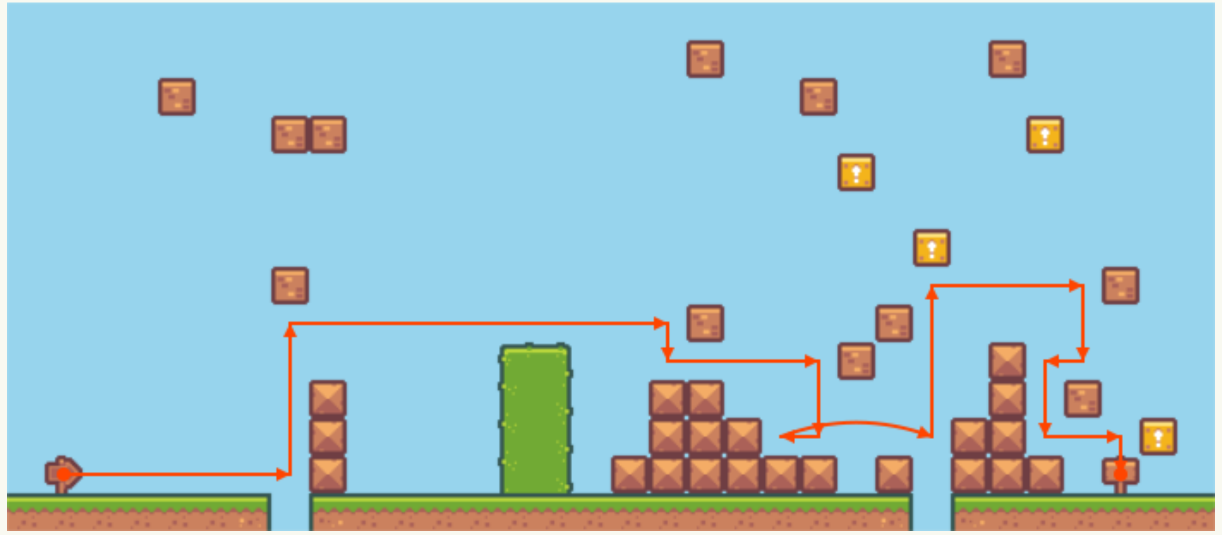
Range

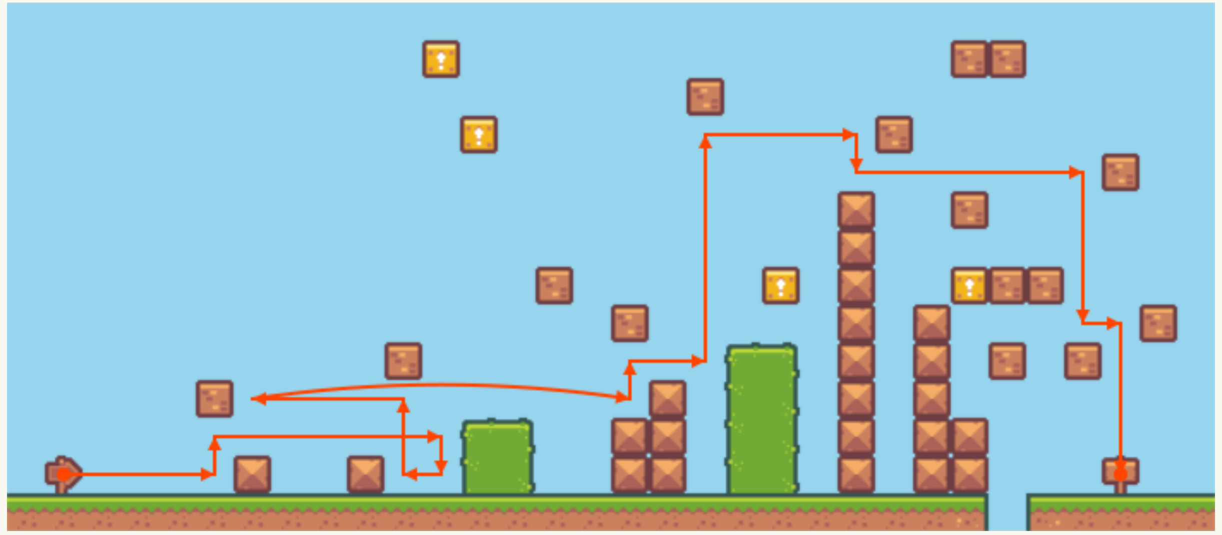


Blending

Patterns and Movement

Platformer played by sliding

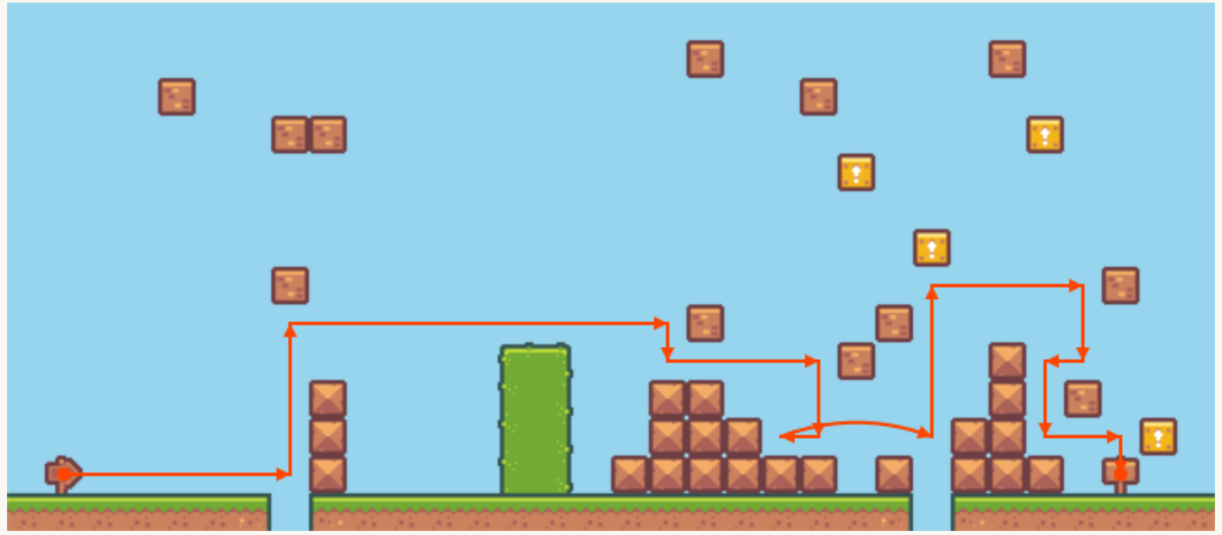


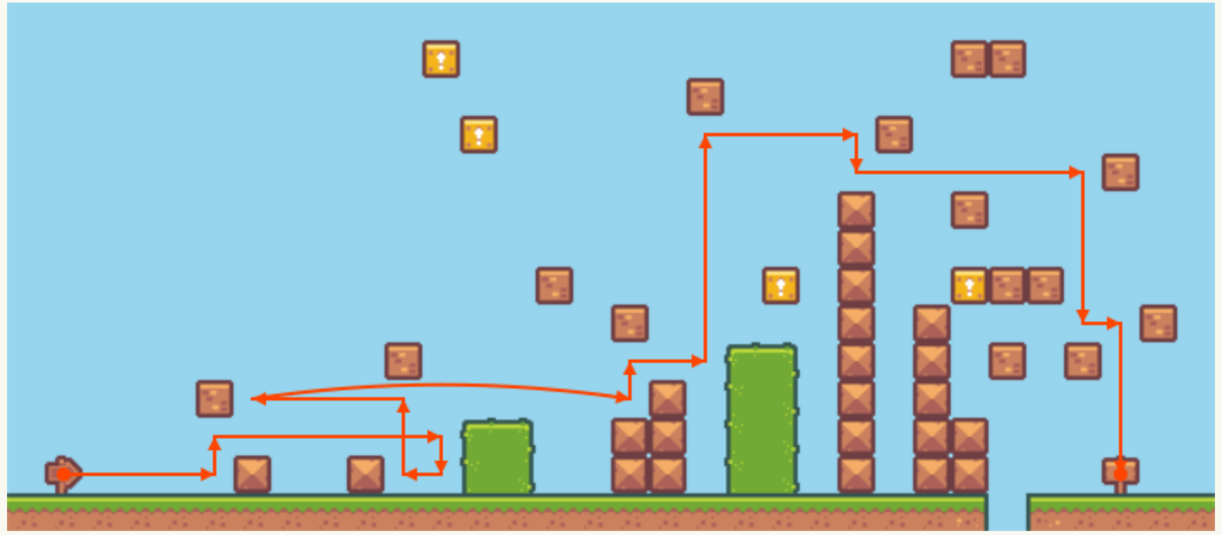


Blending

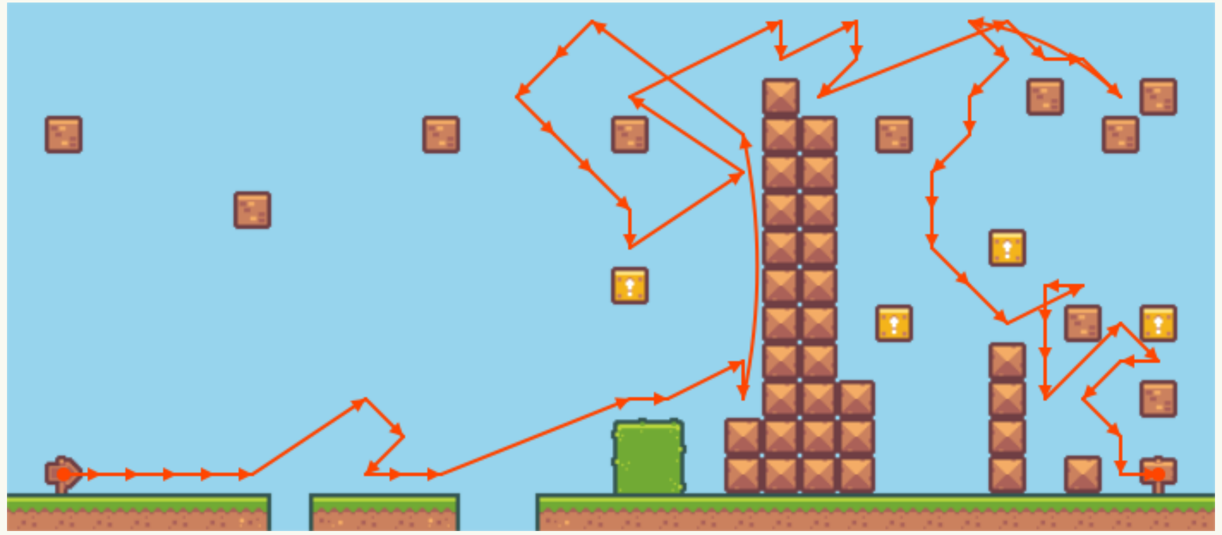
Patterns and Movement

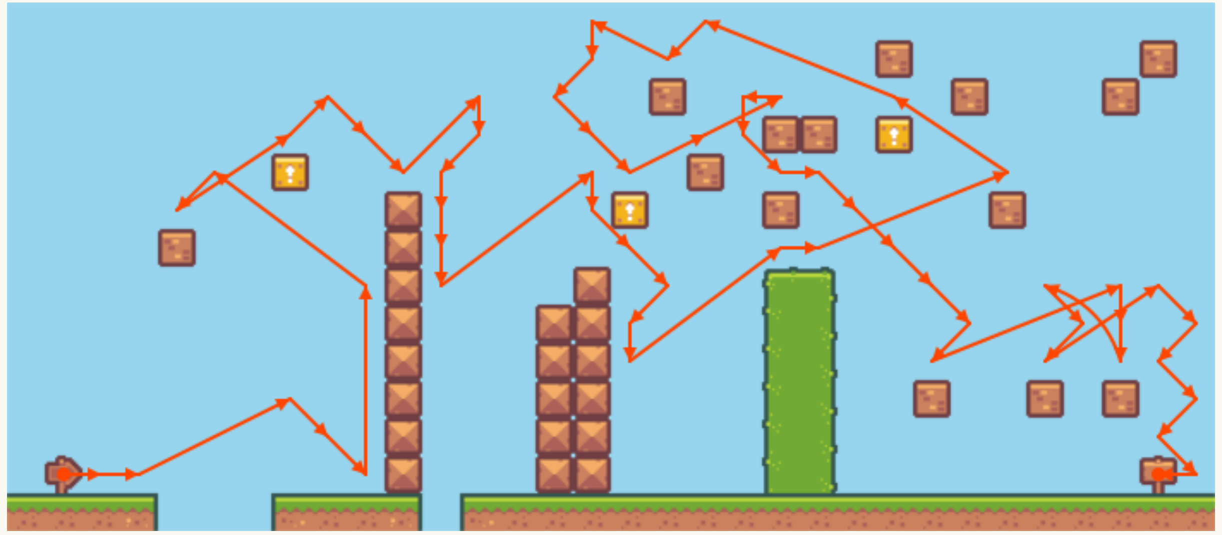
Platformer played by sliding





Platformer played by vertical platformer

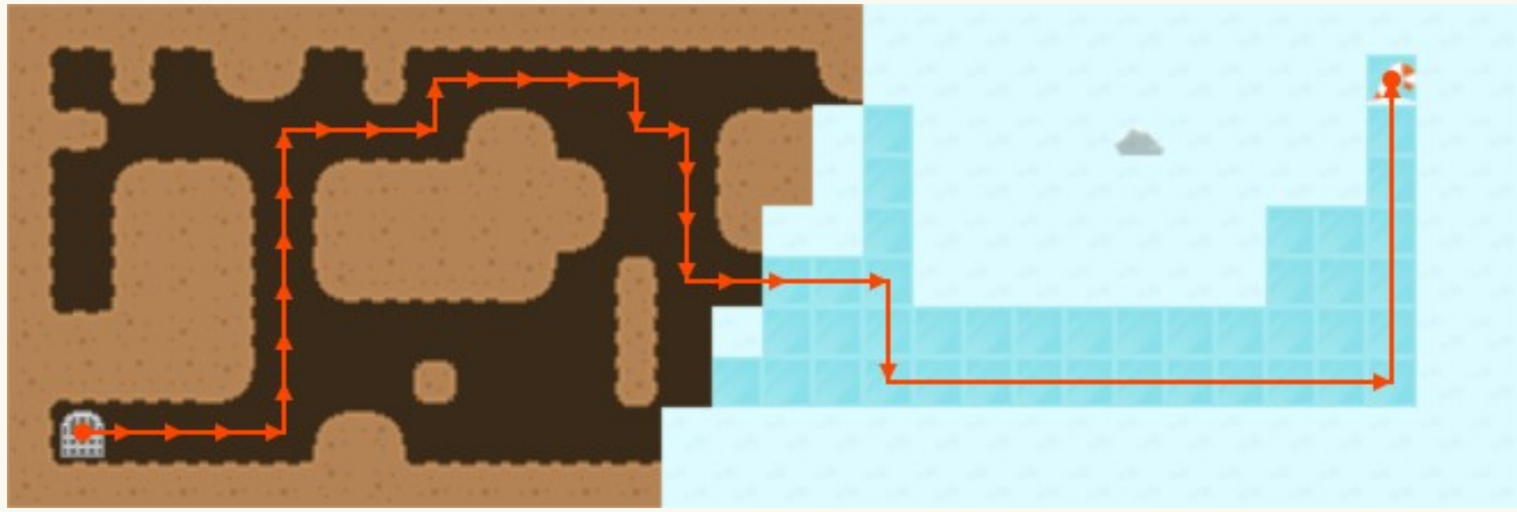




Blending

Multi-Game Levels

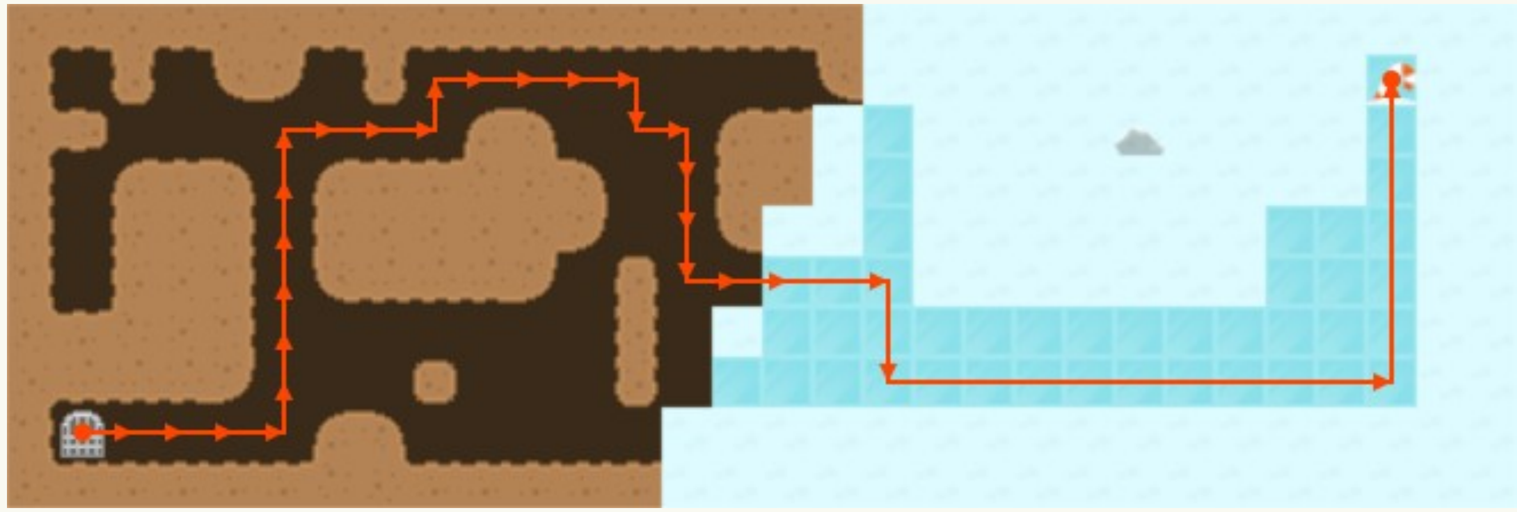
Cave to sliding



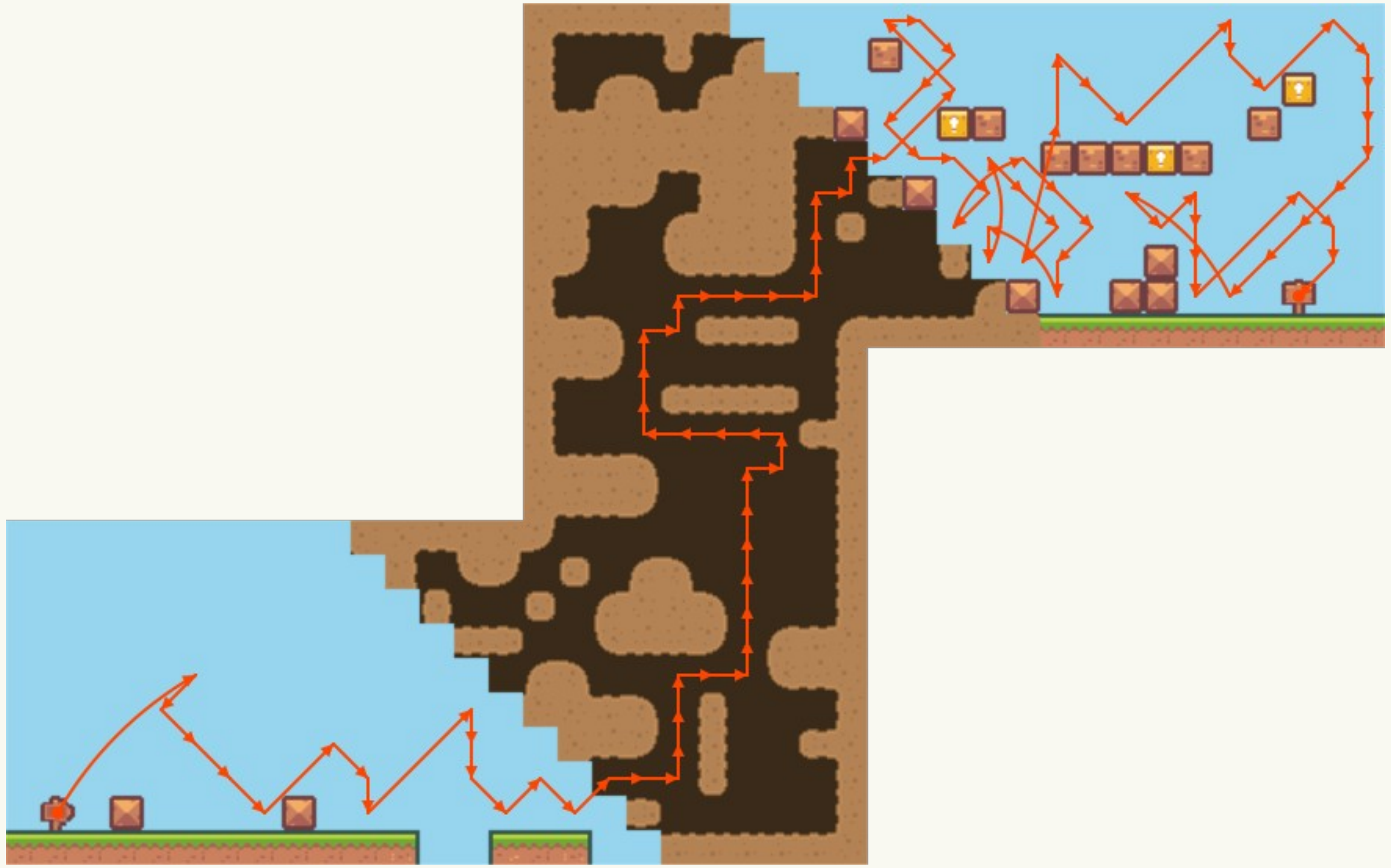
Blending

Multi-Game Levels

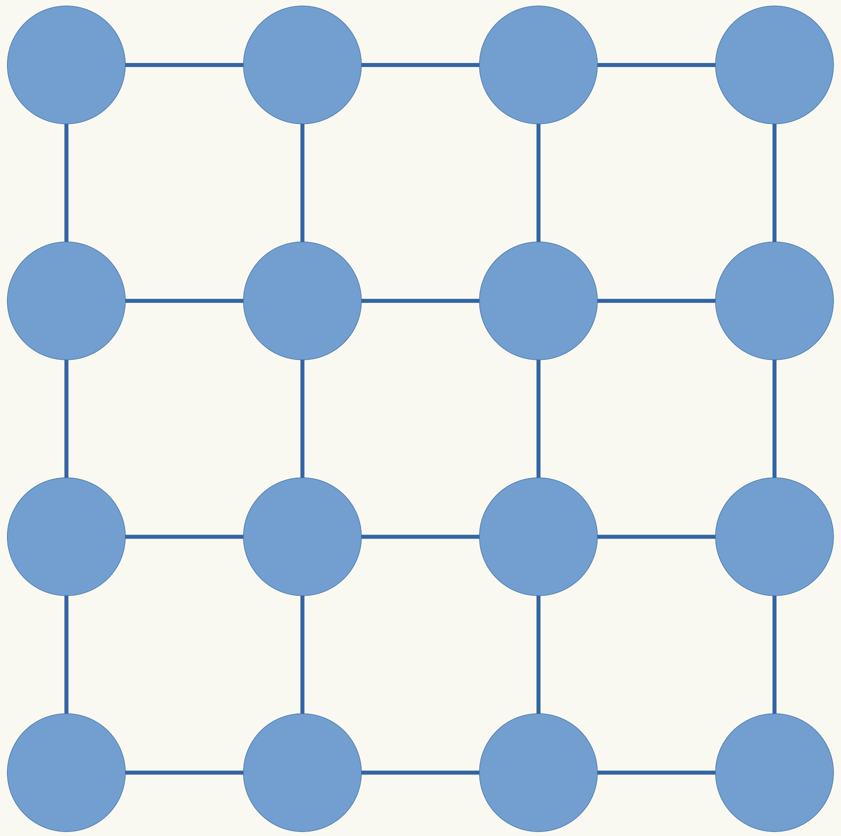
Cave to sliding



Platformer to cave to platformer



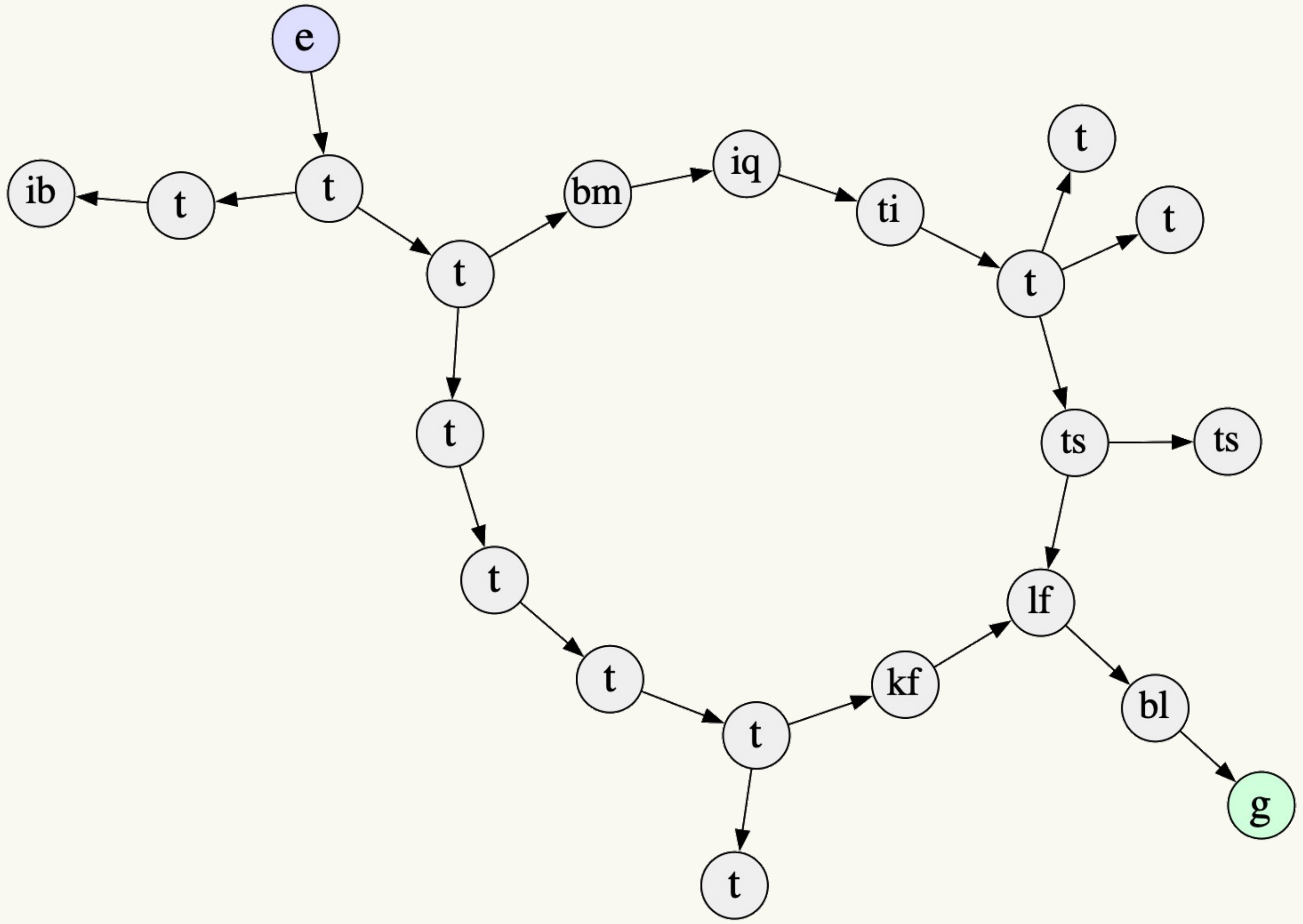
Off the Grid - Graph Generation

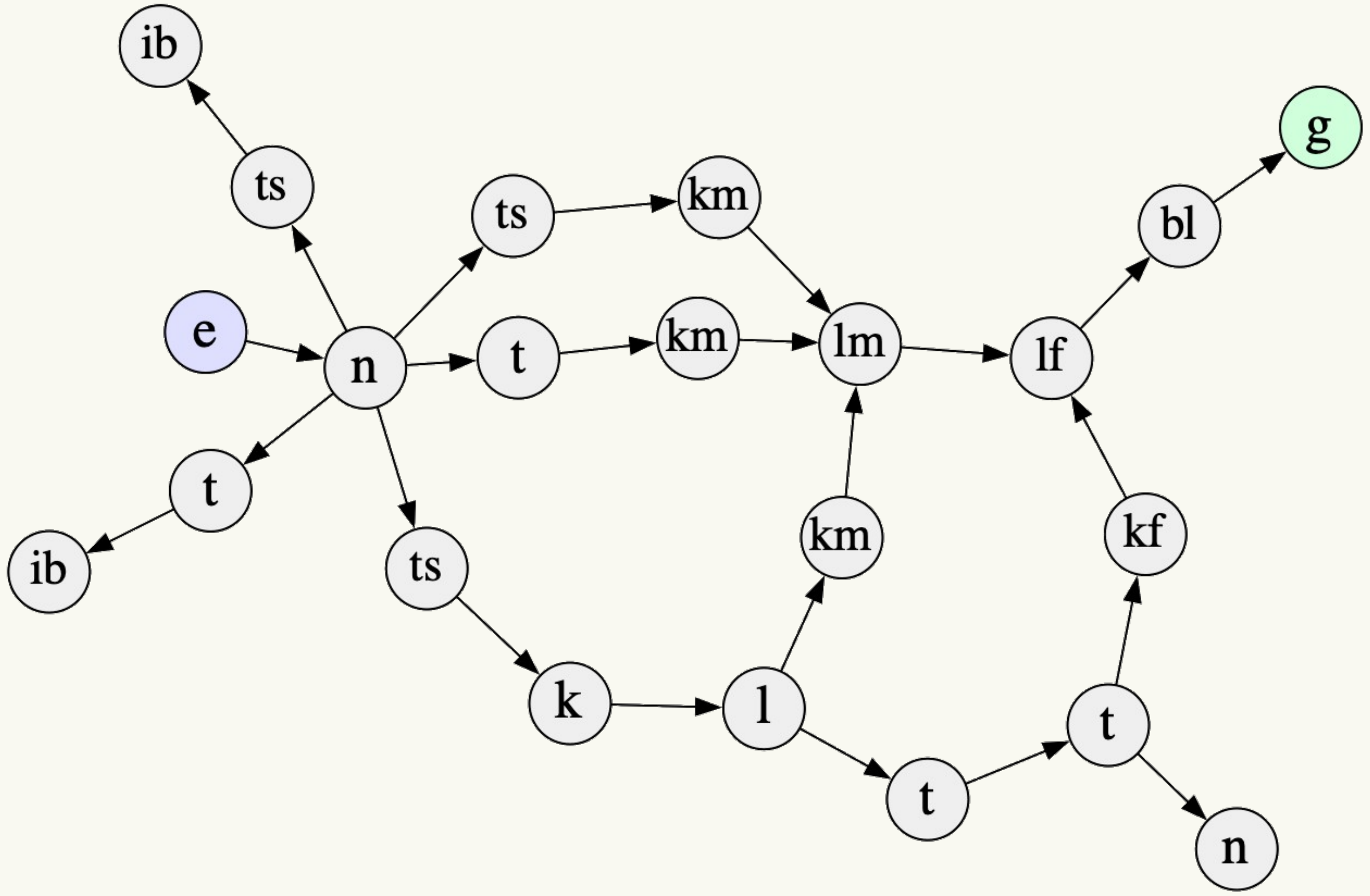


- 2D grid is a special case of graphs
  - (Note: this is not the reachability graph but the tile grid)
- Same general concept:
  - learn local patterns from example(s)
  - generate graphs with only with those patterns
- To learn from / generate graphs:
  - Variables for the graph structure (e.g. how “tiles” neighbor each other)
  - Constraints on structure (e.g. what local connectivity can be, must be connected, be a tree, etc)

Graph Generation

Abstract Missions



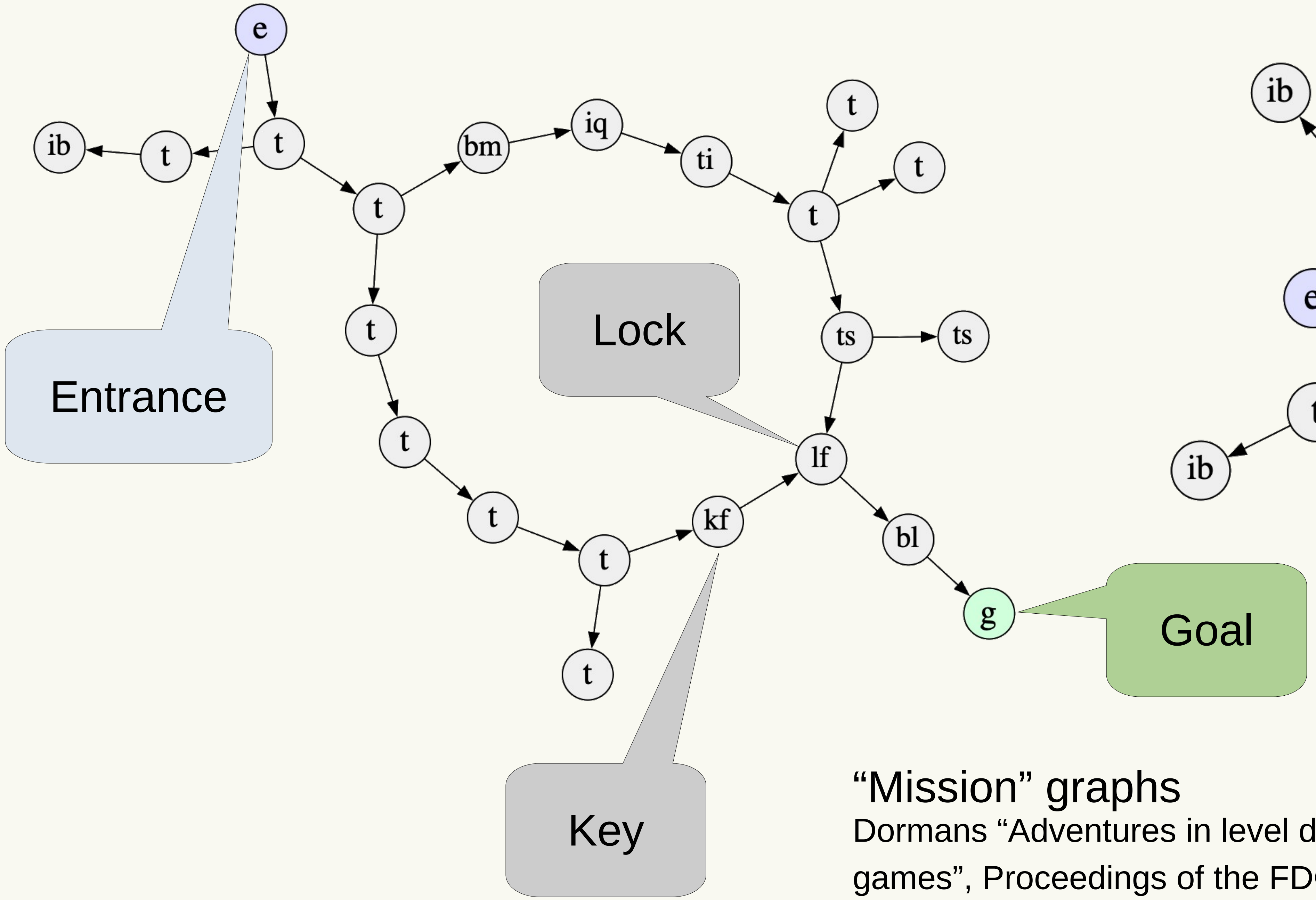


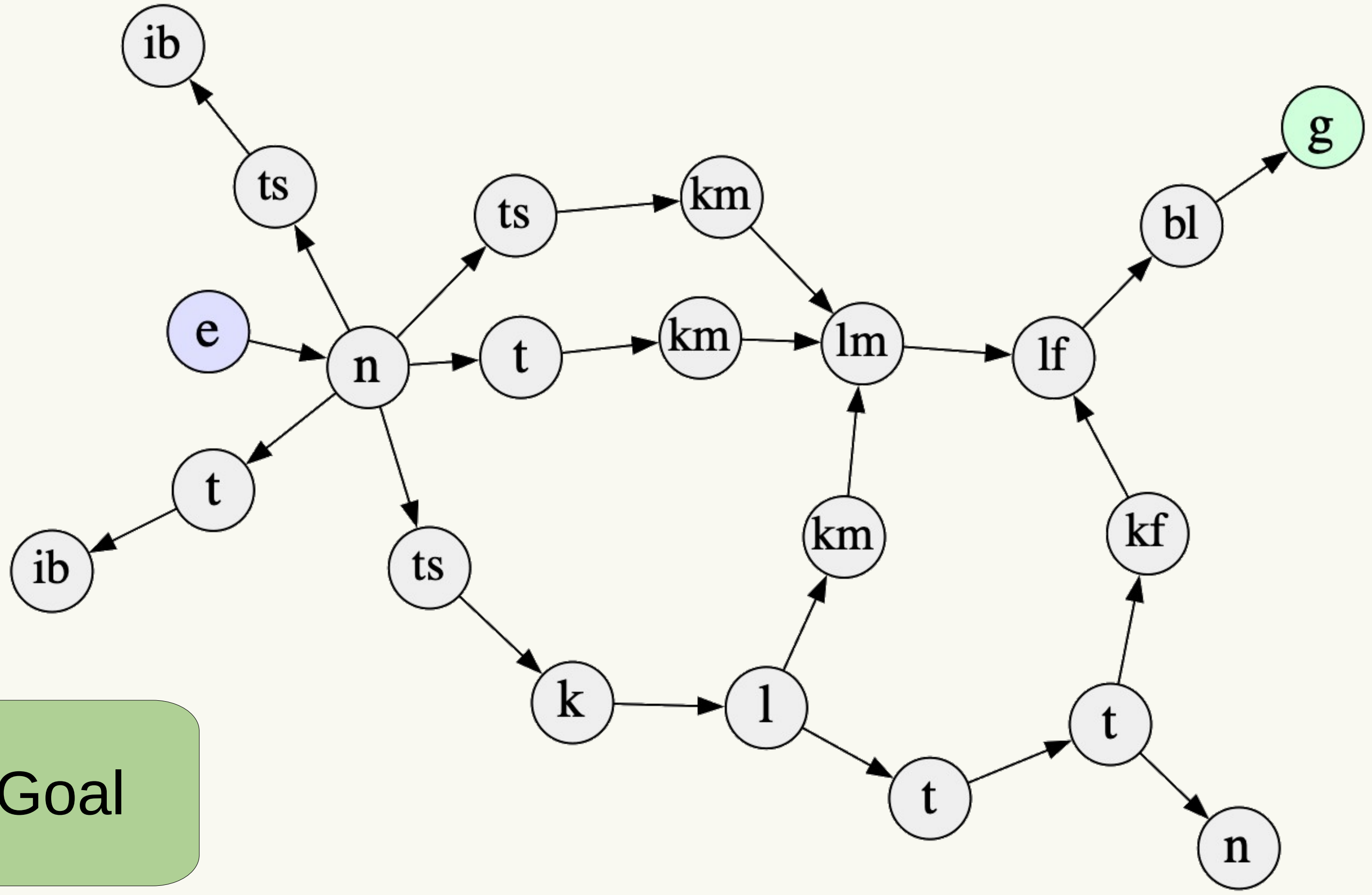
“Mission” graphs

Dormans “Adventures in level design: generating missions and spaces for action adventure games”, Proceedings of the FDG Workshop on Procedural Content Generation (2010)

Graph Generation

Abstract Missions



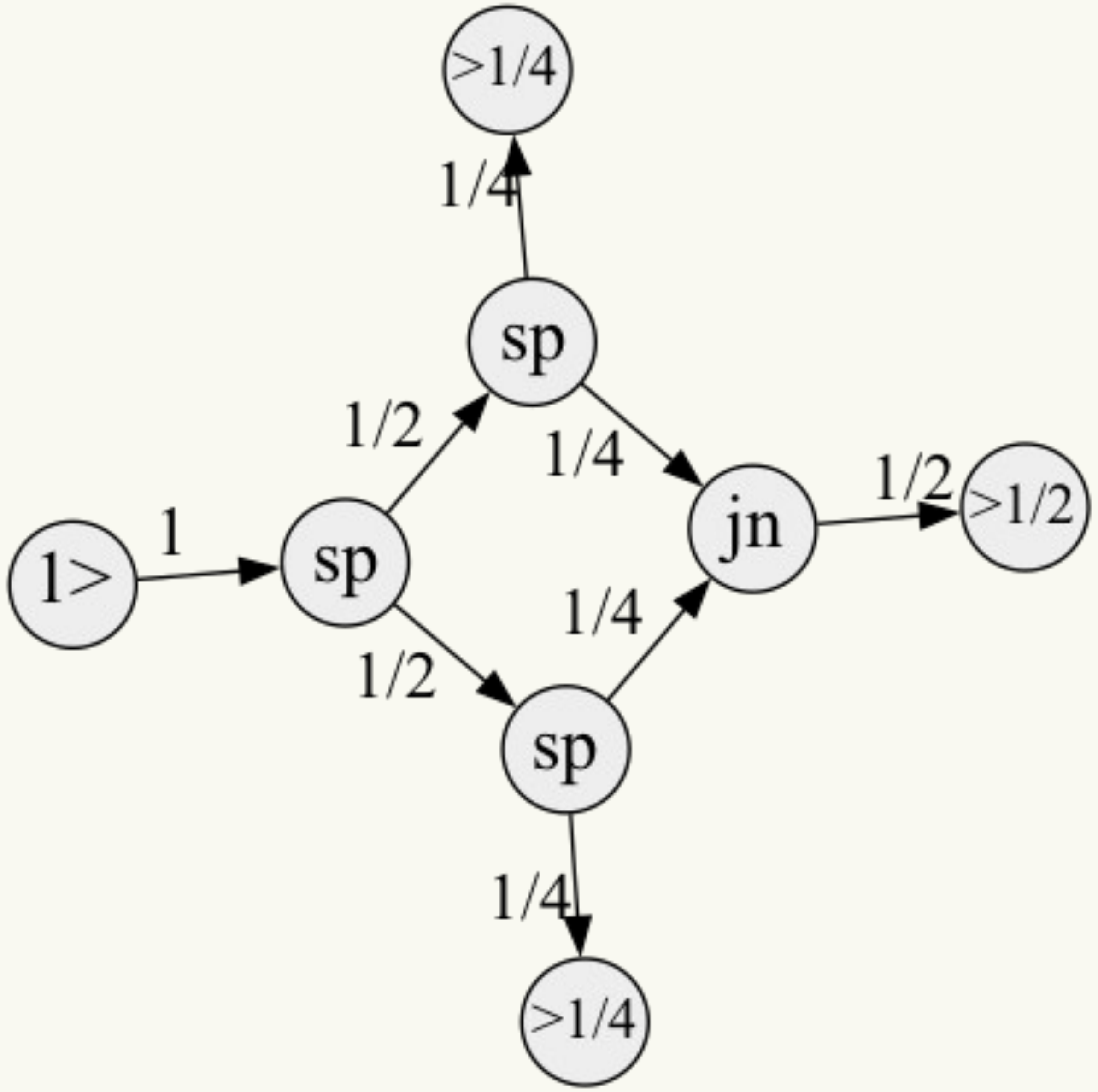


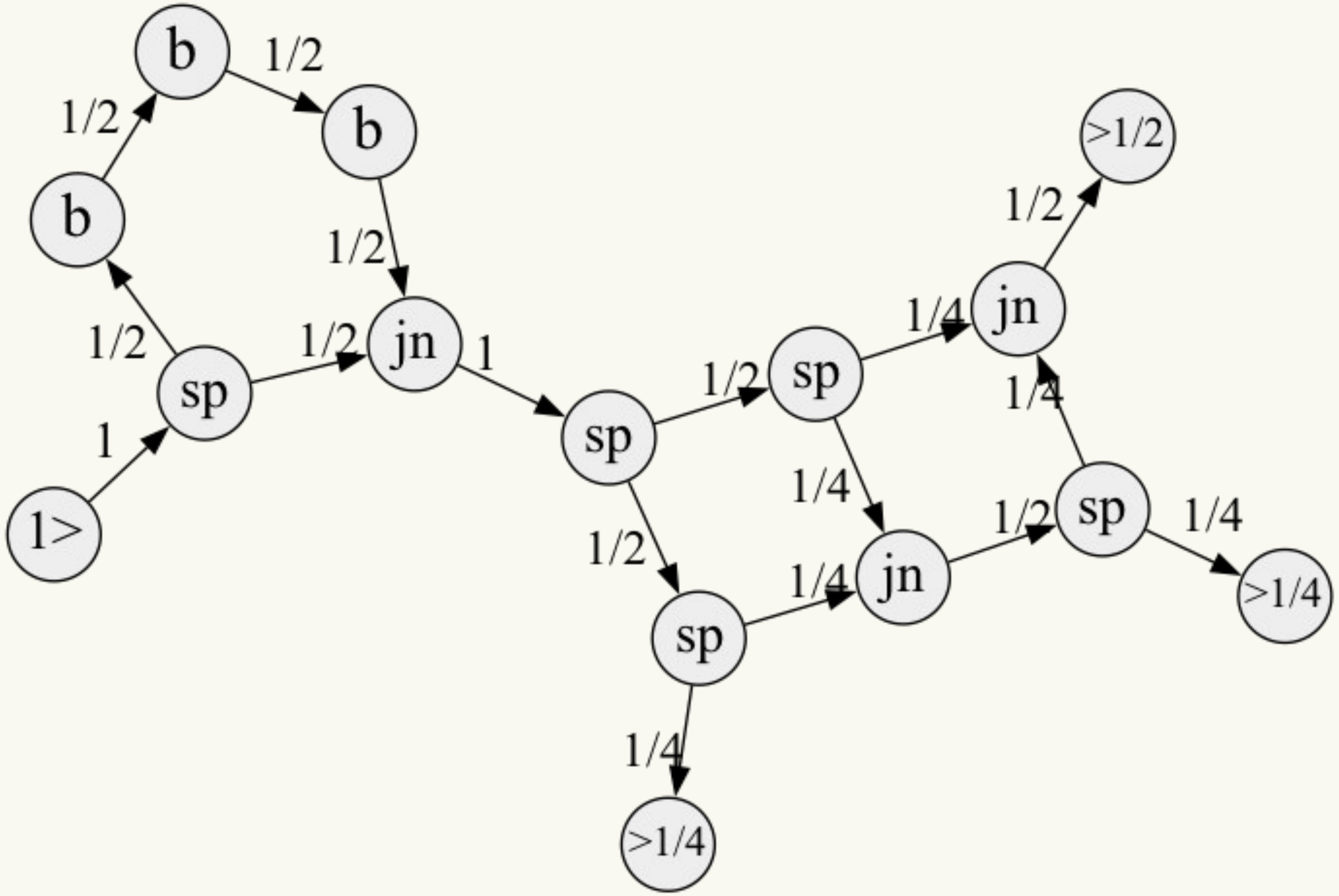
“Mission” graphs

Dormans “Adventures in level design: generating missions and spaces for action adventure games”, Proceedings of the FDG Workshop on Procedural Content Generation (2010)

Graph Generation

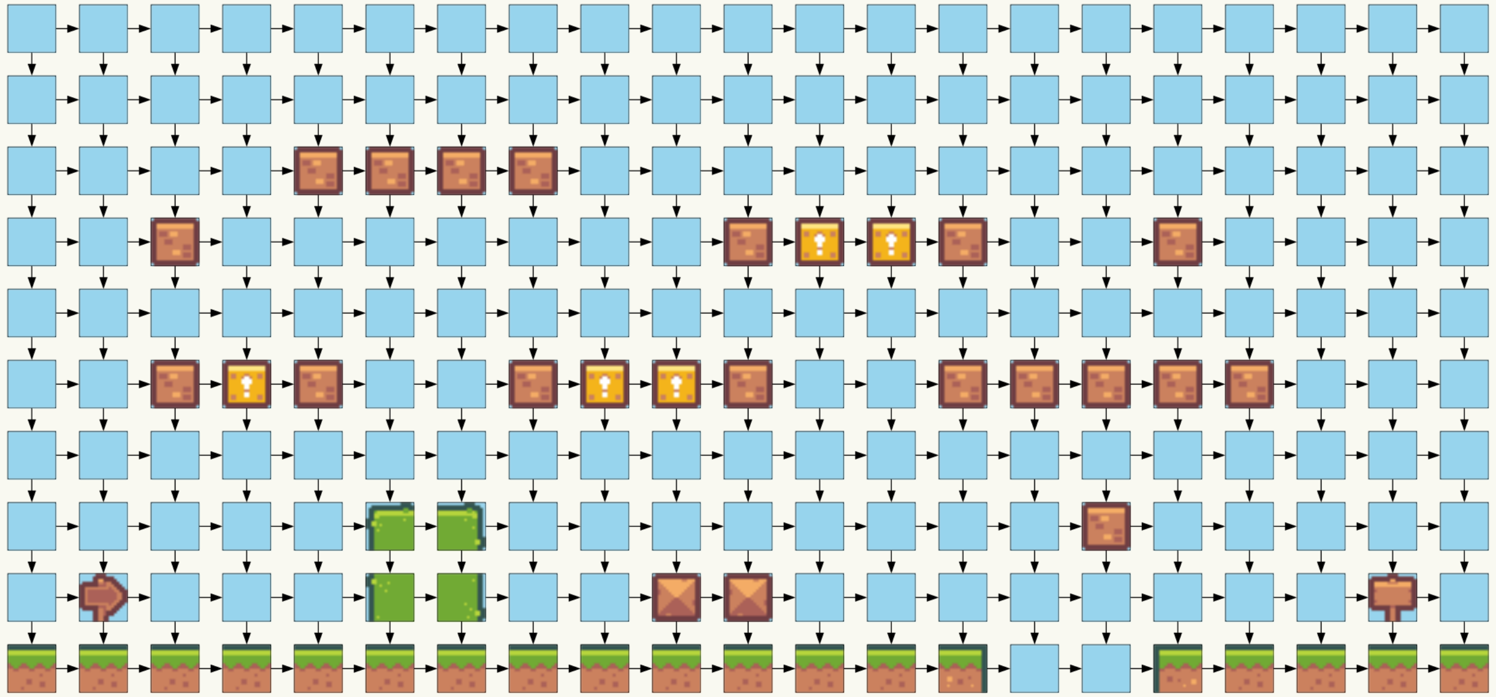
Fractions





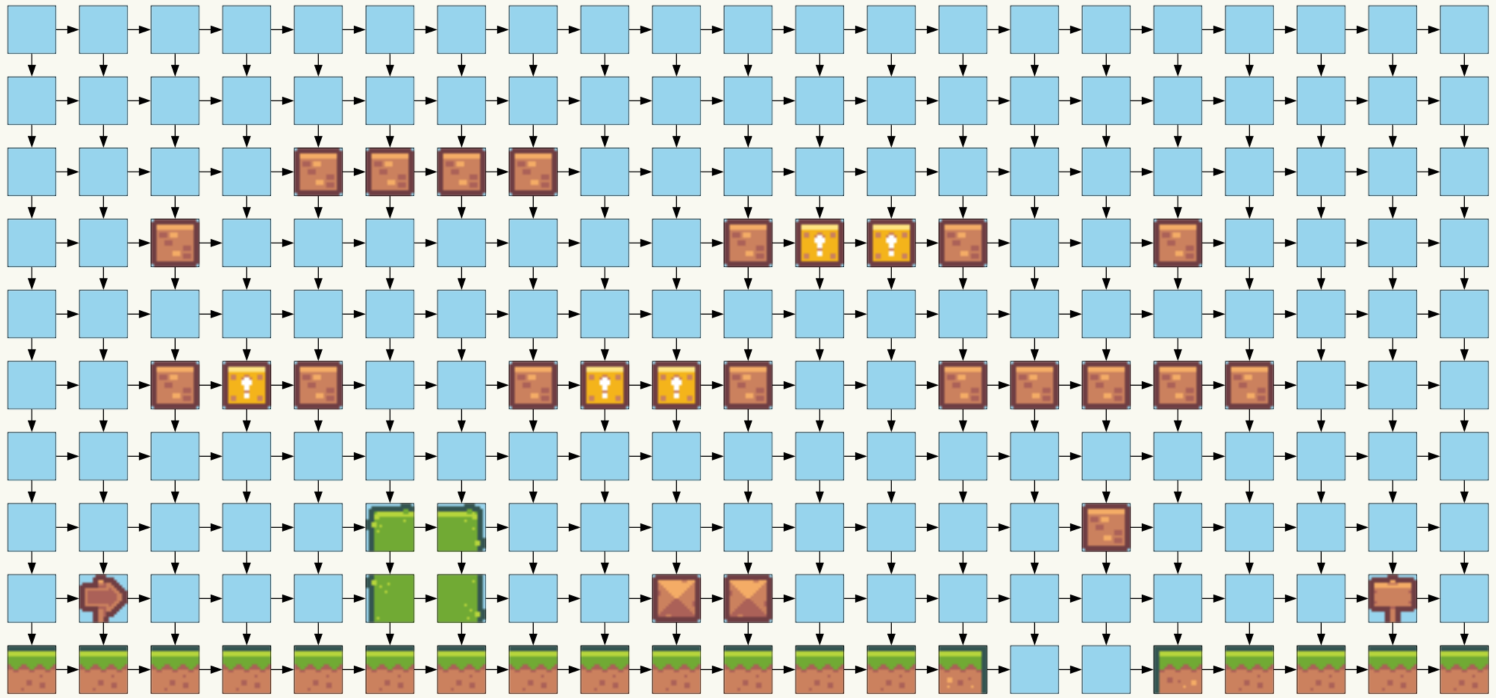
Graph Generation

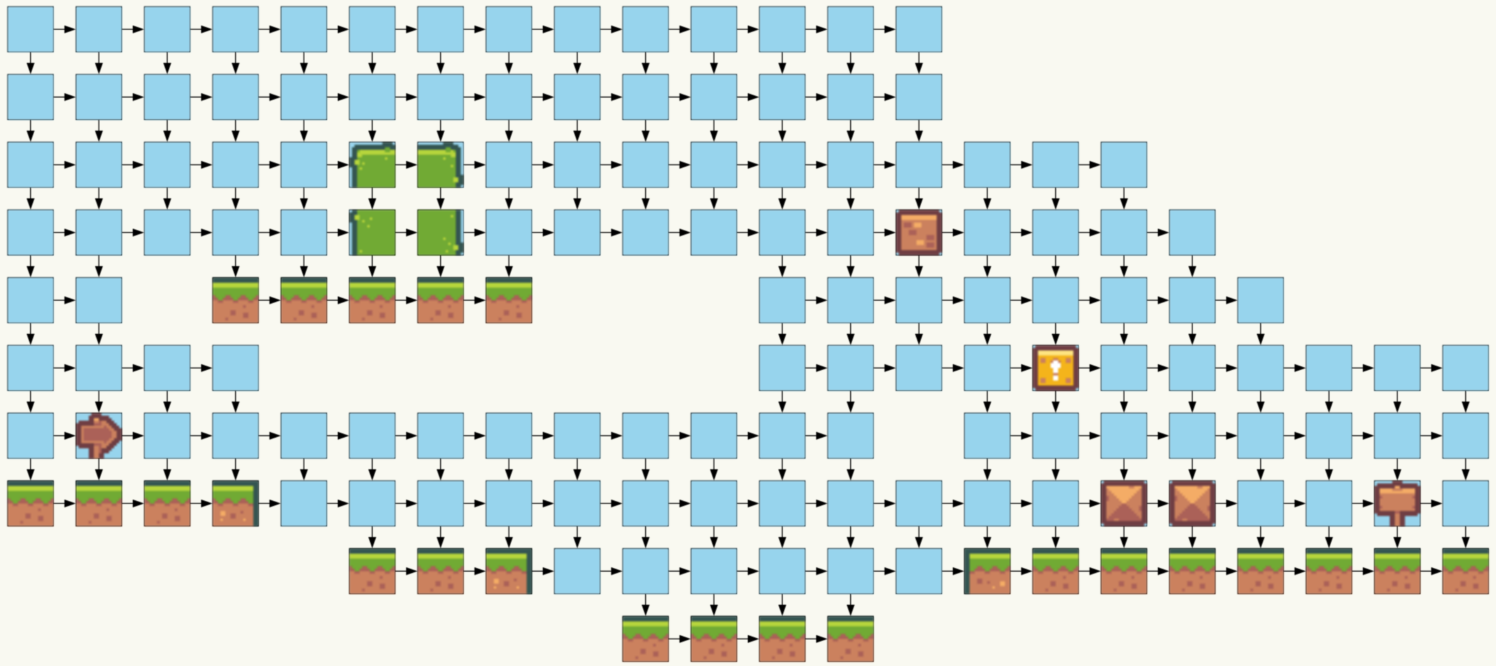
Flexible "Grids"



Graph Generation

Flexible “Grids”

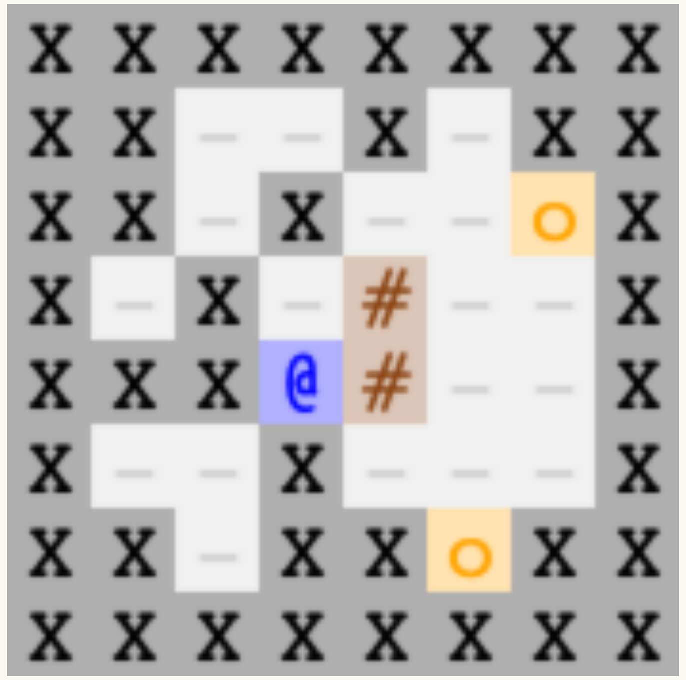


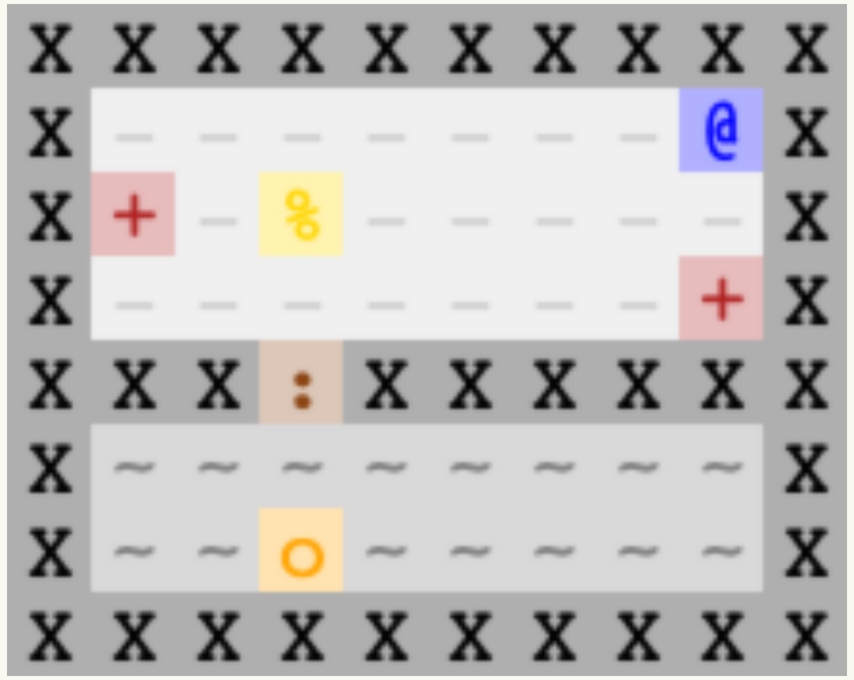


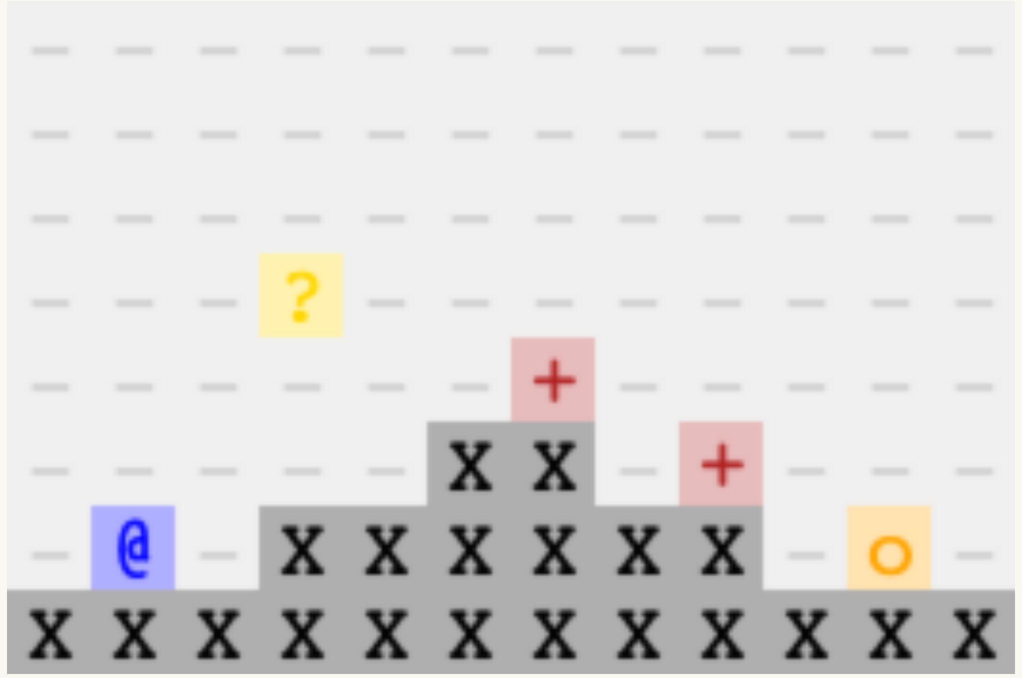
Game Mechanics

- Add another dimension - Time
- Model game mechanics as tile replacement rules
  - Inspired by Gumin's Markov Junior
  - Various ways of grouping and ordering
- Basic setup:
  - Level generation constrains timestep 0
  - Replacement rules constrain changes between timesteps
  - Level must be solved by the last timestep
- Solution is a level **and** example playthrough that level is completable!

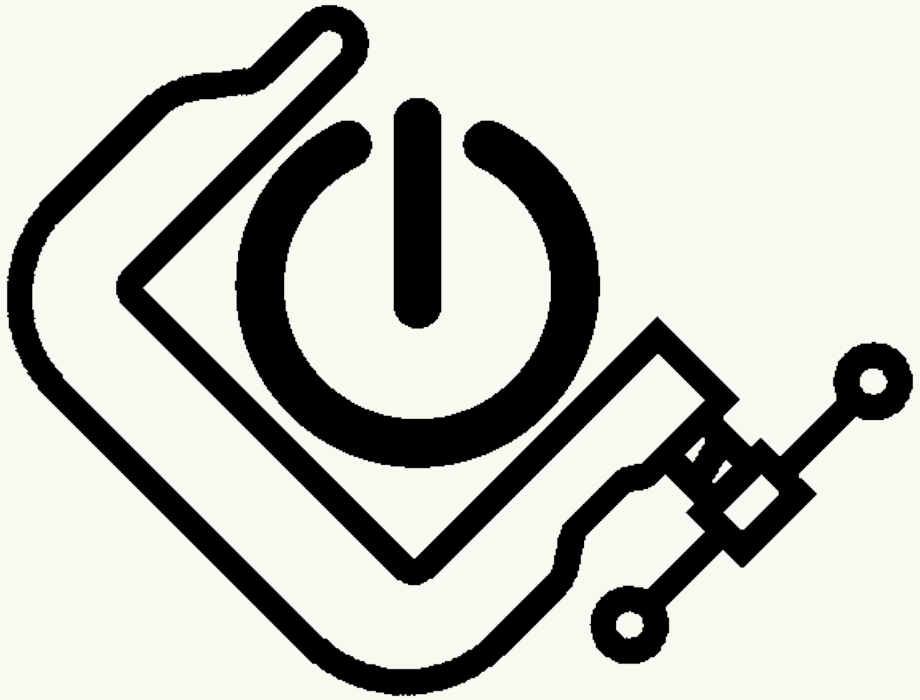
Game Mechanics





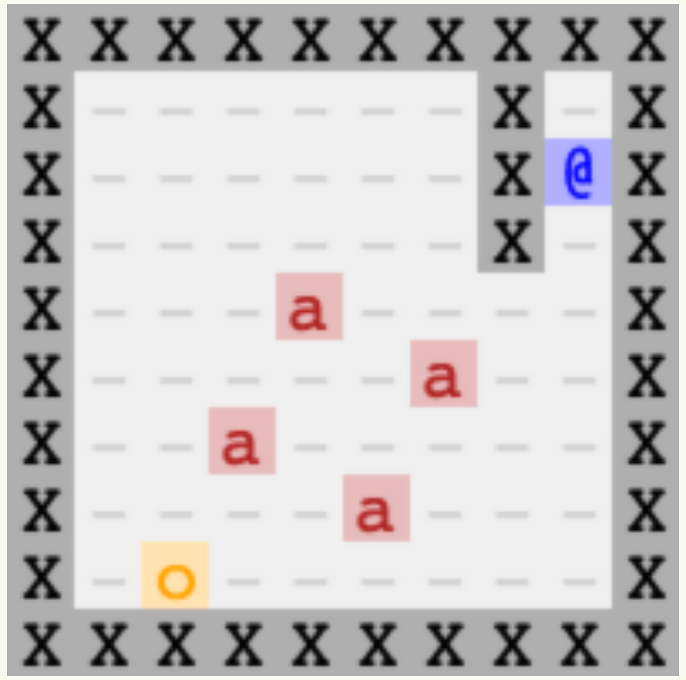


Summary



- Constraint solving can be a powerful and flexible technique for level generation (and editing)
- Can learn from few examples and provide guarantees on generated contented (e.g. path through level)
- Application of general solvers allows a variety of design constraints to be expressed, may benefit from general improvements

Thanks!



https://github.com/crowdgames/sturgeon-pub

Seth Cooper
http://sethcooper.net/
seth.cooper@gmail.com

Image tiles from Kenney:
https://www.kenney.nl/

Thanks to:
Colan Biemer, Anurag Sarkar,
Adam Smith, Pete Manolios, Andrew Walter,
Northeastern Game Research Seminar

- Seth Cooper. **Sturgeon: tile-based procedural level generation via learned and designed constraints.** Proceedings of the Eighteenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (2022).
- Seth Cooper. **Constraint-based 2D tile game blending in the Sturgeon system.** Proceedings of the Experimental AI in Games Workshop (2022).
- Seth Cooper. **Sturgeon-GRAPH: Constrained Graph Generation from Examples.** Proceedings of the 17th International Conference on the Foundations of Digital Games (2023, to appear).
- Seth Cooper. **Sturgeon-MKIII: Simultaneous Level and Example Playthrough Generation via Constraint Satisfaction with Tile Rewrite Rules.** Proceedings of the FDG Workshop on Procedural Content Generation (2023, accepted)