University of BRISTOL

The computational complexity of computer games
Making Mario work hard

Benjamin Sach

Disclaimer: Many of the diagrams in this talk are taken (with or without modification) from the original papers which are cited at the end.


Radioactive girl! 'Superhero' scientist


Orville the Duck ventriloquist Keith


The SNP are 'openly racist', says Farage


Lottery winner killed by train had


Are YOU stressed? Here's the 10


Dog wardens to use DNA tests on


Firing to kill

It's official - Candy Crush is HARD: Study reveals that game belongs to group of complex mathematical problems

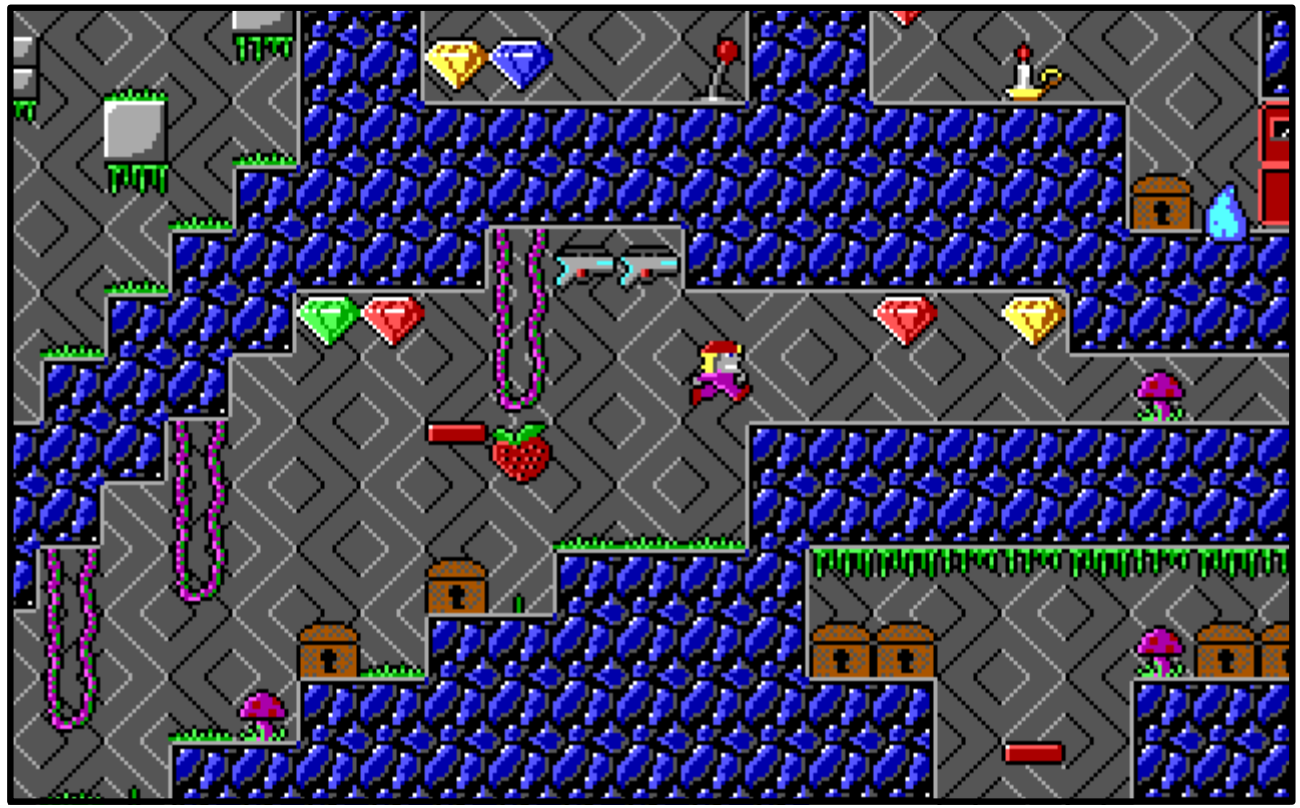
- Candy Crush Saga involves swapping candies to make rows of three
- Study wanted to test whether this game was what's called NP-hard
- NP is a problem that is difficult to check, or can only be solved with a PC
- To test this theory, a researcher constructed a theoretical board of candies
- They then tried to establish a generic formula for solving each puzzle
- This involved trying to prove whether a certain score could be achieved with a set number of swaps
- It wasn't easy to solve, meaning the game was classified as NP-complete

By VICTORIA WOOLLASTON

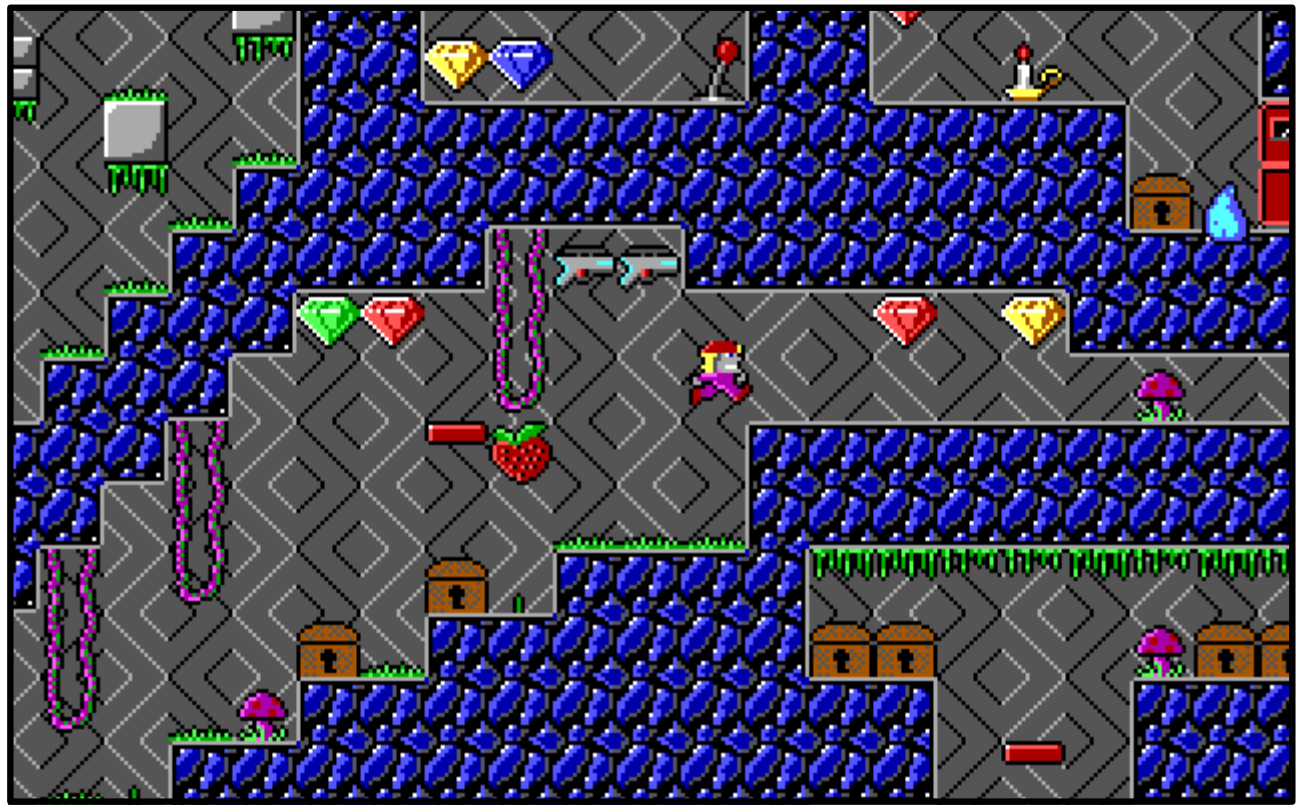
PUBLISHED: 17:52, 12 March 2014 | UPDATED: 12:24, 13 March 2014

Next time someone criticises you for playing too much Candy Crush, you can tell them you're

Crystal Caves

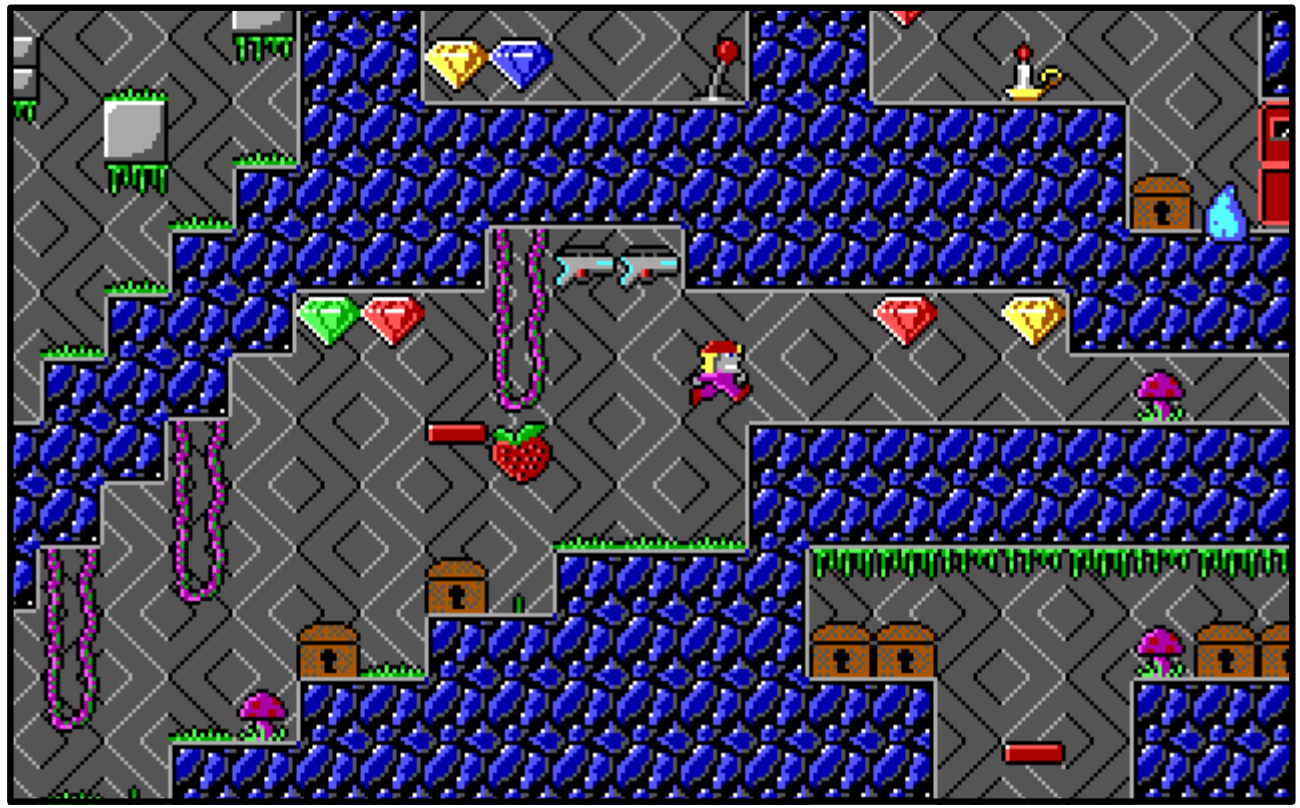


Crystal Caves



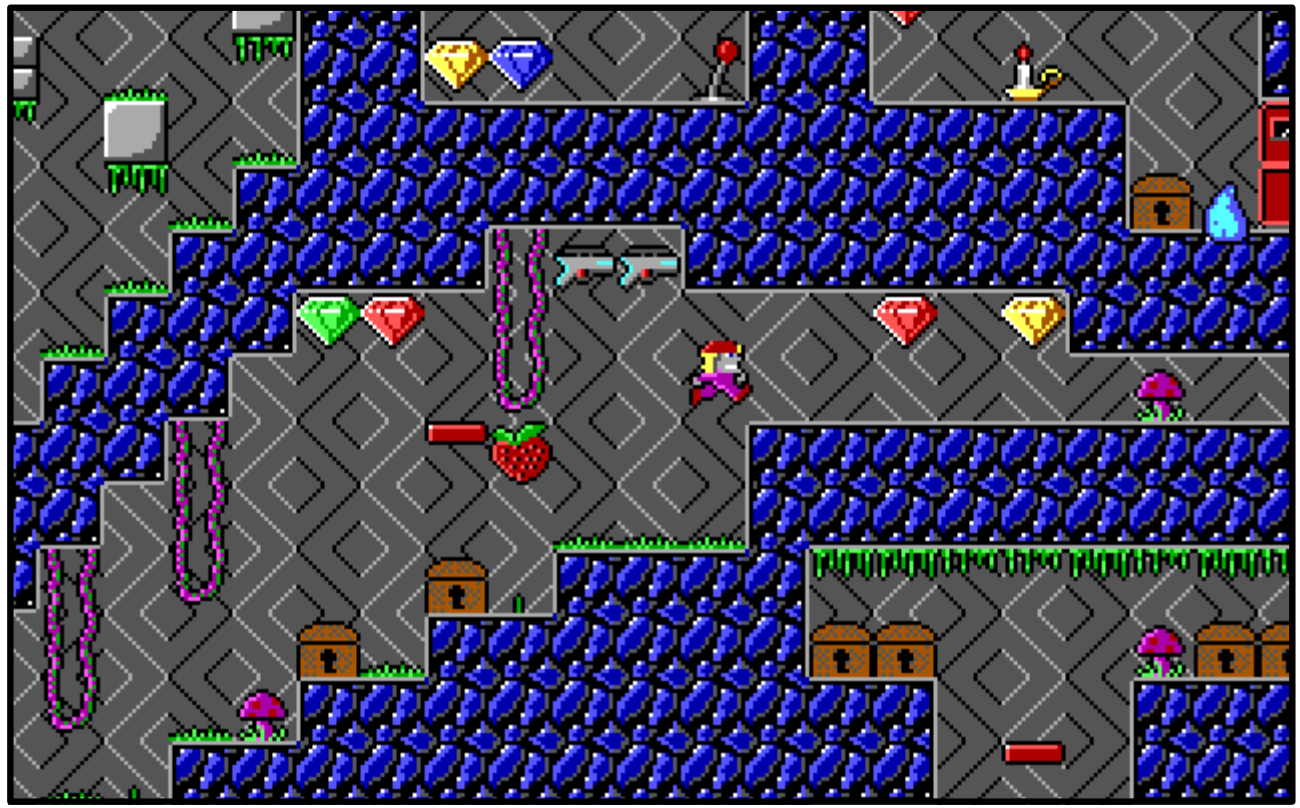
We will prove that Crystal Caves is NP-hard by reduction from 3-SAT

Crystal Caves



We will prove that Crystal Caves is NP-hard by reduction from 3-SAT

Crystal Caves

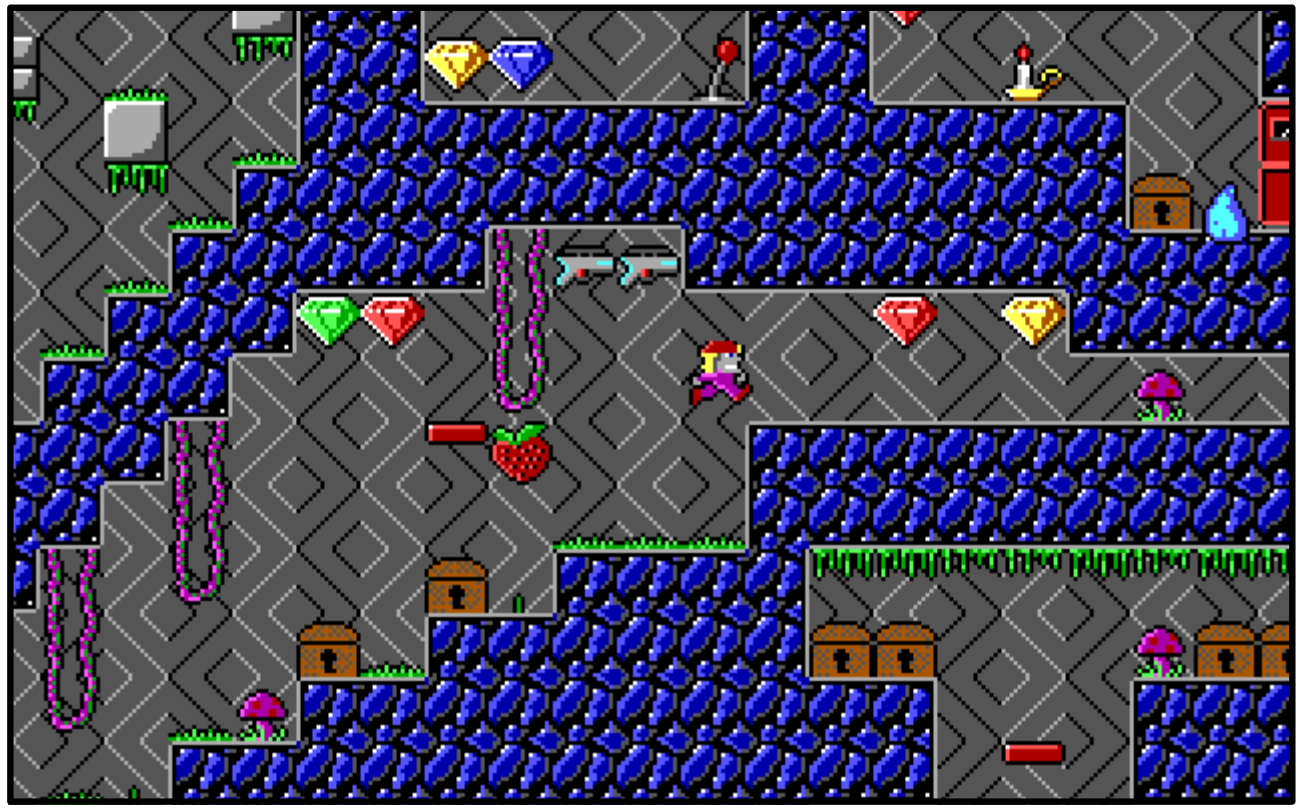


We will prove that Crystal Caves is NP-hard by reduction from 3-SAT

i.e. that Crystal Caves is at least as hard to solve as any problem in NP

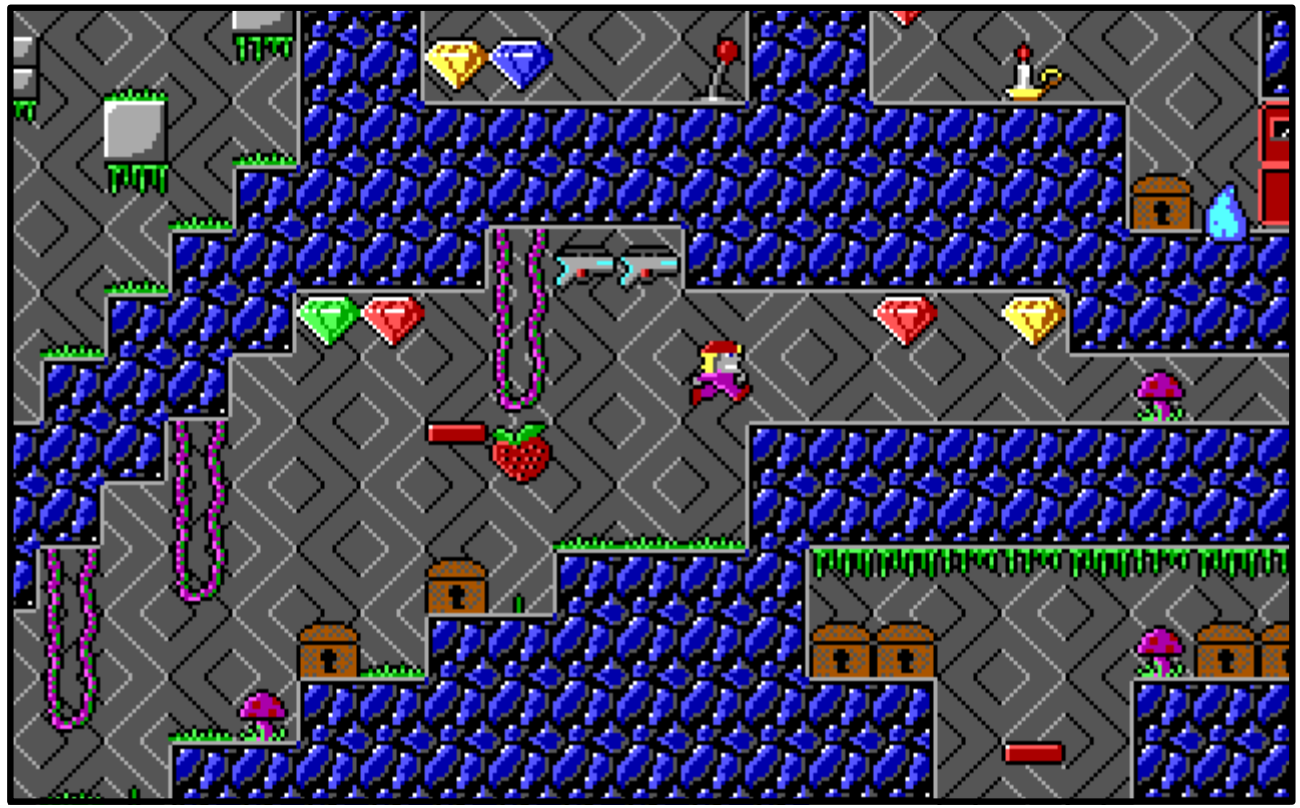
(a problem is in NP *if the answer can be checked in polynomial time)*

Crystal Caves



We will prove that Crystal Caves is NP-hard by reduction from 3-SAT

Crystal Caves



We will prove that Crystal Caves is NP-hard by reduction from 3-SAT

The approach is via a reduction,
we will show that if we can 'solve' Crystal Caves efficiently
then we can solve 3-SAT efficiently

What is 3-SAT?

The 3-SAT problem is to decide whether a given boolean formula can be satisfied.

i.e. can we assign variables so that the formula is satisfied (**True**)*?*

$$(x \vee y \vee z) \wedge (\overline{x} \vee x \vee z) \wedge (\overline{y} \vee \overline{z} \vee \overline{z})$$

What is 3-SAT?

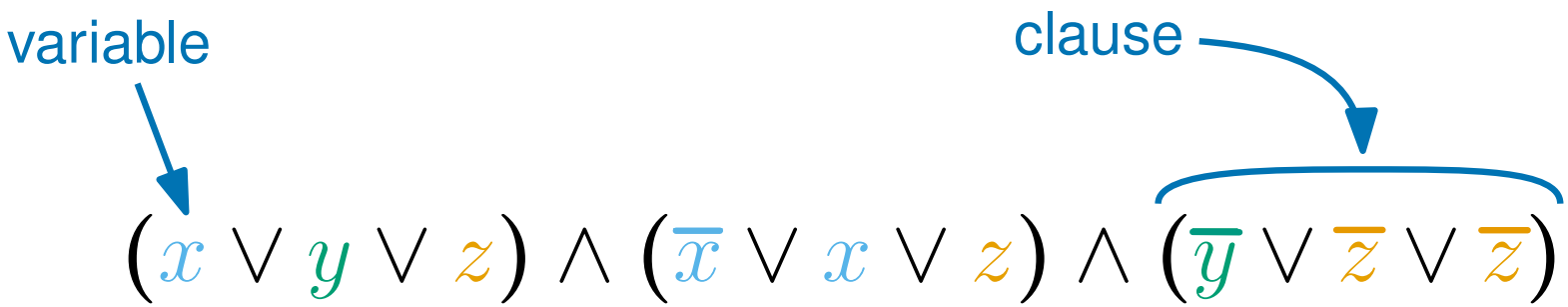
The 3-SAT problem is to decide whether a given boolean formula can be satisfied.

i.e. can we assign variables so that the formula is satisfied (**True**)*?*

What is 3-SAT?

The 3-SAT problem is to decide whether a given boolean formula can be satisfied.

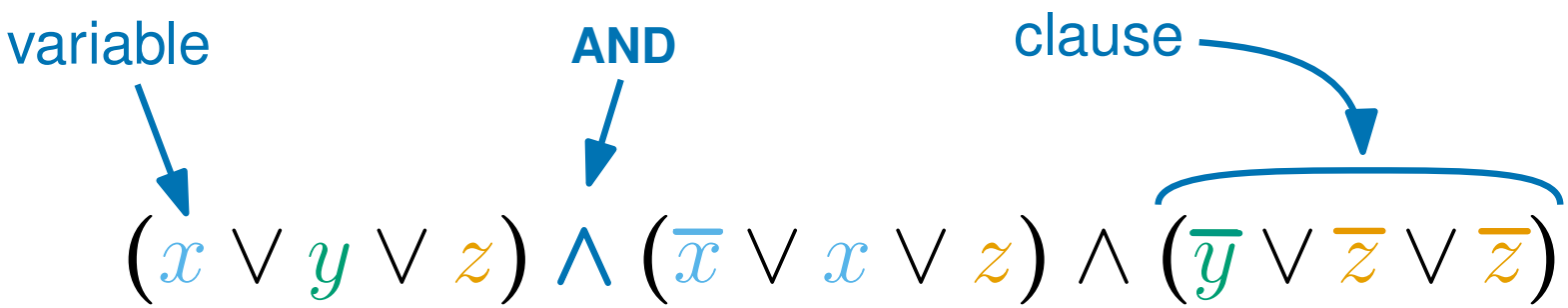
i.e. can we assign variables so that the formula is satisfied (**True**)*?*



What is 3-SAT?

The 3-SAT problem is to decide whether a given boolean formula can be satisfied.

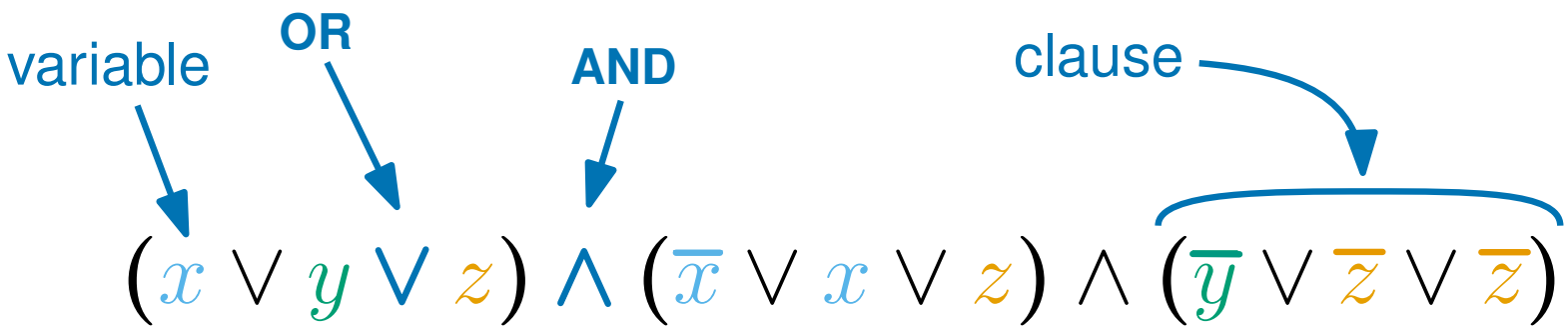
i.e. can we assign variables so that the formula is satisfied (**True**)*?*



What is 3-SAT?

The 3-SAT problem is to decide whether a given boolean formula can be satisfied.

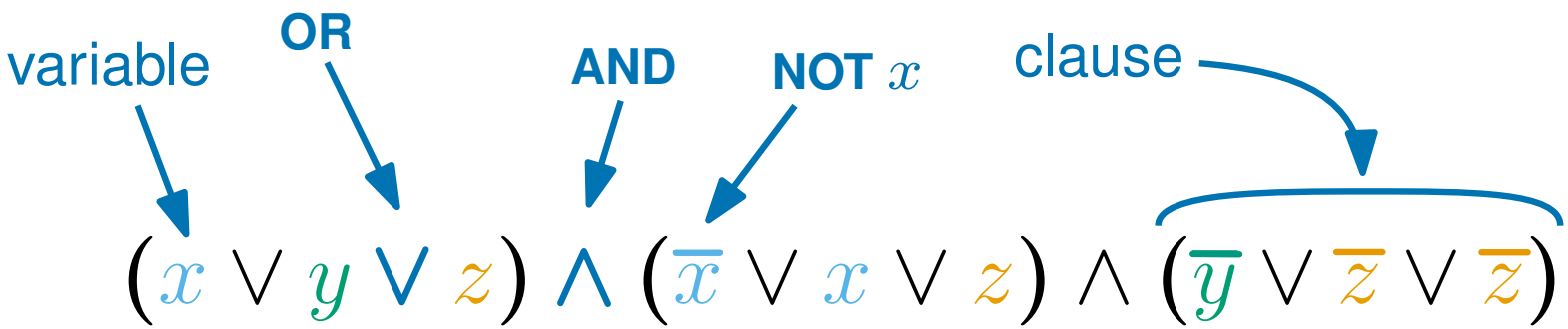
i.e. can we assign variables so that the formula is satisfied (**True**)*?*



What is 3-SAT?

The 3-SAT problem is to decide whether a given boolean formula can be satisfied.

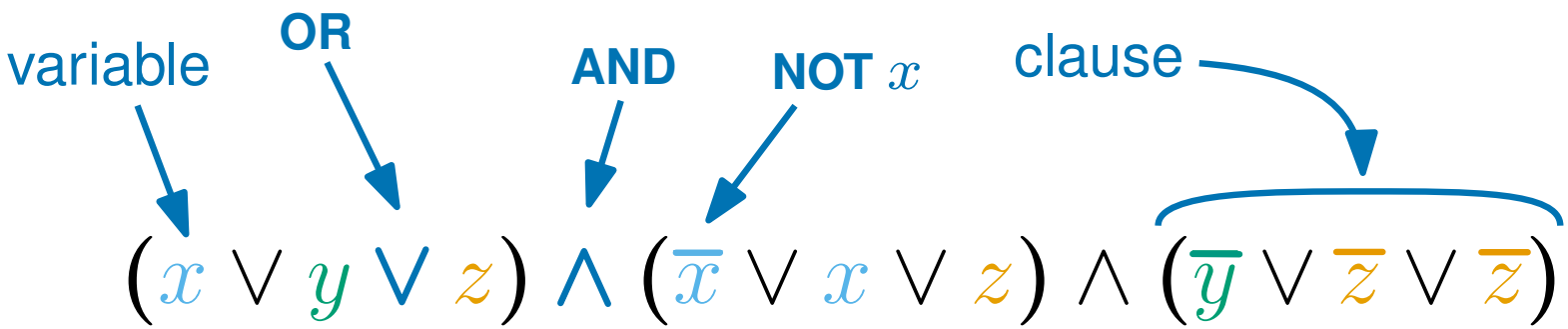
i.e. can we assign variables so that the formula is satisfied (**True**)*?*



What is 3-SAT?

The 3-SAT problem is to decide whether a given boolean formula can be satisfied.

i.e. can we assign variables so that the formula is satisfied (**True**)*?*

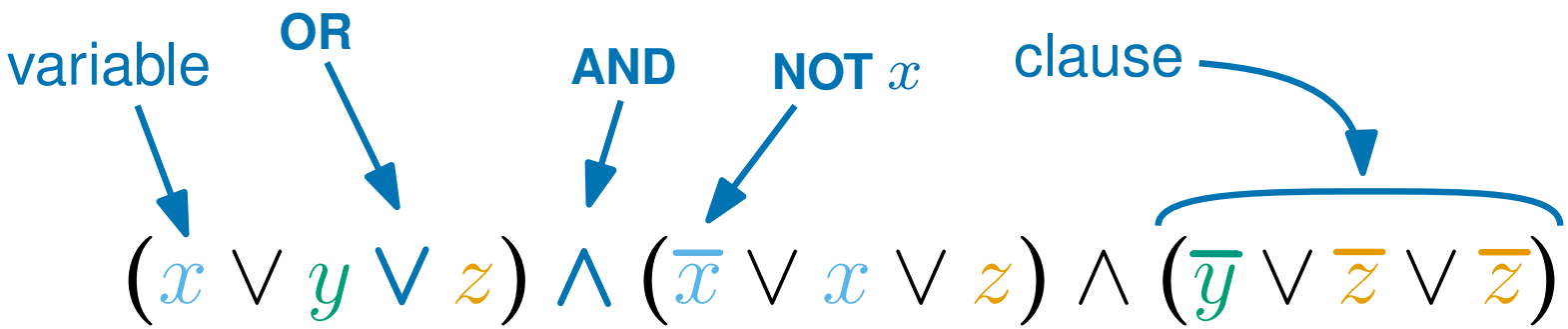


A **literal** is either a variable (e.g. y) or its negation (e.g. $\overline{y}$)

What is 3-SAT?

The 3-SAT problem is to decide whether a given boolean formula can be satisfied.

i.e. can we assign variables so that the formula is satisfied (**True**)*?*



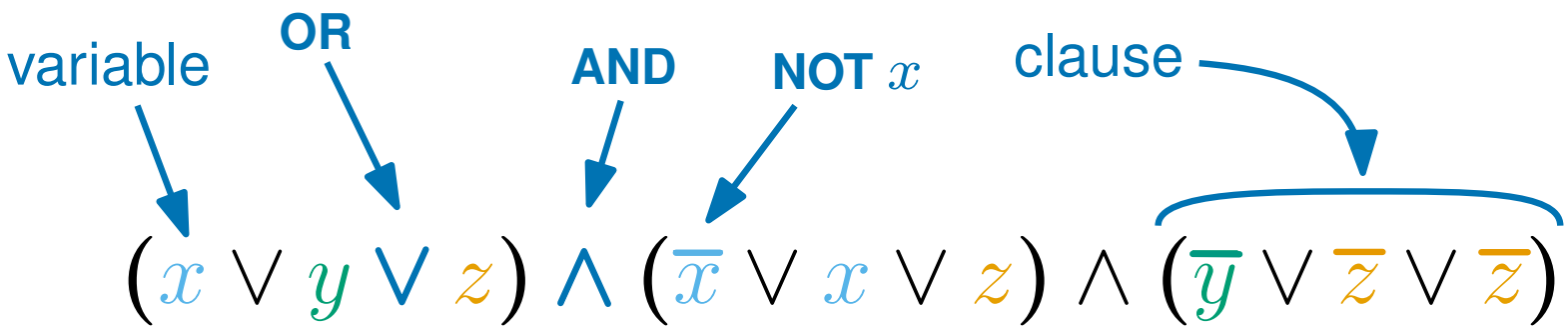
A **literal** is either a variable (e.g. y) or its negation (e.g. $\overline{y}$)

In 3-SAT each clause has exactly three literals

What is 3-SAT?

The 3-SAT problem is to decide whether a given boolean formula can be satisfied.

i.e. can we assign variables so that the formula is satisfied (**True**)*?*



A **literal** is either a variable (e.g. y) or its negation (e.g. $\overline{y}$)

In 3-SAT each clause has exactly three literals

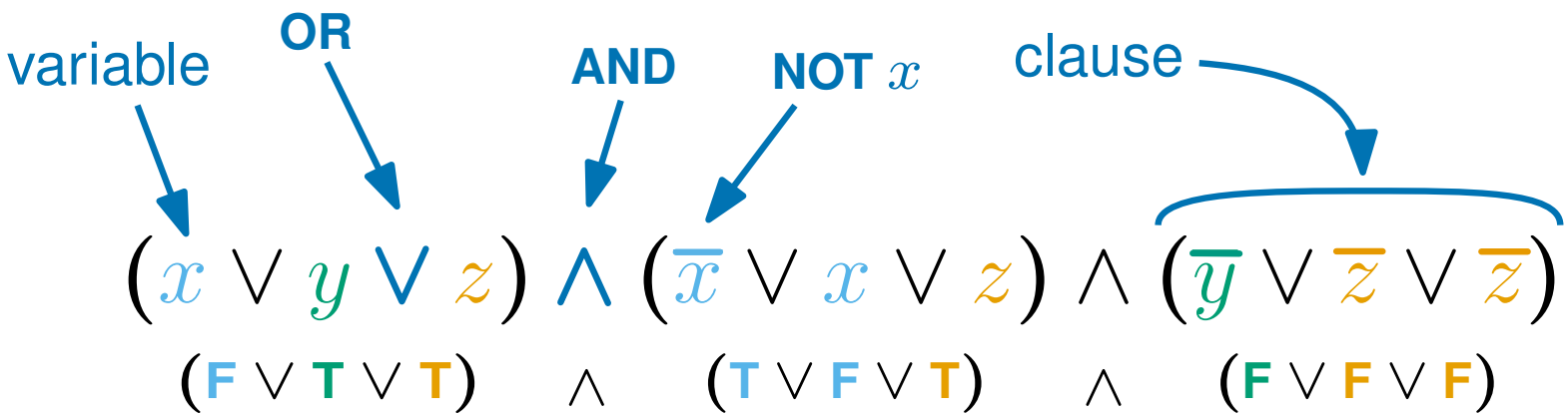
One possible **assignment** is

$x =$ **False**, $y =$ **True**, $z =$ **True**

What is 3-SAT?

The 3-SAT problem is to decide whether a given boolean formula can be satisfied.

i.e. can we assign variables so that the formula is satisfied (**True**)*?*



A **literal** is either a variable (e.g. y) or its negation (e.g. $\overline{y}$)

In 3-SAT each clause has exactly three literals

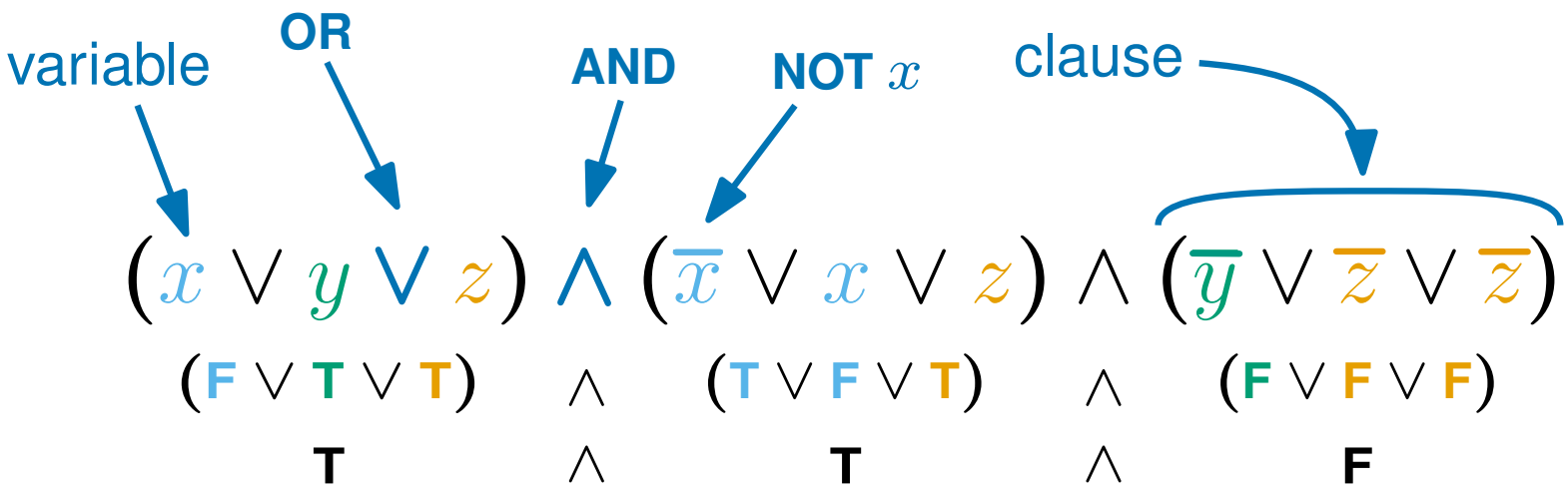
One possible **assignment** is

$x =$ **False**, $y =$ **True**, $z =$ **True**

What is 3-SAT?

The 3-SAT problem is to decide whether a given boolean formula can be satisfied.

i.e. can we assign variables so that the formula is satisfied (**True**)*?*



A **literal** is either a variable (e.g. y) or its negation (e.g. $\overline{y}$)

In 3-SAT each clause has exactly three literals

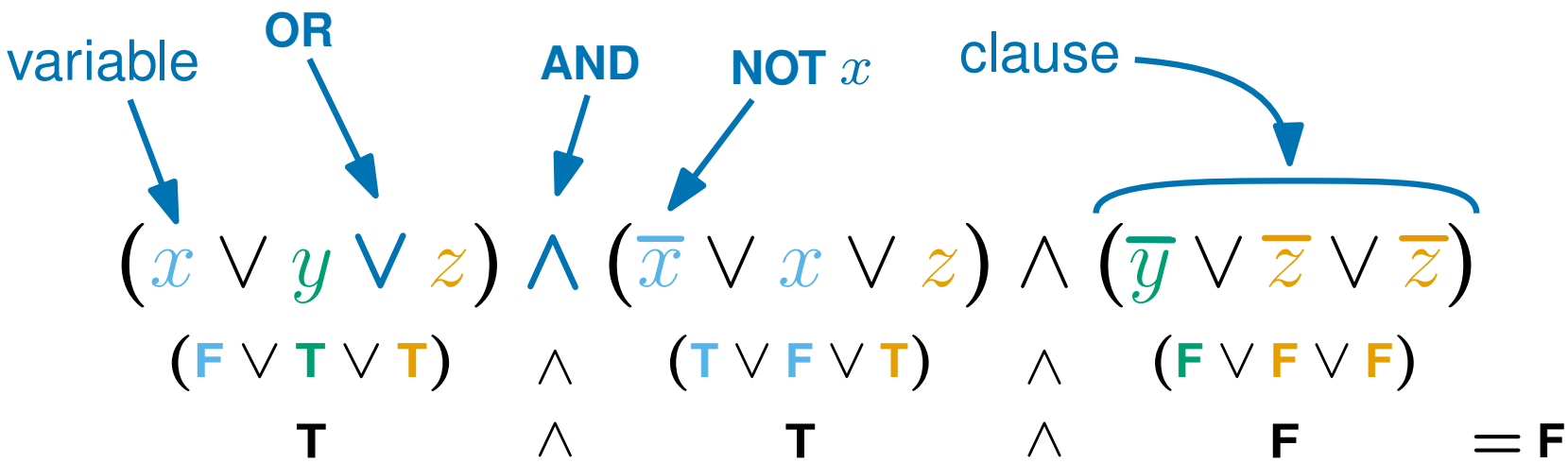
One possible **assignment** is

$x =$ **False**, $y =$ **True**, $z =$ **True**

What is 3-SAT?

The 3-SAT problem is to decide whether a given boolean formula can be satisfied.

i.e. can we assign variables so that the formula is satisfied (**True**)*?*



A **literal** is either a variable (e.g. y) or its negation (e.g. $\overline{y}$)

In 3-SAT each clause has exactly three literals

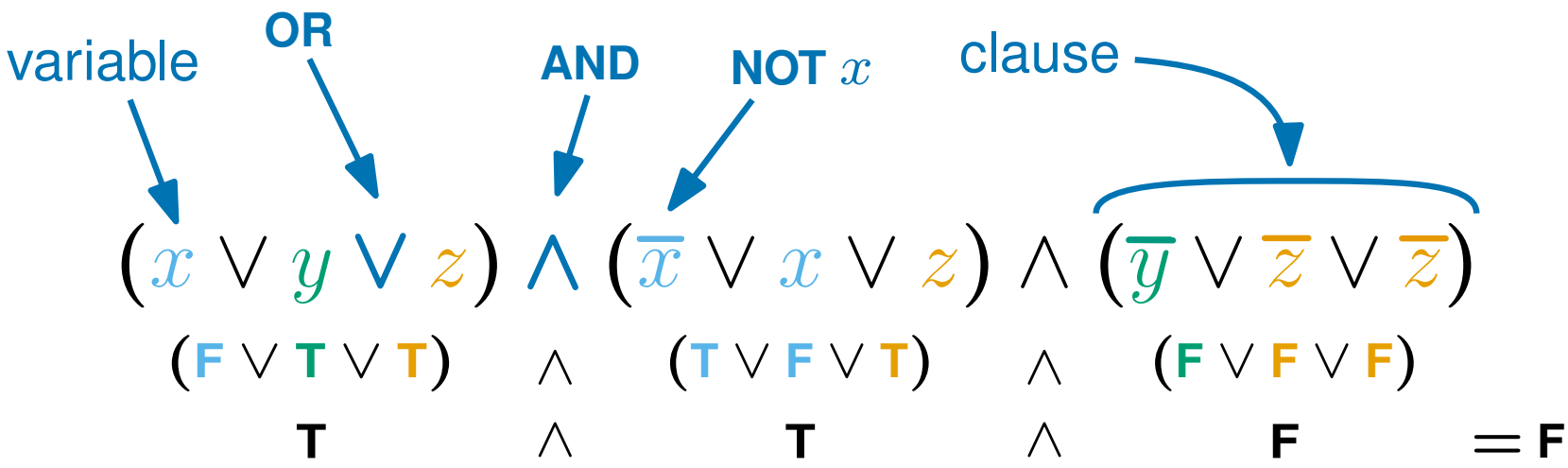
One possible **assignment** is

$x =$ **False**, $y =$ **True**, $z =$ **True**

What is 3-SAT?

The 3-SAT problem is to decide whether a given boolean formula can be satisfied.

i.e. can we assign variables so that the formula is satisfied (**True**)*?*



A **literal** is either a variable (e.g. y) or its negation (e.g. $\overline{y}$)

In 3-SAT each clause has exactly three literals

One possible **assignment** is

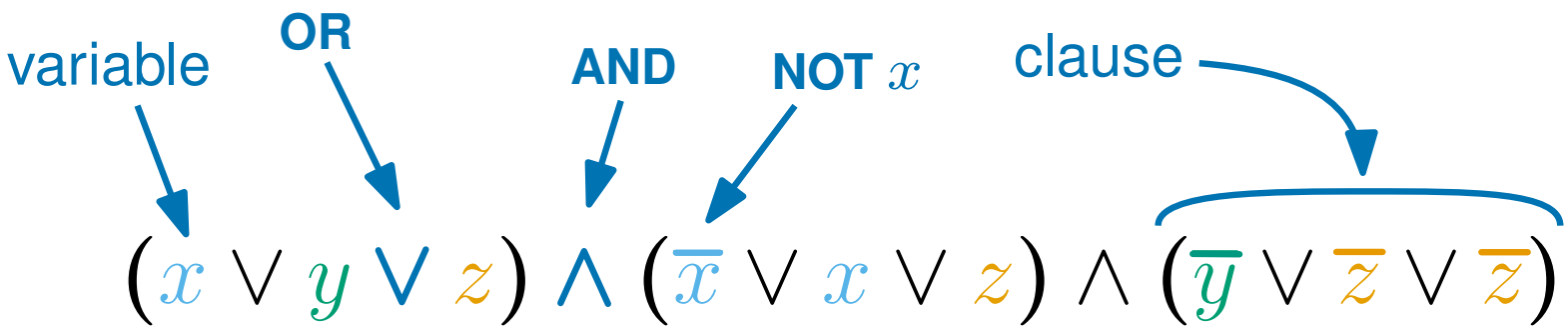
$x =$ **False**, $y =$ **True**, $z =$ **True**

(which doesn't satisfy the formula)

What is 3-SAT?

The 3-SAT problem is to decide whether a given boolean formula can be satisfied.

i.e. can we assign variables so that the formula is satisfied (**True**)*?*



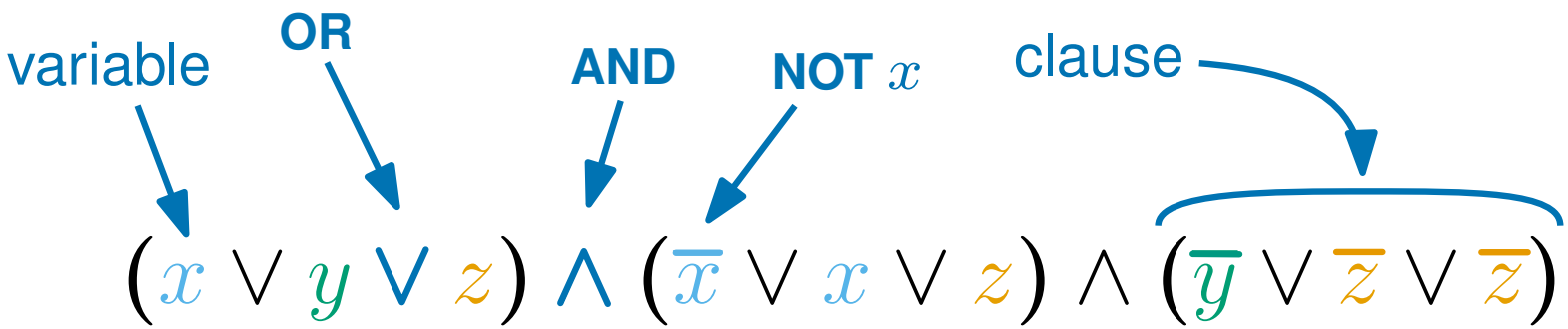
A **literal** is either a variable (e.g. y) or its negation (e.g. $\overline{y}$)

In 3-SAT each clause has exactly three literals

What is 3-SAT?

The 3-SAT problem is to decide whether a given boolean formula can be satisfied.

i.e. can we assign variables so that the formula is satisfied (**True**)*?*



A **literal** is either a variable (e.g. y) or its negation (e.g. $\overline{y}$)

In 3-SAT each clause has exactly three literals

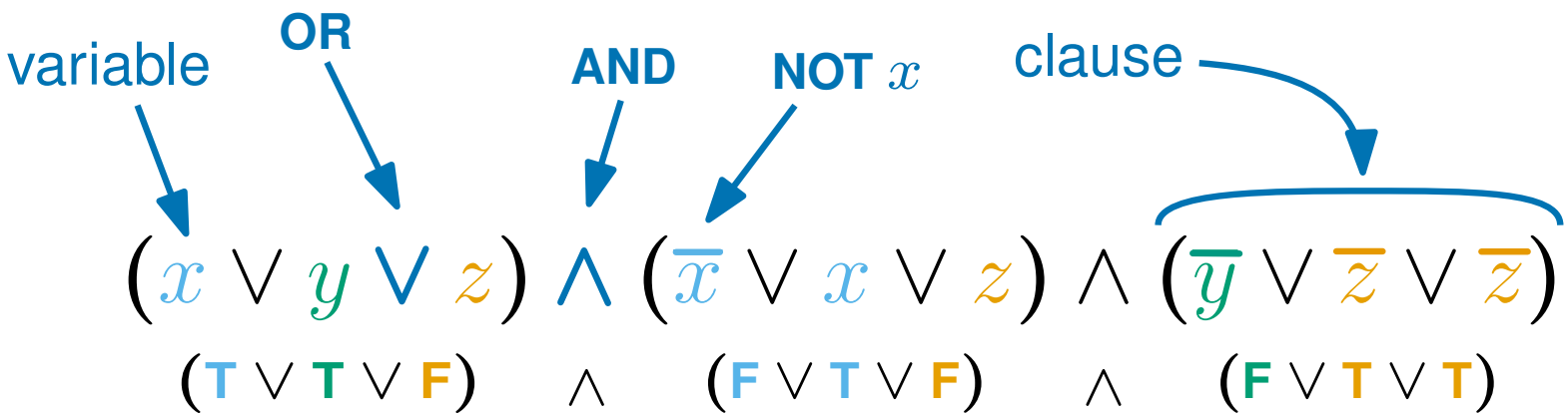
Another possible **assignment** is

$x =$ **True**, $y =$ **True**, $z =$ **False**

What is 3-SAT?

The 3-SAT problem is to decide whether a given boolean formula can be satisfied.

i.e. can we assign variables so that the formula is satisfied (**True**)*?*



A **literal** is either a variable (e.g. y) or its negation (e.g. $\overline{y}$)

In 3-SAT each clause has exactly three literals

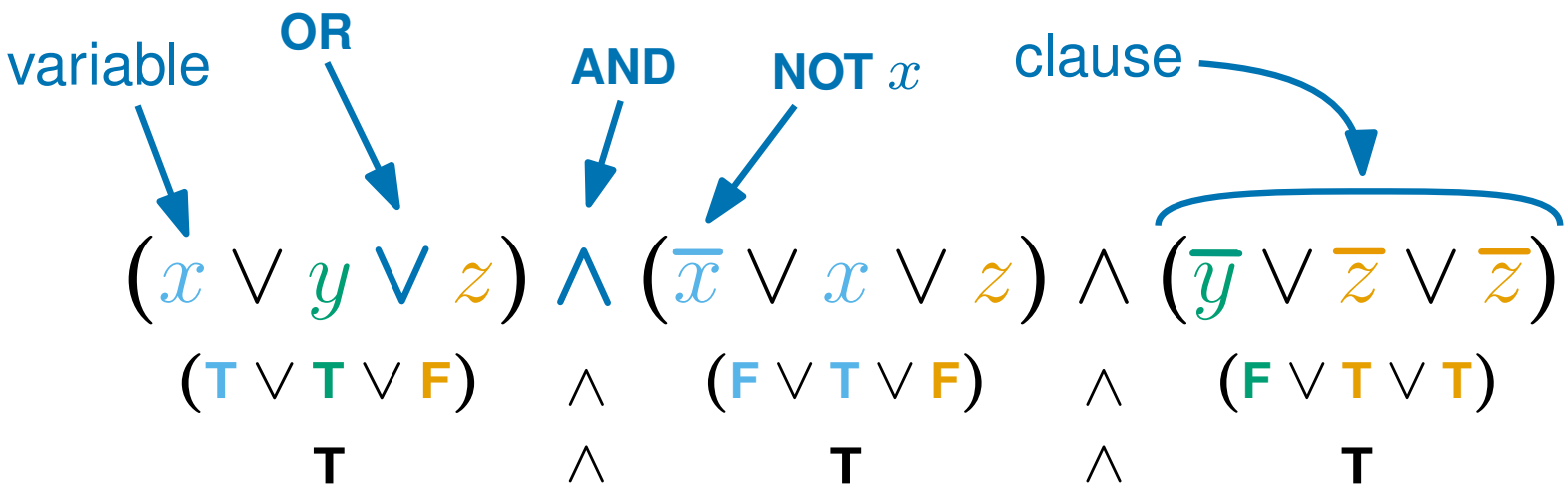
Another possible **assignment** is

$x =$ **True**, $y =$ **True**, $z =$ **False**

What is 3-SAT?

The 3-SAT problem is to decide whether a given boolean formula can be satisfied.

i.e. can we assign variables so that the formula is satisfied (**True**)*?*



A **literal** is either a variable (e.g. y) or its negation (e.g. $\overline{y}$)

In 3-SAT each clause has exactly three literals

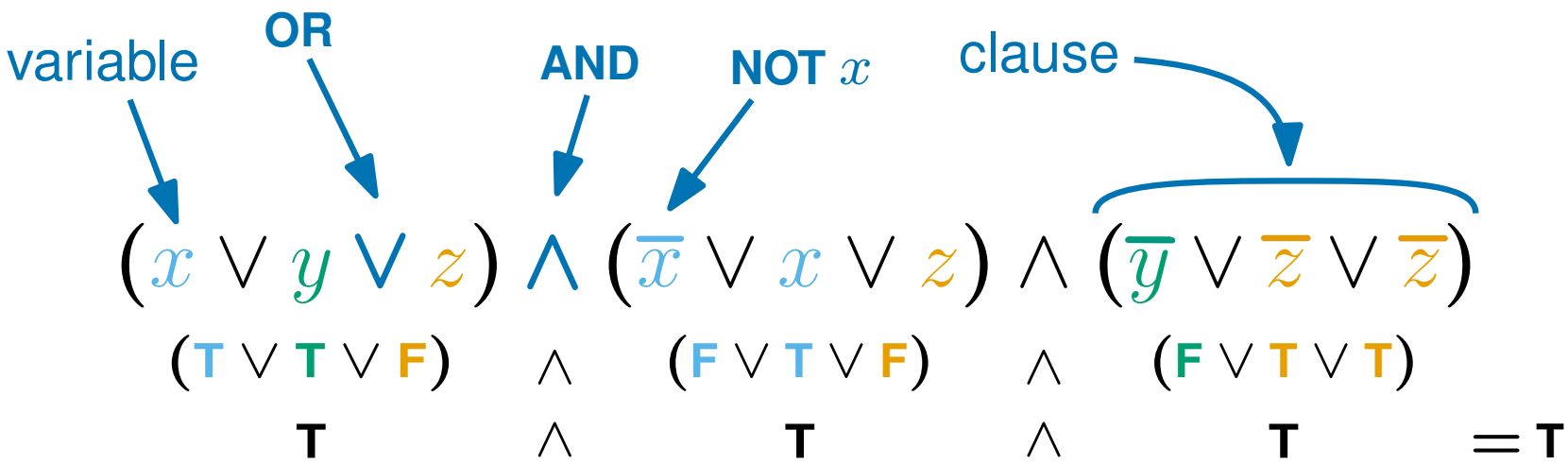
Another possible **assignment** is

$x =$ **True**, $y =$ **True**, $z =$ **False**

What is 3-SAT?

The 3-SAT problem is to decide whether a given boolean formula can be satisfied.

i.e. can we assign variables so that the formula is satisfied (**True**)*?*



A **literal** is either a variable (e.g. y) or its negation (e.g. $\overline{y}$)

In 3-SAT each clause has exactly three literals

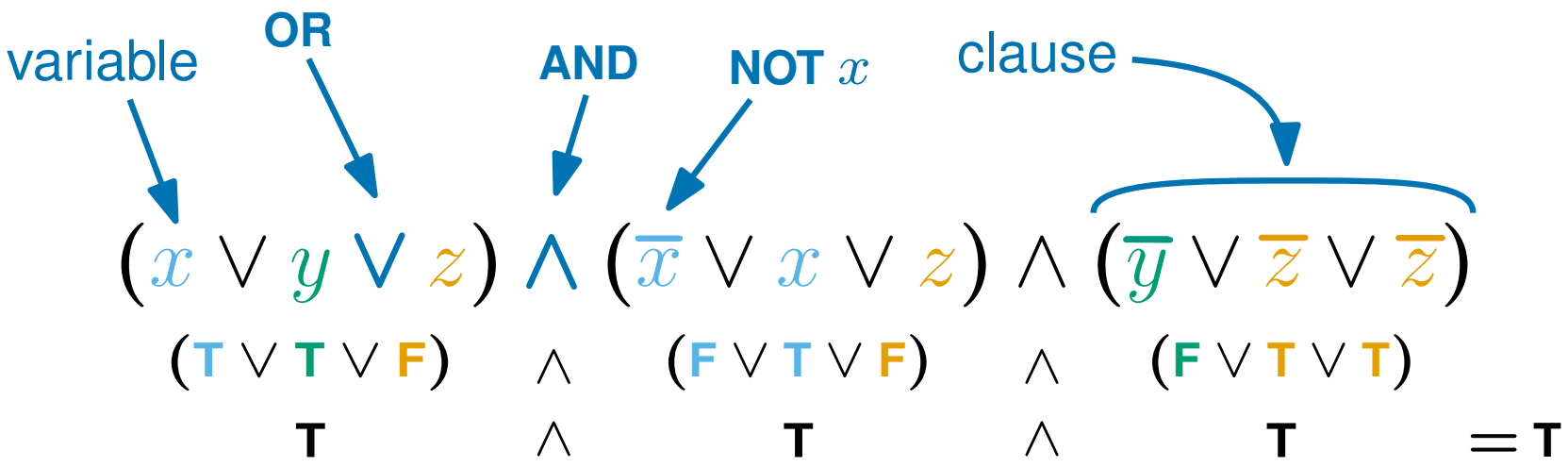
Another possible **assignment** is

$x =$ **True**, $y =$ **True**, $z =$ **False**

What is 3-SAT?

The 3-SAT problem is to decide whether a given boolean formula can be satisfied.

i.e. can we assign variables so that the formula is satisfied (**True**)*?*



A **literal** is either a variable (e.g. y) or its negation (e.g. $\overline{y}$)

In 3-SAT each clause has exactly three literals

Another possible **assignment** is

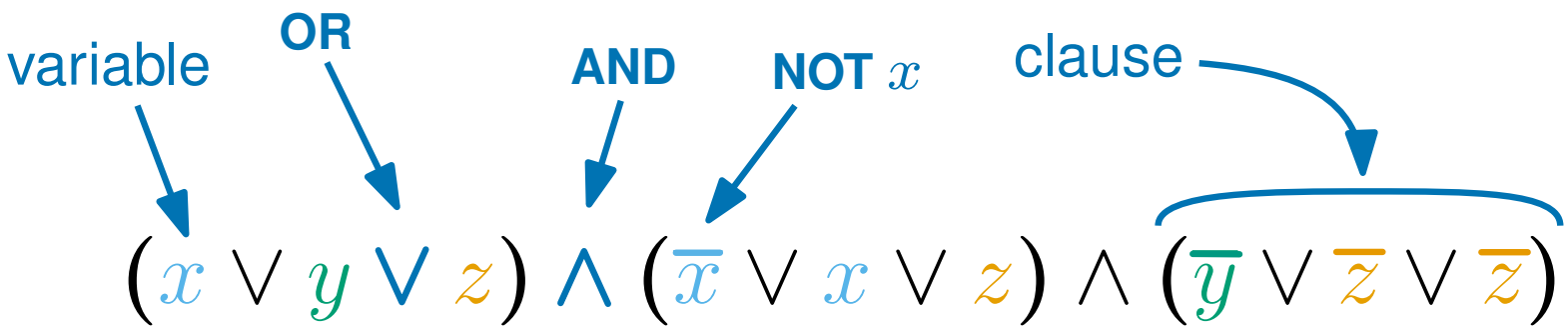
$x =$ **True**, $y =$ **True**, $z =$ **False**

(which satisfies the formula)

What is 3-SAT?

The 3-SAT problem is to decide whether a given boolean formula can be satisfied.

i.e. can we assign variables so that the formula is satisfied (**True**)*?*



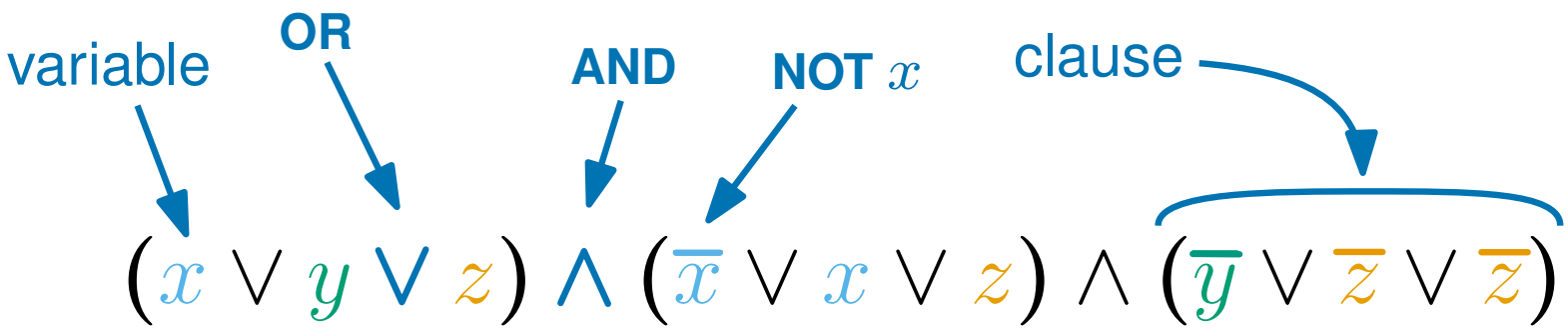
A **literal** is either a variable (e.g. y) or its negation (e.g. $\overline{y}$)

In 3-SAT each clause has exactly three literals

What is 3-SAT?

The 3-SAT problem is to decide whether a given boolean formula can be satisfied.

i.e. can we assign variables so that the formula is satisfied (**True**)*?*



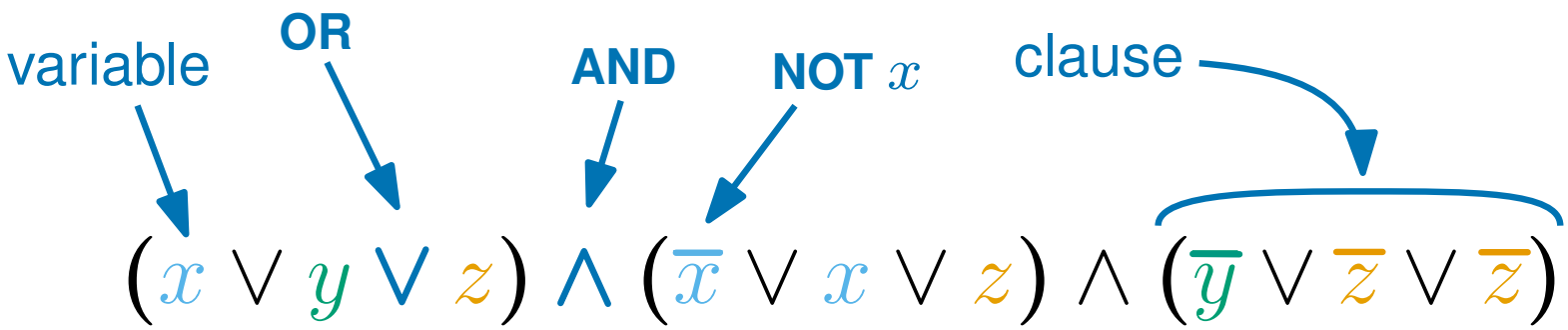
A **literal** is either a variable (e.g. y) or its negation (e.g. $\overline{y}$)

In 3-SAT each clause has exactly three literals

What is 3-SAT?

The 3-SAT problem is to decide whether a given boolean formula can be satisfied.

i.e. can we assign variables so that the formula is satisfied (**True**)*?*



A **literal** is either a variable (e.g. y) or its negation (e.g. $\overline{y}$)

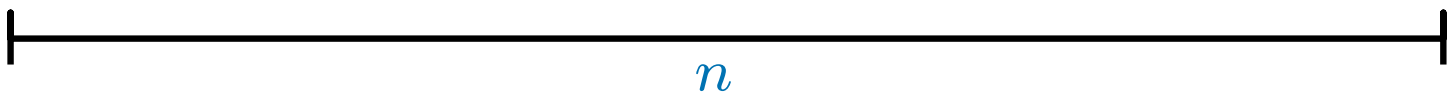
In 3-SAT each clause has exactly three literals

The 3-SAT problem is NP-Complete *(it's NP-Hard and in NP)*

The approach

3-SAT: Decide whether a given boolean formula can be satisfied

$$(x \vee y \vee z) \wedge (\overline{x} \vee x \vee z) \wedge (\overline{y} \vee \overline{z} \vee \overline{z})$$



i.e. can we assign variables so that the formula is satisfied (**True**)*?*

Remember that each clause has exactly three literals

Given a 3-SAT formula we will construct a Crystal Caves level...

which can be completed *if and only if* the formula can be satisfied

(and the size of the level will be polynomial in n)

The approach

3-SAT: Decide whether a given boolean formula can be satisfied

$$(x \vee y \vee z) \wedge (\overline{x} \vee x \vee z) \wedge (\overline{y} \vee \overline{z} \vee \overline{z})$$

n

i.e. can we assign variables so that the formula is satisfied (**True**)*?*

Remember that each clause has exactly three literals

Given a 3-SAT formula we will construct a Crystal Caves level. . .

which can be completed *if and only if* the formula can be satisfied

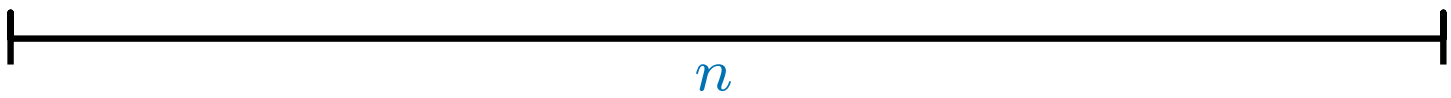
(and the size of the level will be polynomial in n)

so if we had a polynomial time Crystal Caves 'solver', we could use it to solve 3-SAT in polynomial time

The approach

3-SAT: Decide whether a given boolean formula can be satisfied

$$(x \vee y \vee z) \wedge (\overline{x} \vee x \vee z) \wedge (\overline{y} \vee \overline{z} \vee \overline{z})$$



i.e. can we assign variables so that the formula is satisfied (**True**)*?*

Remember that each clause has exactly three literals

Given a 3-SAT formula we will construct a Crystal Caves level. . .

which can be completed *if and only if* the formula can be satisfied

(and the size of the level will be polynomial in n)

a program which decides whether a given level can be completed

so if we had a polynomial time Crystal Caves 'solver', we could use it to solve 3-SAT in polynomial time

The approach

3-SAT: Decide whether a given boolean formula can be satisfied

$$(x \vee y \vee z) \wedge (\overline{x} \vee x \vee z) \wedge (\overline{y} \vee \overline{z} \vee \overline{z})$$

n

i.e. can we assign variables so that the formula is satisfied (**True**)*?*

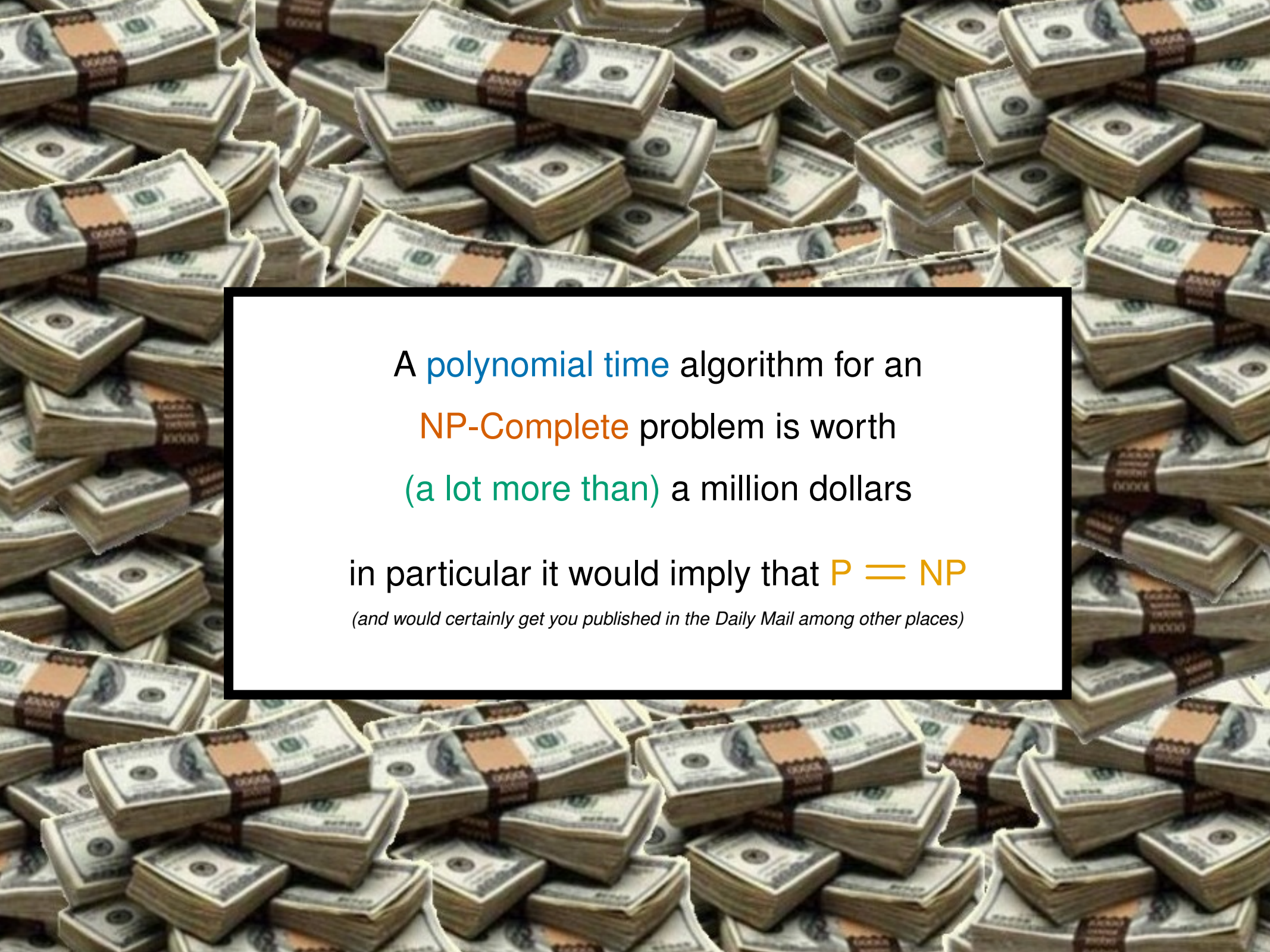
Remember that each clause has exactly three literals

Given a 3-SAT formula we will construct a Crystal Caves level. . .

which can be completed *if and only if* the formula can be satisfied

(and the size of the level will be polynomial in n)

so if we had a polynomial time Crystal Caves 'solver', we could use it to solve 3-SAT in polynomial time


A polynomial time algorithm for an
NP-Complete problem is worth
(a lot more than) a million dollars
in particular it would imply that P = NP
(and would certainly get you published in the Daily Mail among other places)

The approach

3-SAT: Decide whether a given boolean formula can be satisfied

$$(x \vee y \vee z) \wedge (\overline{x} \vee x \vee z) \wedge (\overline{y} \vee \overline{z} \vee \overline{z})$$

n

i.e. can we assign variables so that the formula is satisfied (**True**)*?*

Remember that each clause has exactly three literals

Given a 3-SAT formula we will see how to construct a Crystal Caves level. . .

which can be completed *if and only if* the formula can be satisfied

(and the size of the level will be polynomial in n)

so if we had a polynomial time Crystal Caves 'solver', we could use it to solve 3-SAT in polynomial time

The approach

3-SAT: Decide whether a given boolean formula can be satisfied

$$(x \vee y \vee z) \wedge (\overline{x} \vee x \vee z) \wedge (\overline{y} \vee \overline{z} \vee \overline{z})$$

n

i.e. can we assign variables so that the formula is satisfied (**True**)*?*

Remember that each clause has exactly three literals

Given a 3-SAT formula we will see how to construct a Crystal Caves level. . .

which can be completed *if and only if* the formula can be satisfied

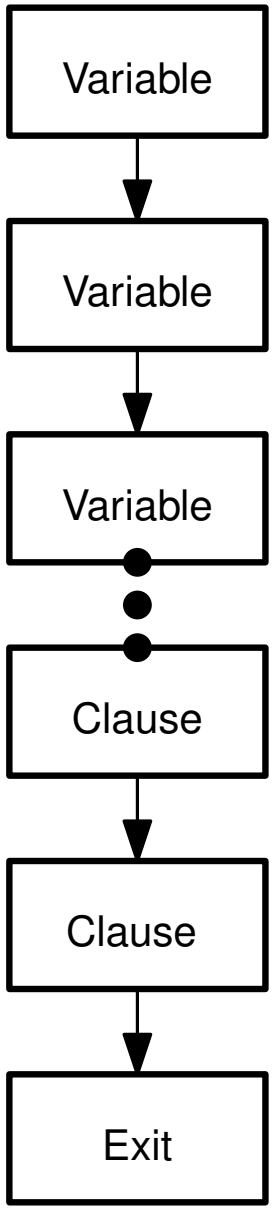
(and the size of the level will be polynomial in n)

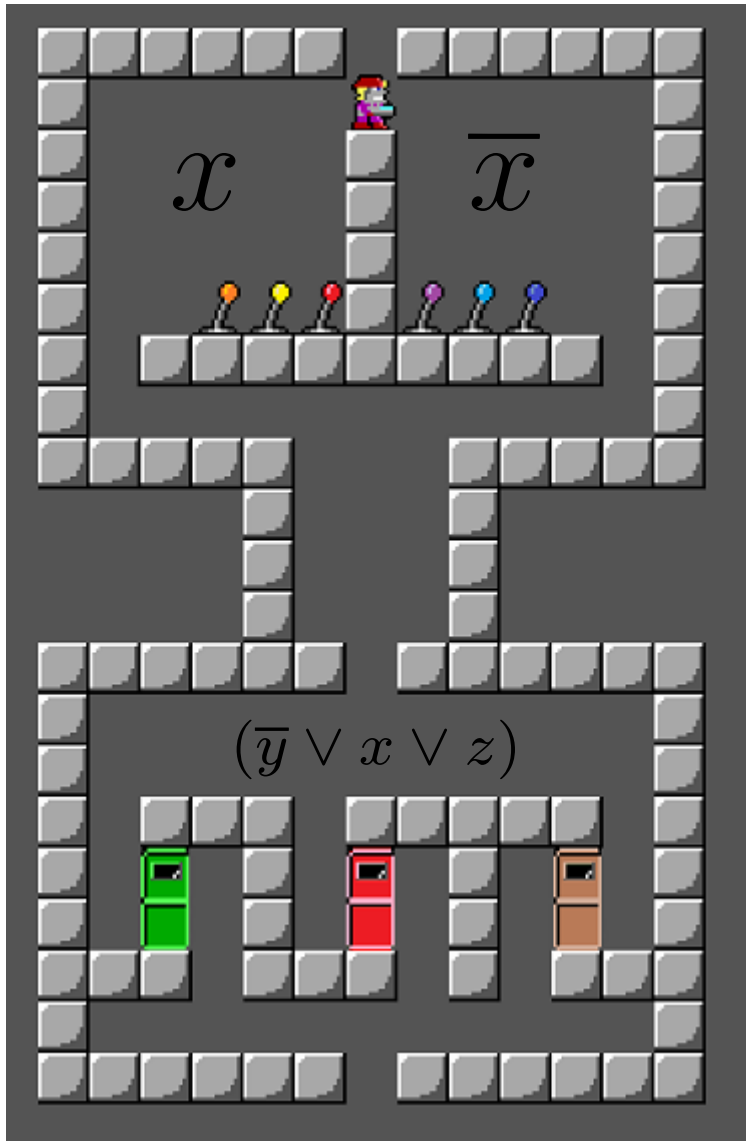
so if we had a polynomial time Crystal Caves 'solver', we could use it to solve 3-SAT in polynomial time

this means that Crystal Caves is *at least as hard as* 3-SAT

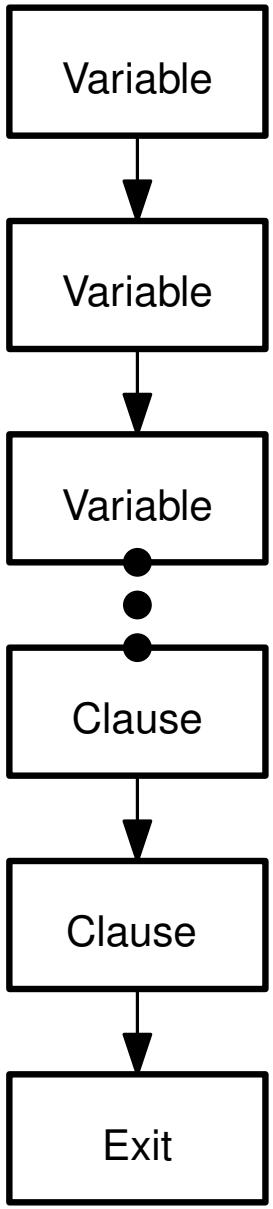
in other words, Crystal Caves is NP-hard

Embedding 3-SAT in Crystal Caves

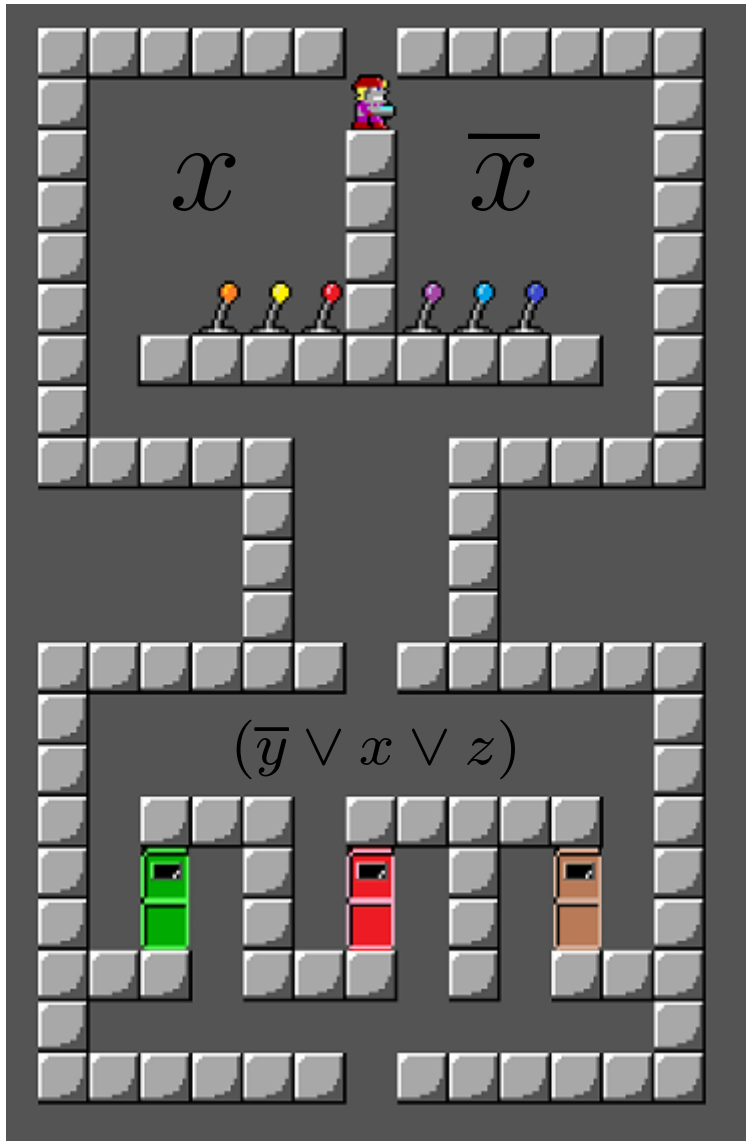




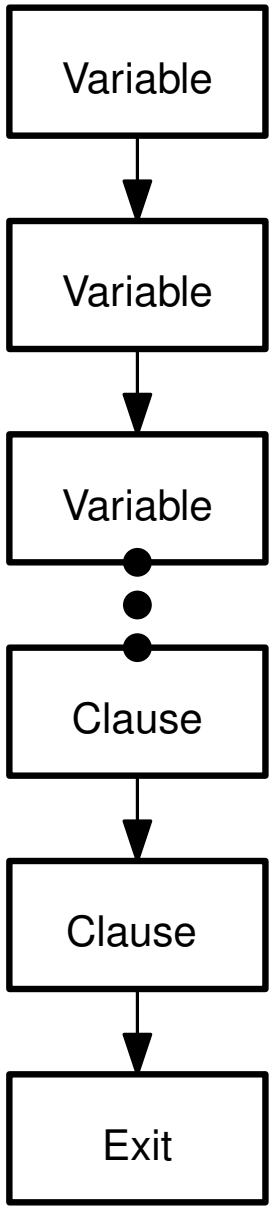
Embedding 3-SAT in Crystal Caves



The miner must either go left (x) or right ($\overline{x}$) at each variable gadget



Embedding 3-SAT in Crystal Caves

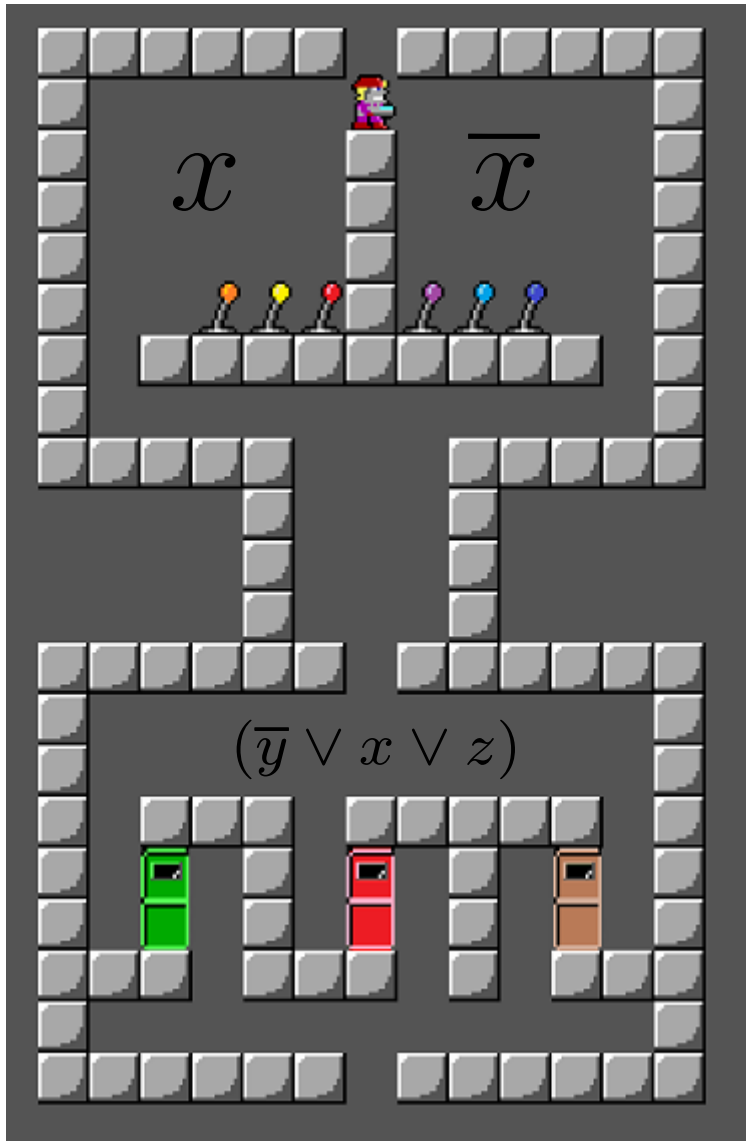


The miner must either go

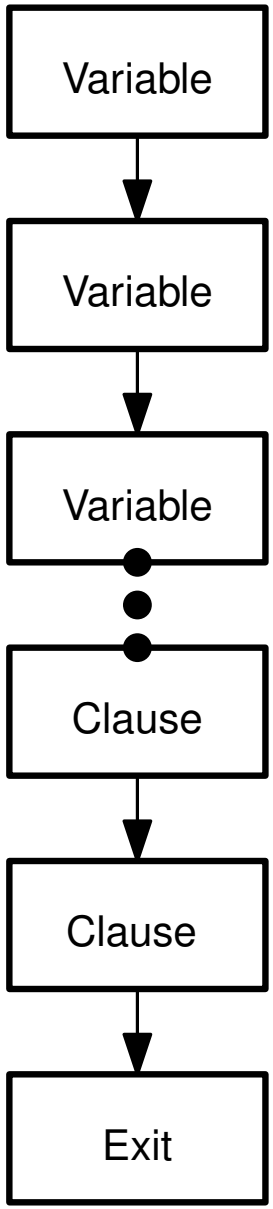
left (x) or right ($\overline{x}$)

at each variable gadget

and can't jump back up



Embedding 3-SAT in Crystal Caves



The miner must either go

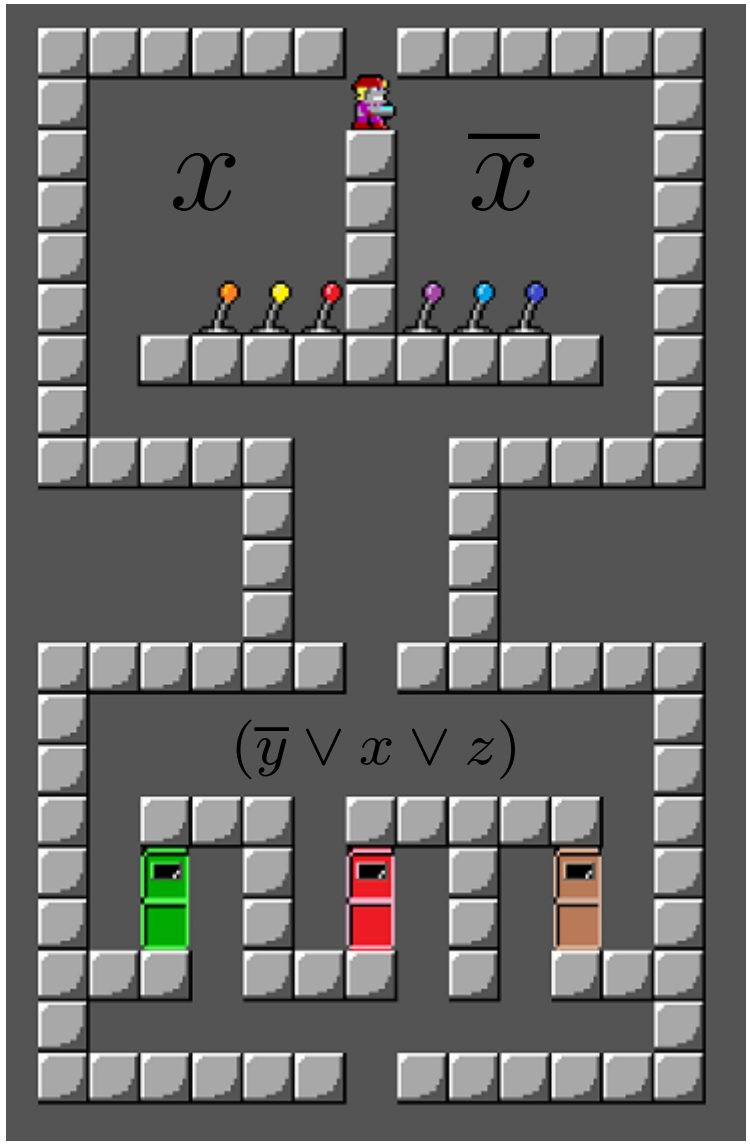
left (x) or right ($\overline{x}$)

at each variable gadget

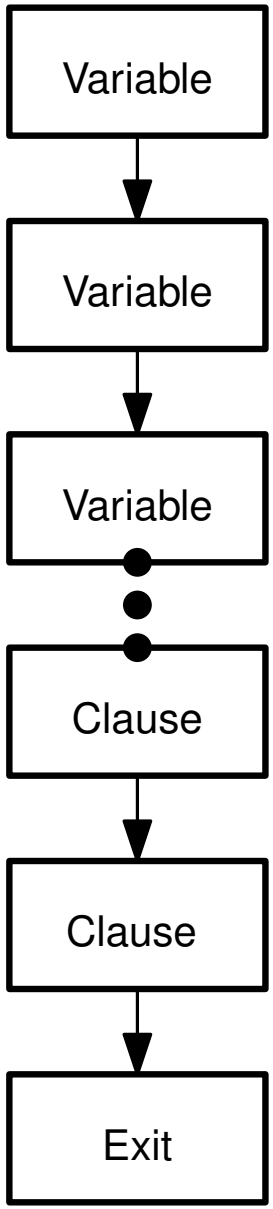
and can't jump back up

Variable gadgets have switches which

open doors in clause gadgets



Embedding 3-SAT in Crystal Caves

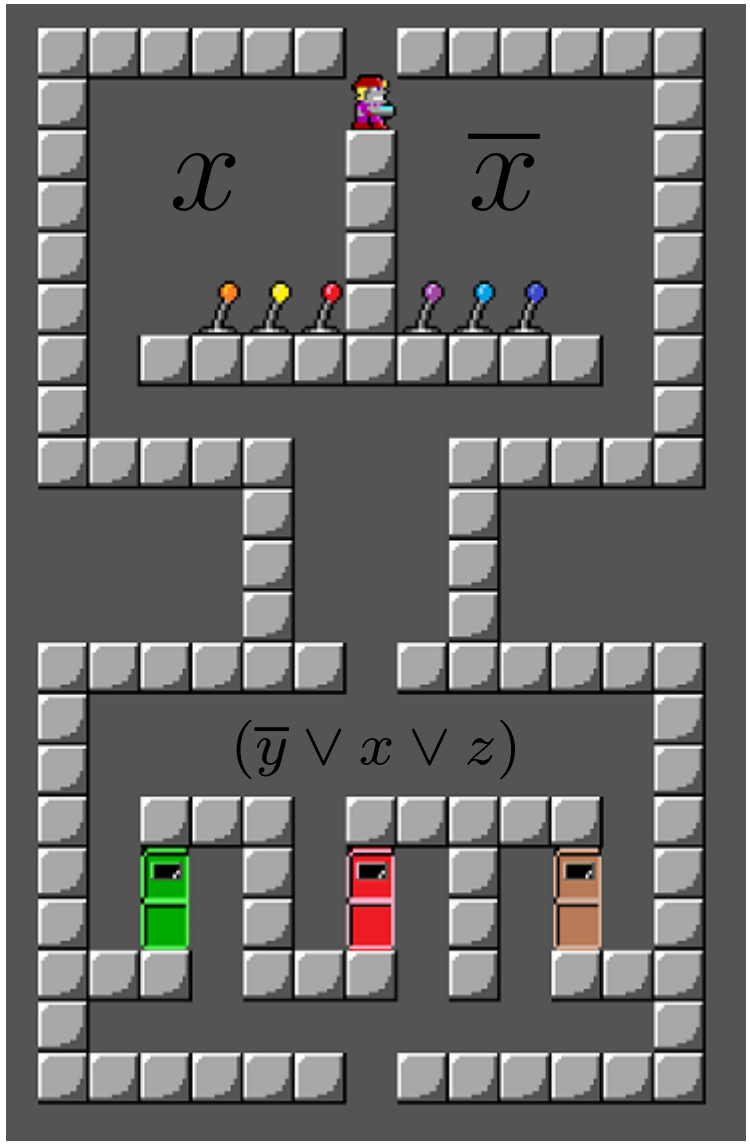


The miner must either go left (x) or right ($\overline{x}$) at each variable gadget

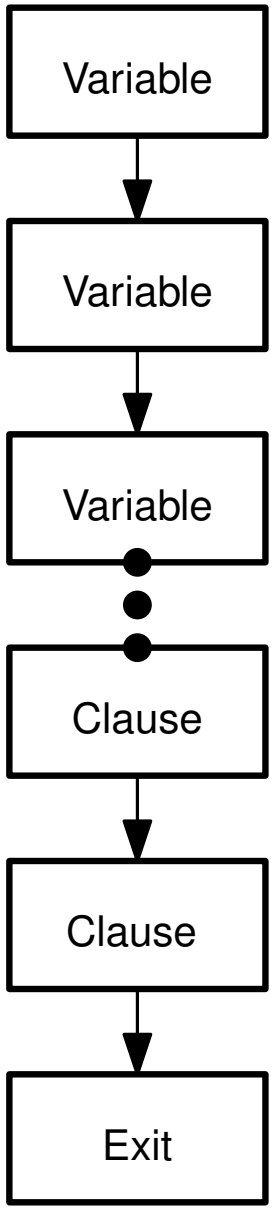
and can't jump back up

Variable gadgets have switches which open doors in clause gadgets

Each clause gadget has three doors:



Embedding 3-SAT in Crystal Caves



The miner must either go

left (x) or right ($\overline{x}$)

at each variable gadget

and can't jump back up

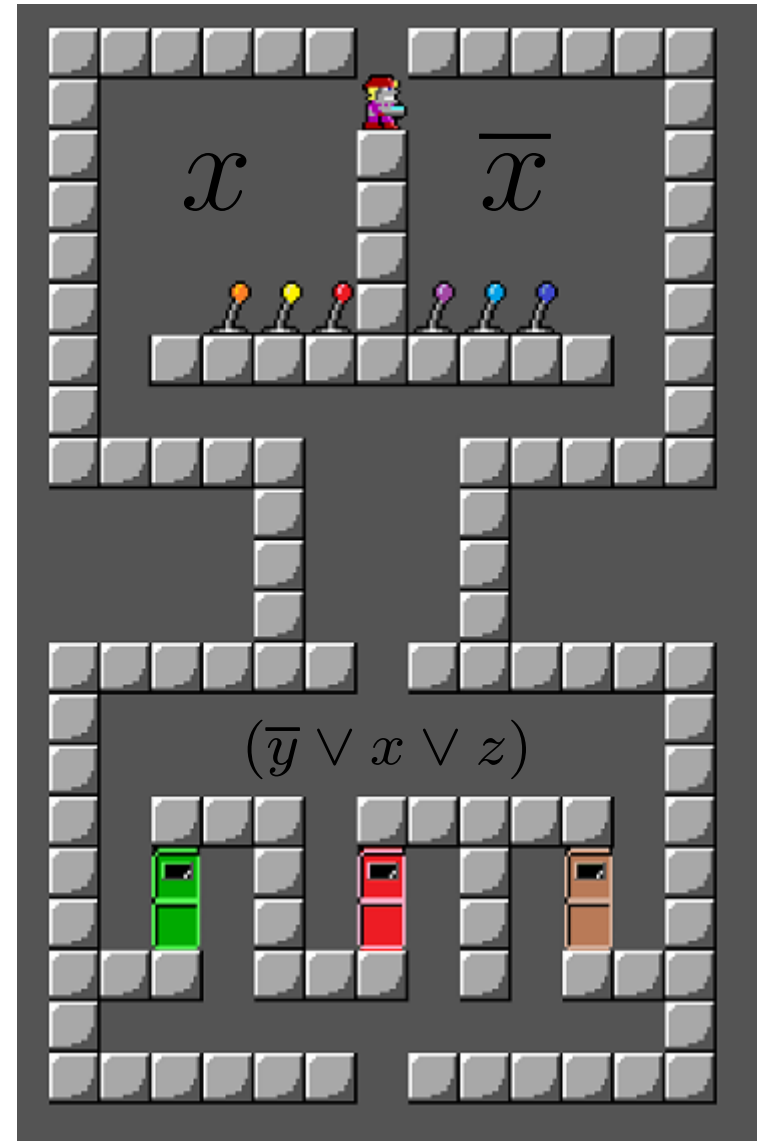
Variable gadgets have switches which

open doors in clause gadgets

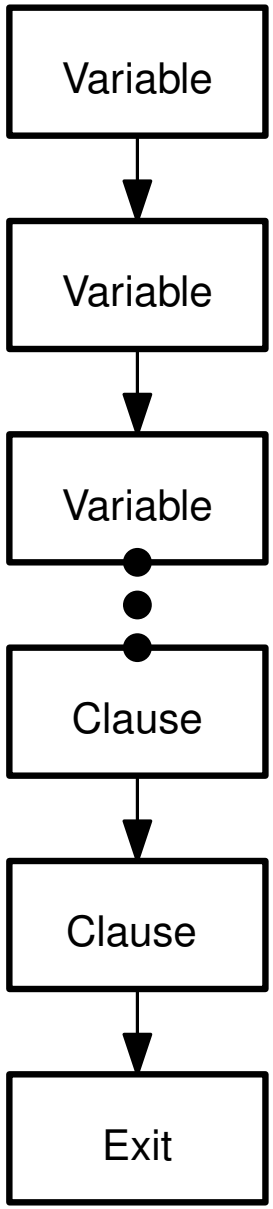
Each clause gadget has three doors:

Every 'x door' has a switch
on the 'x side' of the x variable gadget

Every '$\overline{x}$ door' has a switch
on the '$\overline{x}$ side' of the x variable gadget



Embedding 3-SAT in Crystal Caves



The miner must either go
left (x) or right ($\overline{x}$)
at each variable gadget

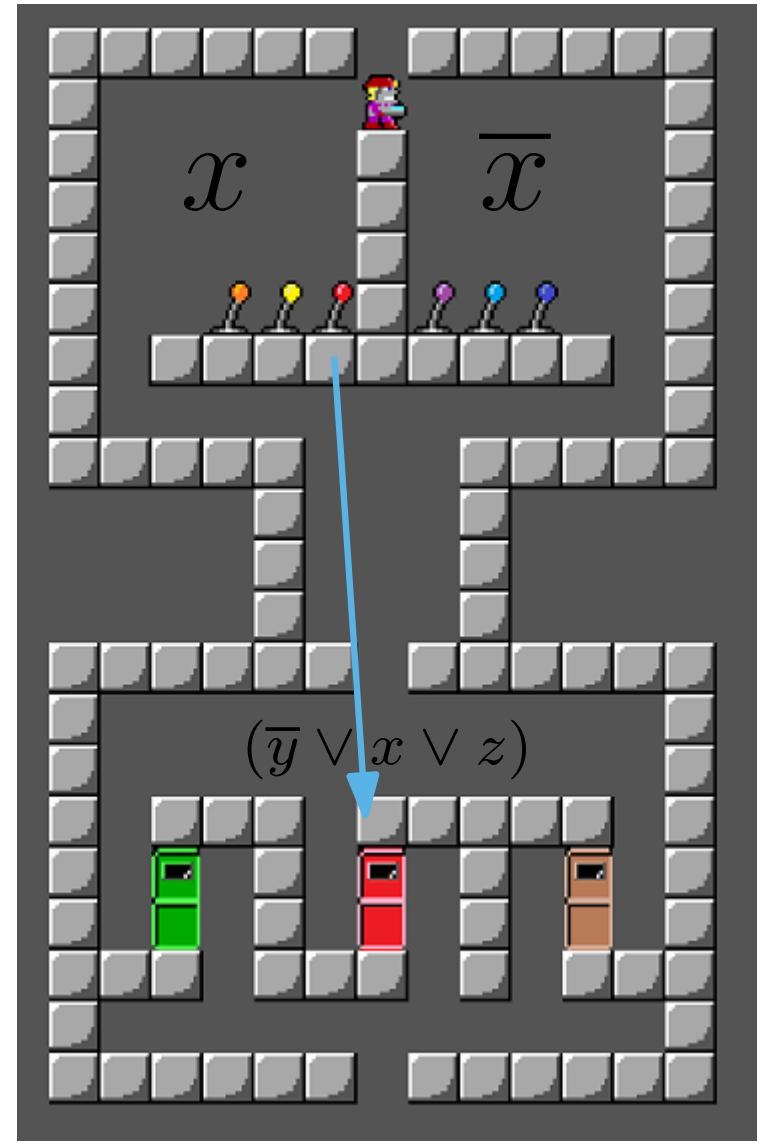
and can't jump back up

Variable gadgets have switches which
open doors in clause gadgets

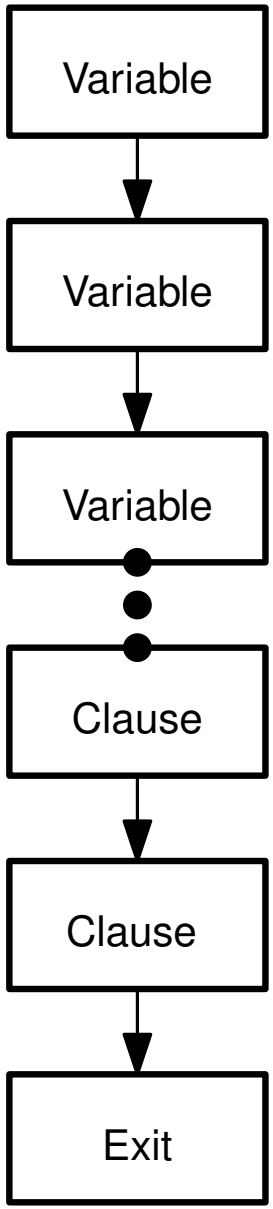
Each clause gadget has three doors:

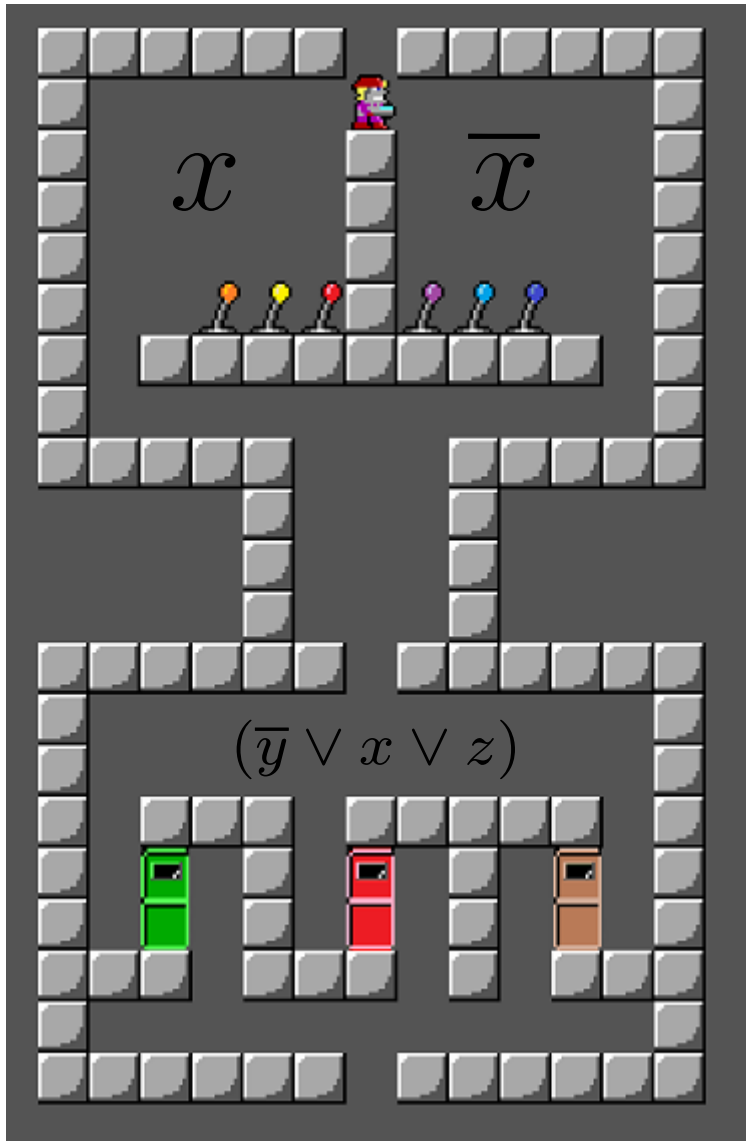
Every 'x door' has a switch
on the 'x side' of the x variable gadget

Every '$\overline{x}$ door' has a switch
on the '$\overline{x}$ side' of the x variable gadget

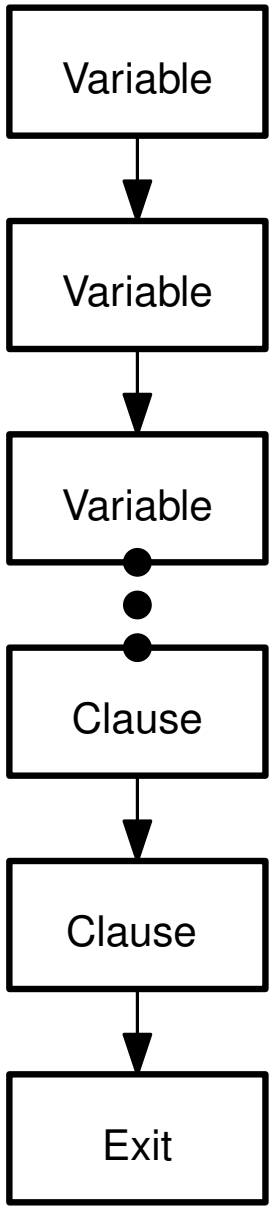


Embedding 3-SAT in Crystal Caves



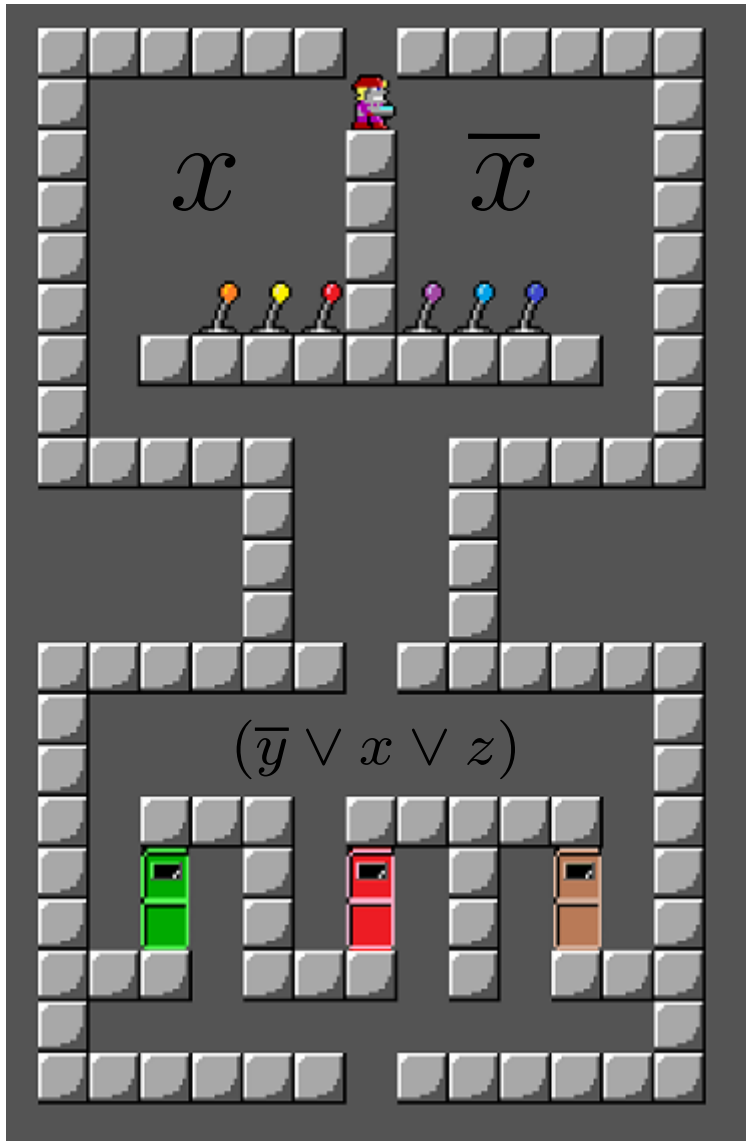


Embedding 3-SAT in Crystal Caves

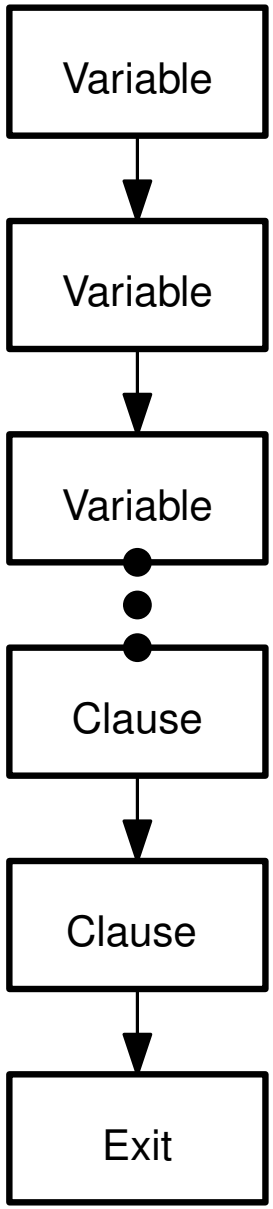


The miner can exit the level

iff the formula can be satisfied



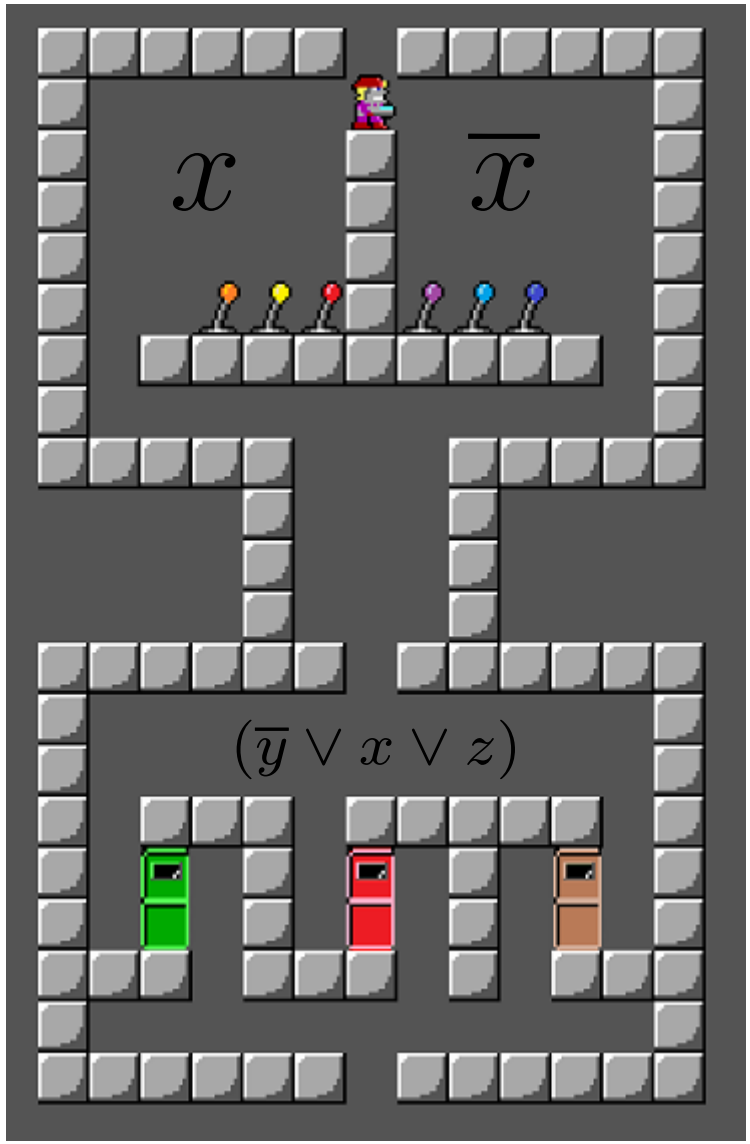
Embedding 3-SAT in Crystal Caves



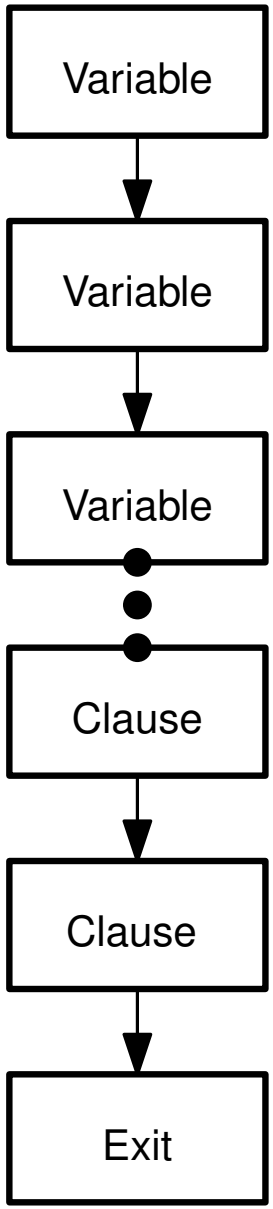
The miner can exit the level

iff the formula can be satisfied

However



Embedding 3-SAT in Crystal Caves

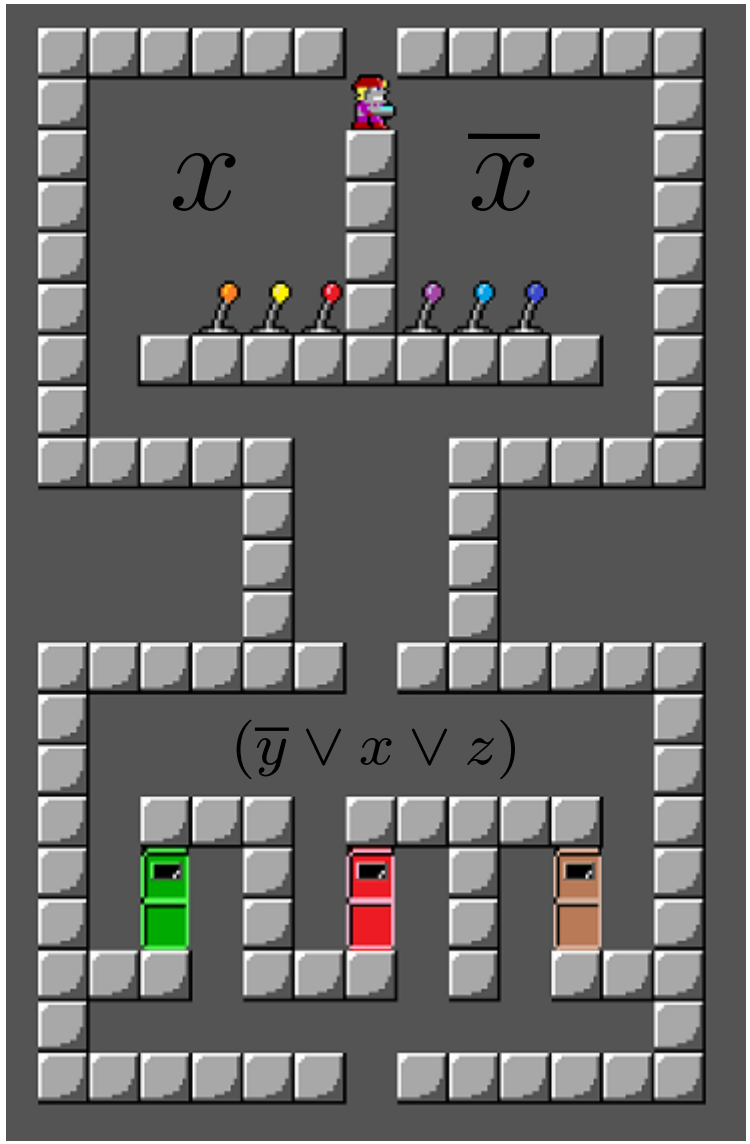


The miner can exit the level

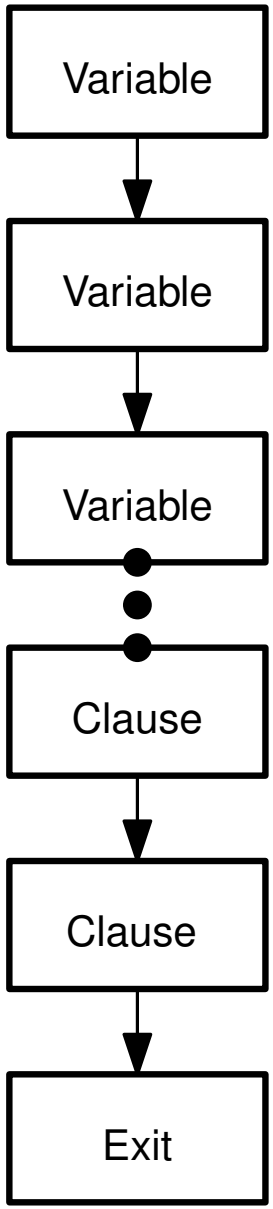
iff the formula can be satisfied

However

There is a problem. . .



Embedding 3-SAT in Crystal Caves



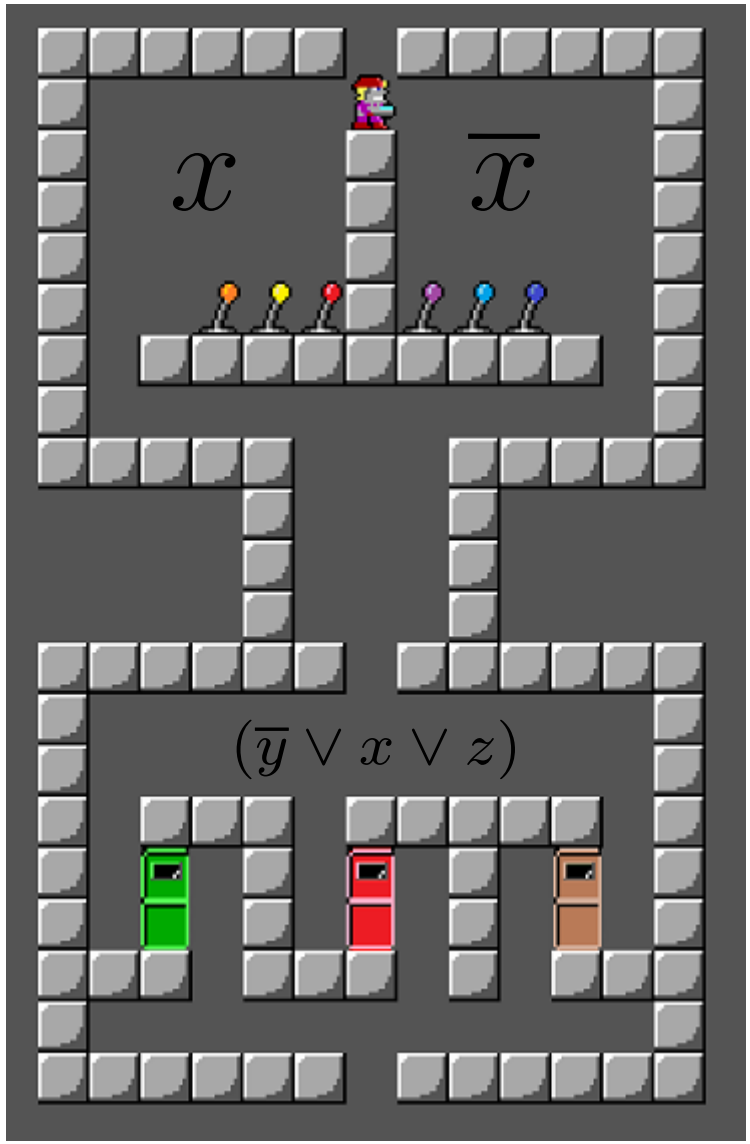
The miner can exit the level

iff the formula can be satisfied

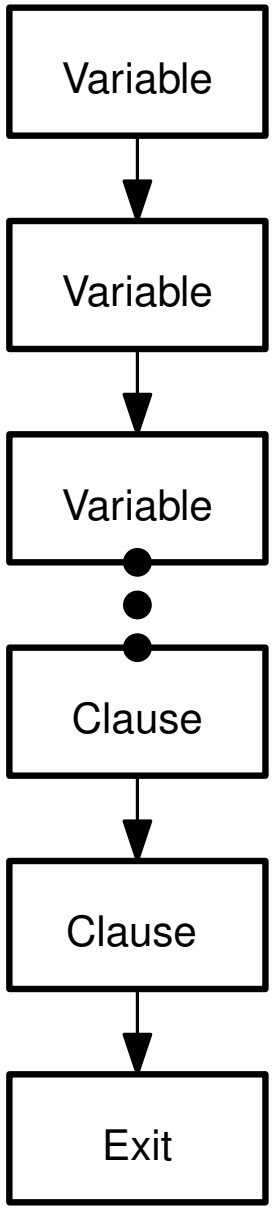
However

There is a problem. . .

How many colours do we use?



Embedding 3-SAT in Crystal Caves



The miner can exit the level

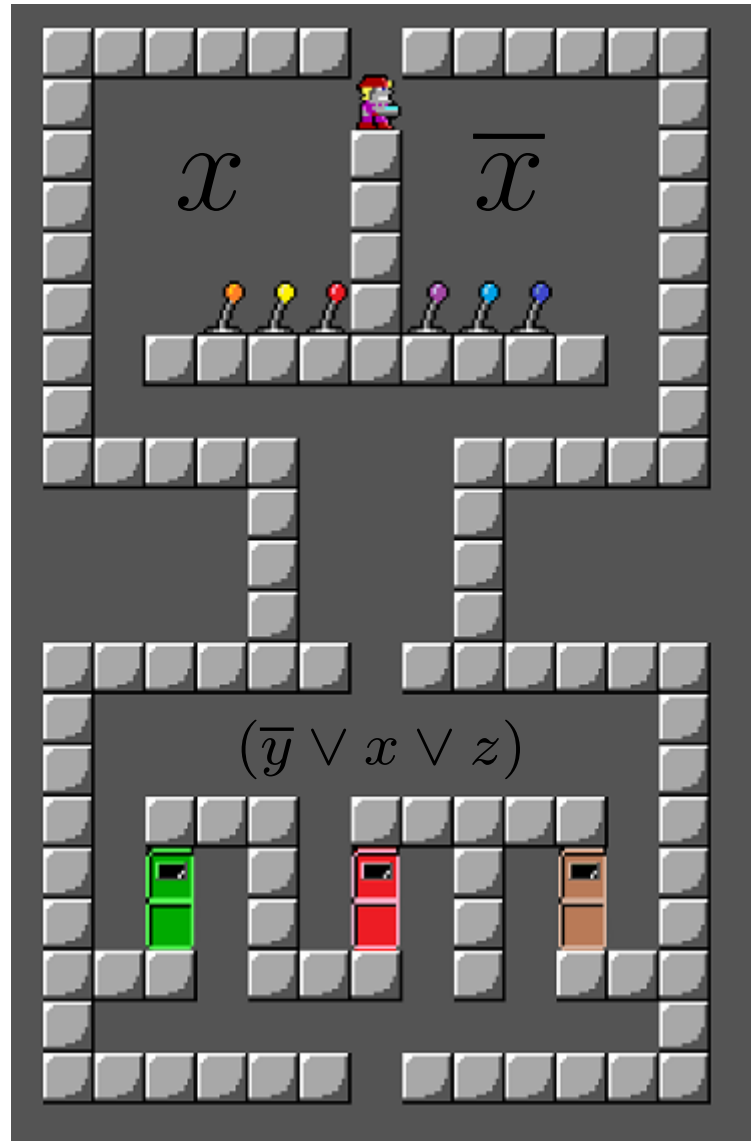
iff the formula can be satisfied

However

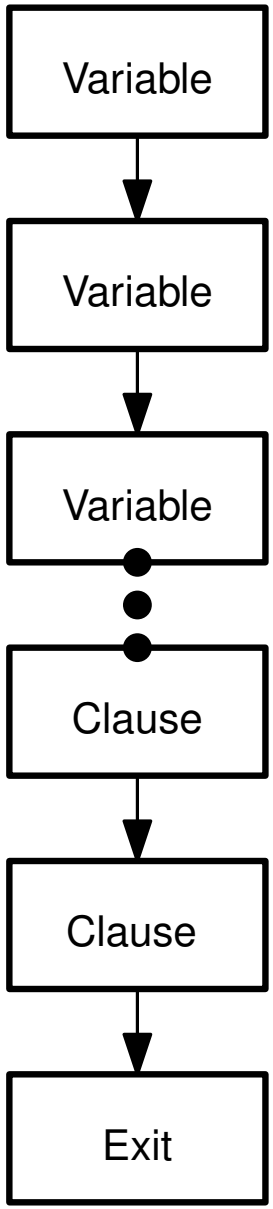
There is a problem. . .

How many colours do we use?

three in every clause!



Embedding 3-SAT in Crystal Caves



The miner can exit the level

iff the formula can be satisfied

However

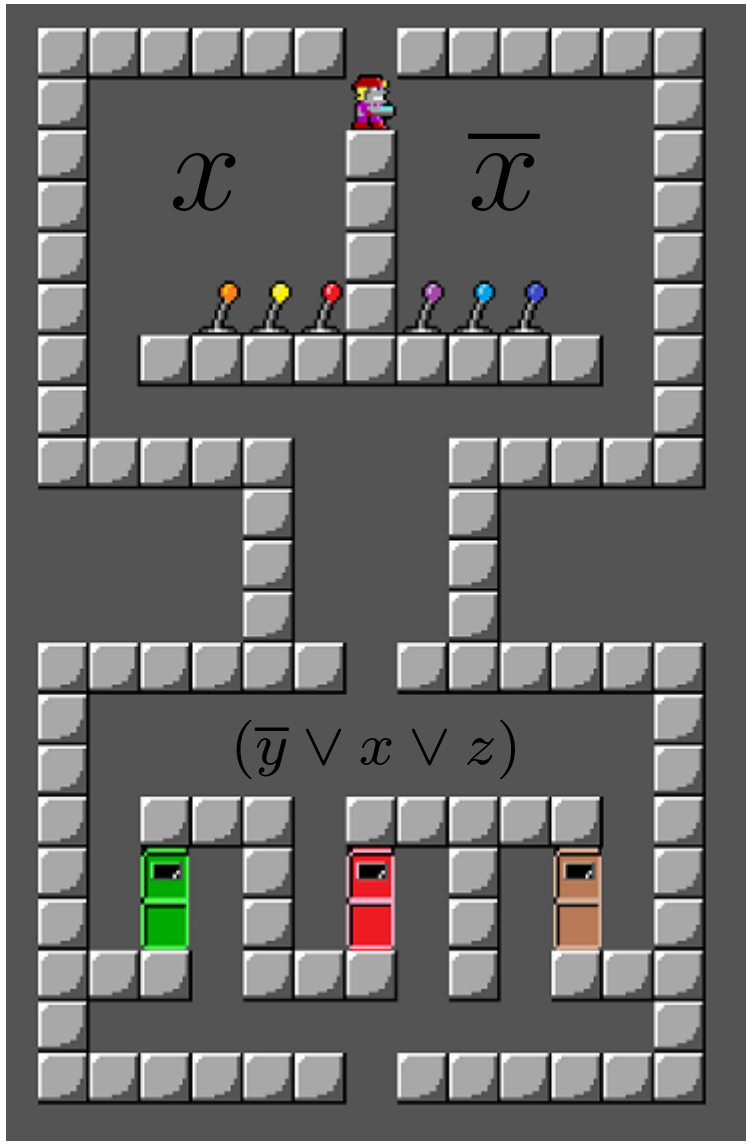
There is a problem. . .

How many colours do we use?

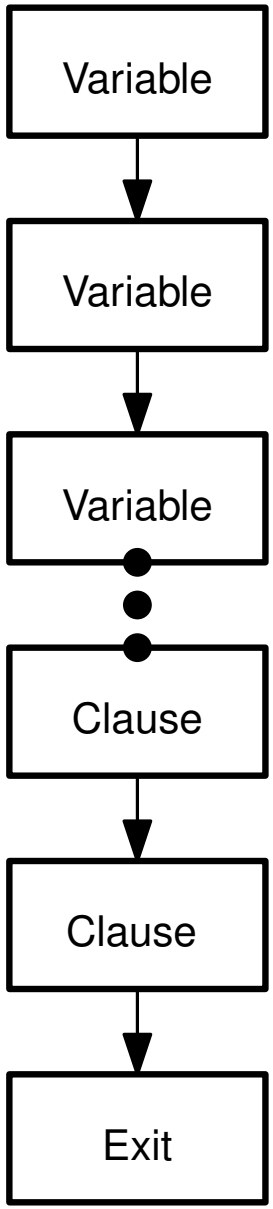
three in every clause!

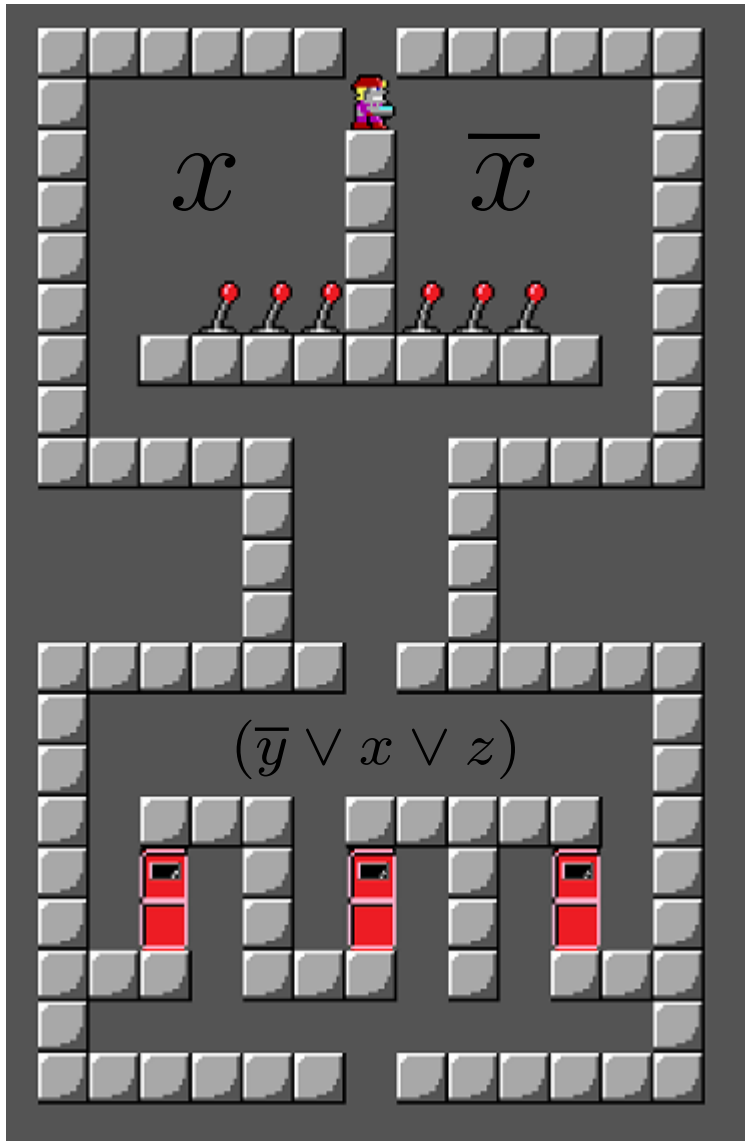
Crystal Caves is EGA

(64 colours in total)

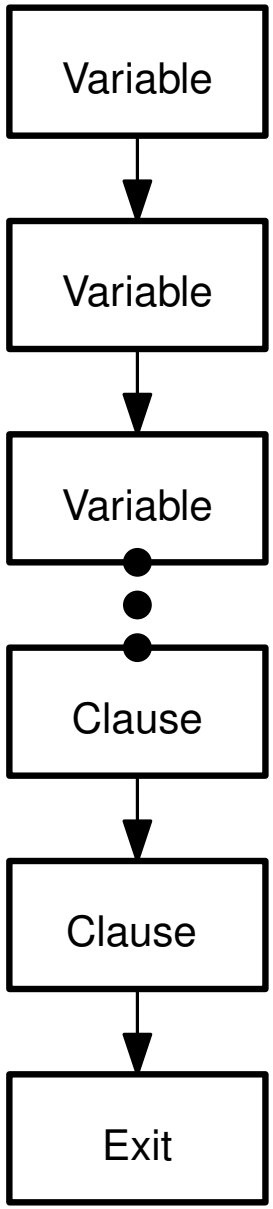


Embedding 3-SAT in Crystal Caves (with modified switches)

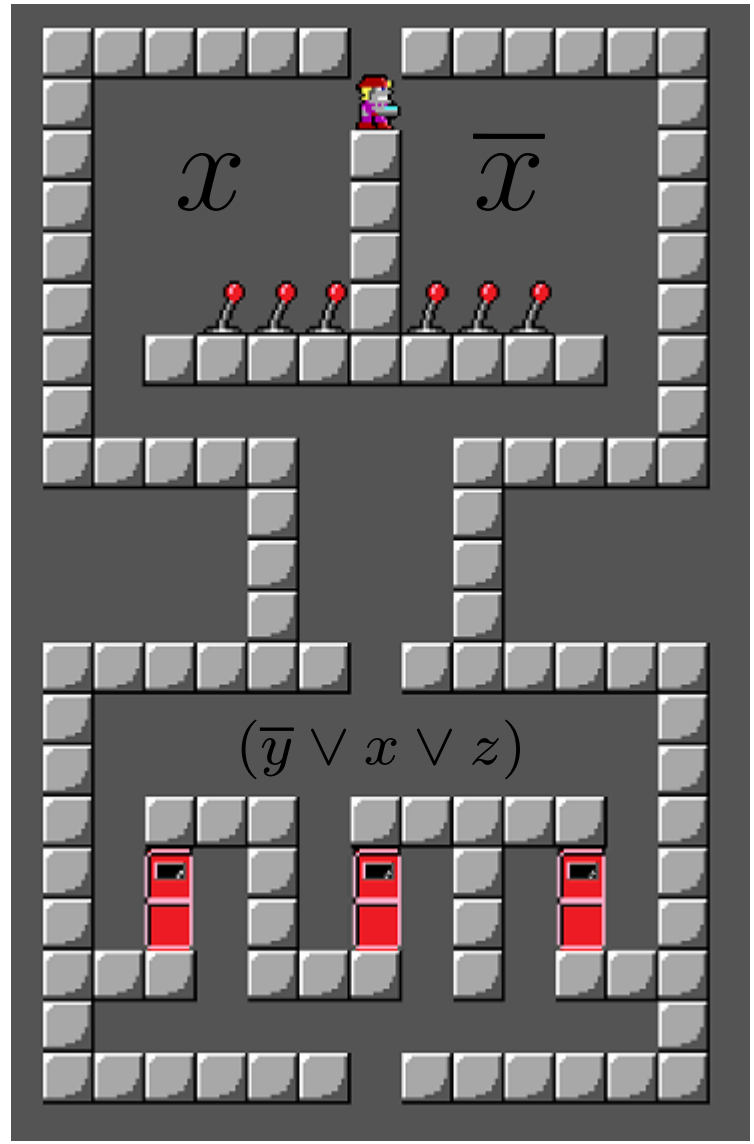




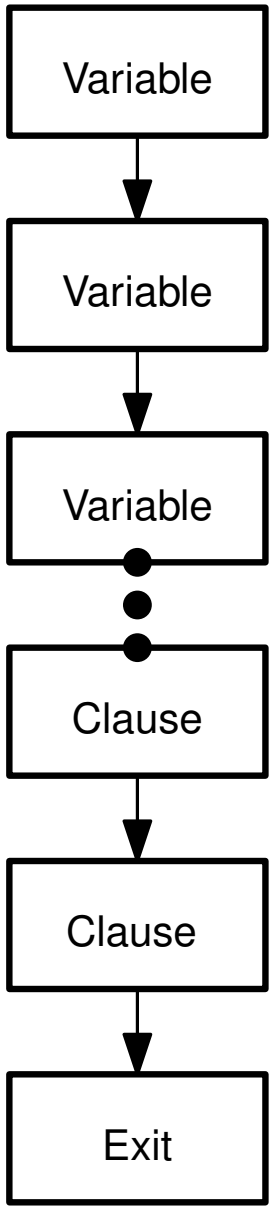
Embedding 3-SAT in Crystal Caves (with modified switches)



Make all the switches and doors the same colour

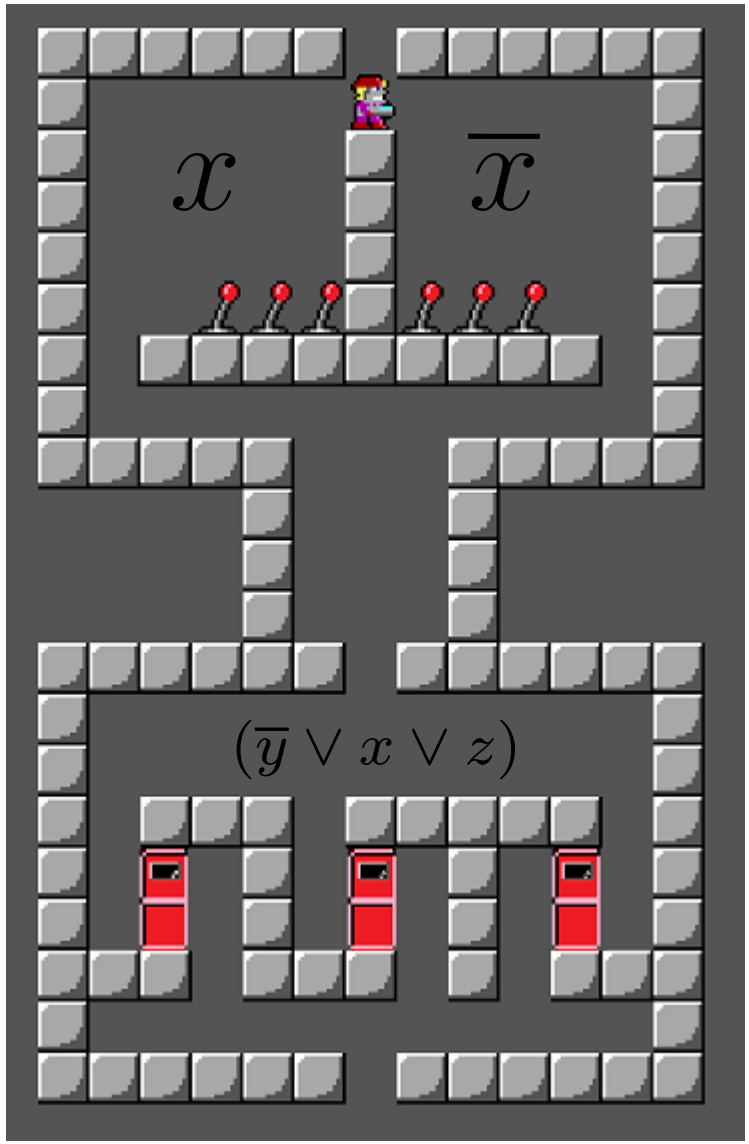


Embedding 3-SAT in Crystal Caves (with modified switches)

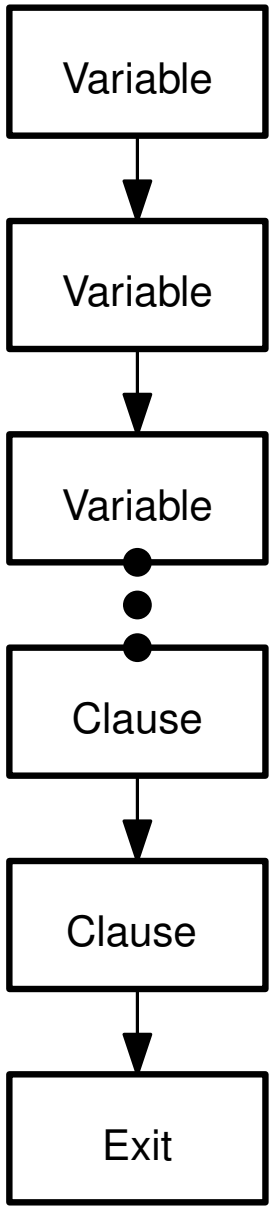


Make all the switches and doors the same colour

. . . but still connect them in the same way

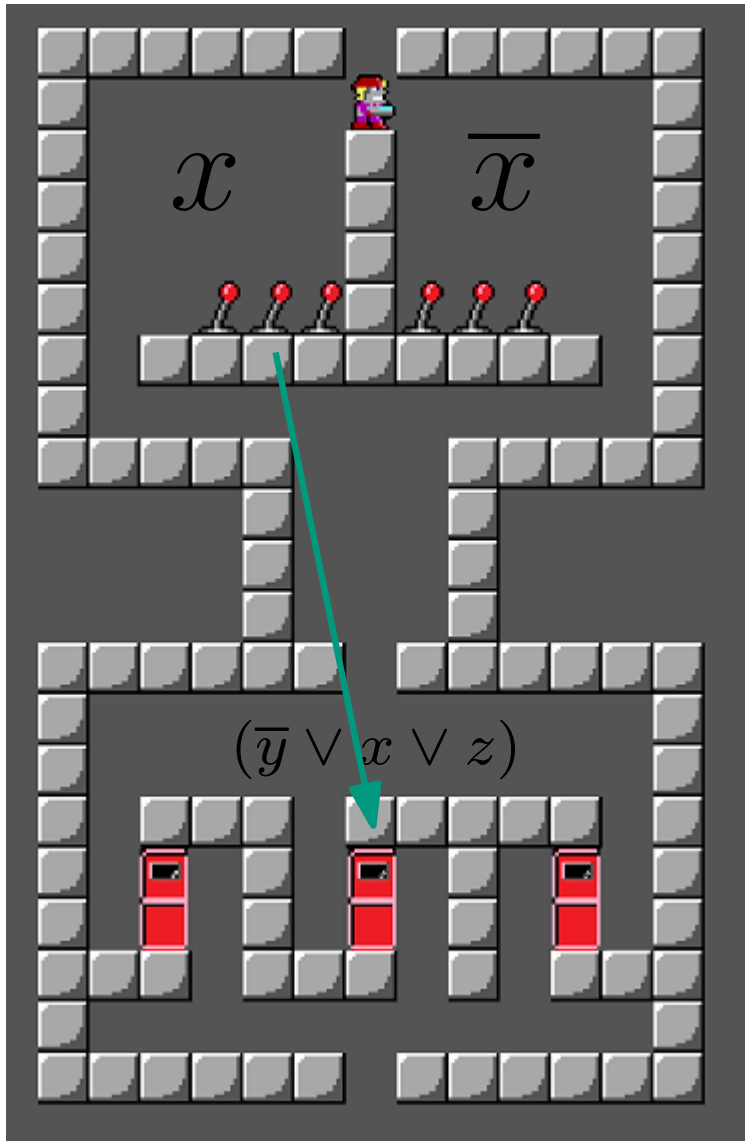


Embedding 3-SAT in Crystal Caves (with modified switches)

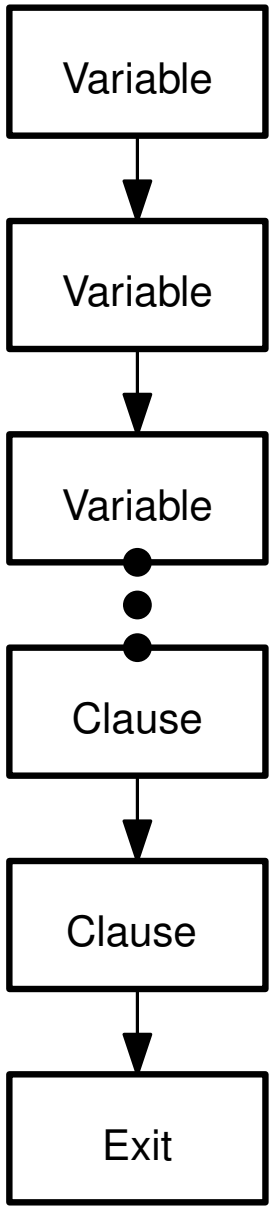


Make all the switches and doors the same colour

. . . but still connect them in the same way



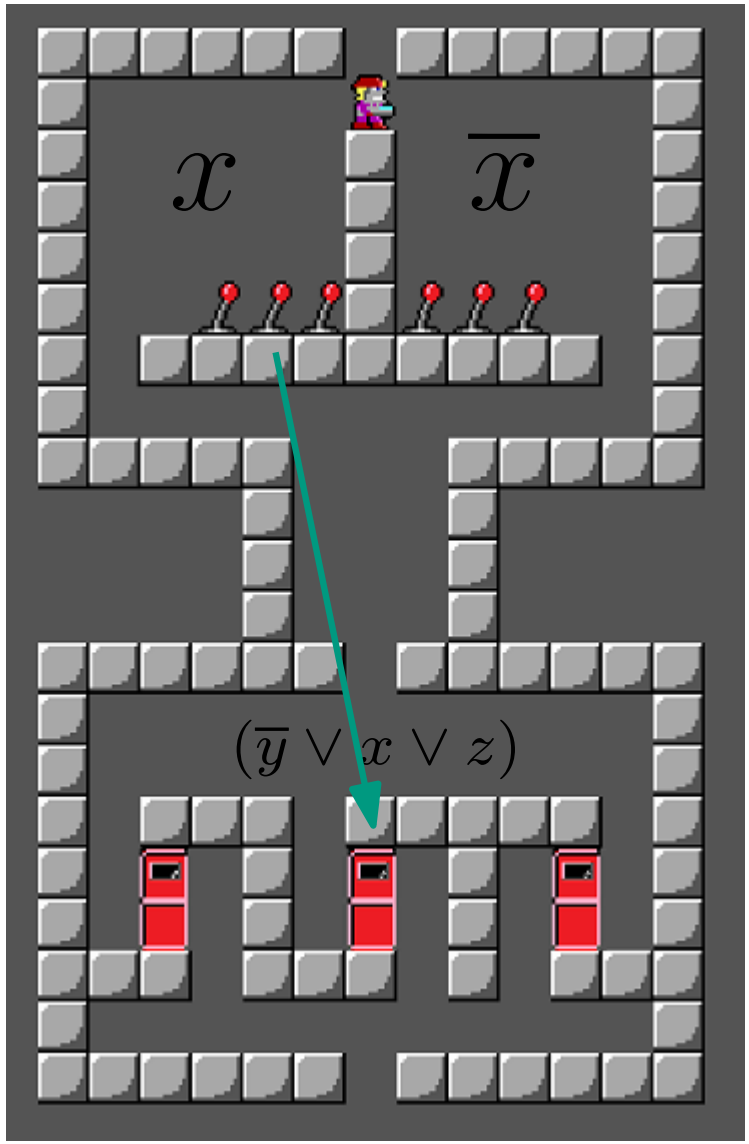
Embedding 3-SAT in Crystal Caves (with modified switches)



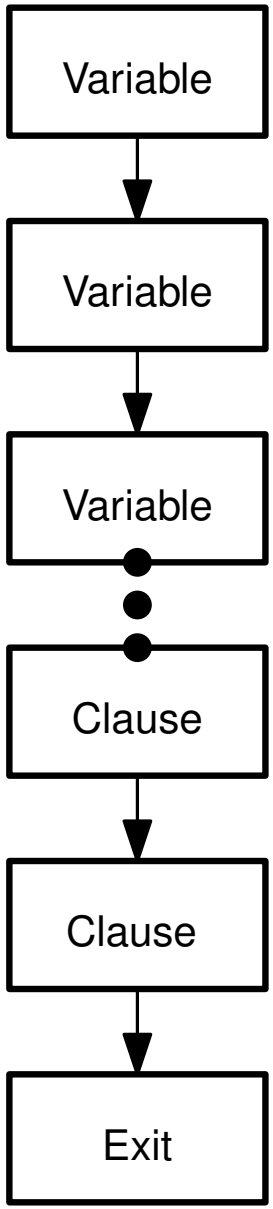
Make all the switches and doors the same colour

. . . but still connect them in the same way

Many games have switches like this (or keys)



Embedding 3-SAT in Crystal Caves (with modified switches)

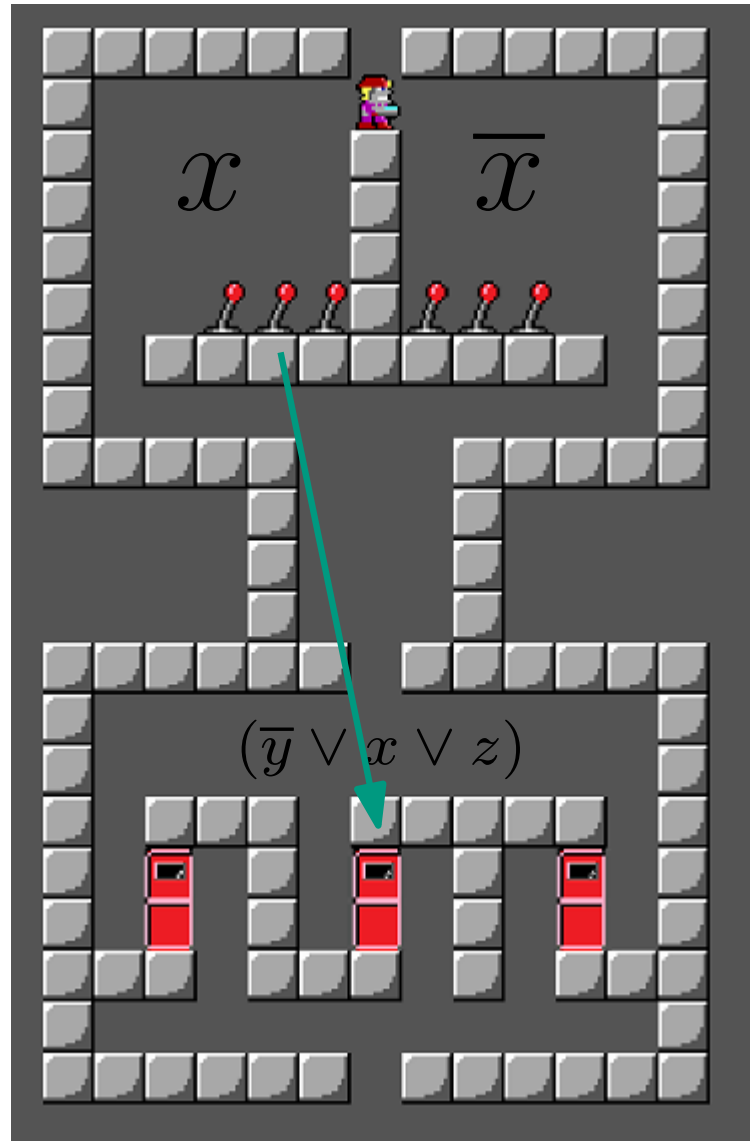


Make all the switches and doors the same colour

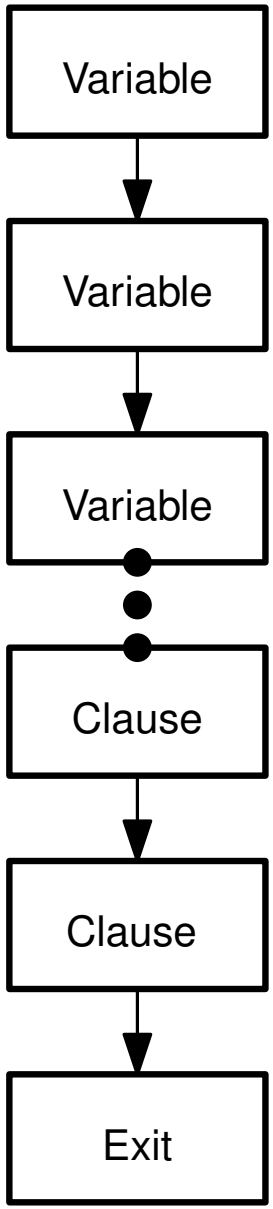
. . . but still connect them in the same way

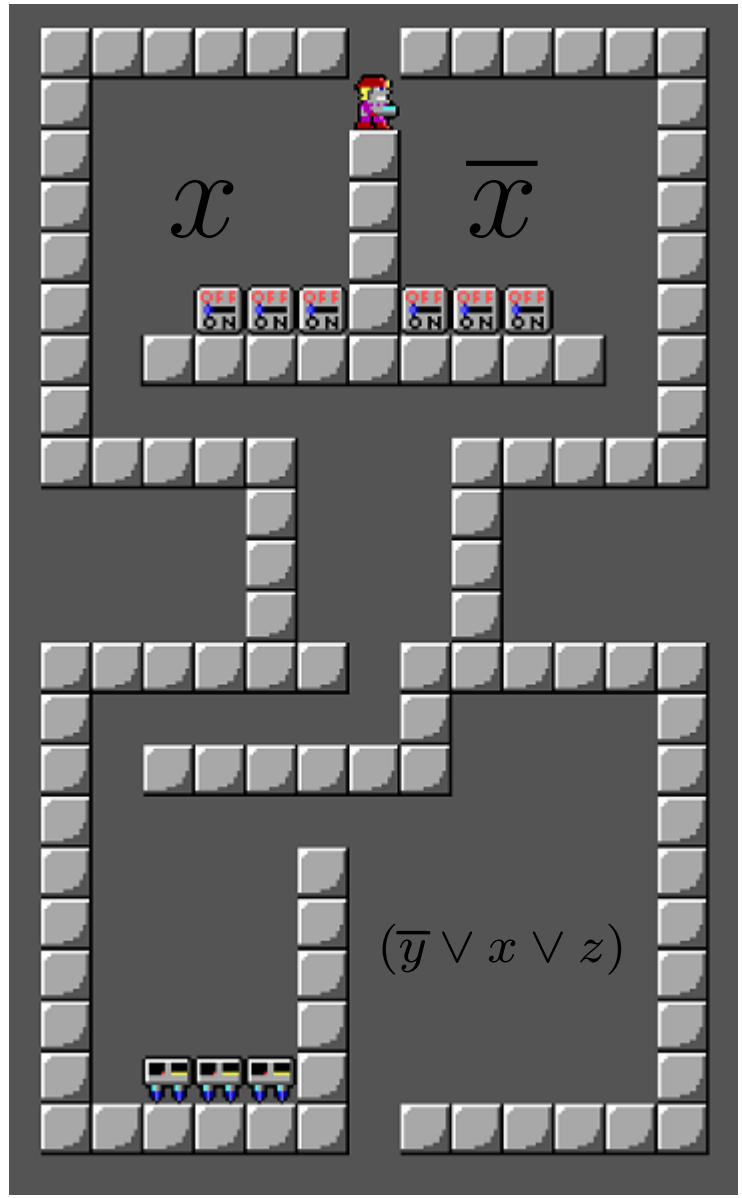
Many games have switches like this (or keys)

Crystal Caves doesn't

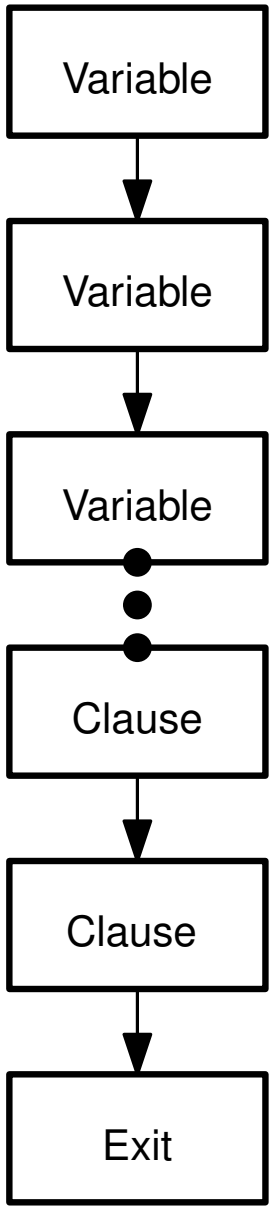


Embedding 3-SAT in Crystal Caves



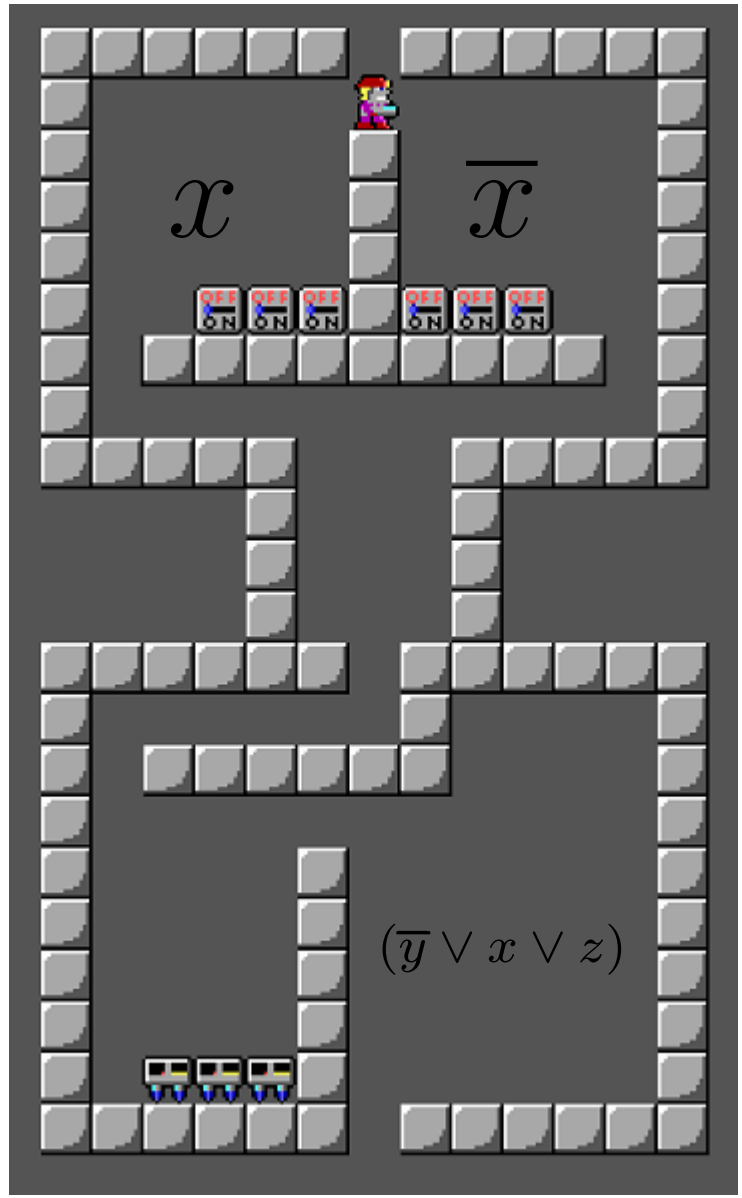


Embedding 3-SAT in Crystal Caves

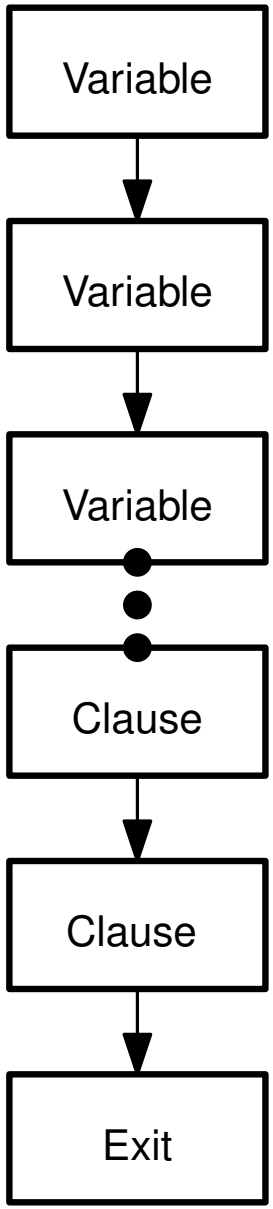


Crystal Caves has switches

which work arbitrary platforms

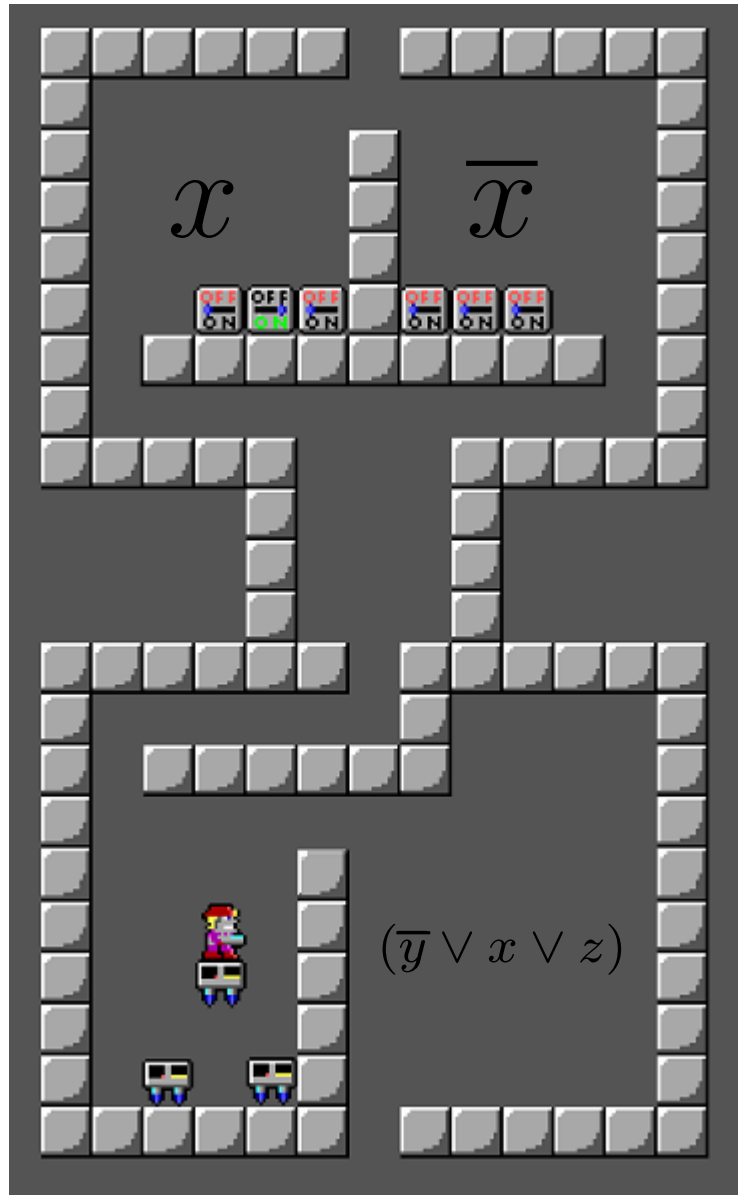


Embedding 3-SAT in Crystal Caves

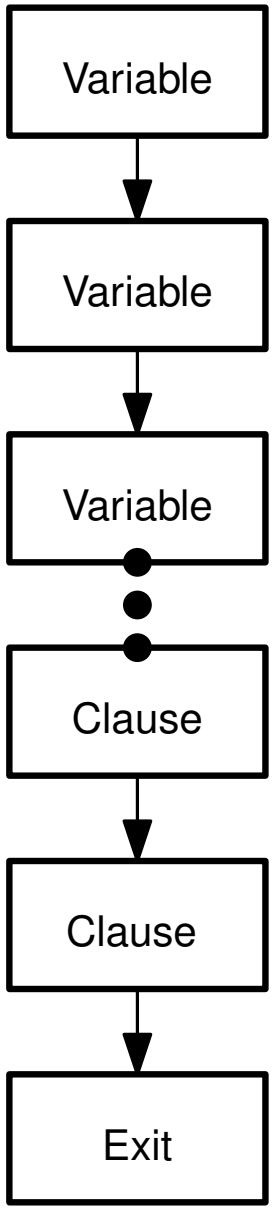


Crystal Caves has switches

which work arbitrary platforms

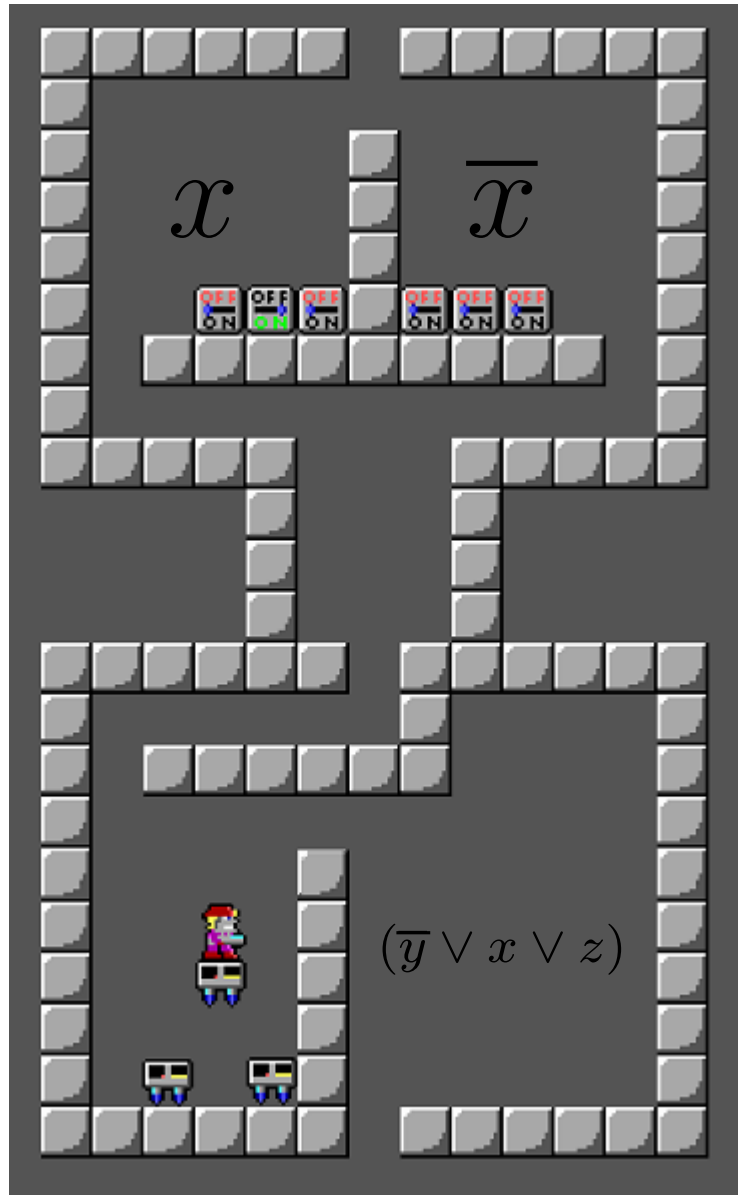


Embedding 3-SAT in Crystal Caves

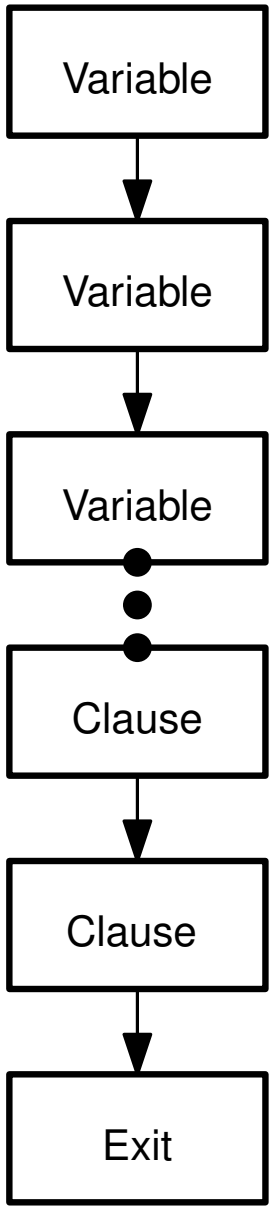


Crystal Caves has switches which work arbitrary platforms

The miner can jump over iff at least one platform is active



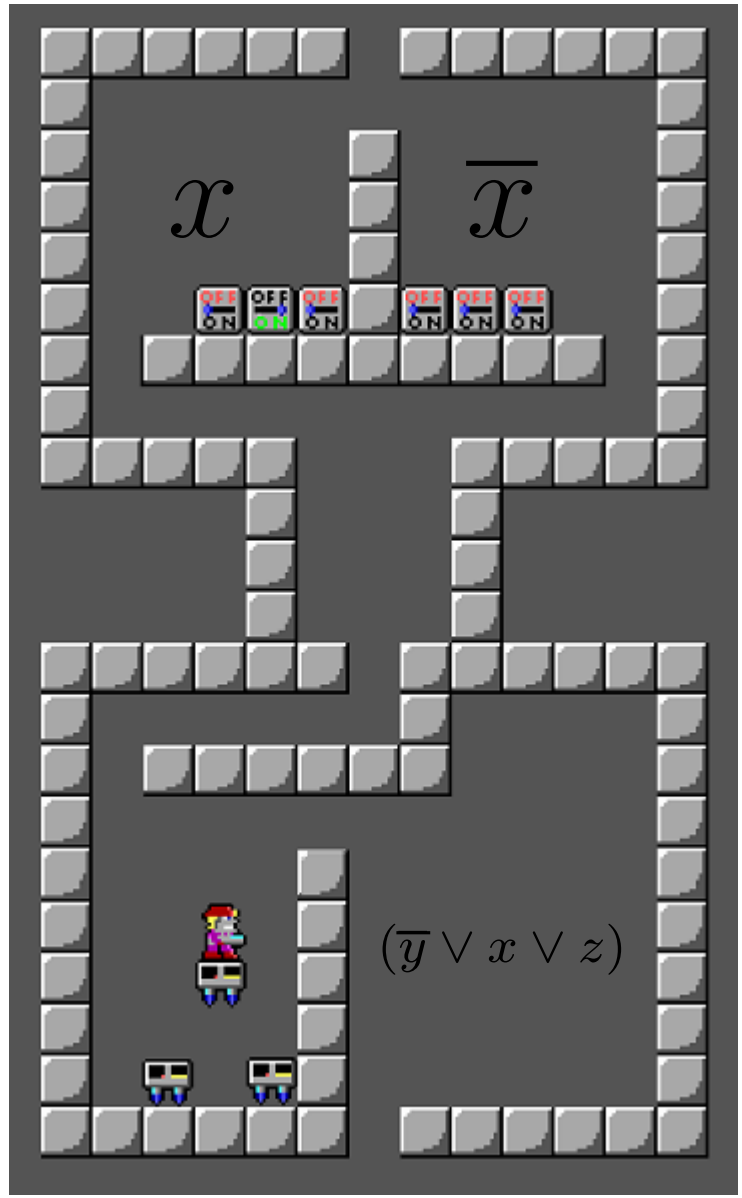
Embedding 3-SAT in Crystal Caves



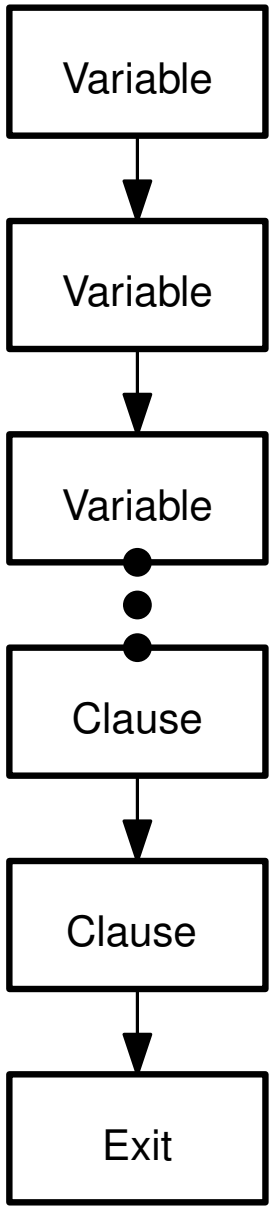
Crystal Caves has switches which work arbitrary platforms

The miner can jump over iff at least one platform is active

So Crystal Caves is NP-hard



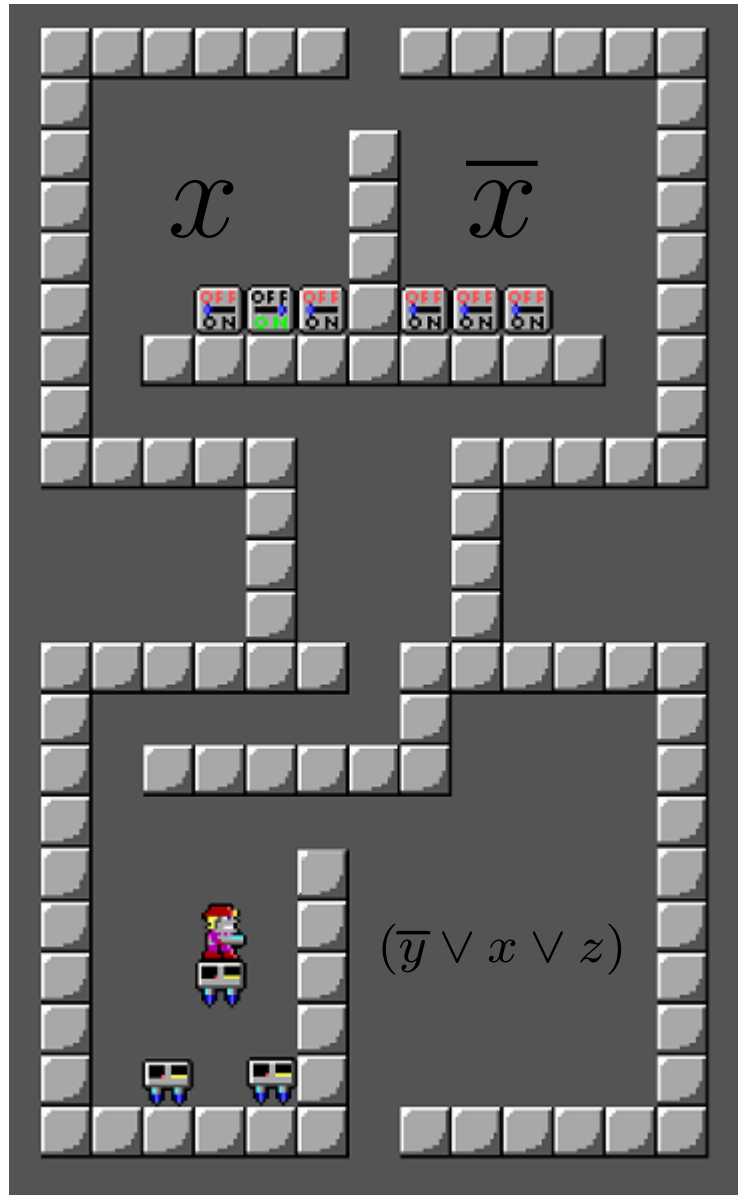
Embedding 3-SAT in Crystal Caves



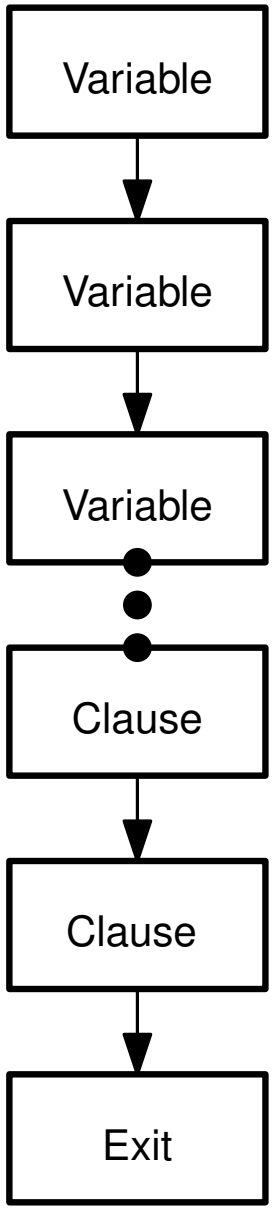
Crystal Caves has switches which work arbitrary platforms

The miner can jump over iff at least one platform is active

So Crystal Caves is NP-hard



Embedding 3-SAT in Crystal Caves

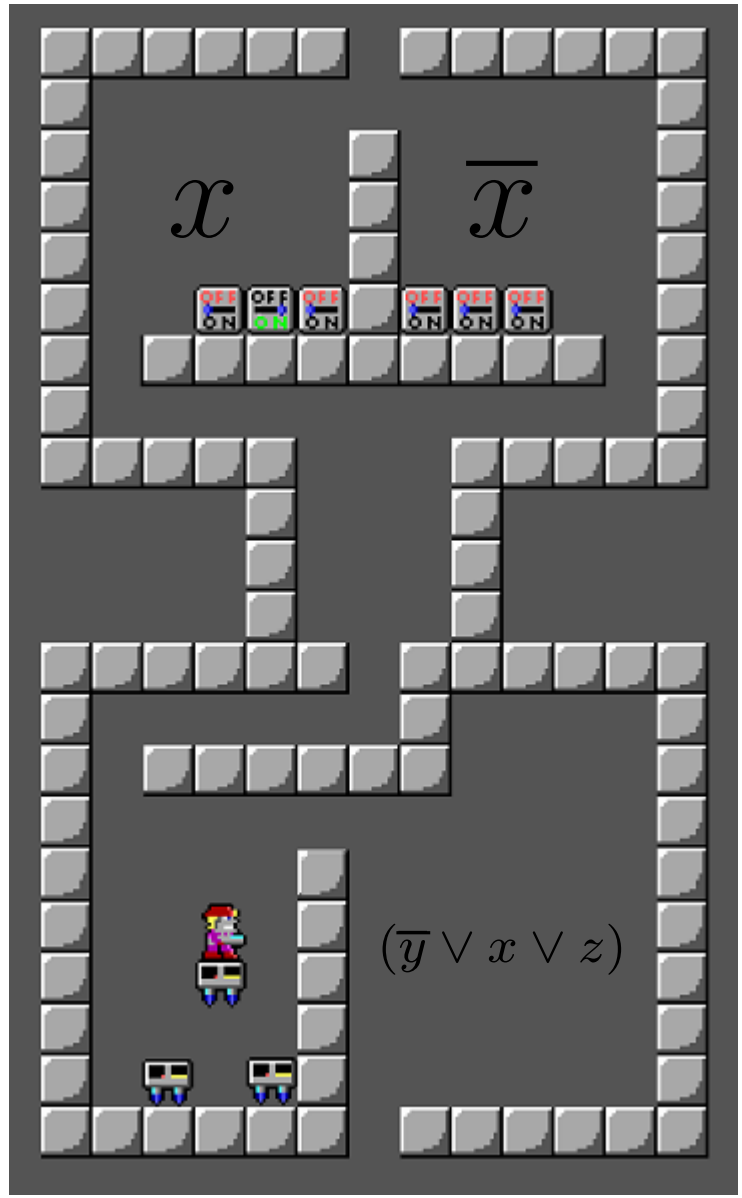


Crystal Caves has switches which work arbitrary platforms

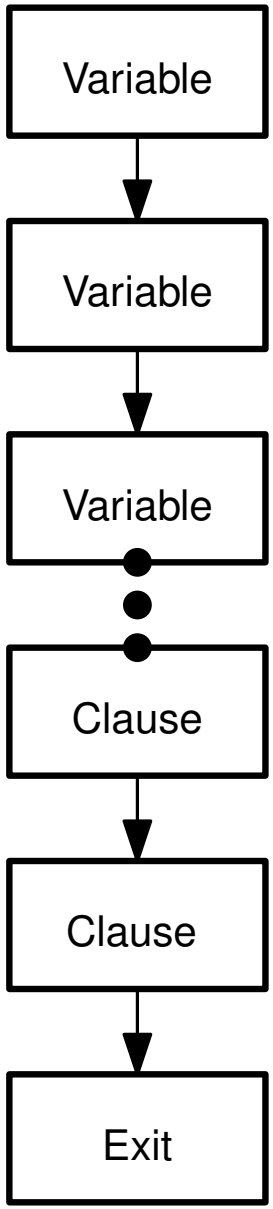
The miner can jump over iff at least one platform is active

So Crystal Caves is NP-hard

A similar approach works for many other platform games with keys/switches and locked doors



Embedding 3-SAT in Crystal Caves



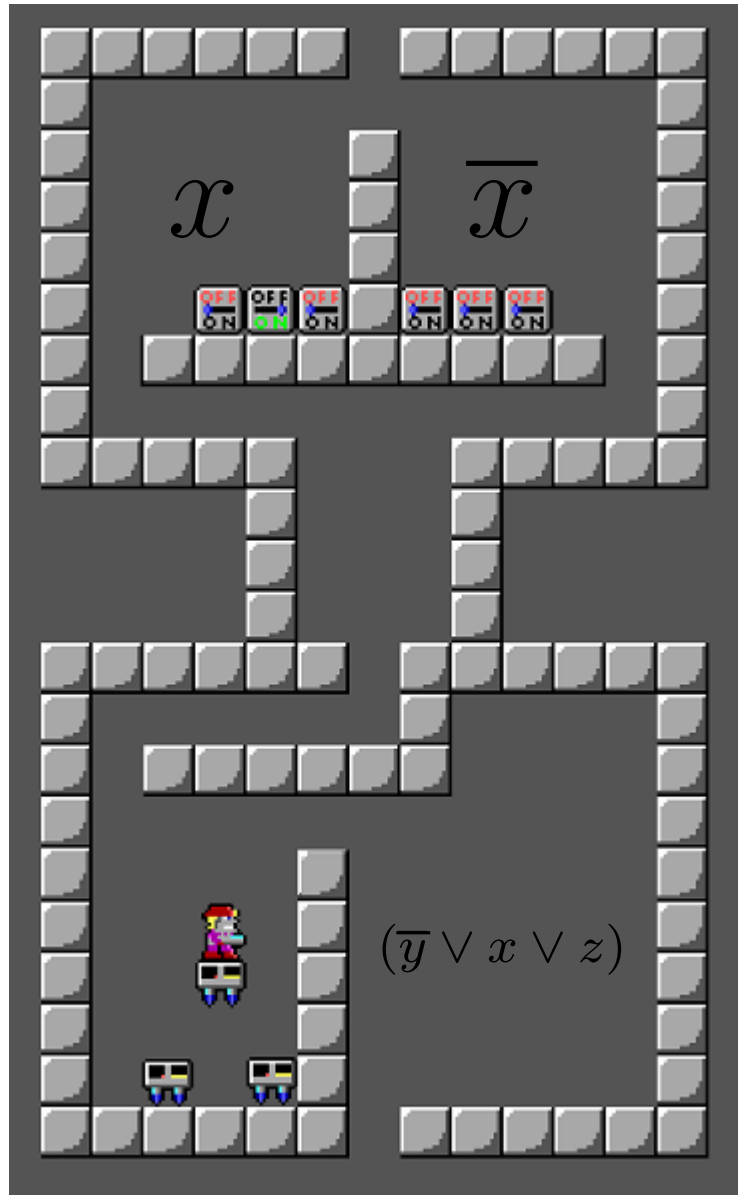
Crystal Caves has switches which work arbitrary platforms

The miner can jump over iff at least one platform is active

So Crystal Caves is NP-hard

A similar approach works for many other platform games with keys/switches and locked doors

but what about Mario?



The approach

3-SAT: Decide whether a given boolean formula can be satisfied

$$(x \vee y \vee z) \wedge (\overline{x} \vee x \vee z) \wedge (\overline{y} \vee \overline{z} \vee \overline{z})$$

n

i.e. can we assign variables so that the formula is satisfied (**True**)*?*

Remember that each clause has exactly three literals

Given a 3-SAT formula we *construct* a **Super Mario Bros.** level. . .

which can be completed *if and only if* the formula can be satisfied

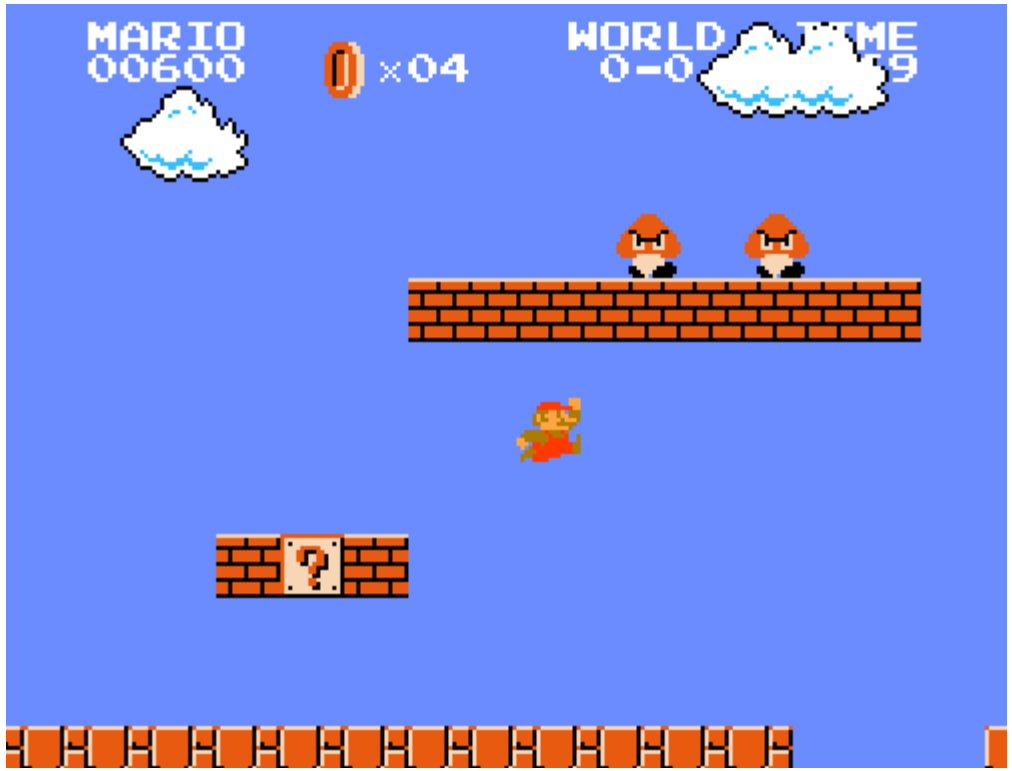
(and the size of the level will be polynomial in n)

so if we had a polynomial time **Super Mario Bros.** 'solver', we could use it to solve 3-SAT in polynomial time

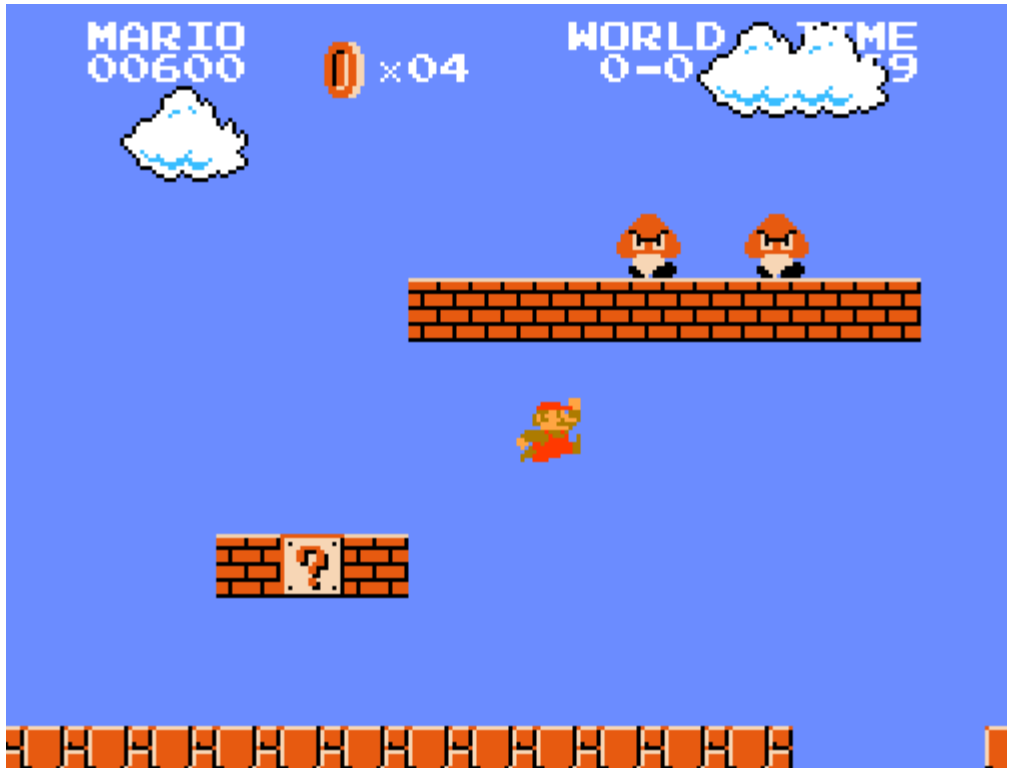
this means that **Super Mario Bros.** is *at least as hard as* 3-SAT

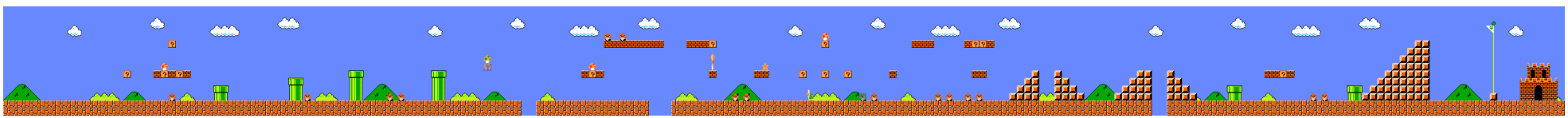
in other words, **Super Mario Bros.** is NP-hard

Super Mario Bros.

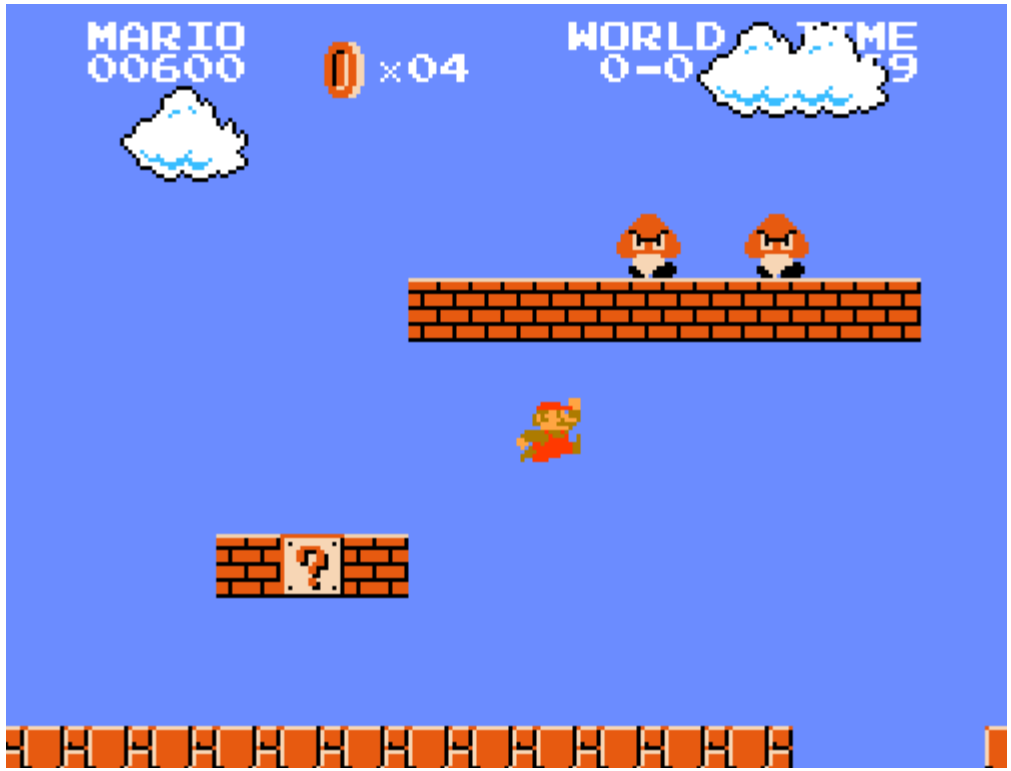


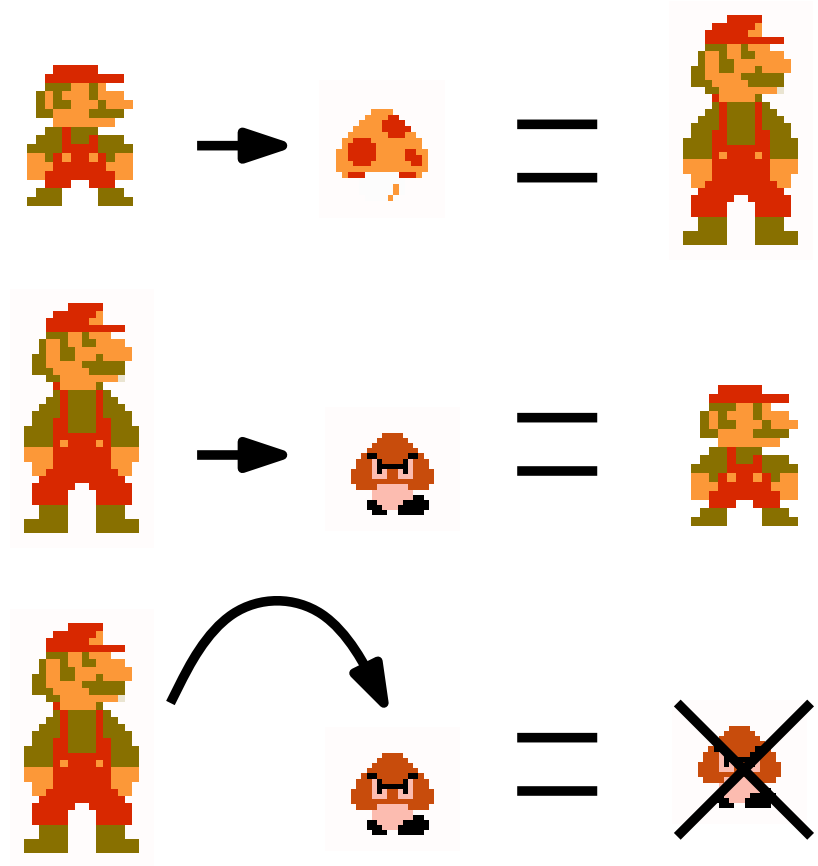
Super Mario Bros.

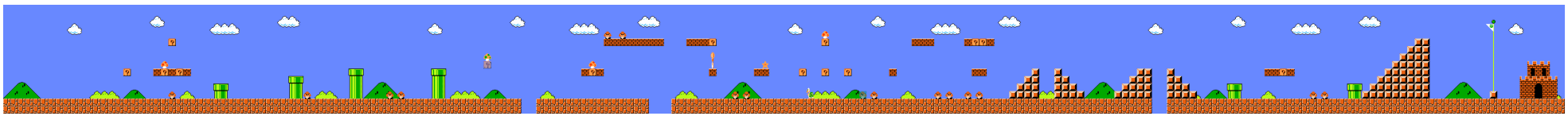




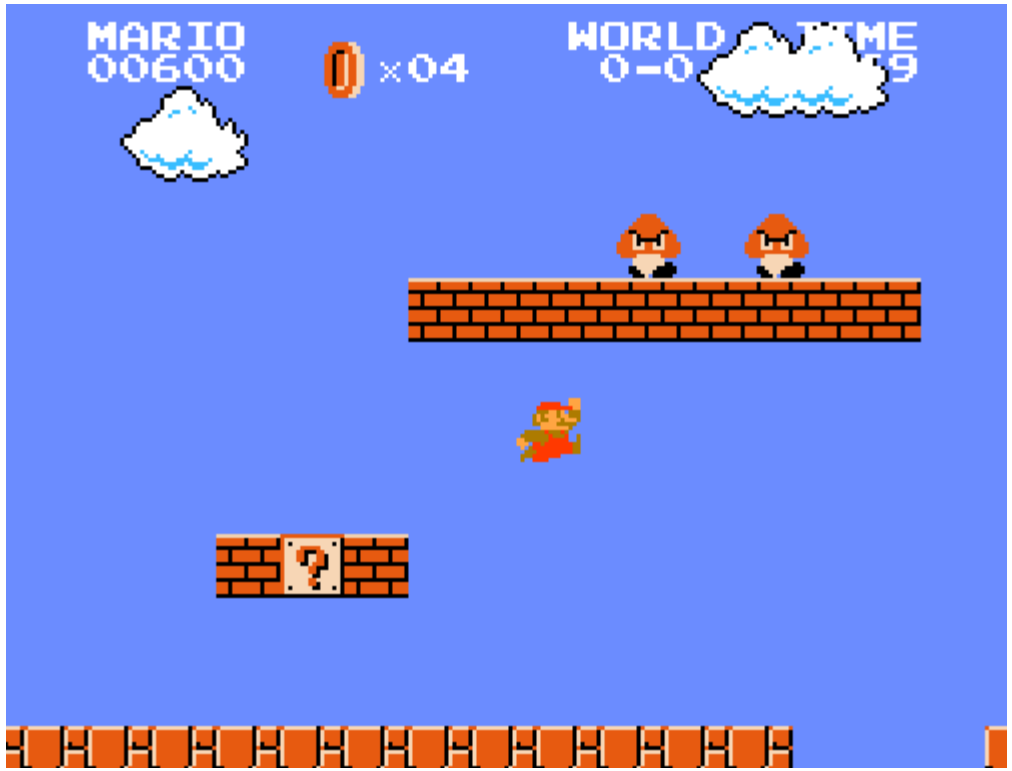
Super Mario Bros.

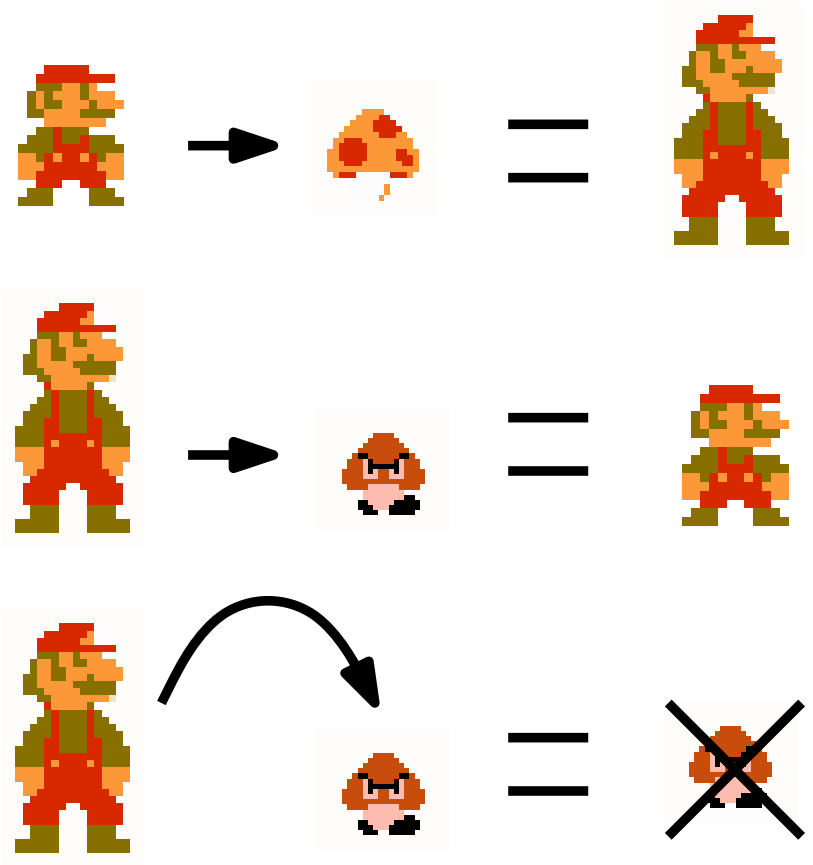






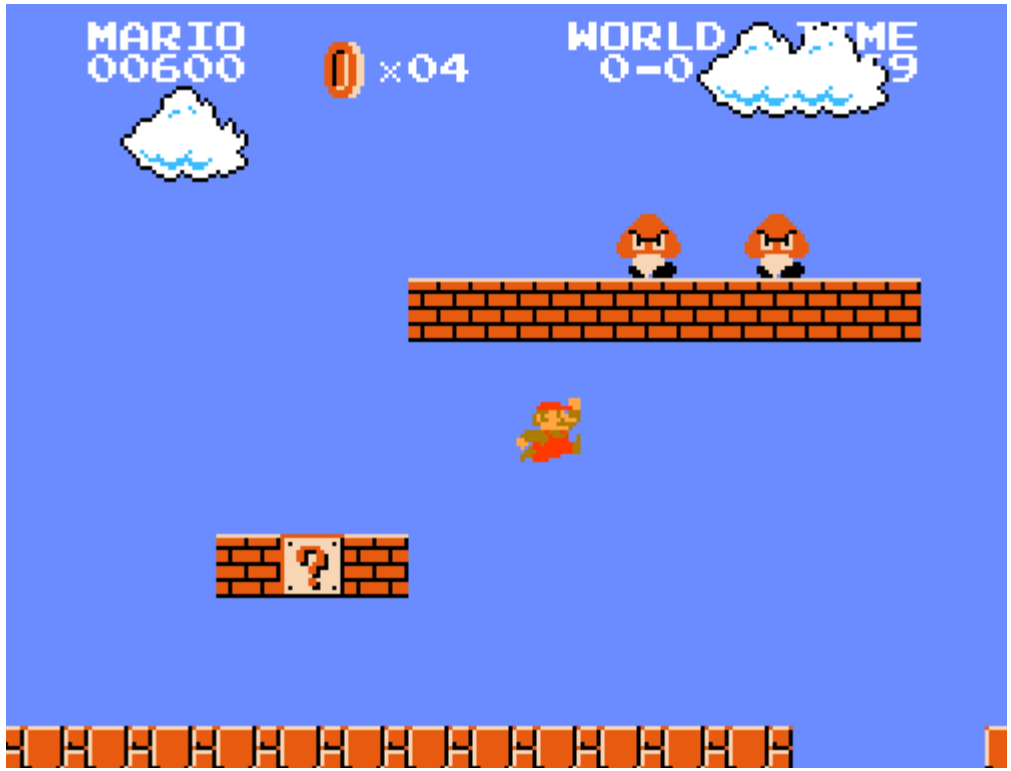
Super Mario Bros.

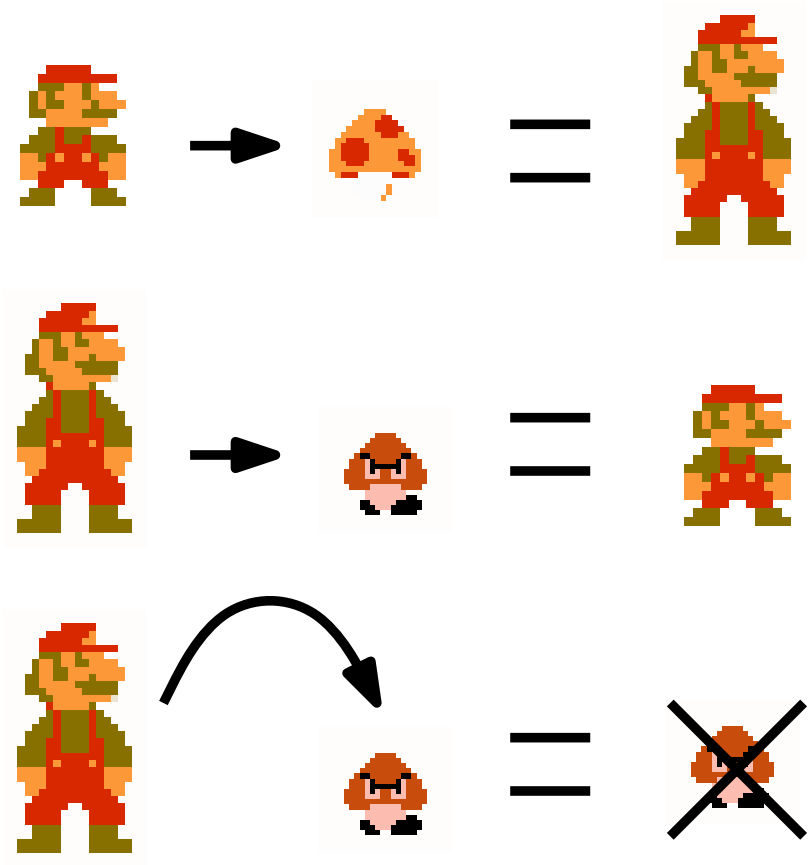




We will again prove NP-hardness by reduction from 3-SAT

Super Mario Bros.

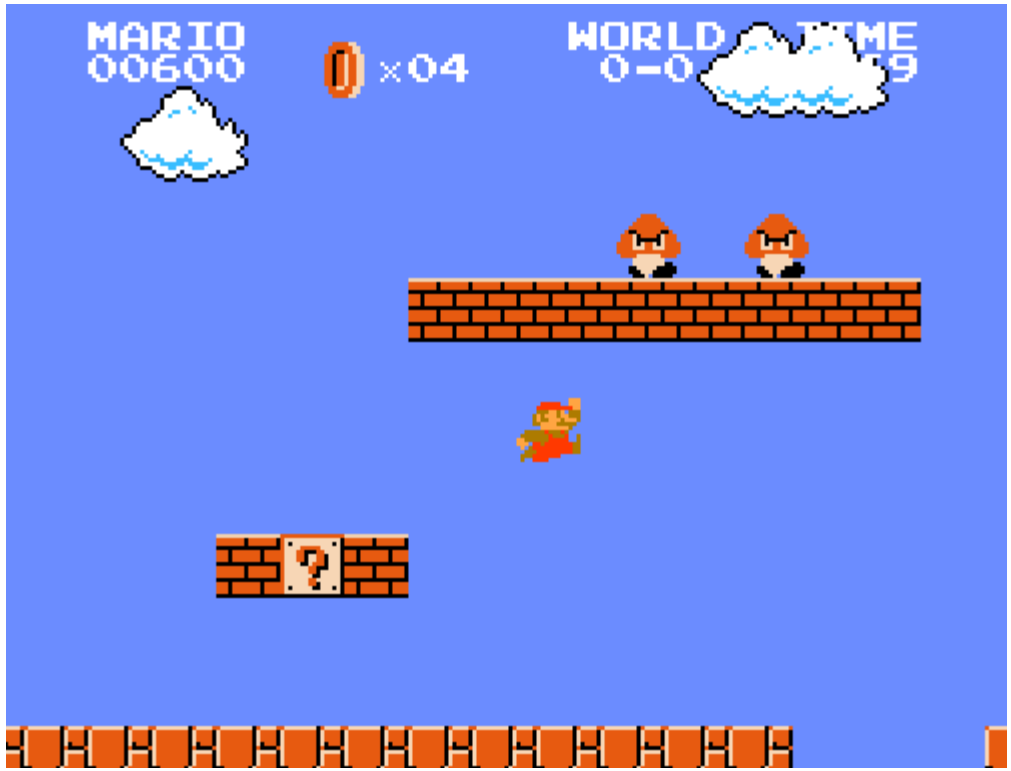


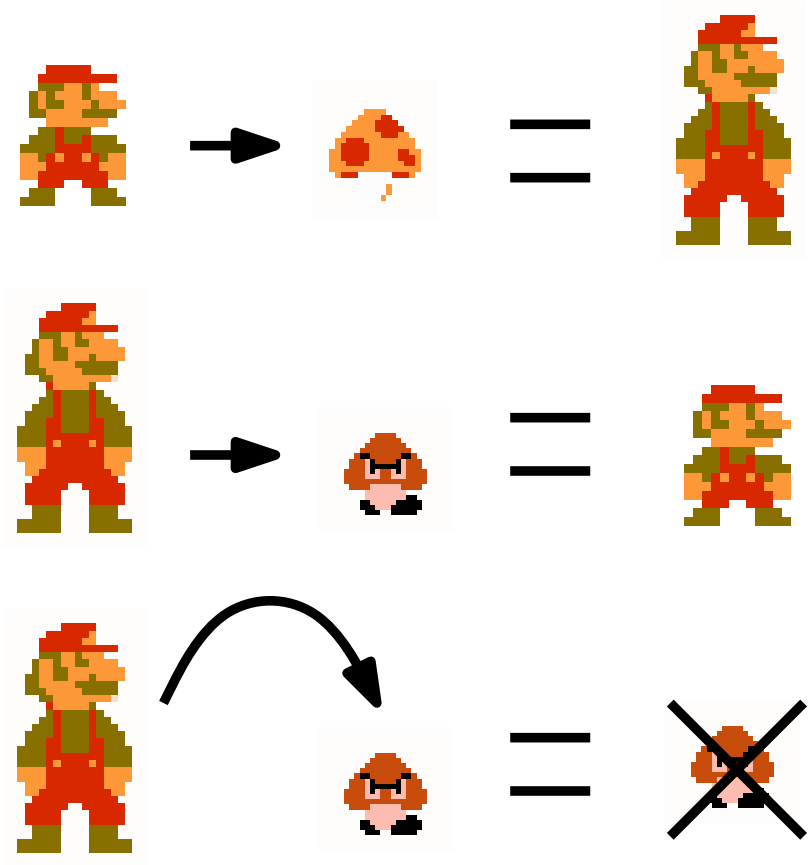


We will again prove NP-hardness by reduction from 3-SAT

. . . but without using keys/switches etc.

Super Mario Bros.





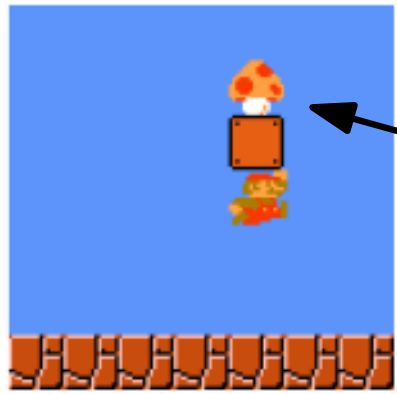
We will again prove NP-hardness by reduction from 3-SAT

. . . but without using keys/switches etc.

(because there aren't any)

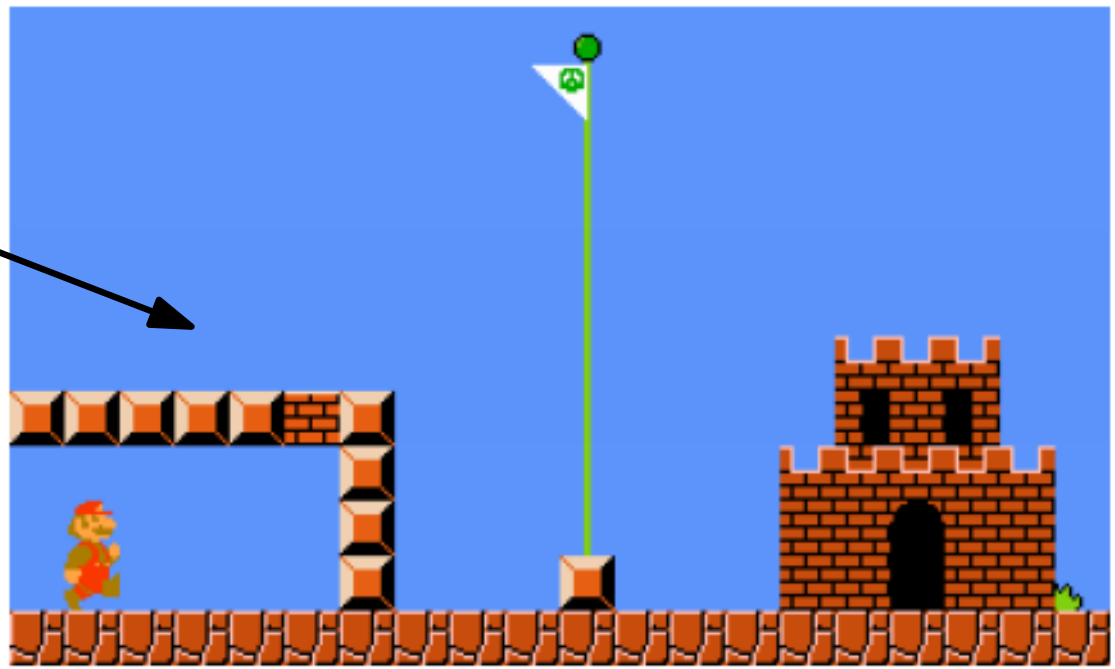
Super Mario Bros.

start gadget



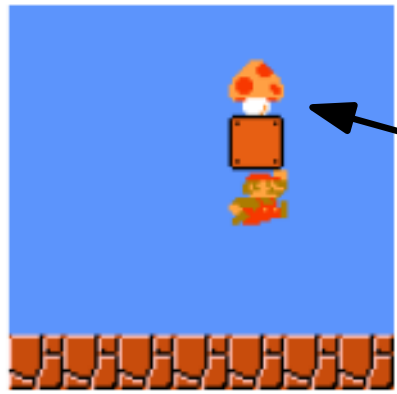
The only Mushroom

finish gadget



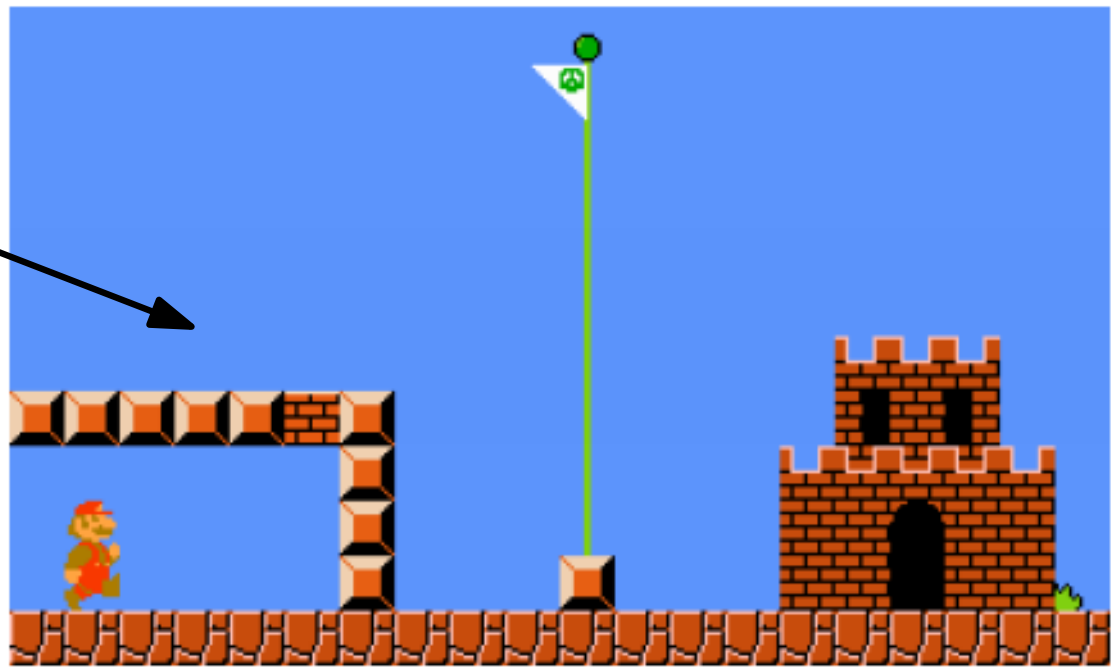
Super Mario Bros.

start gadget



The only Mushroom

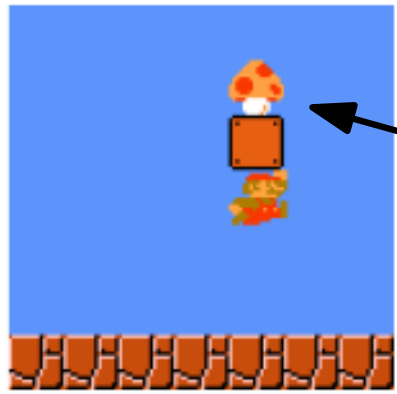
finish gadget



Little Mario can't smash blocks. . .

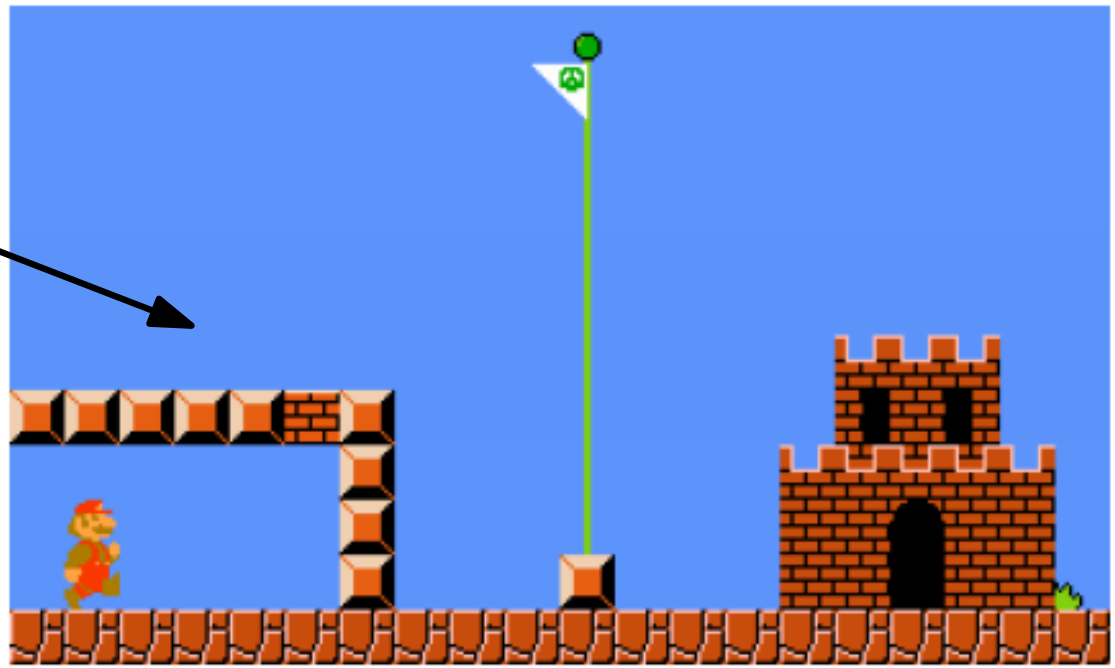
Super Mario Bros.

start gadget



The only Mushroom

finish gadget

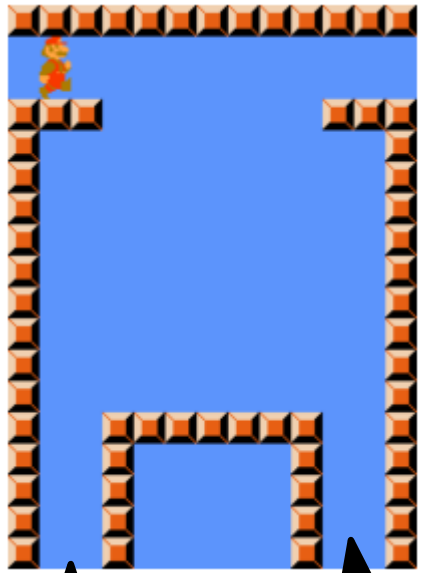


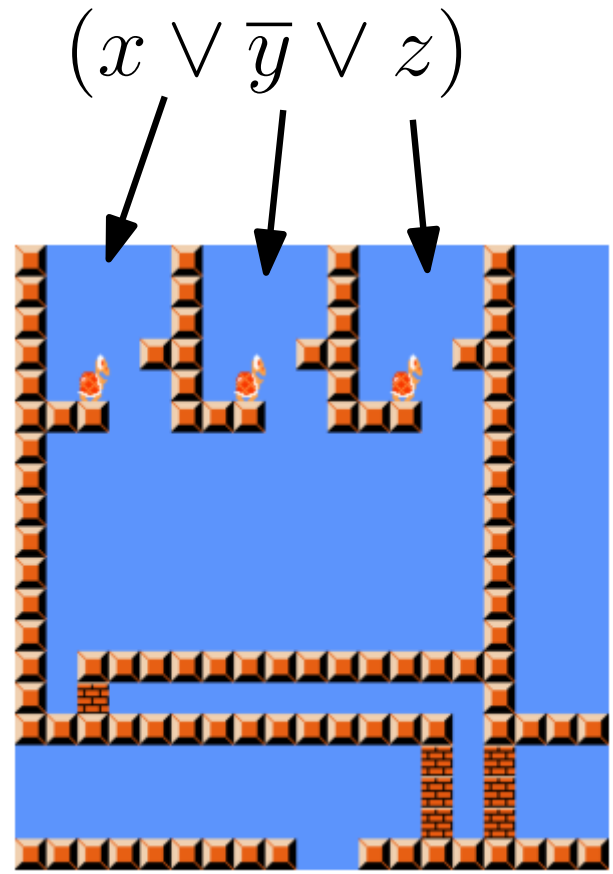
Little Mario can't smash blocks. . .

so you have to stay big to win

Variable and Clause Gadgets (Sneak Preview)

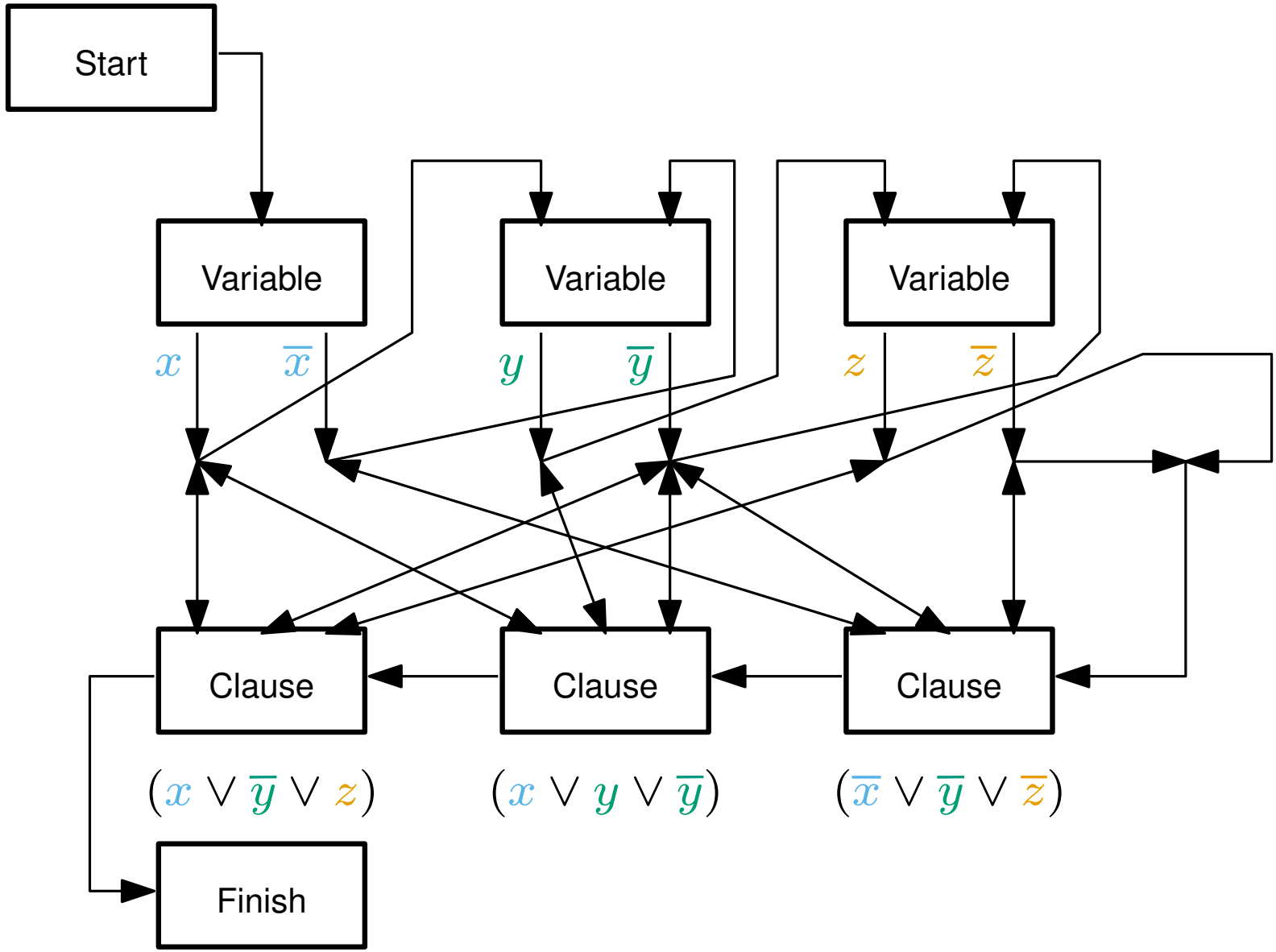
variable gadget





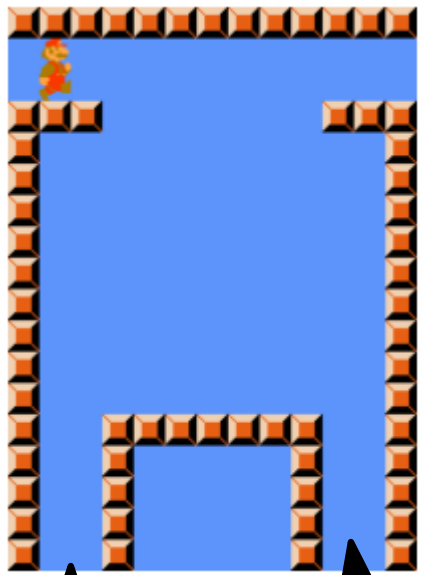
clause gadget

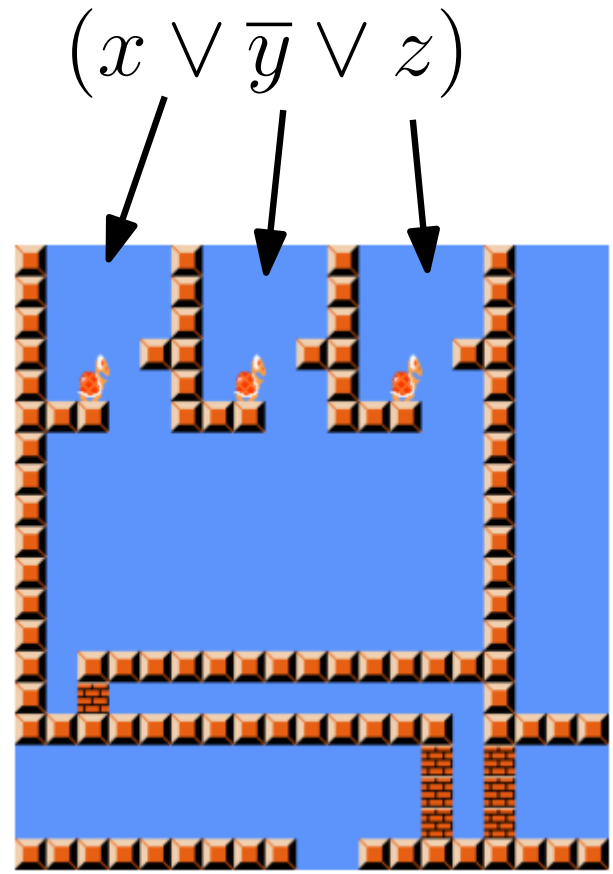
Embedding Overview



Variable and Clause Gadgets

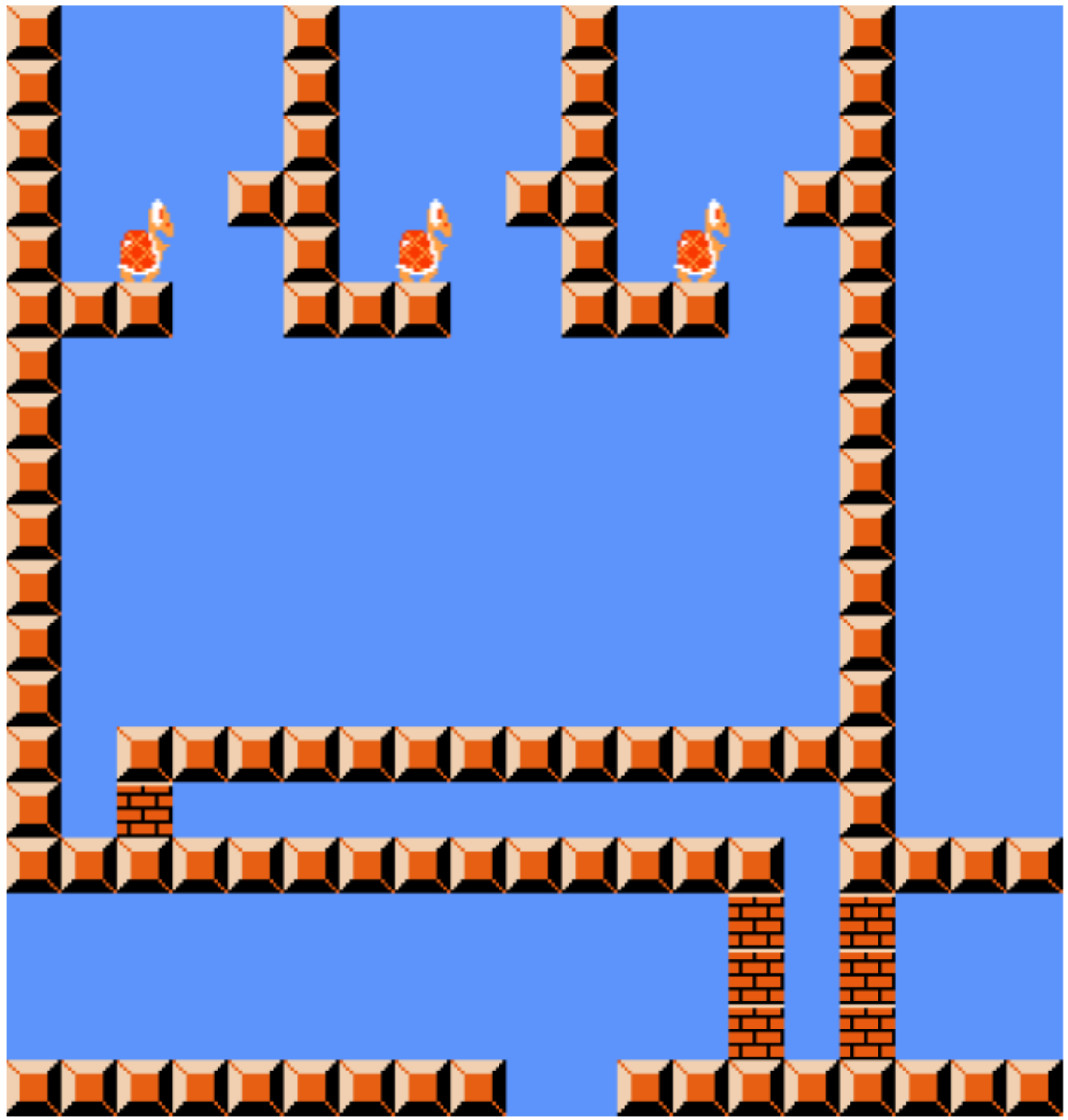
variable gadget



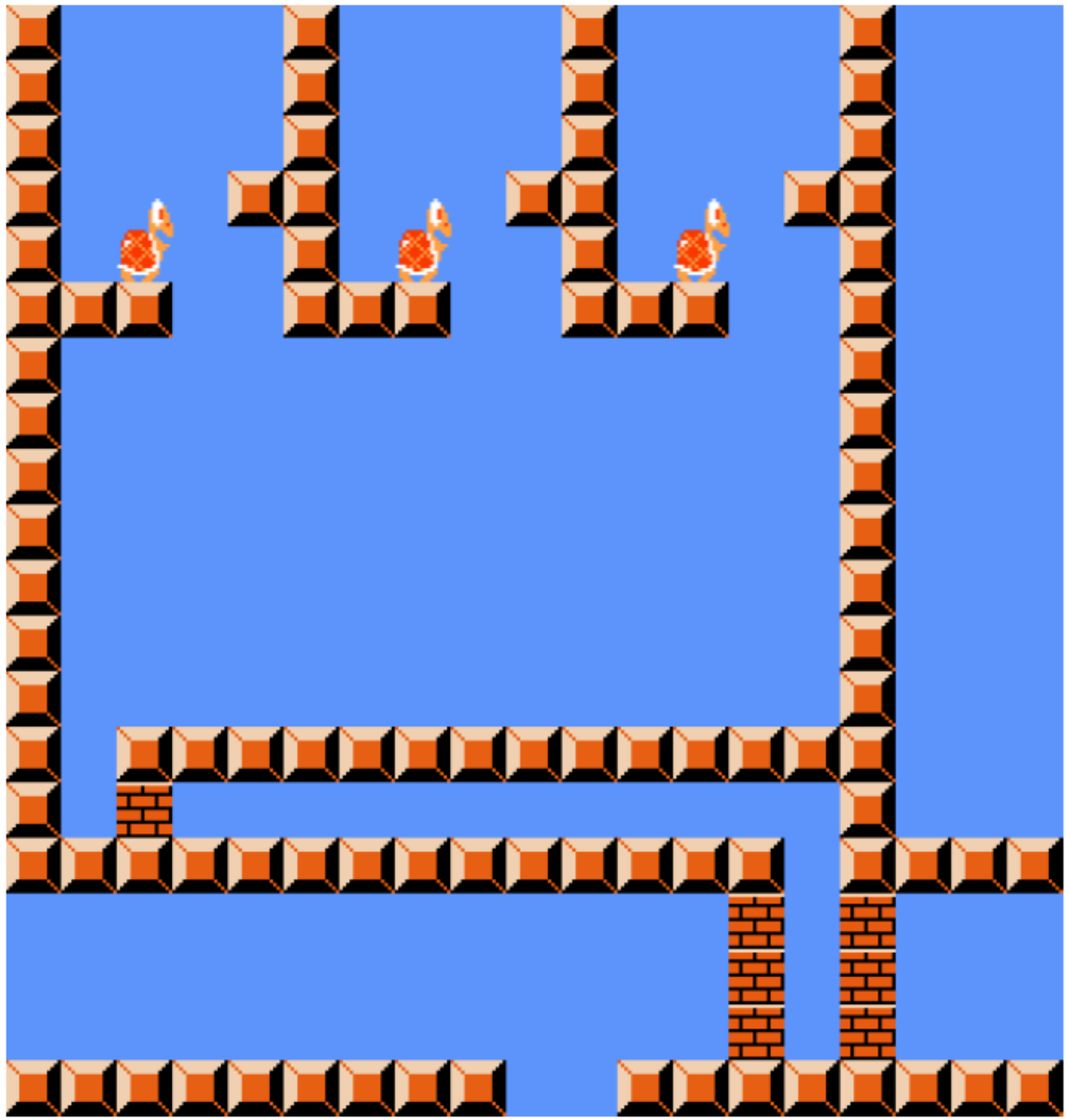


clause gadget

Variable and Clause Gadgets

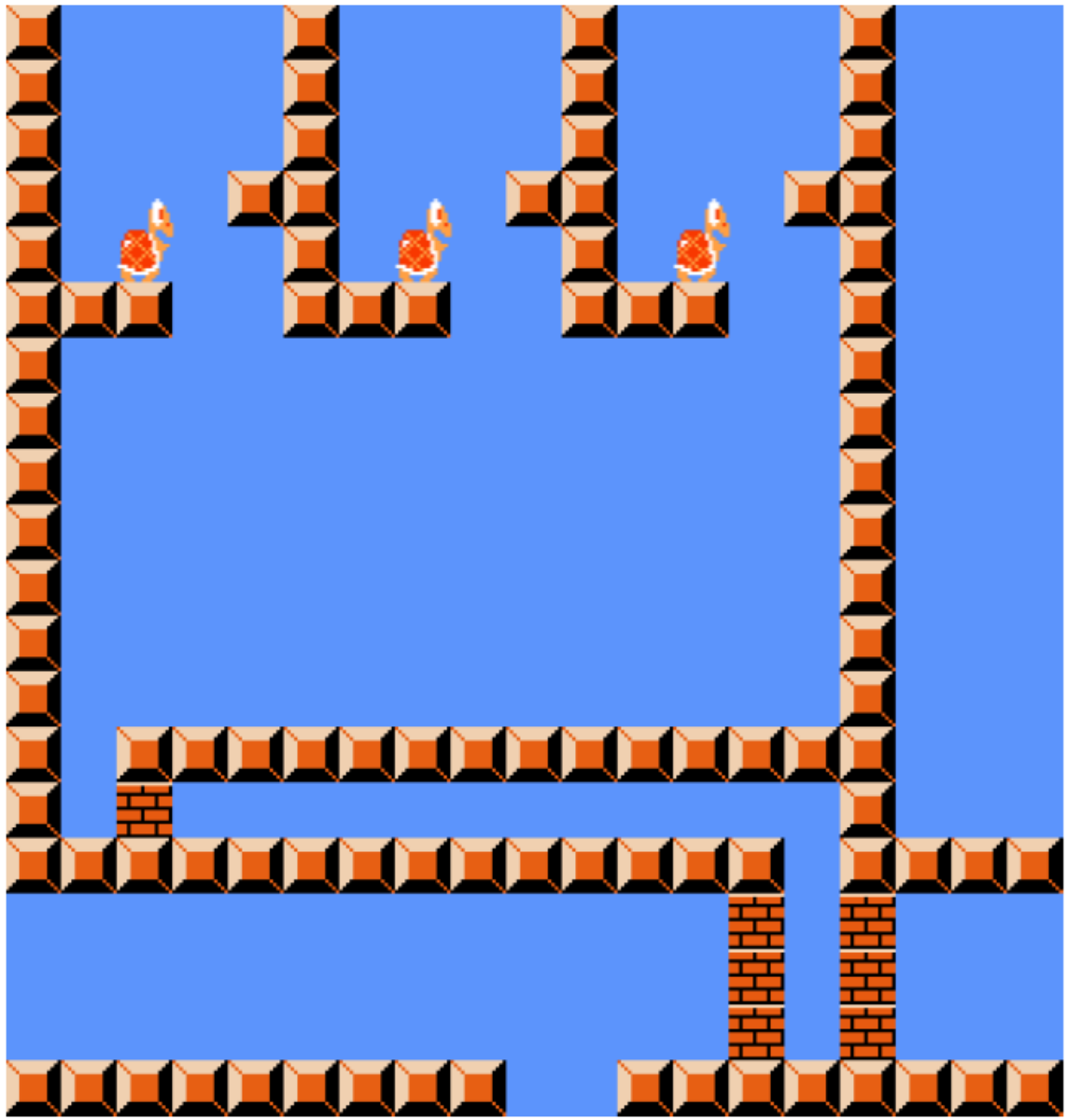


Variable and Clause Gadgets



Red Turtles don't fall off cliffs

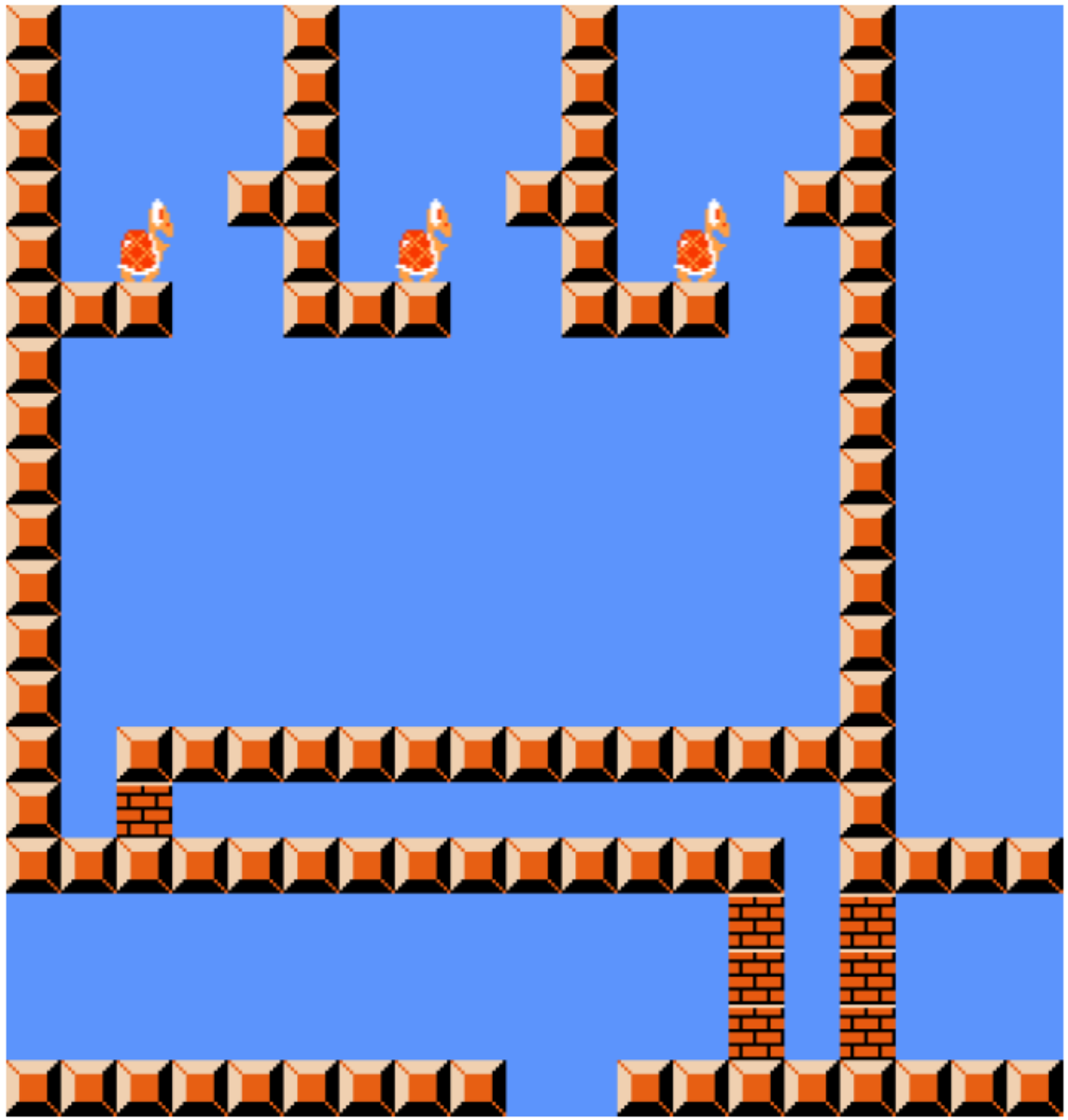
Variable and Clause Gadgets



Red Turtles don't fall off cliffs

Jumping on Turtles turns them into shells

Variable and Clause Gadgets

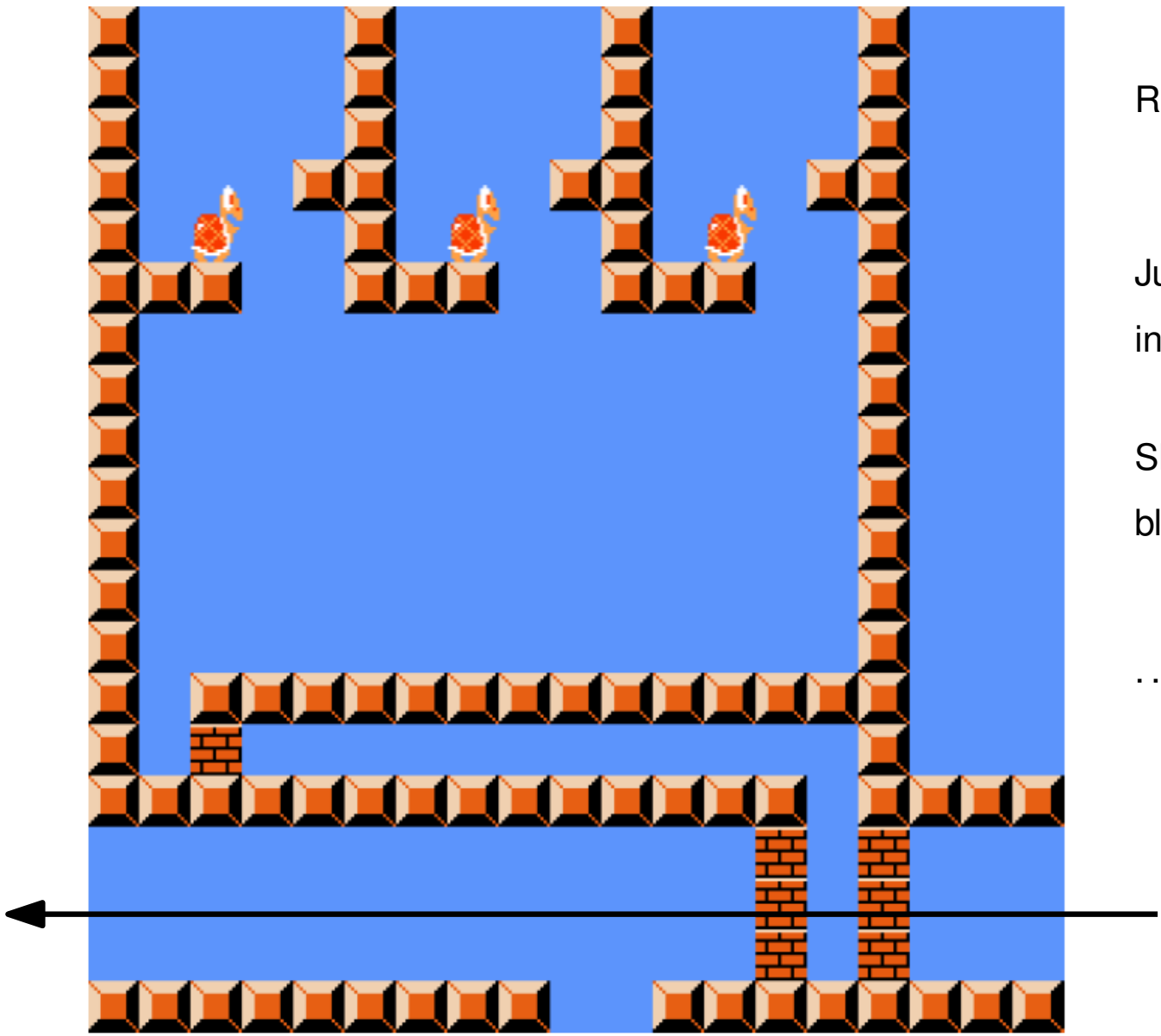


Red Turtles don't fall off cliffs

Jumping on Turtles turns them into shells

Shells fall off cliffs and break blocks

Variable and Clause Gadgets



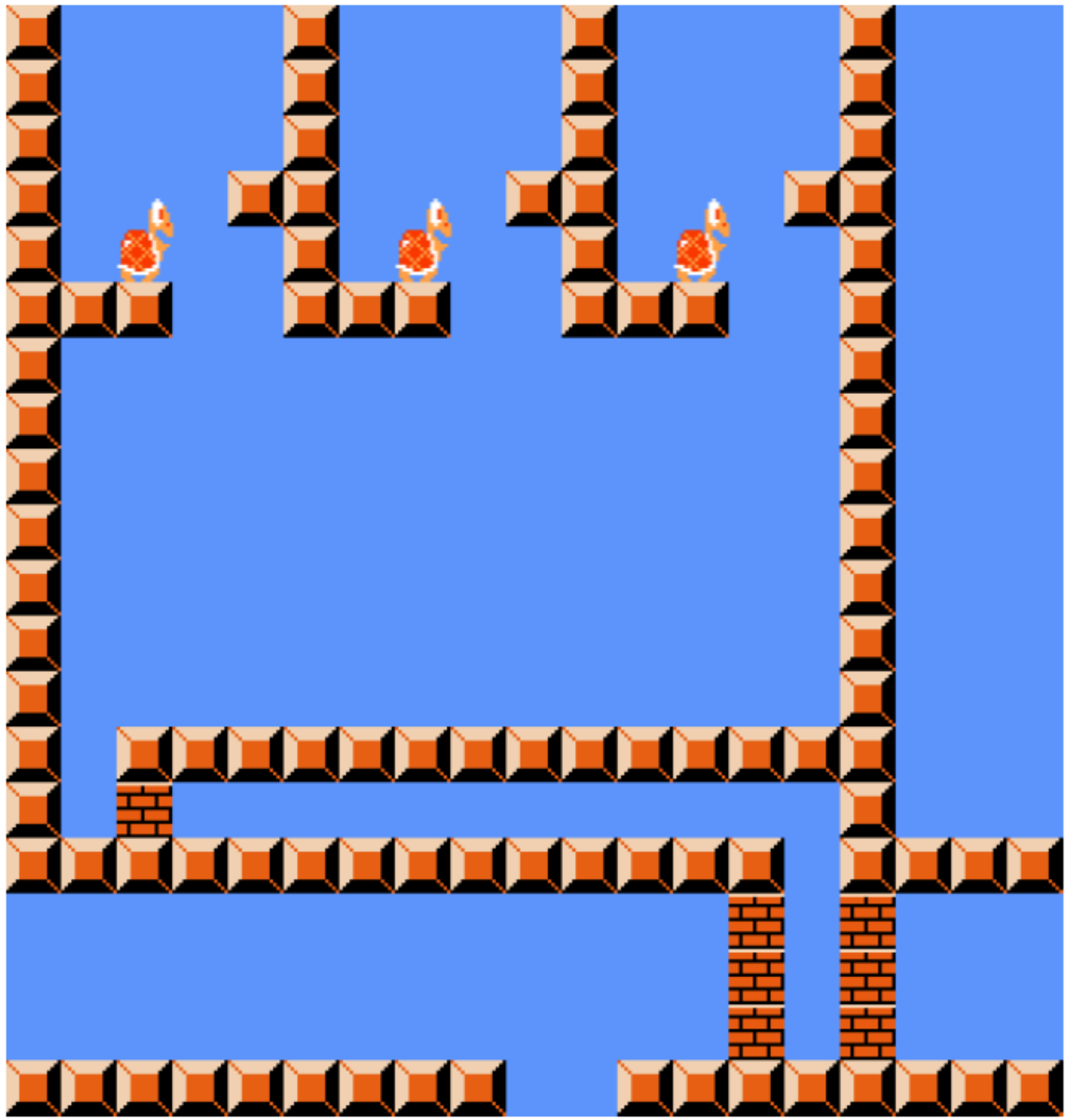
Red Turtles don't fall off cliffs

Jumping on Turtles turns them into shells

Shells fall off cliffs and break blocks

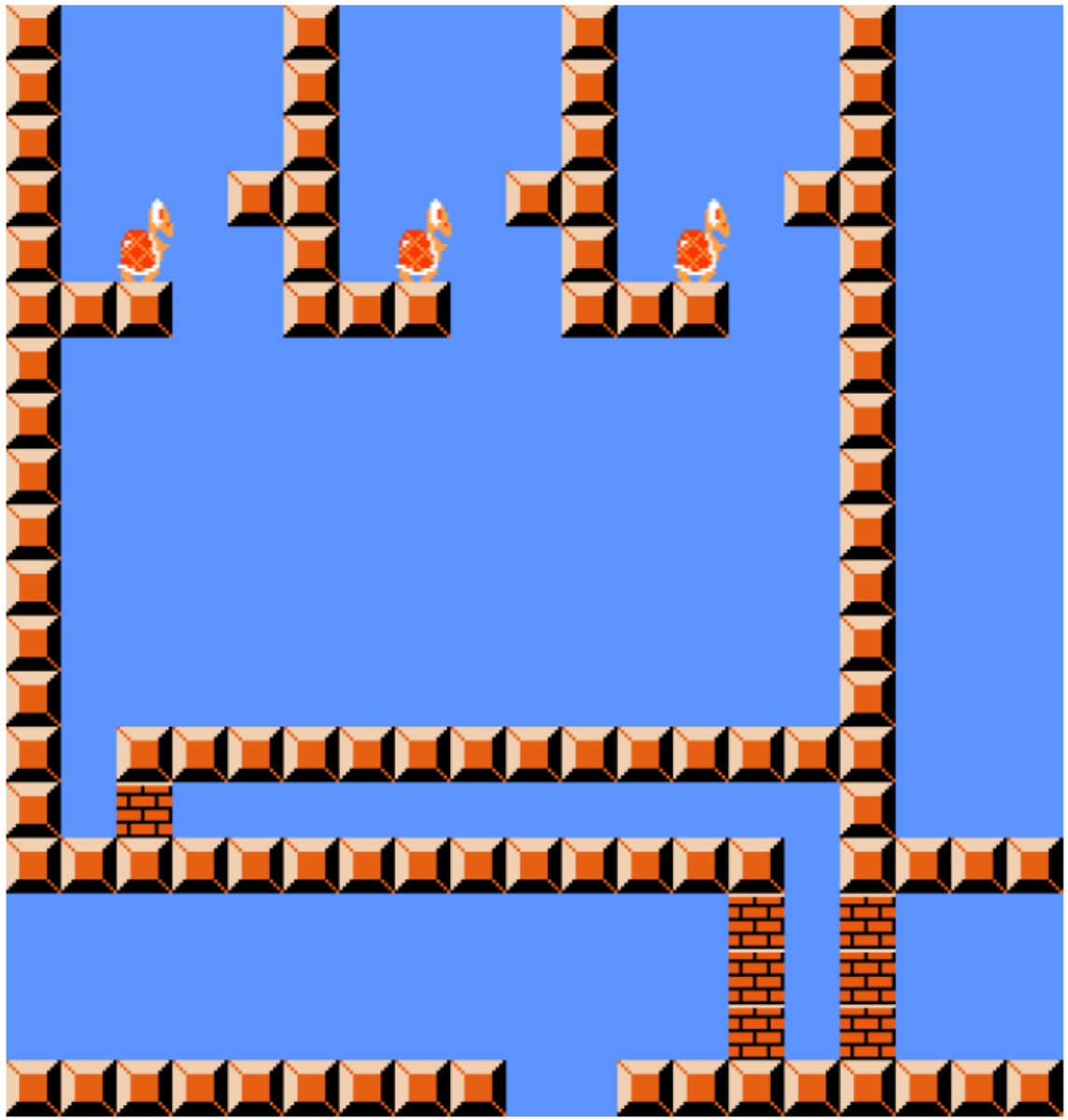
. . . opening the bottom path

Variable and Clause Gadgets



Big Mario can't get to the bottom

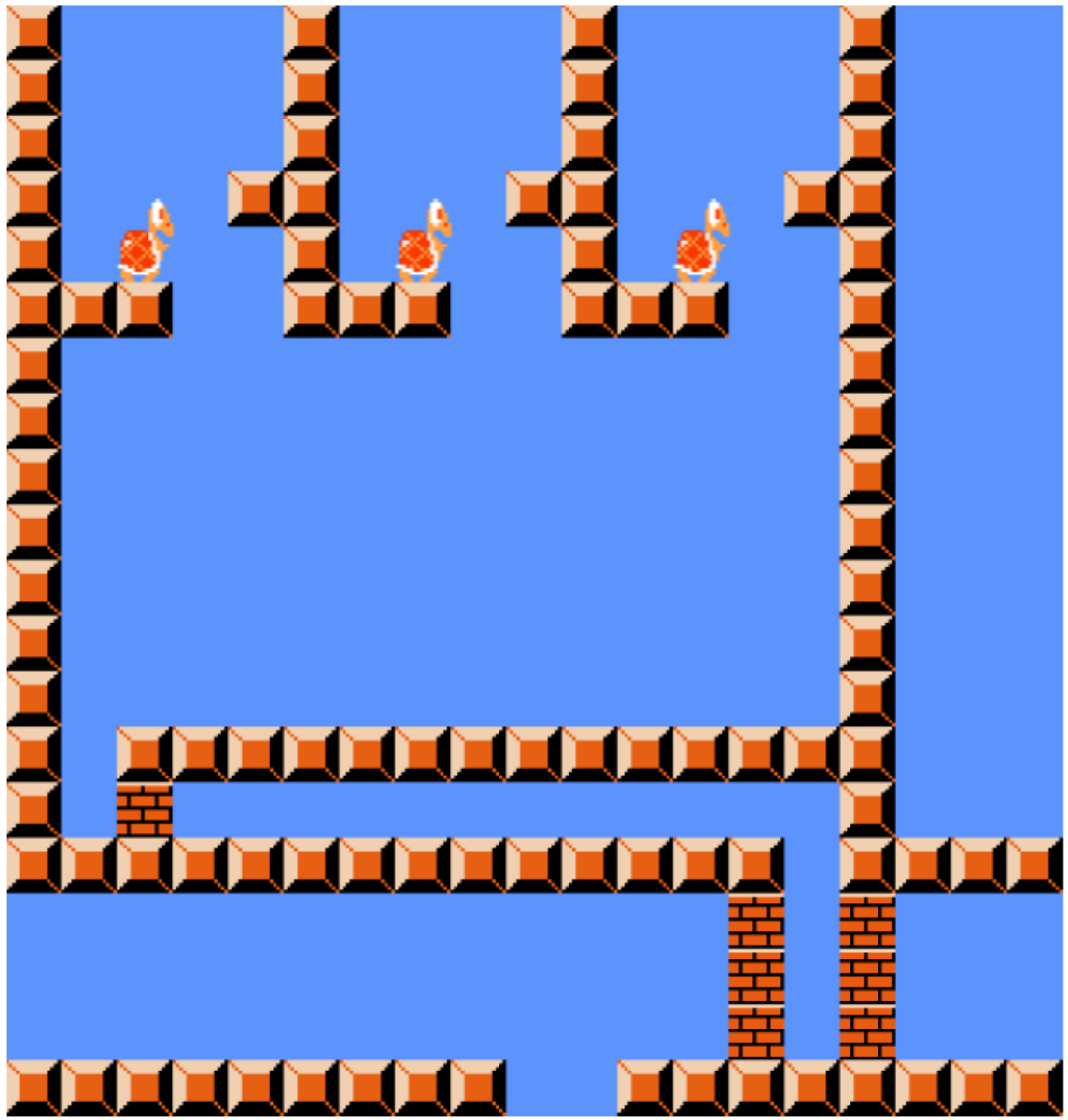
Variable and Clause Gadgets



Big Mario can't get to the bottom

Small Mario can't win the level

Variable and Clause Gadgets

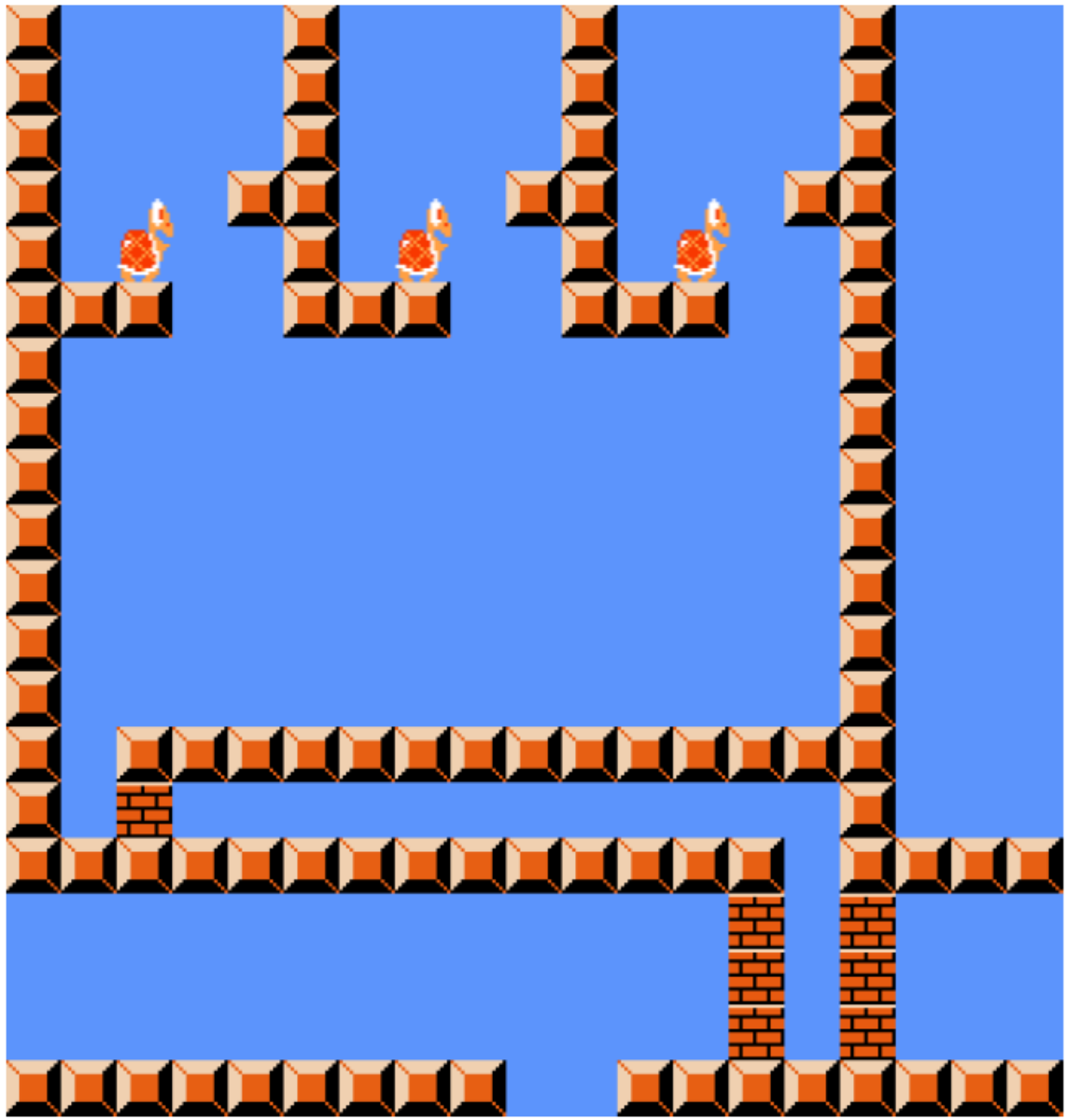


Big Mario can't get to the bottom

Small Mario can't win the level

(There is only one mushroom)

Variable and Clause Gadgets



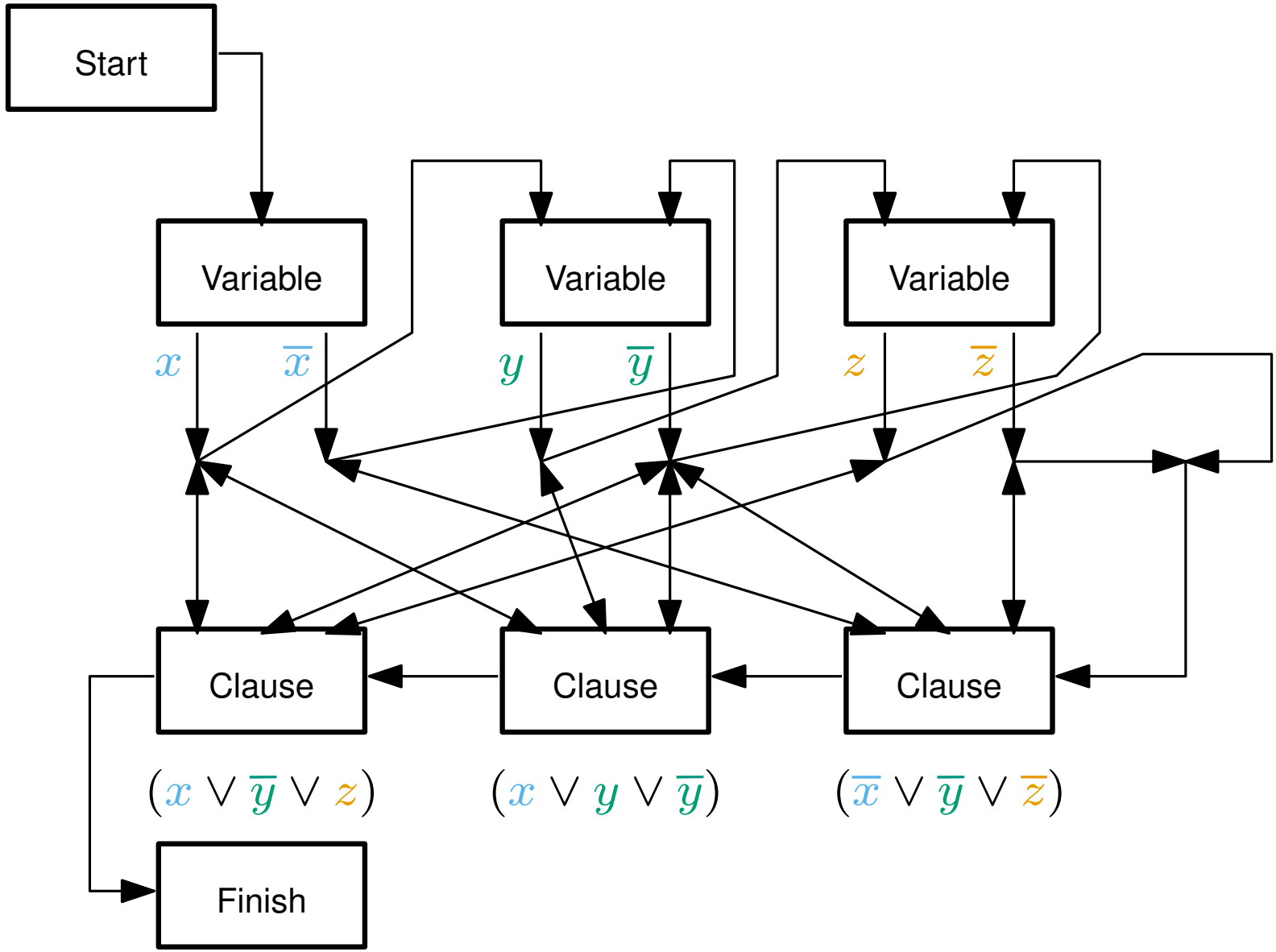
Big Mario can't get to the bottom

Small Mario can't win the level

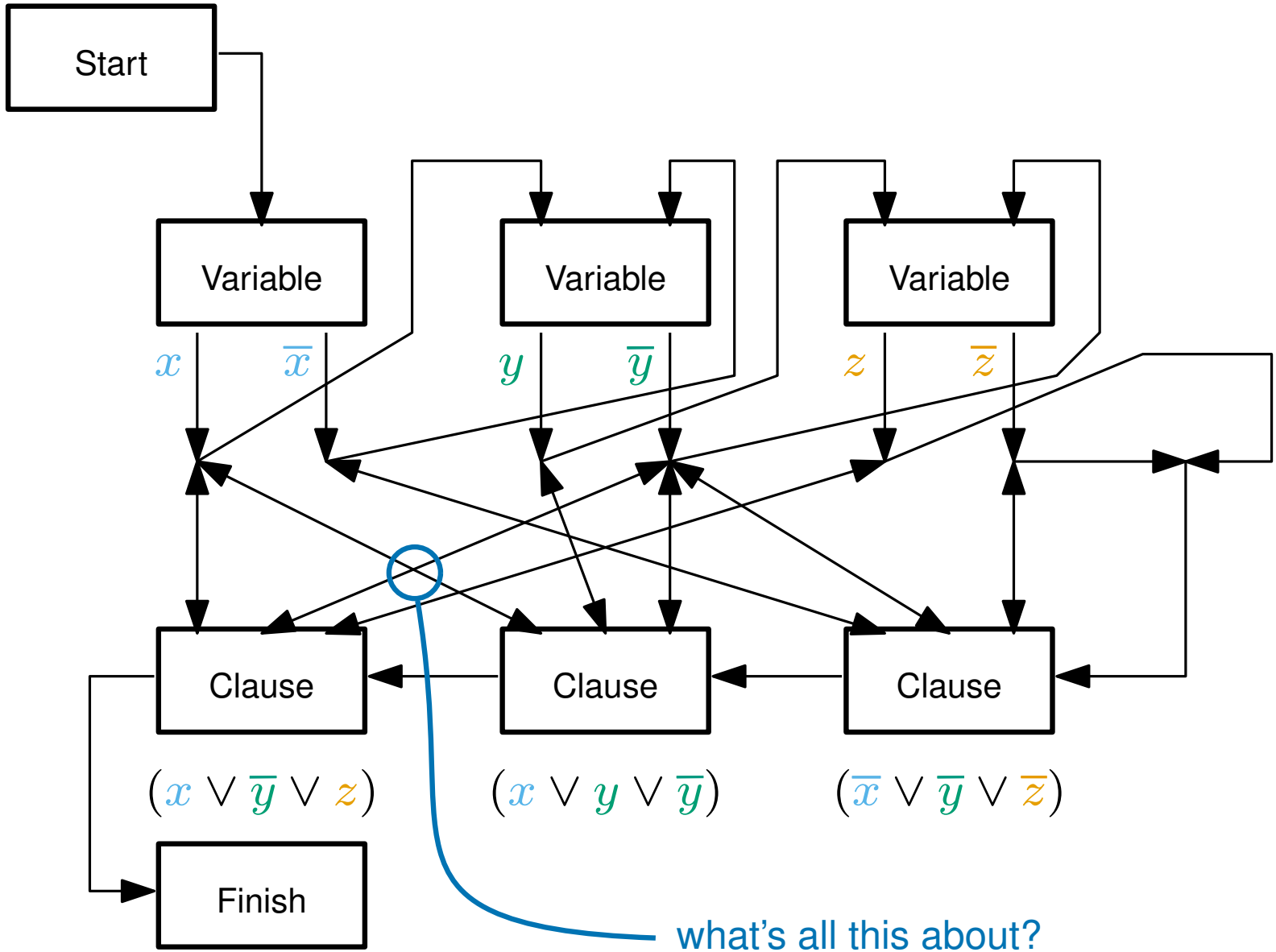
(There is only one mushroom)

The only (sensible) way out is back up where he came from

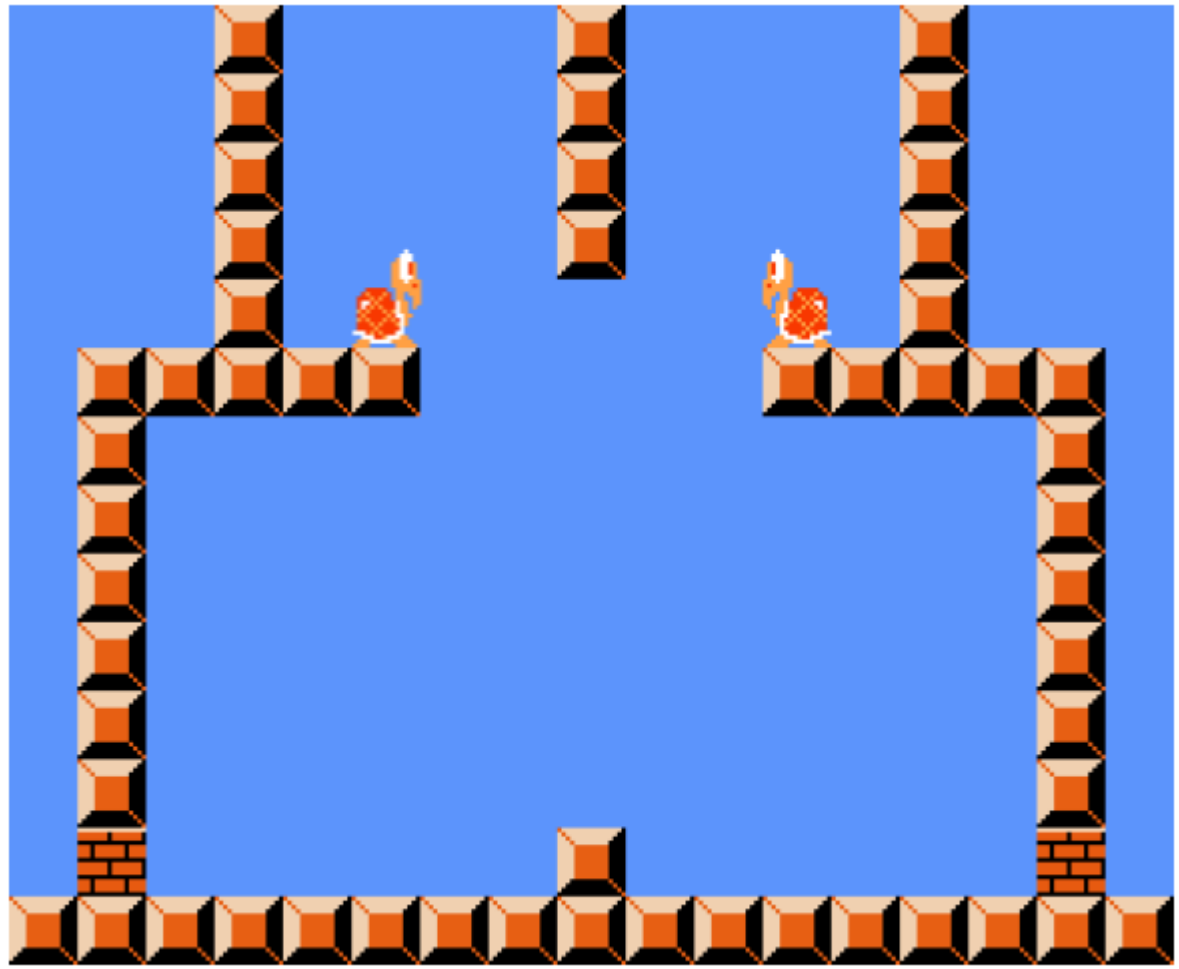
Embedding Overview



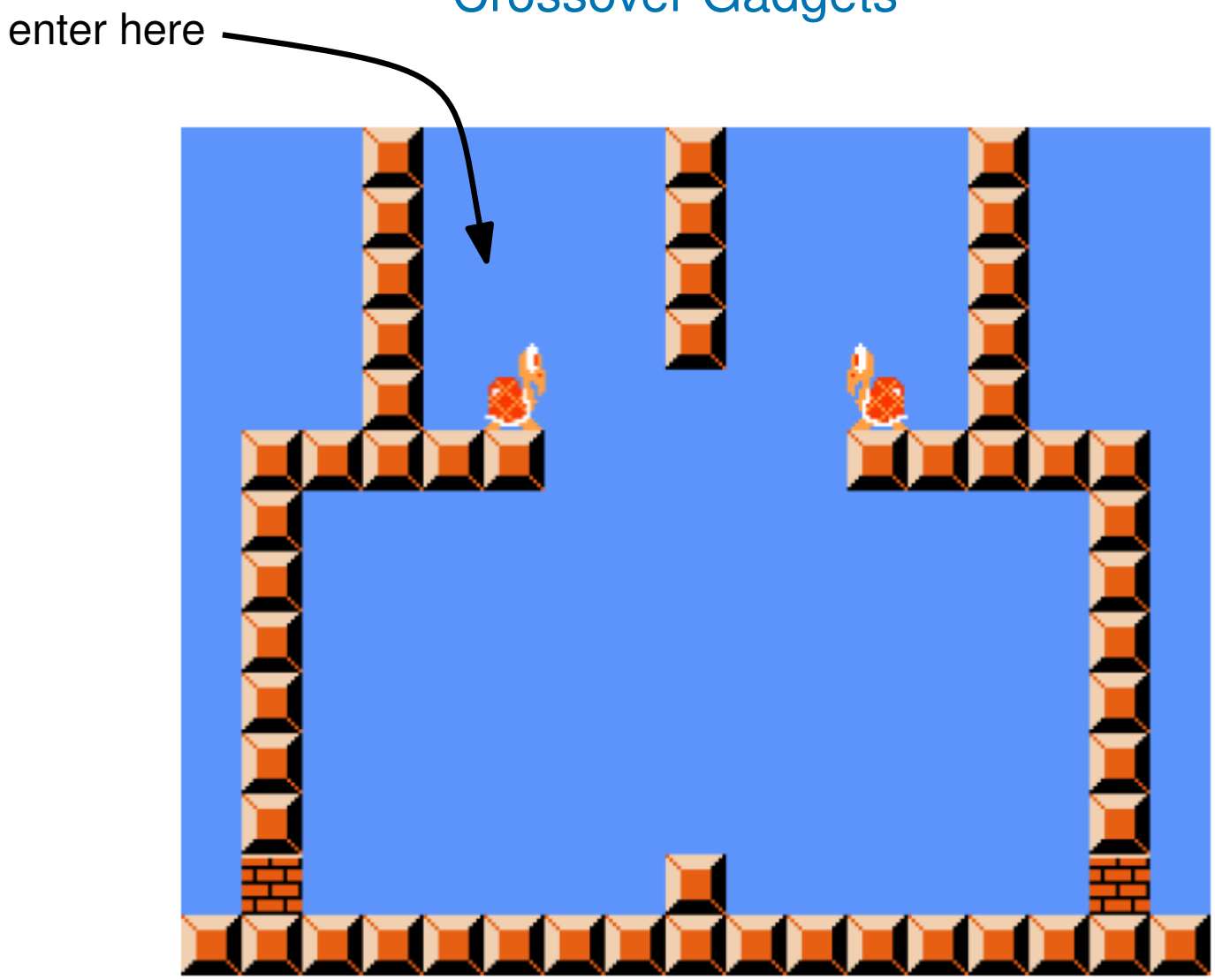
Embedding Overview



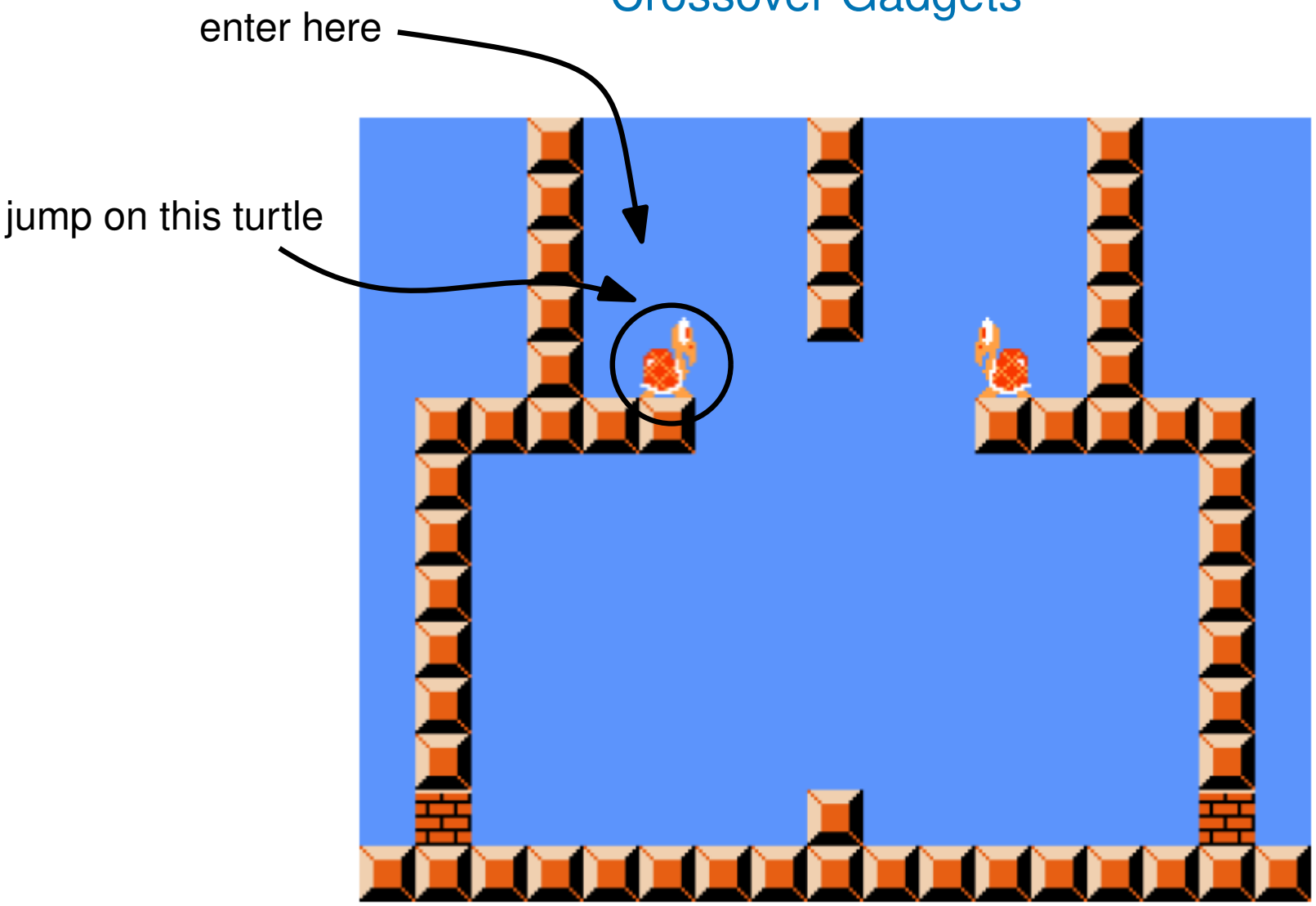
Crossover Gadgets



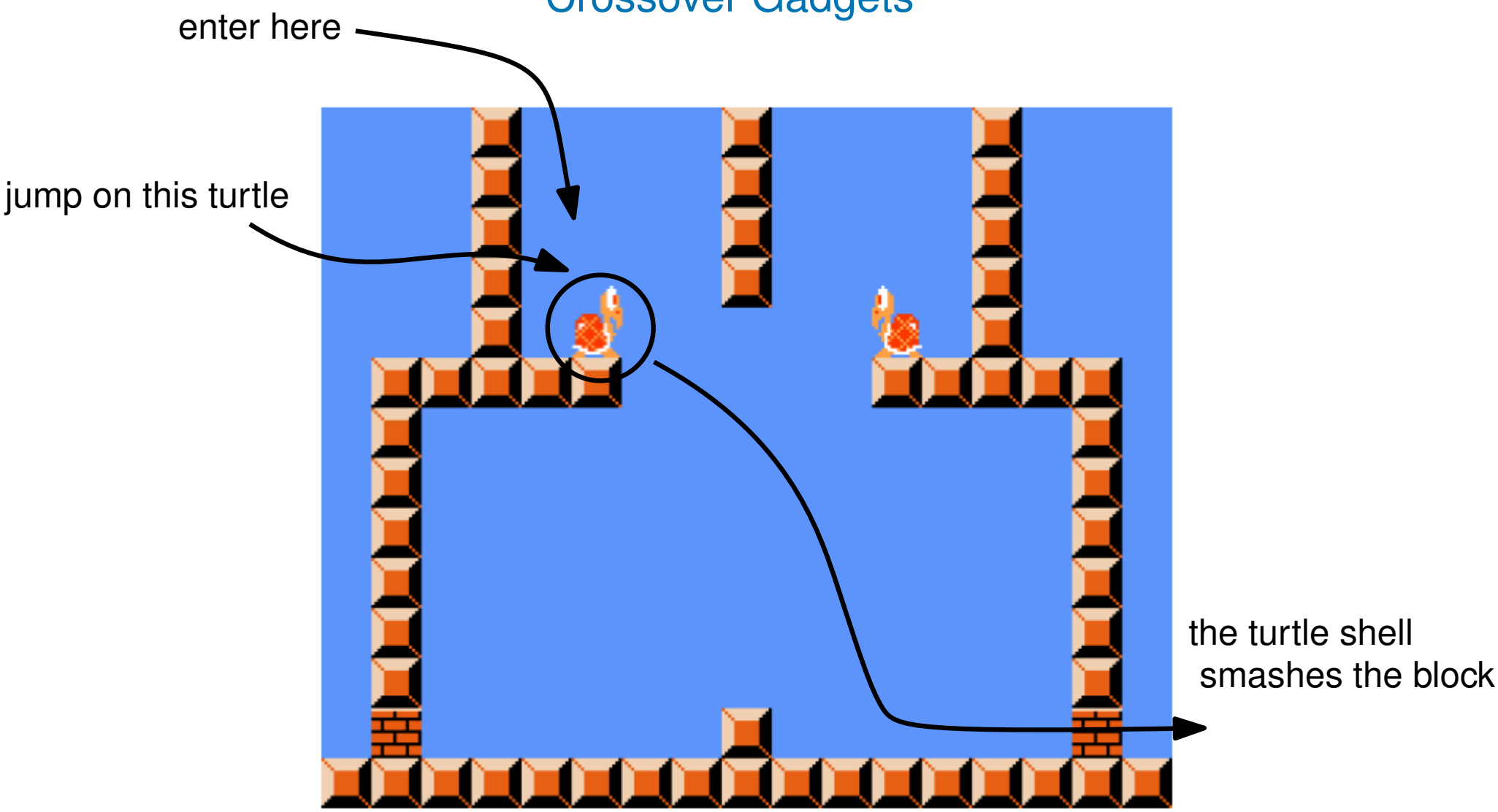
Crossover Gadgets



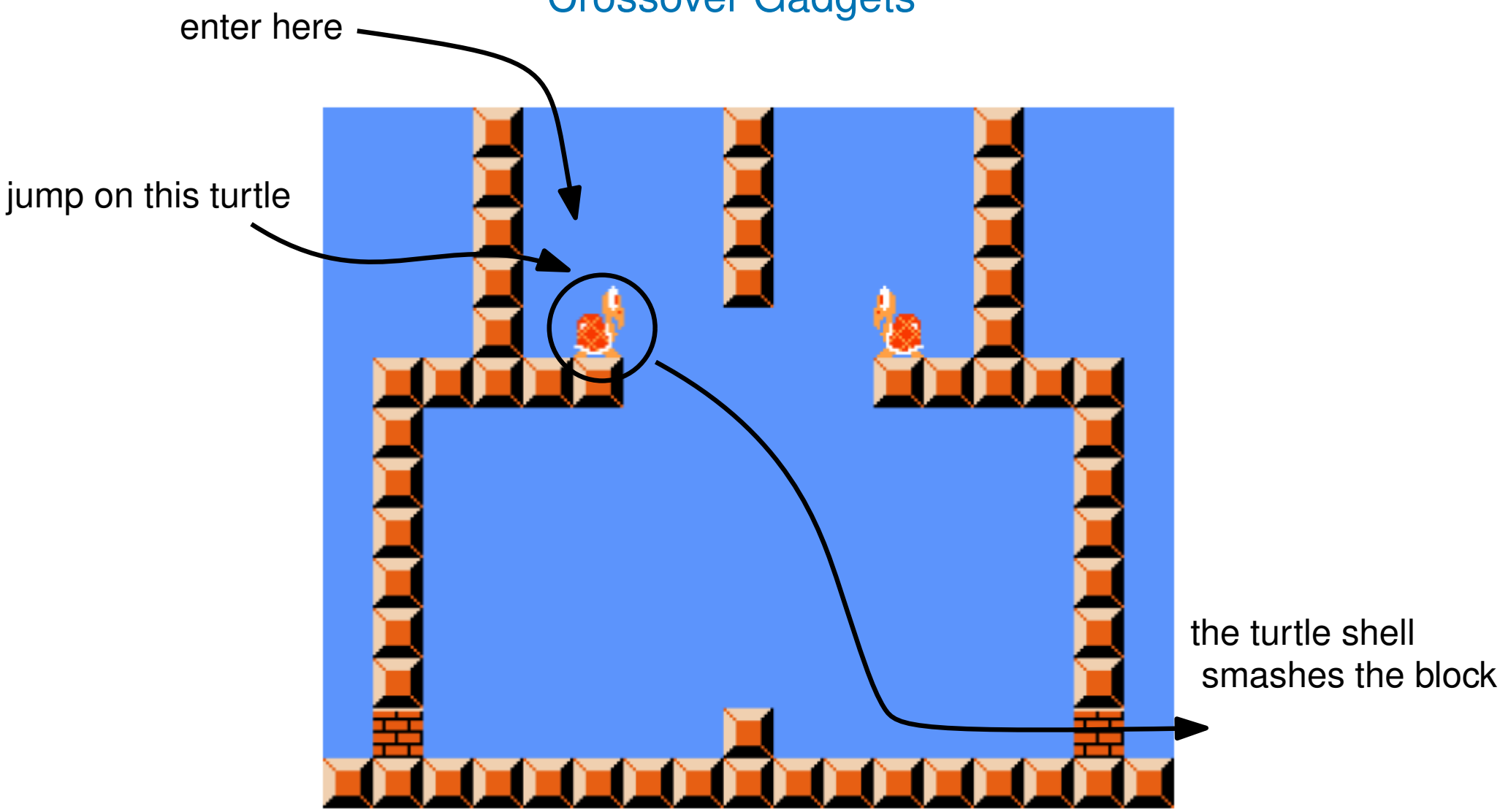
Crossover Gadgets



Crossover Gadgets



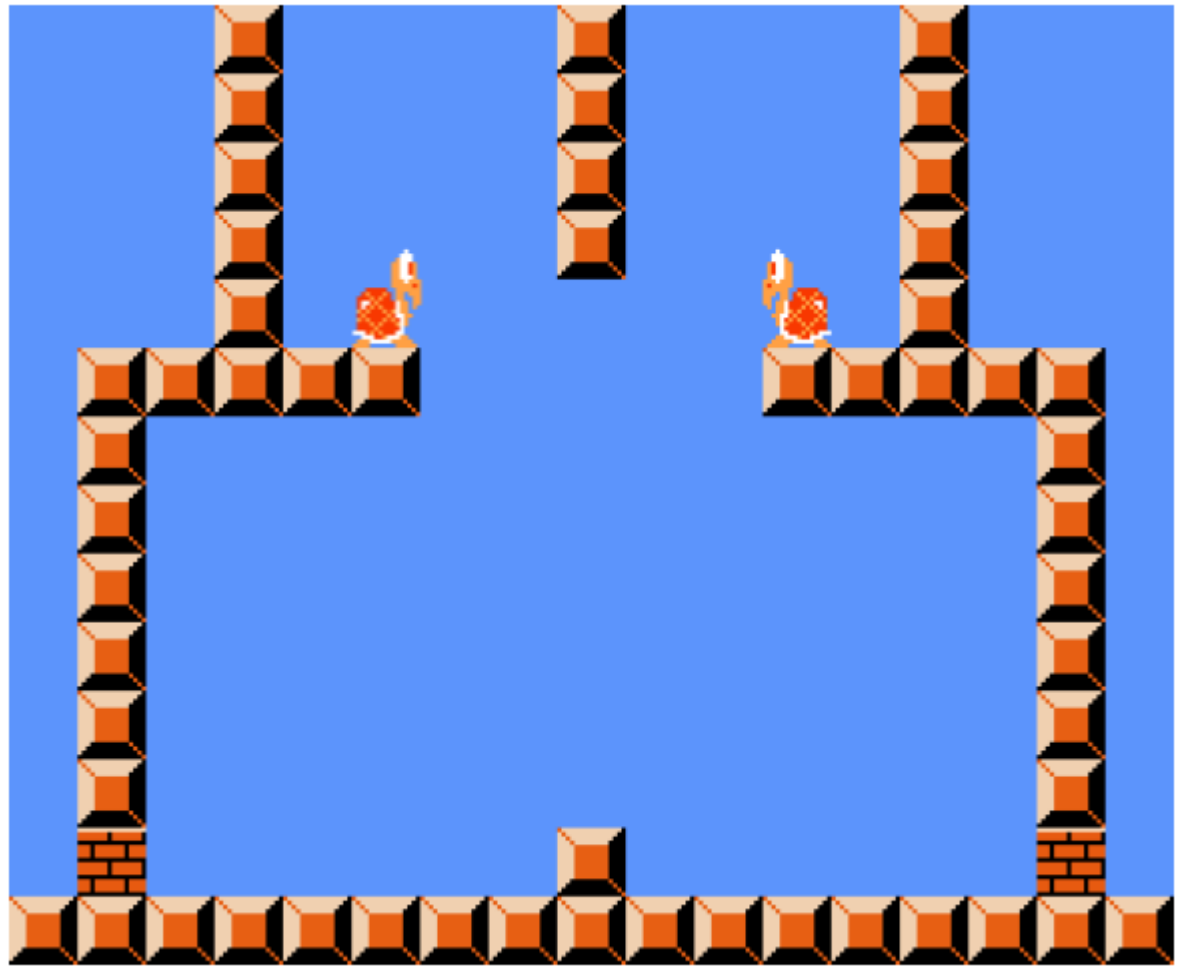
Crossover Gadgets



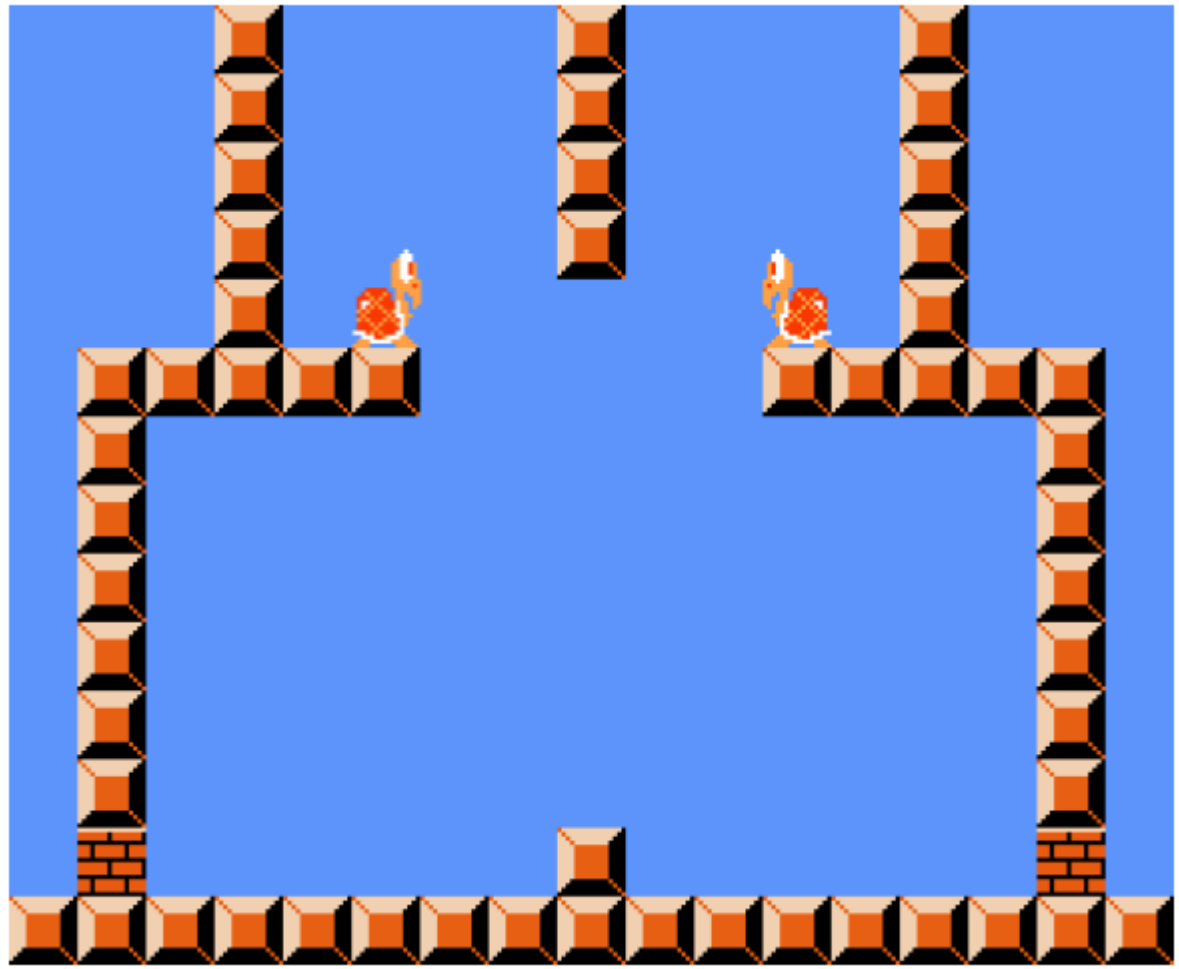
Mario can then ‘crouch-slide’ through the small gap

(but he can’t make the turtle smash the left block)

Crossover Gadgets

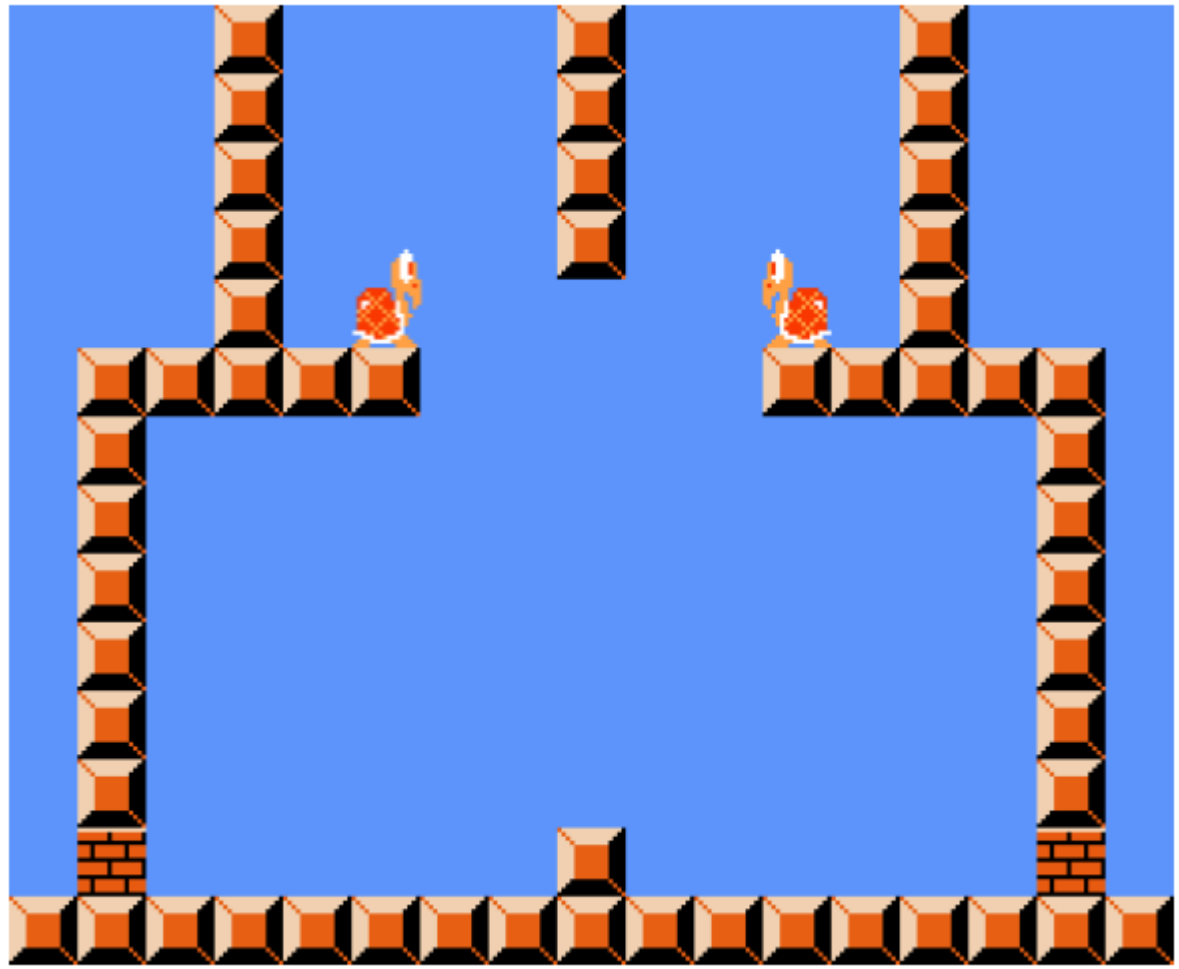


Crossover Gadgets



Super Mario Bros. is NP-Hard in an $n \times n$ world

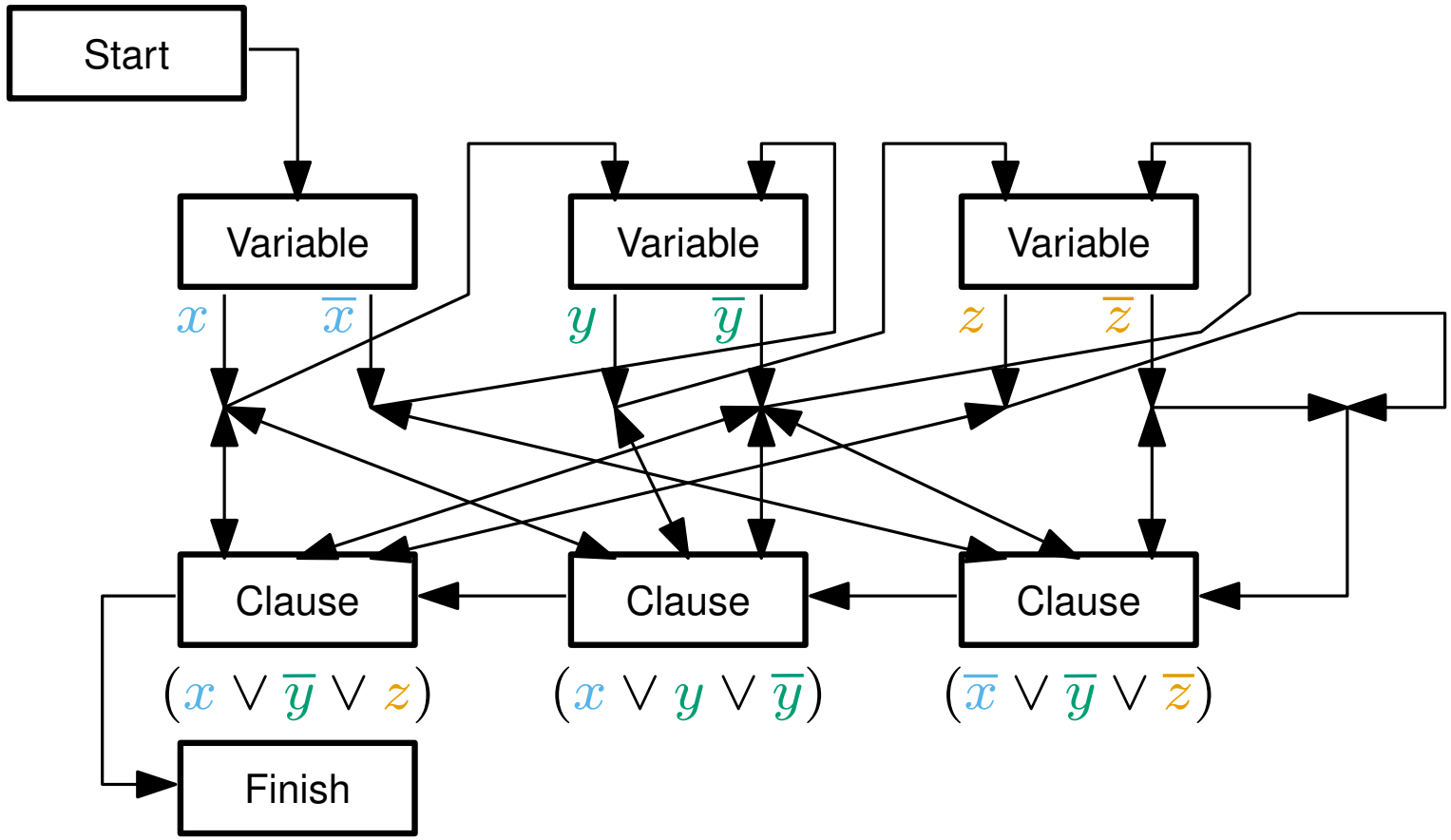
Crossover Gadgets



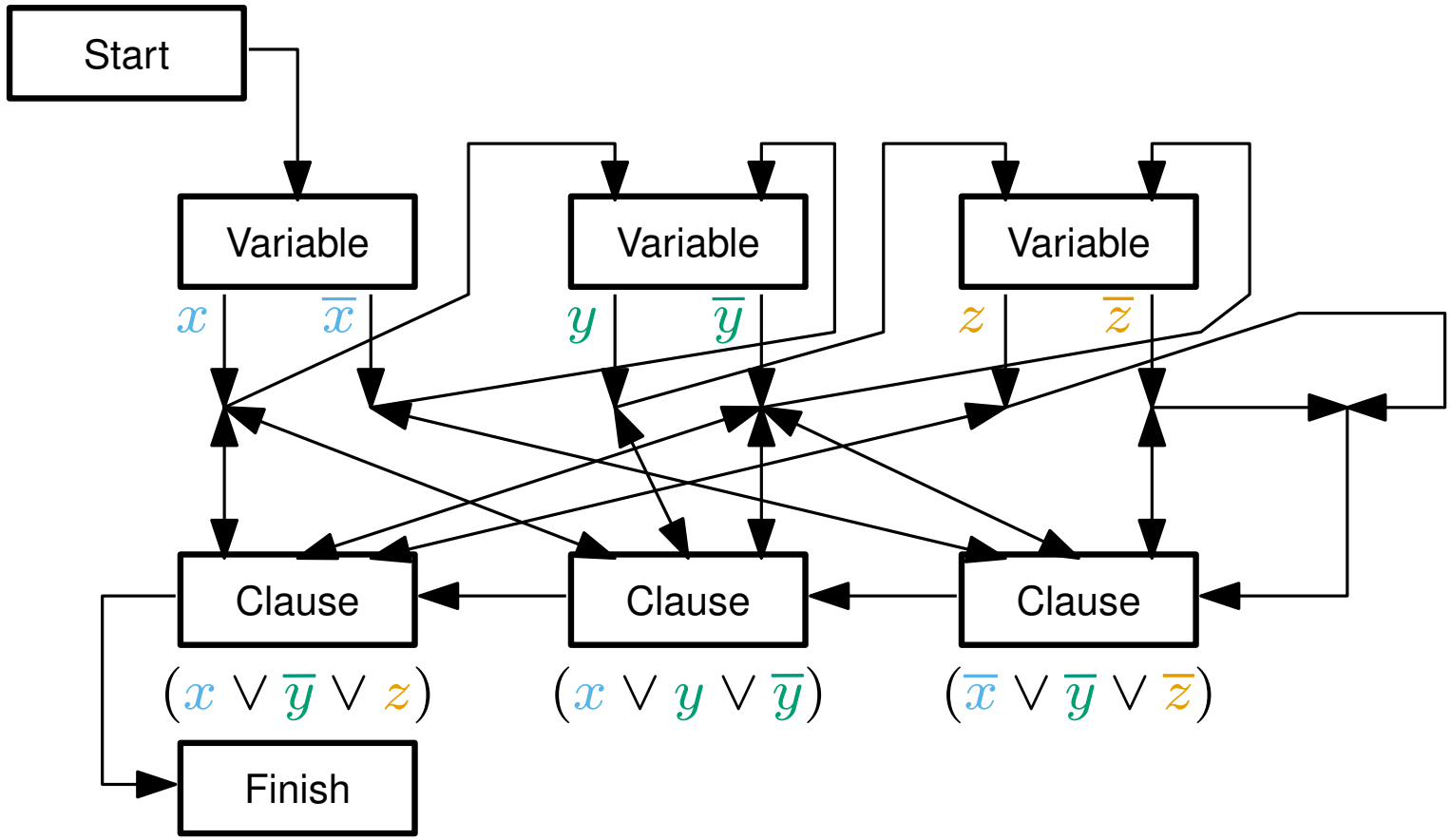
Super Mario Bros. is NP-Hard in an $n \times n$ world

. . . what about constant height worlds?

Embedding Overview (again) (again)

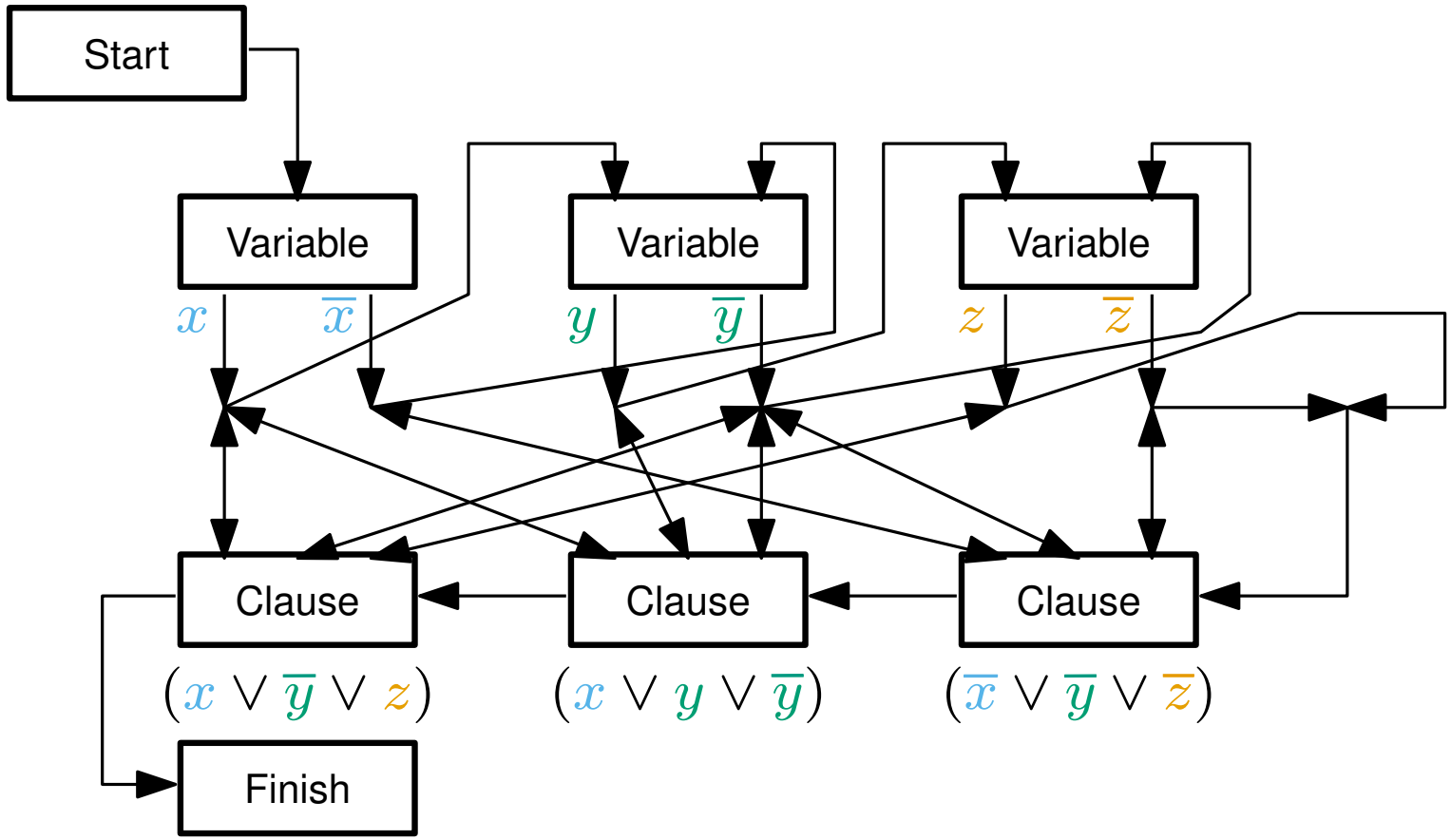


Embedding Overview (again) (again)



How can we embed this in a $n \times h$ *world?* (with small h)

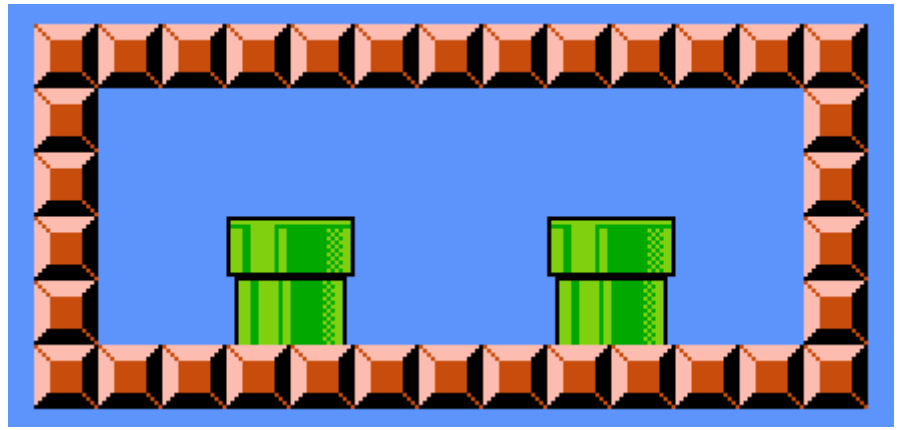
Embedding Overview (again) (again)



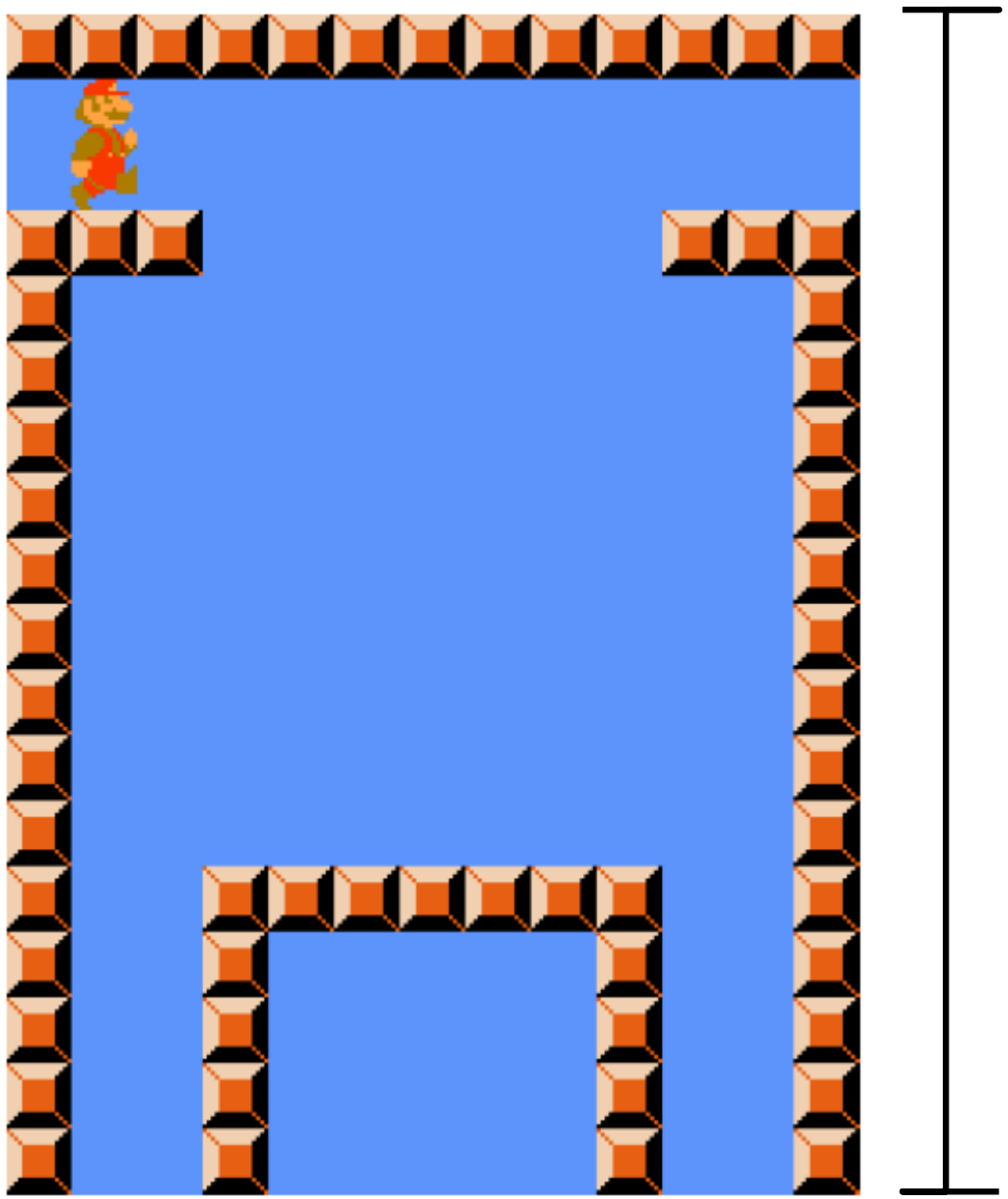
How can we embed this in a $n \times h$ world? (with small h)

. . . we might need lots of height for crossovers

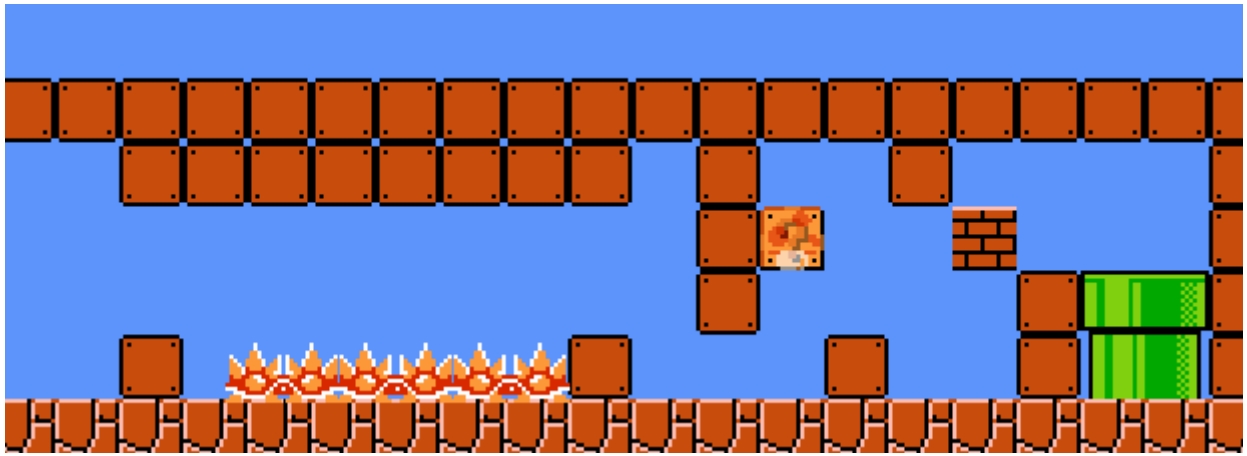
Simplified Crossovers :)

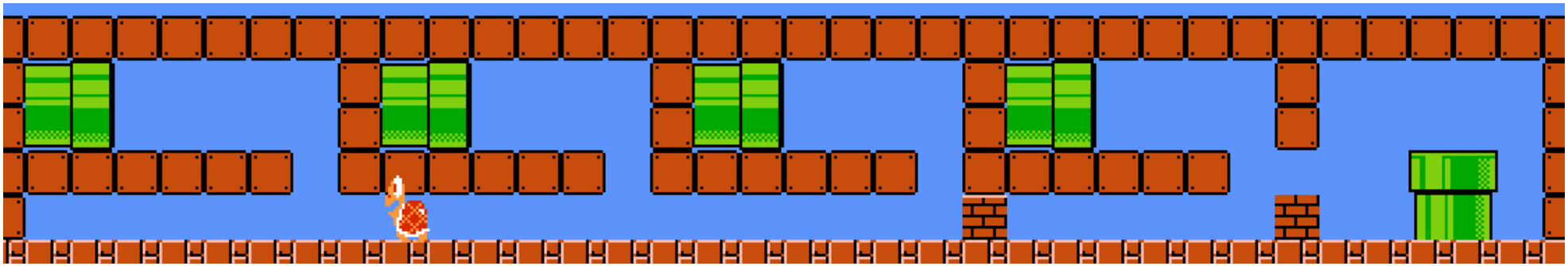


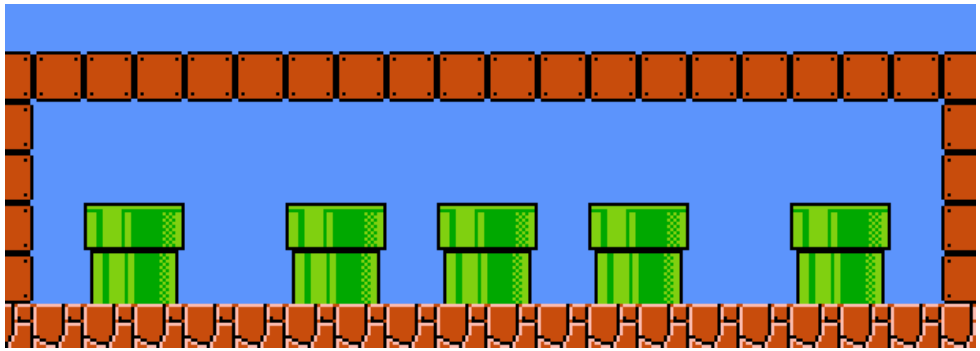
How low can you go?

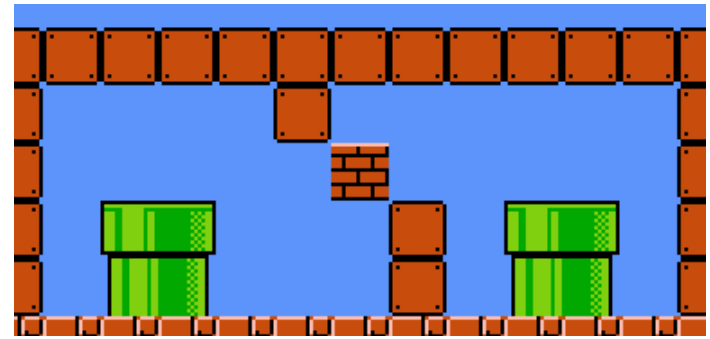


How low can you go?

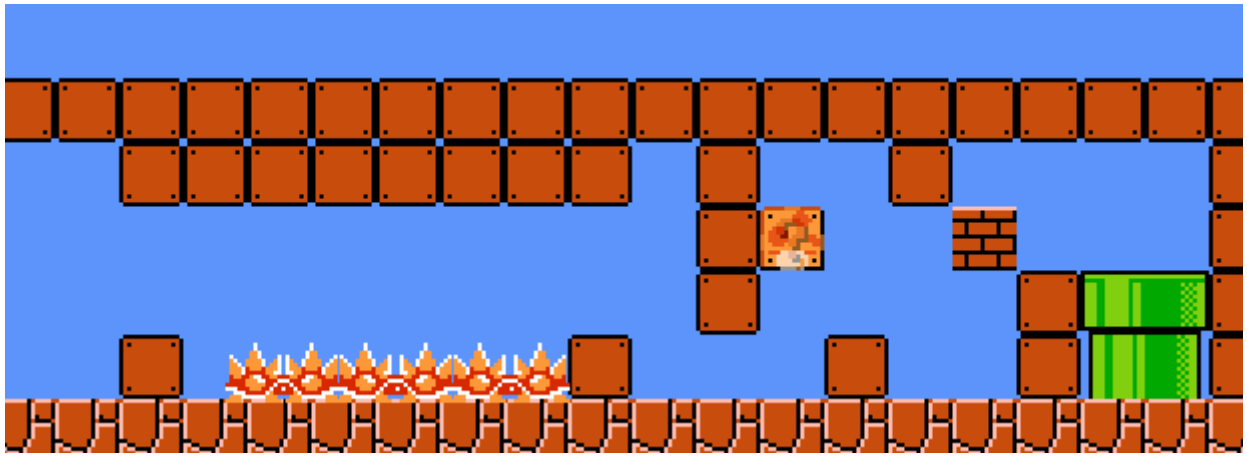








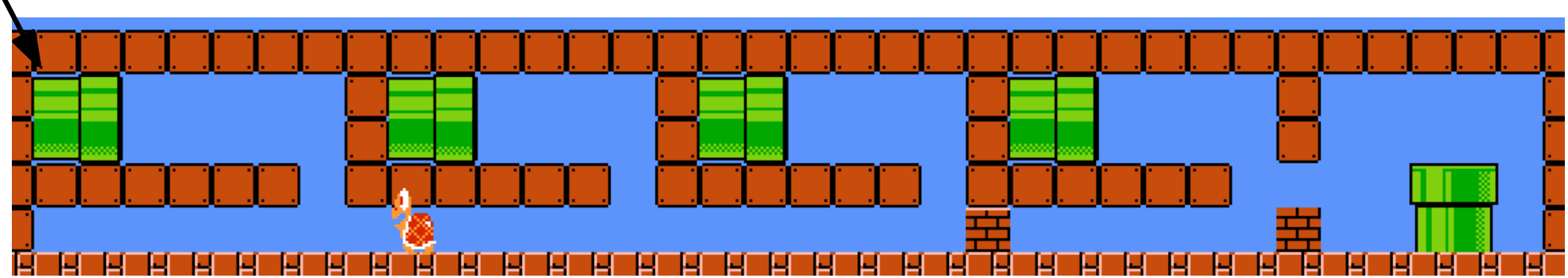
How low can you go?

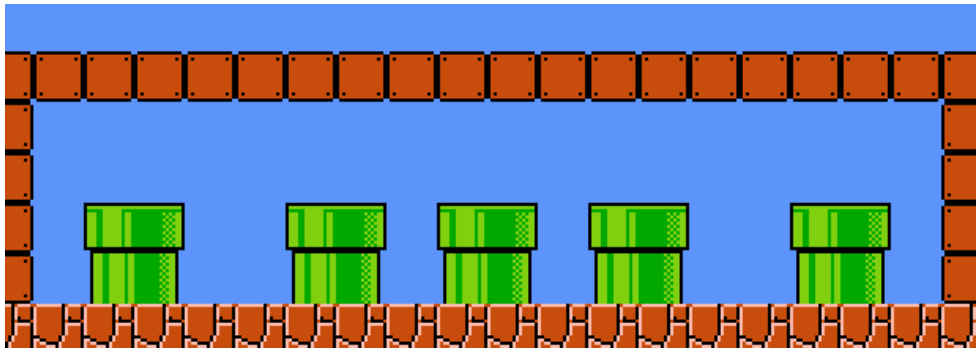


One Way

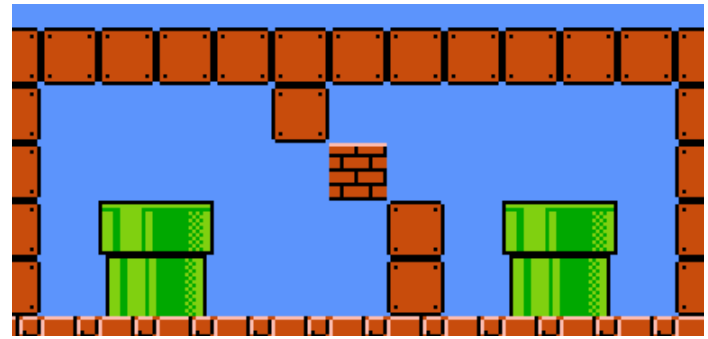
(Left to Right)

Clause



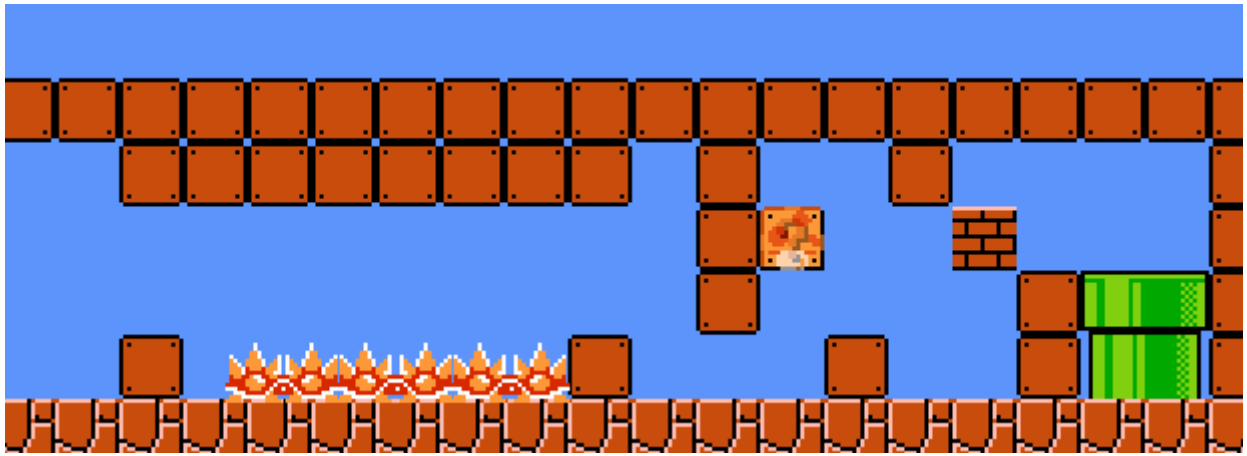


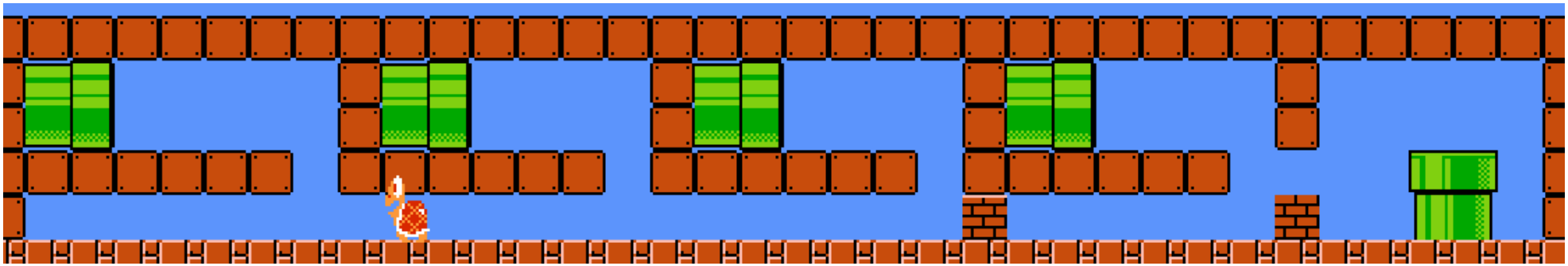
Variable linking

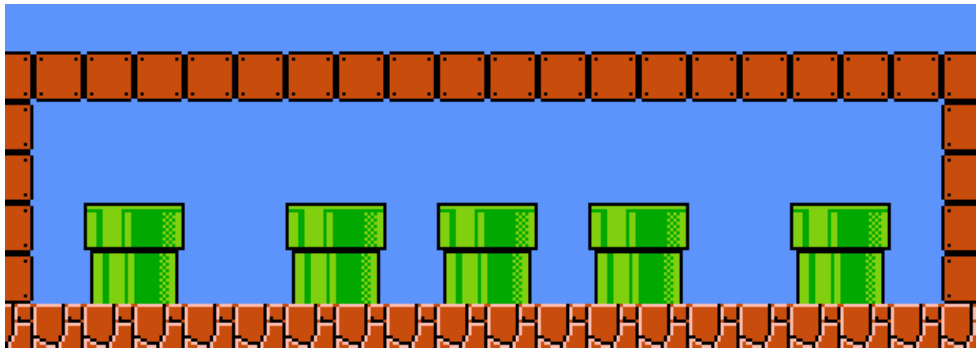


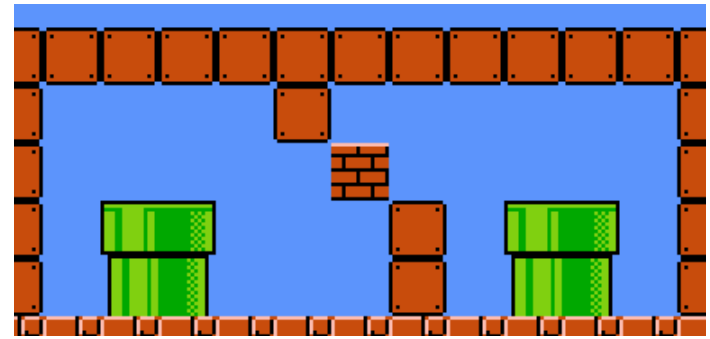
Left to Right first

How low can you go?

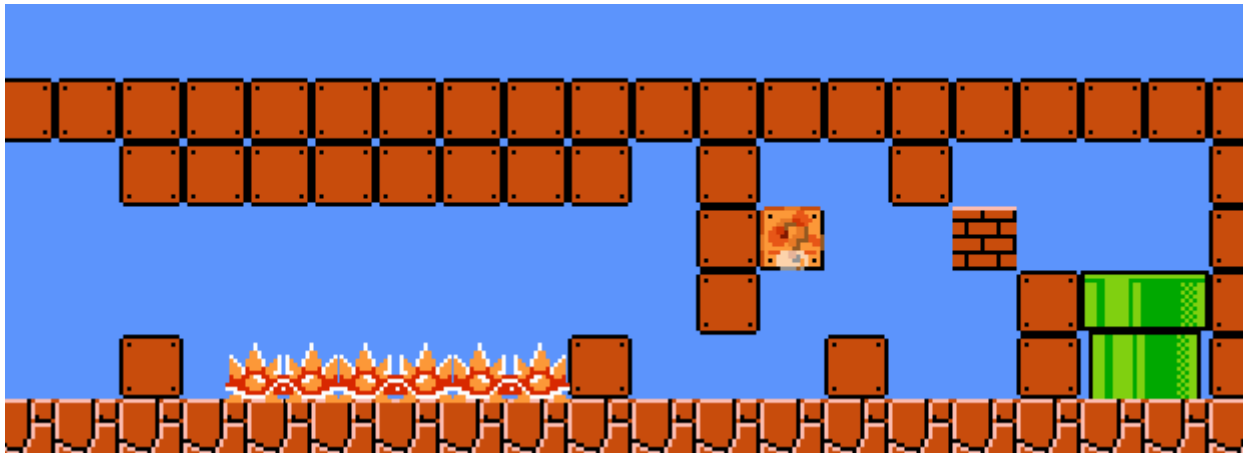


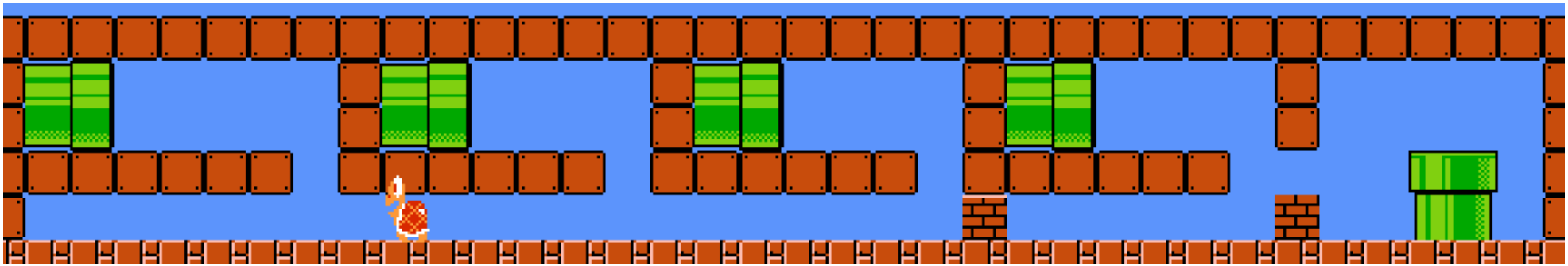


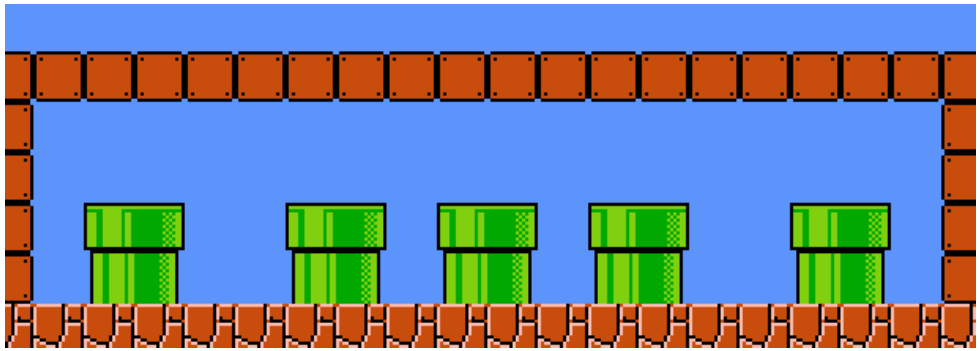


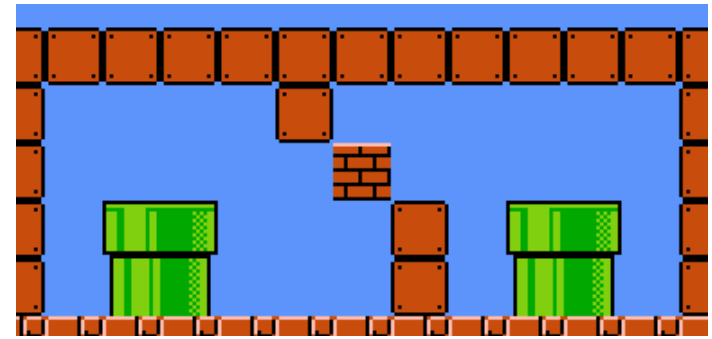


How low can you go?









Using these gadgets, you can prove that $n \times 4$ Mario is NP-hard :)

ZORK I: The Great Underground Empire Copyright (c) 1981, 1982, 1983 Infocom, Inc. All rights reserved. ZORK is a registered trademark of Infocom, Inc. Revision 88 / Serial number 840726

You are in an open field west of a big white house with a boarded front door. There is a small mailbox here.

>

ZORK I: The Great Underground Empire Copyright (c) 1981, 1982, 1983 Infocom, Inc. All rights reserved. ZORK is a registered trademark of Infocom, Inc. Revision 88 / Serial number 840726

You are in an open field west of a big white house with a boarded front door. There is a small mailbox here.

> go north

ZORK I: The Great Underground Empire Copyright (c) 1981, 1982, 1983 Infocom, Inc. All rights reserved. ZORK is a registered trademark of Infocom, Inc. Revision 88 / Serial number 840726

You are in an open field west of a big white house with a boarded front door. There is a small mailbox here.

> go north

You are outside a large, foreboding cave. Next to the entrance is a strange little man in a long cloak.

>

ZORK I: The Great Underground Empire Copyright (c) 1981, 1982, 1983 Infocom, Inc. All rights reserved. ZORK is a registered trademark of Infocom, Inc. Revision 88 / Serial number 840726

You are in an open field west of a big white house with a boarded front door. There is a small mailbox here.

> go north

You are outside a large, foreboding cave. Next to the entrance is a strange little man in a long cloak.

> enter cave

ZORK I: The Great Underground Empire Copyright (c) 1981, 1982, 1983 Infocom, Inc. All rights reserved. ZORK is a registered trademark of Infocom, Inc. Revision 88 / Serial number 840726

You are in an open field west of a big white house with a boarded front door. There is a small mailbox here.

> go north

You are outside a large, foreboding cave. Next to the entrance is a strange little man in a long cloak.

> enter cave

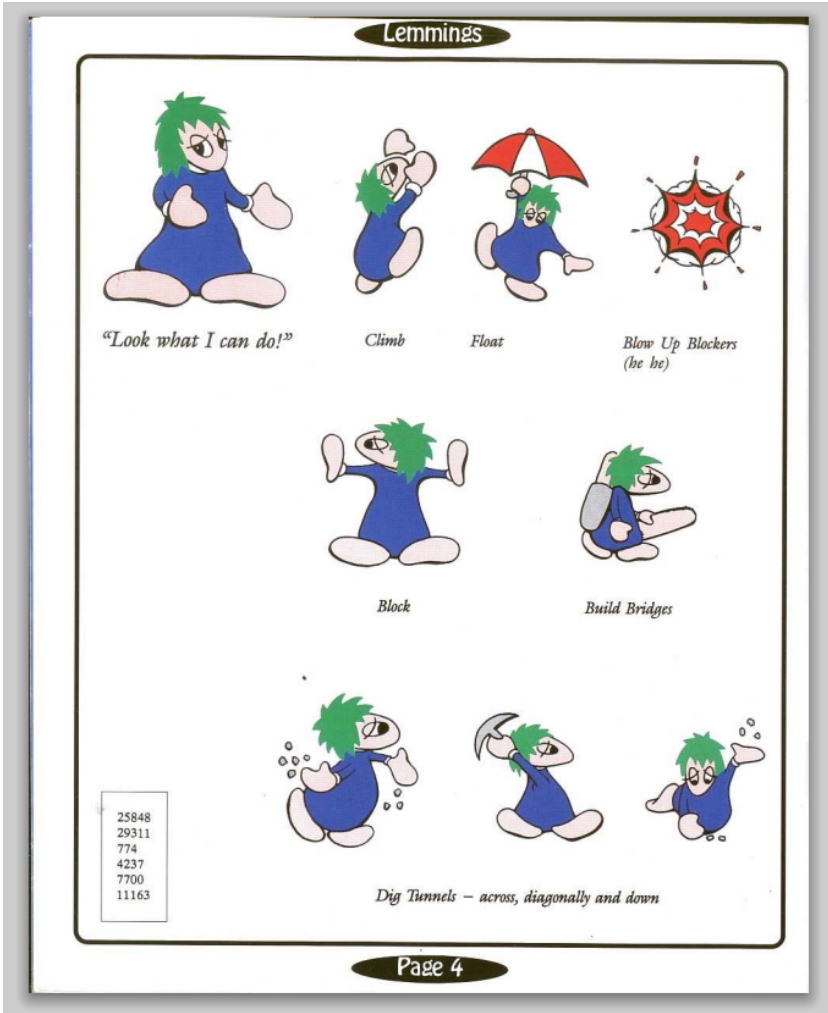
The man stops you and declares:

‘‘I am a wizard and you shall not pass until you give me a satisfying assignment for the following boolean formula...

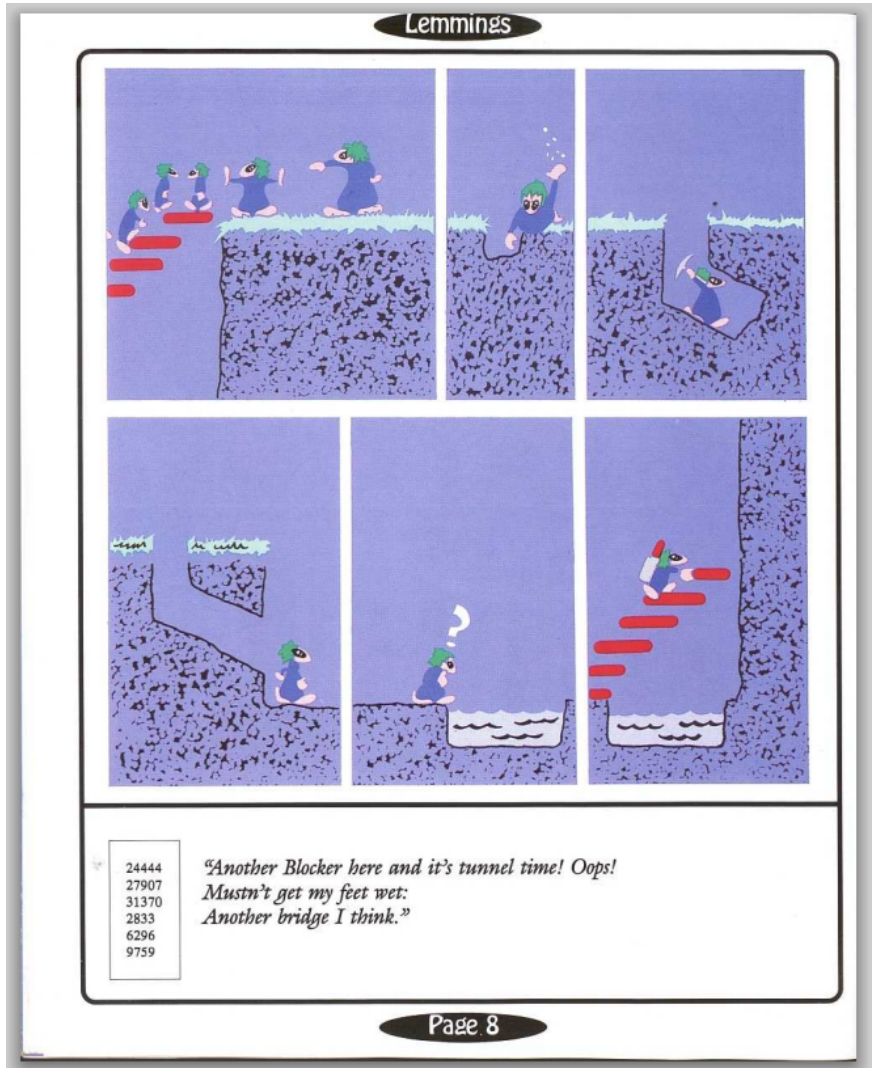
(x OR y OR z) AND (X OR y OR Z) AND (y OR Y OR Z)’’

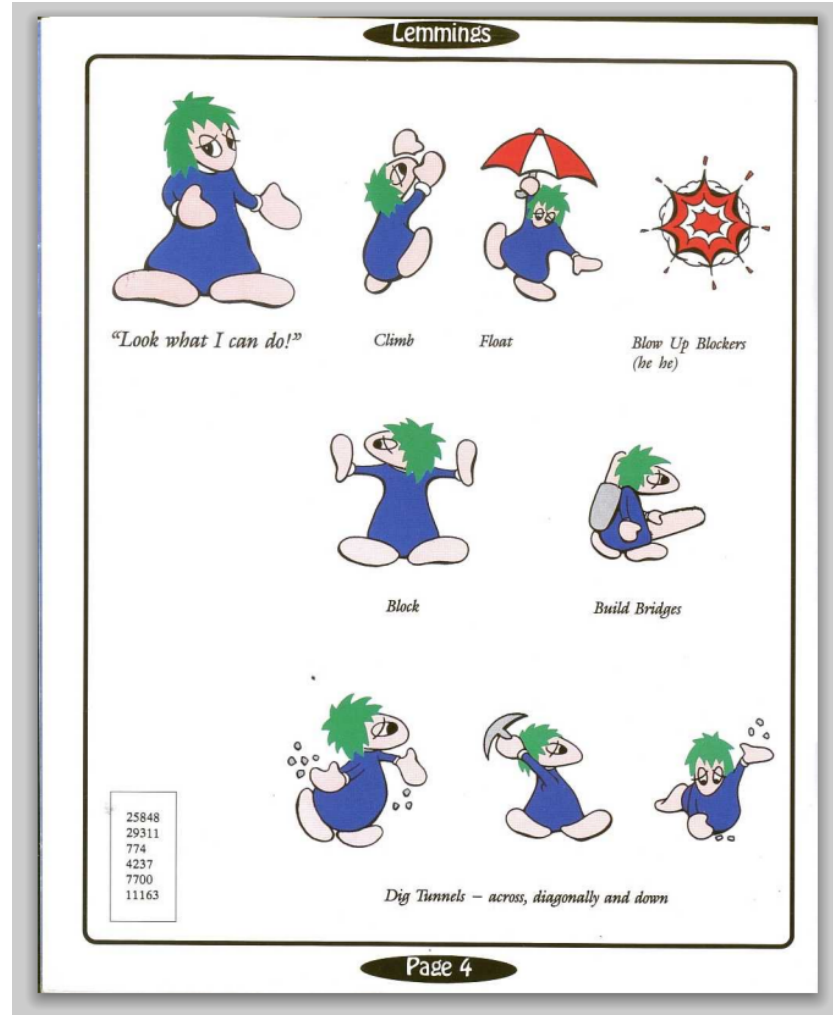
Lemmings!


OUT 4
IN 0%
TIME 4-52


Lemmings
"Look what I can do!"
Climb
Float
Blow Up Blockers
(he he)
Block
Build Bridges
25848
29311
774
4237
7700
11163
Dig Tunnels – across, diagonally and down
Page 4

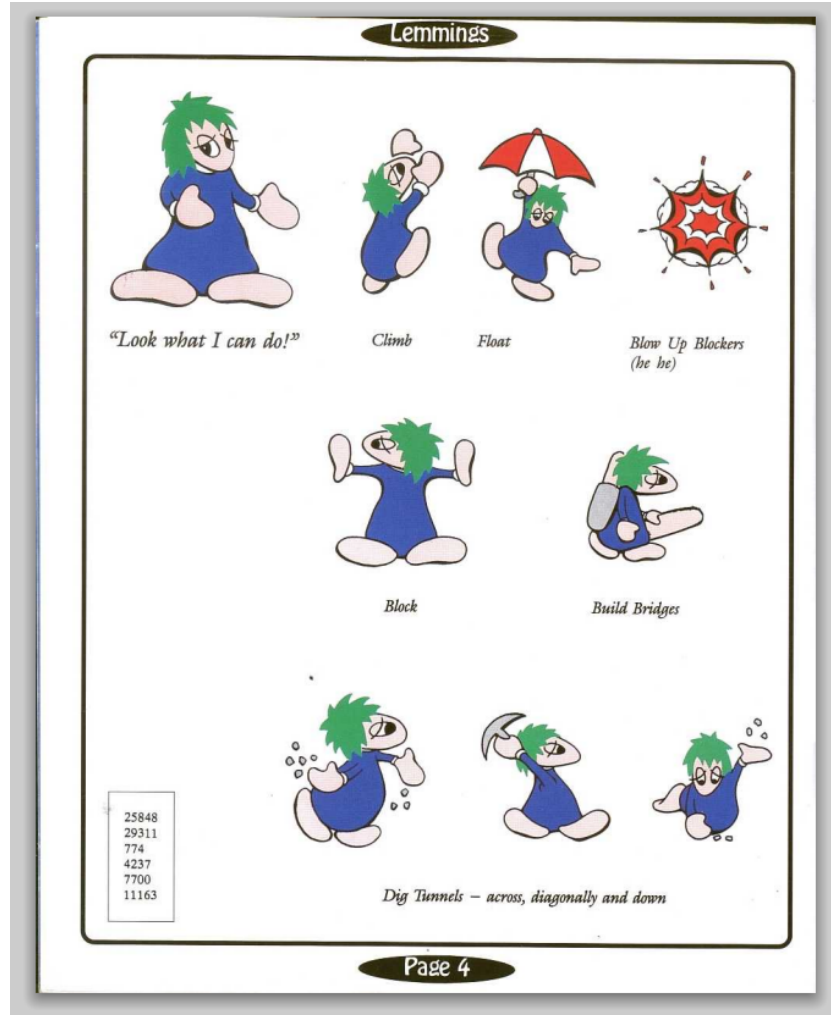
Lemmings!


Lemmings
24444
27907
31370
2833
6296
9759
"Another Blocker here and it's tunnel time! Oops!
Mustn't get my feet wet:
Another bridge I think."
Page 8


Lemmings
"Look what I can do!"
Climb
Float
Blow Up Blockers
(he he)
Block
Build Bridges
25848
29311
774
4237
7700
11163
Dig Tunnels – across, diagonally and down
Page 4

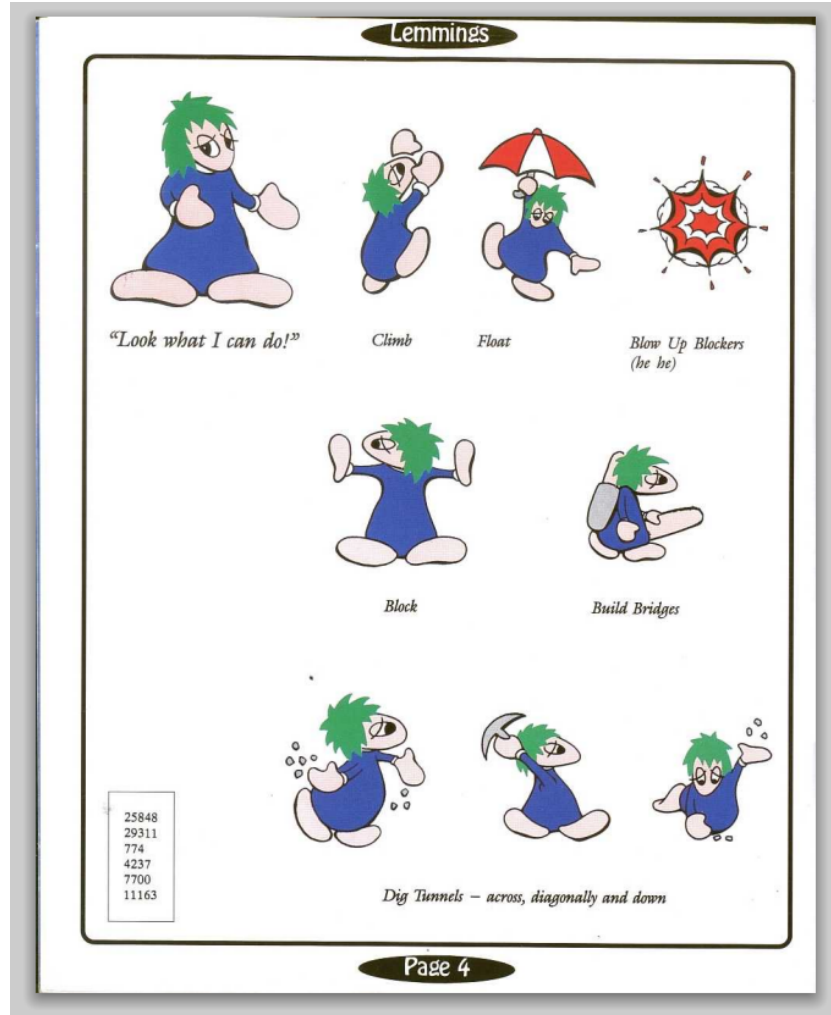
Lemmings!


OUT 28 IN 0% TIME 2-31


Lemmings
"Look what I can do!"
Climb
Float
Blow Up Blockers
(he he)
Block
Build Bridges
25848
29311
774
4237
7700
11163
Dig Tunnels – across, diagonally and down
Page 4

Lemmings!

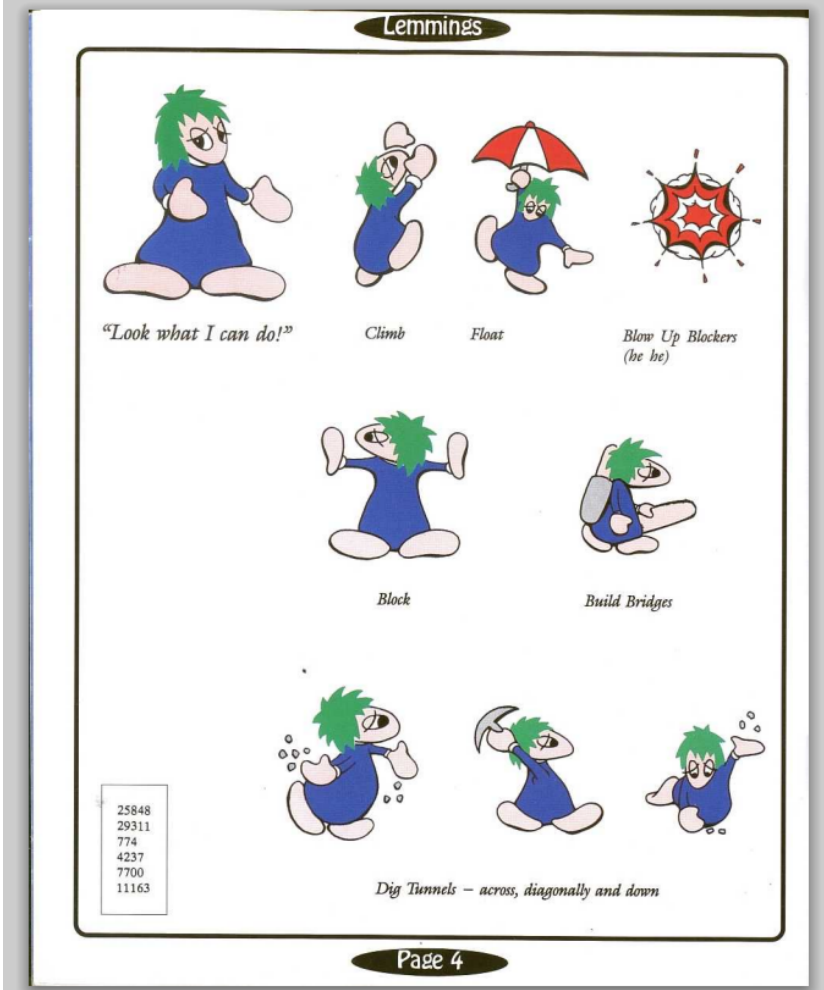
Oh no! More 3-SAT reductions



Benjamin Sach

Lemmings!

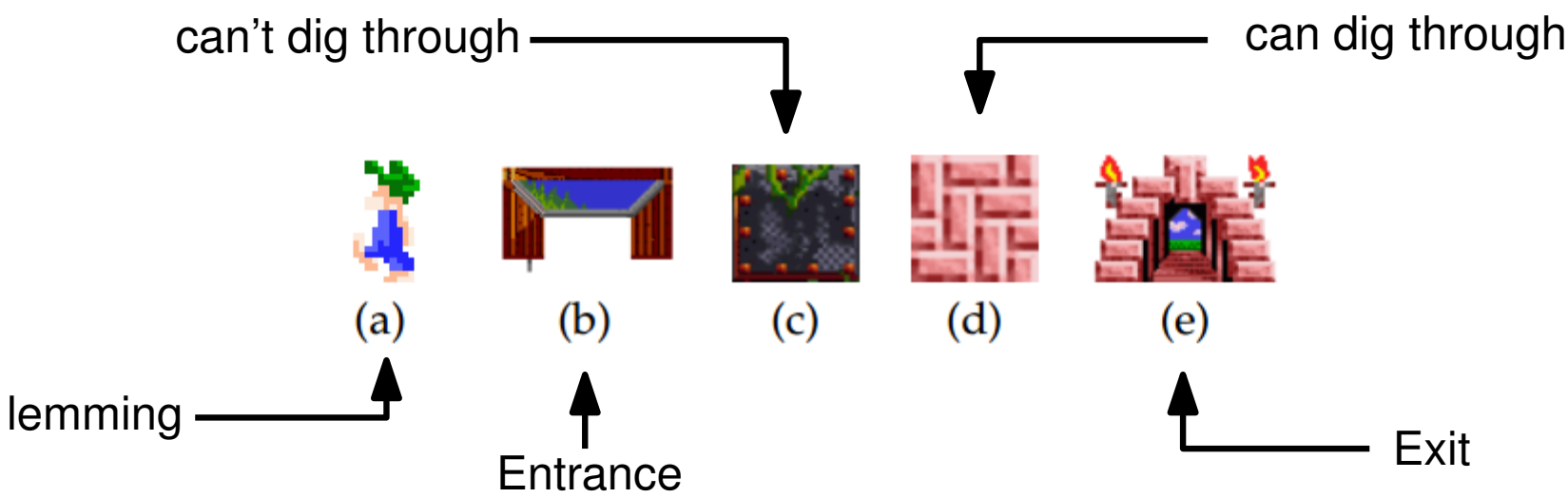
Oh no! More 3-SAT reductions

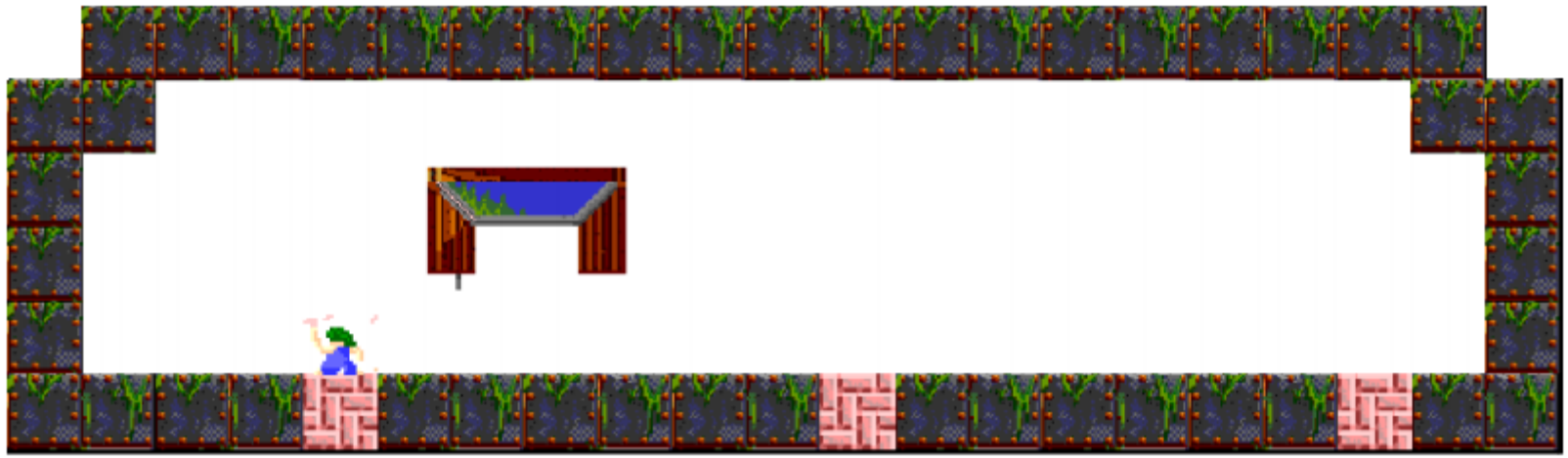


We will show that it is NP-hard to decide whether you can save *all* the lemmings

Benjamin Sach

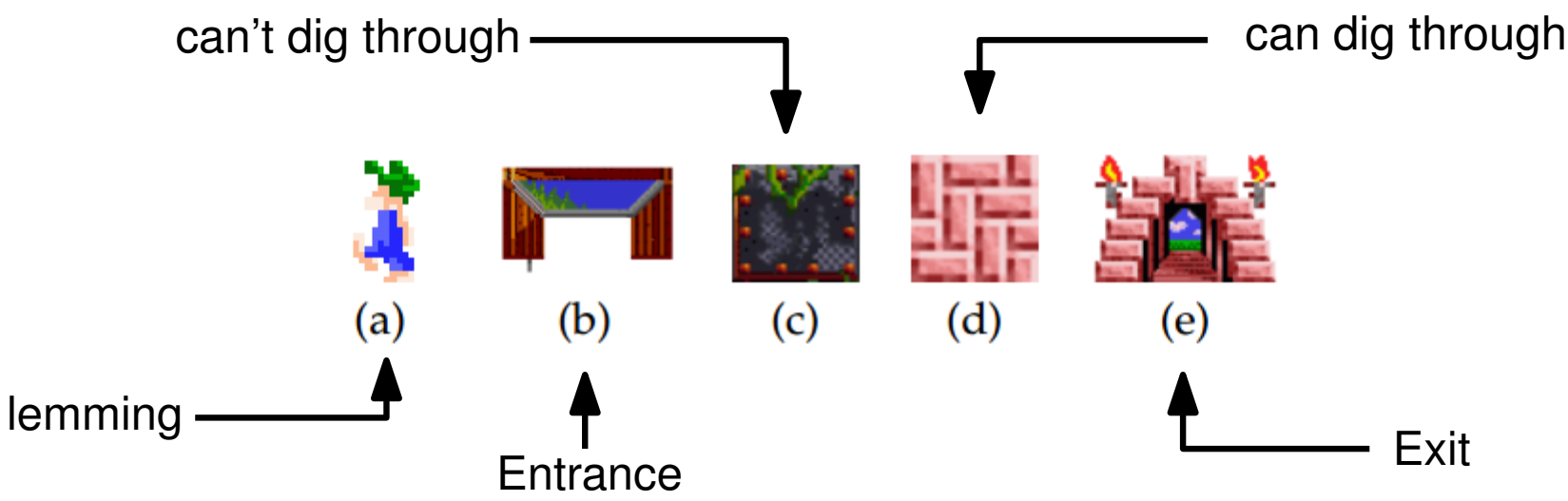
Clauses

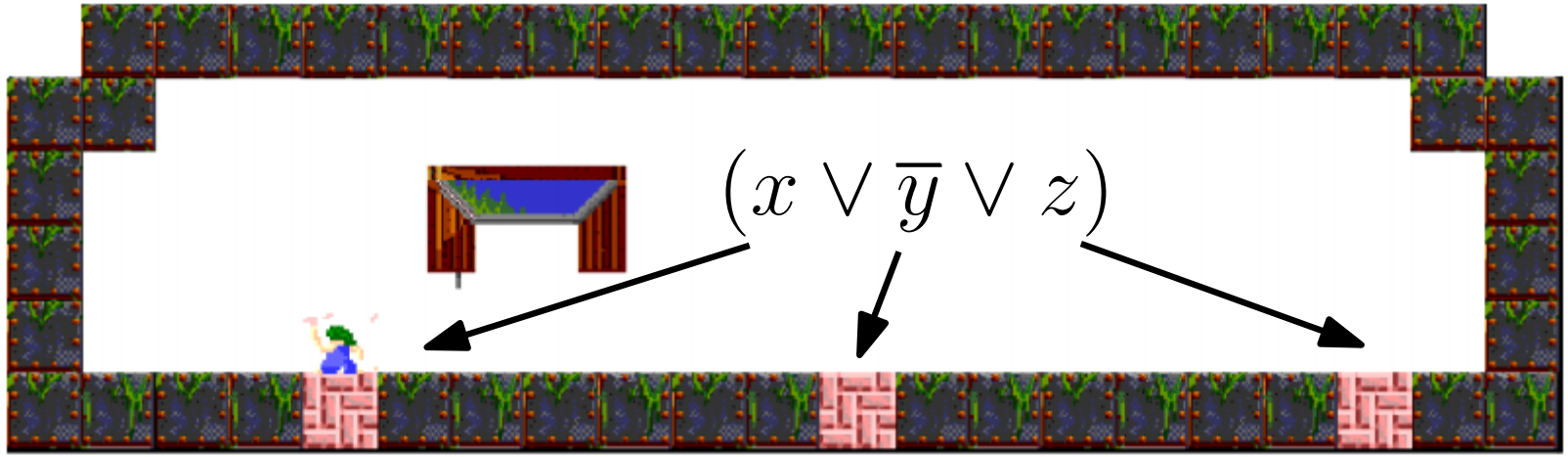




clause gadget (one lemming + one 'allocated' digger)

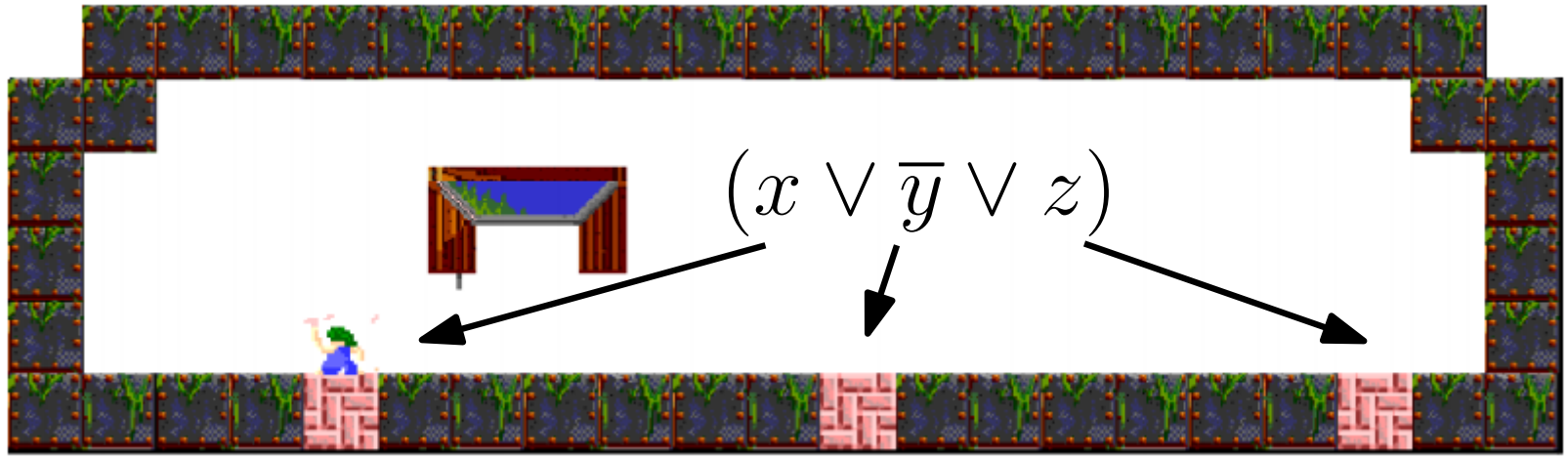
Clauses





clause gadget (one lemming + one 'allocated' digger)

Clauses



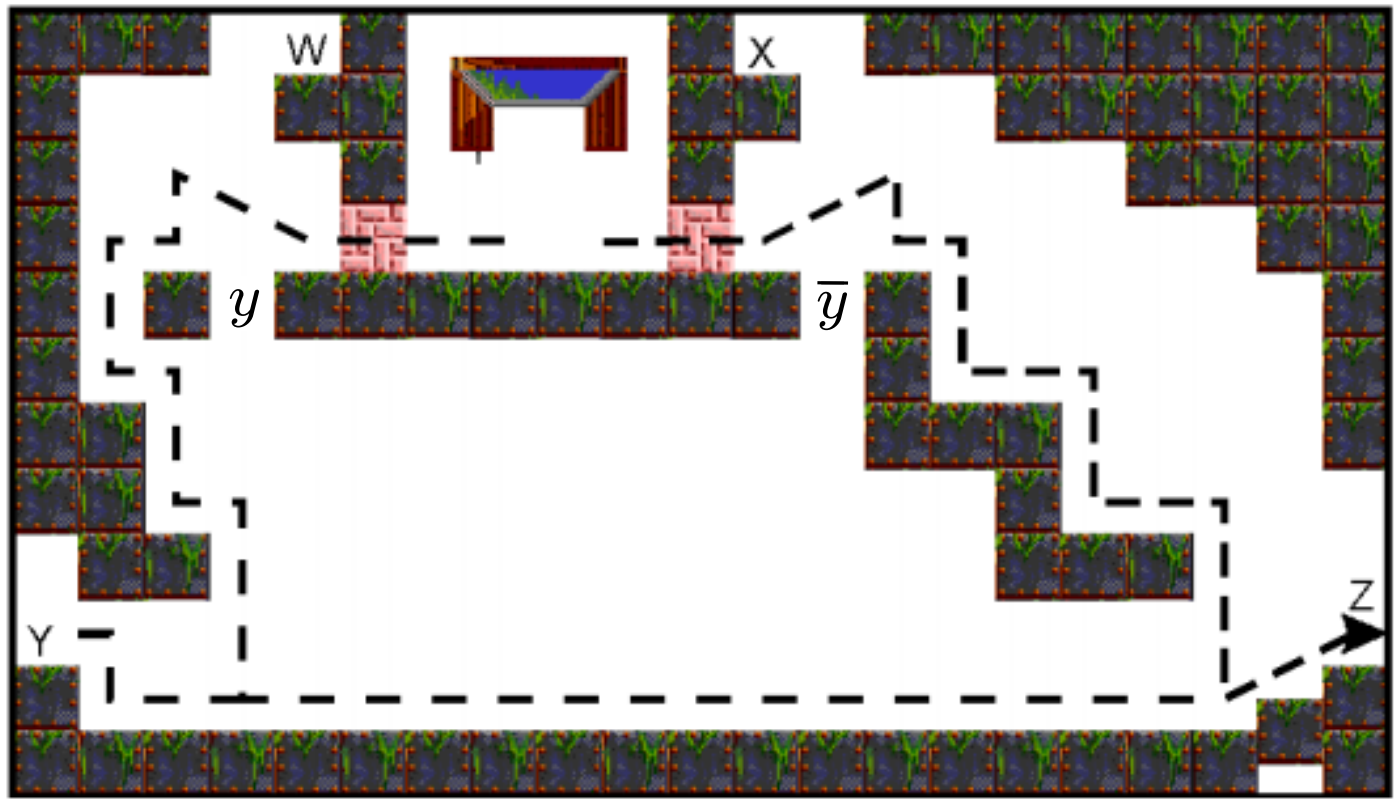
clause gadget (one lemming + one ‘allocated’ digger)

We will ensure that the lemming survives iff it digs through a block corresponding to a *true* literal

(under the chosen assignment)

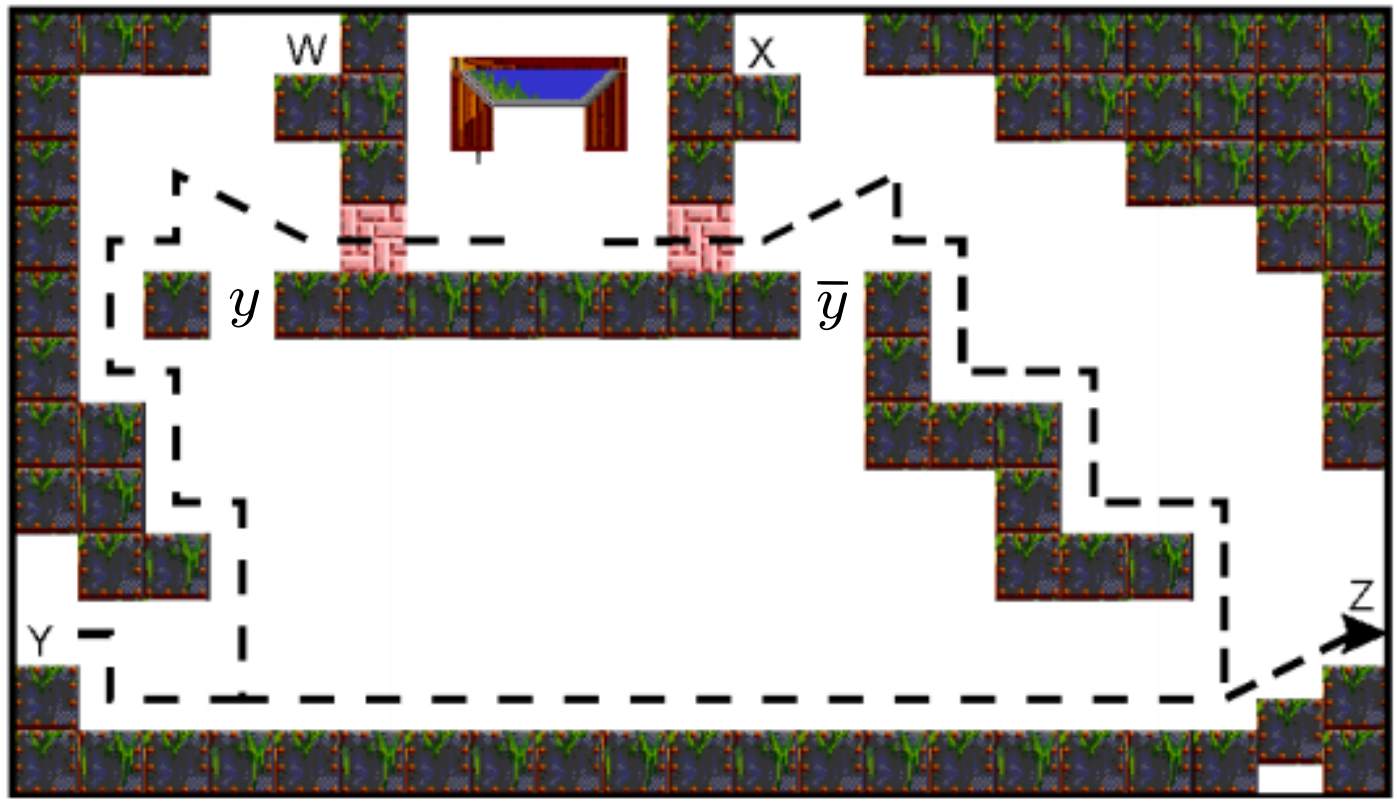
- remember that a literal is x or $\overline{x}$

variable gadgets



(one lemming + one ‘allocated’ builder move + one ‘allocated’ basher move)

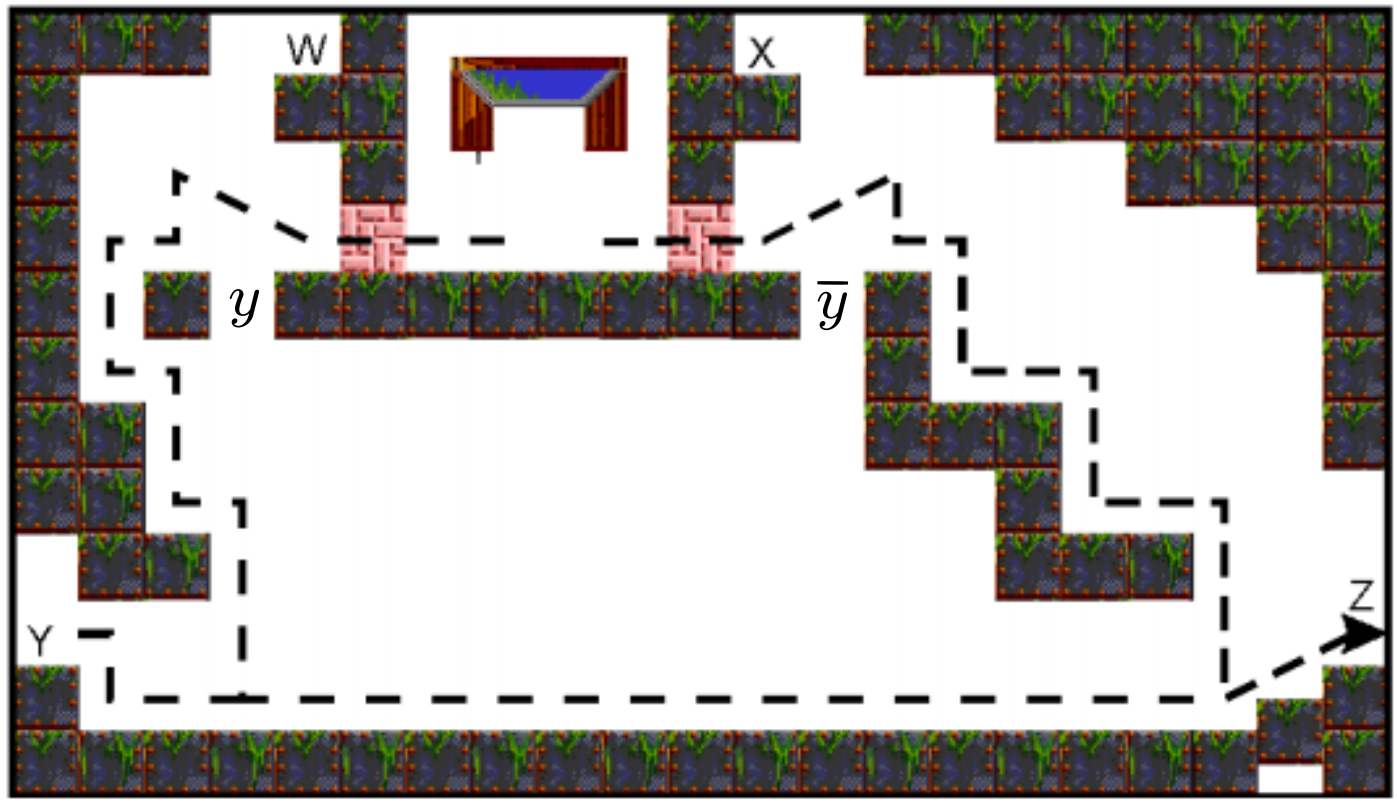
variable gadgets



(one lemming + one ‘allocated’ builder move + one ‘allocated’ basher move)

A lemming entering at Y leaves through Z (without player intervention)

variable gadgets

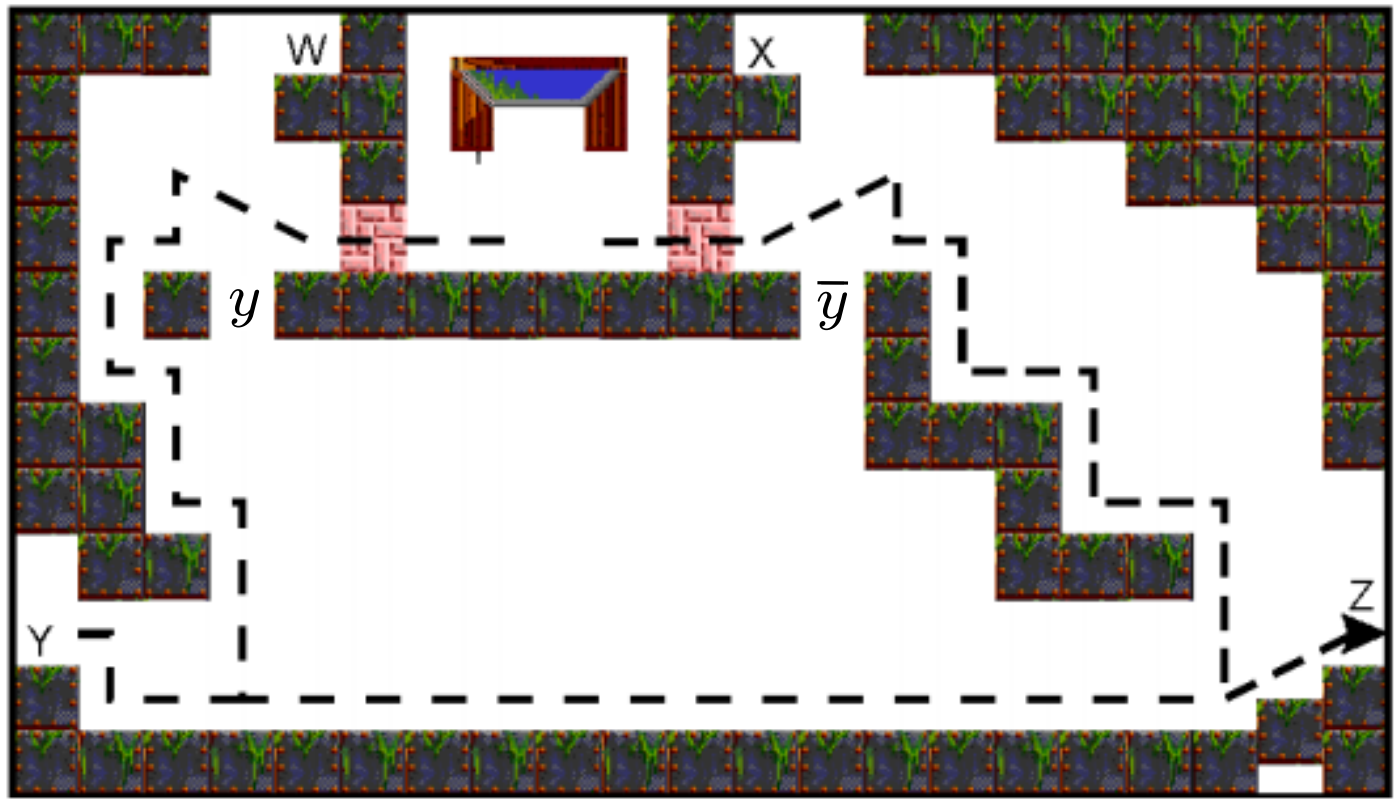


(one lemming + one ‘allocated’ builder move + one ‘allocated’ basher move)

A lemming entering at Y leaves through Z (without player intervention)

. . . .and *eventually* will reach the exit safely

variable gadgets

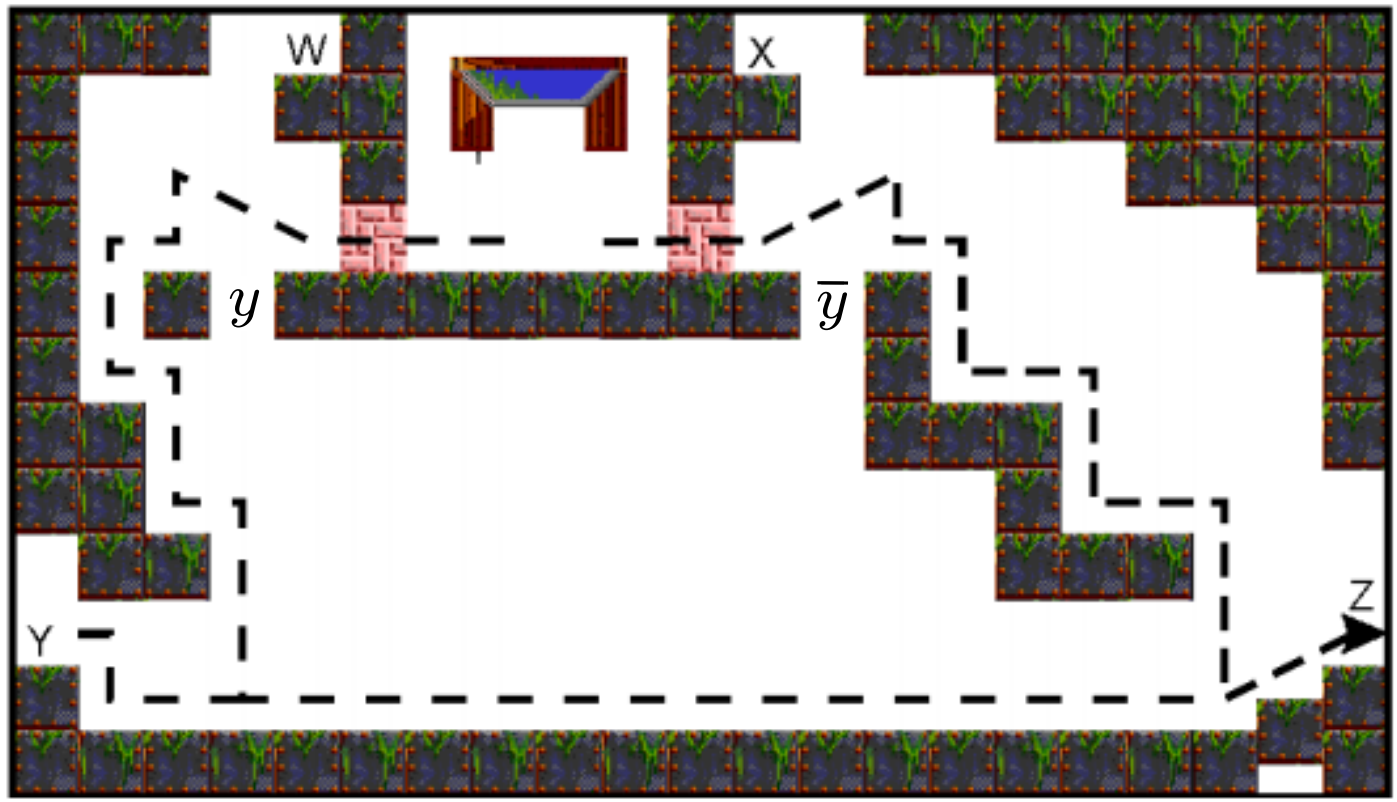


(one lemming + one ‘allocated’ builder move + one ‘allocated’ basher move)

The lemming from the included start either

bashes and bridges left (y) or right ($\overline{y}$)

variable gadgets



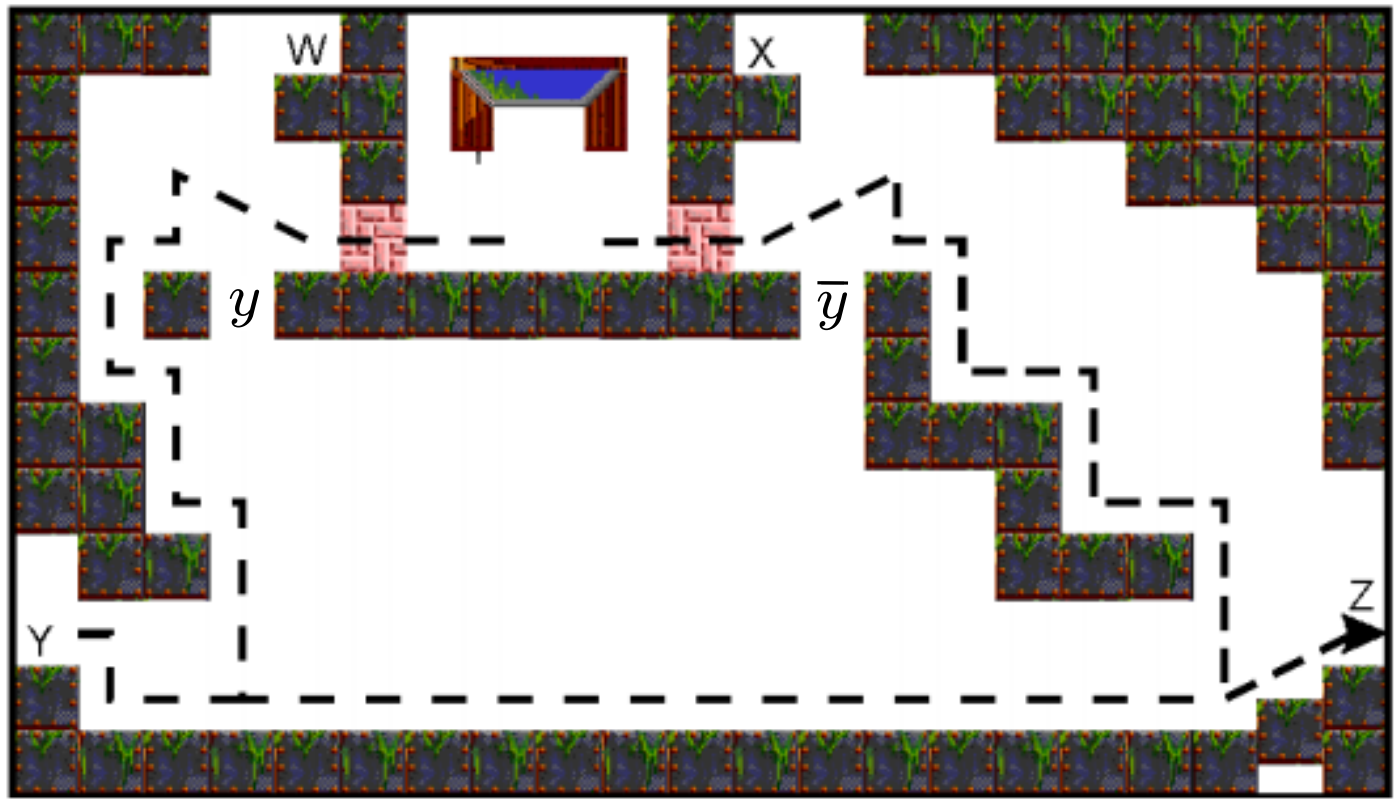
(one lemming + one ‘allocated’ builder move + one ‘allocated’ basher move)

The lemming from the included start either

bashes and bridges left (y) or right ($\overline{y}$)

. . . and then leaves via Z to safety

variable gadgets

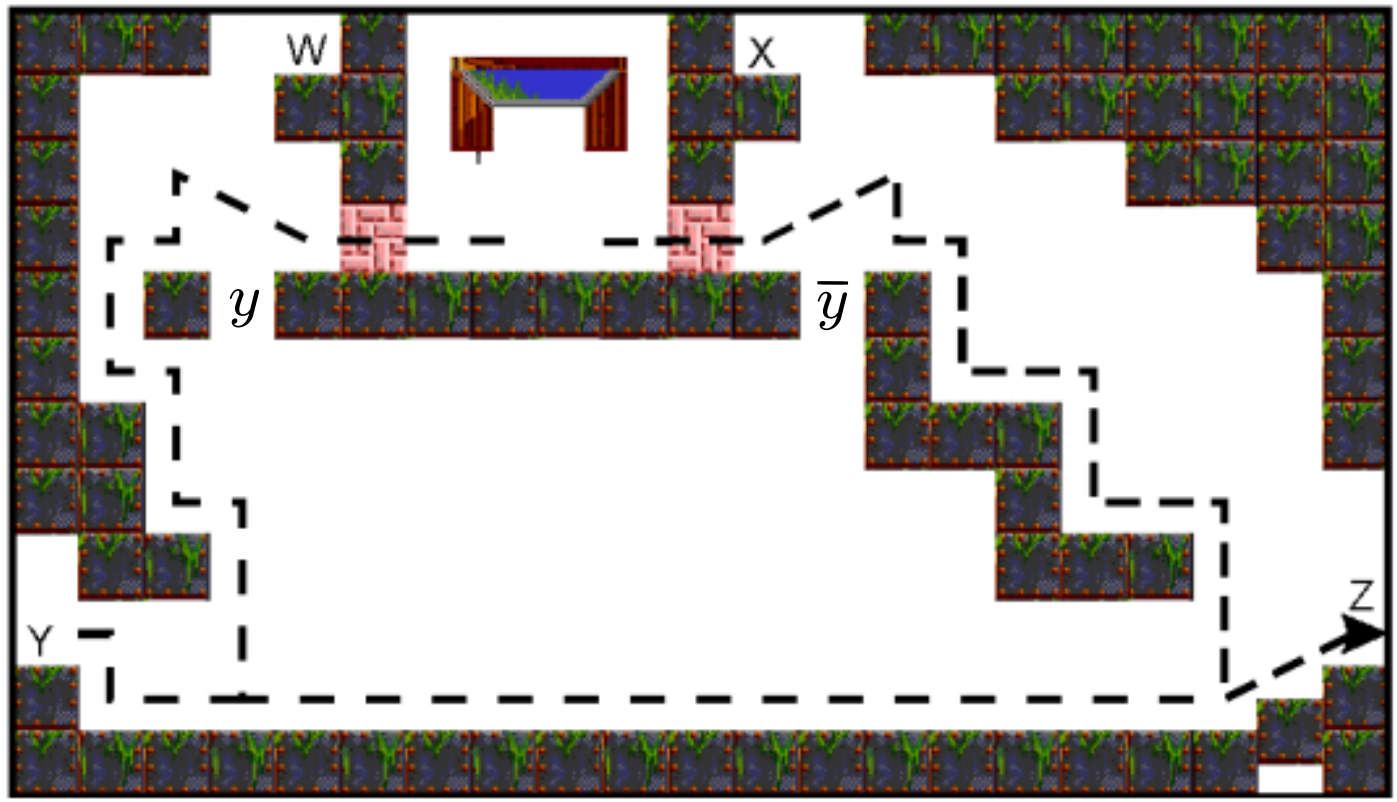


(one lemming + one ‘allocated’ builder move + one ‘allocated’ basher move)

A lemming from W survives iff y was bridged

A lemming from X survives iff $\overline{y}$ was bridged

variable gadgets



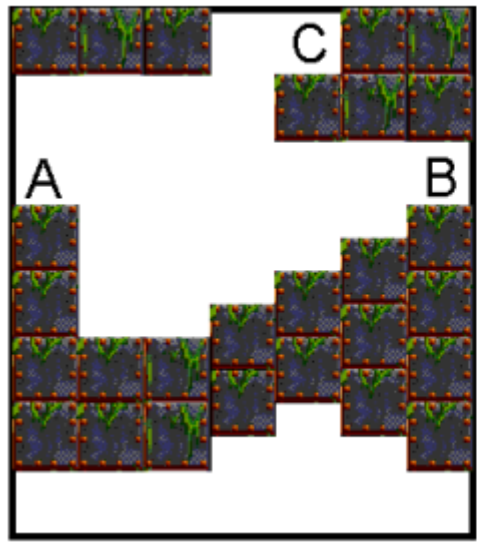
(one lemming + one ‘allocated’ builder move + one ‘allocated’ basher move)

A lemming from W survives iff y was bridged

A lemming from X survives iff $\overline{y}$ was bridged

. . . and then leaves via Z to safety

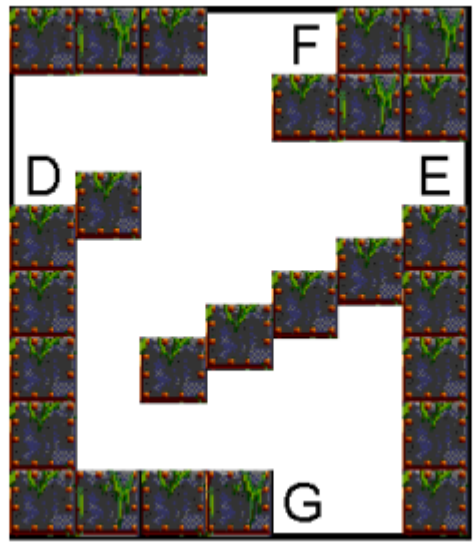
Other bits . . .



Junction

$A \rightarrow B$

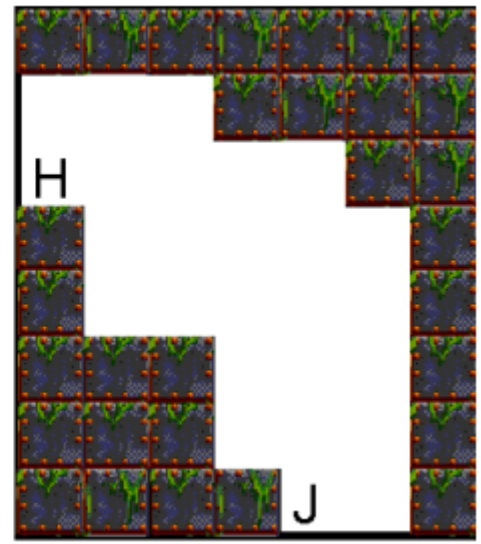
$C \rightarrow B$



Crossover

$D \rightarrow E$

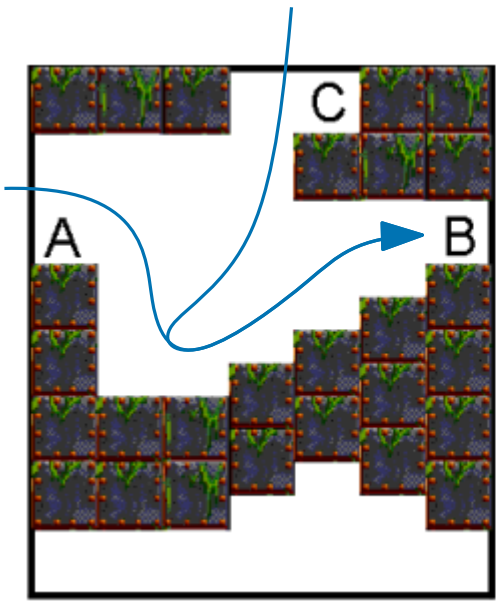
$F \rightarrow G$



Corner

$H \rightarrow J$

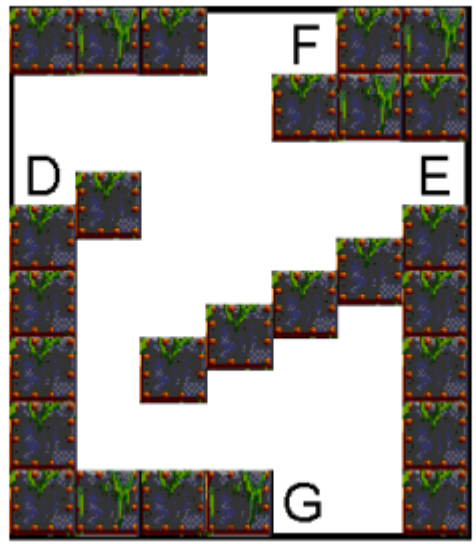
Other bits . . .



Junction

A → B

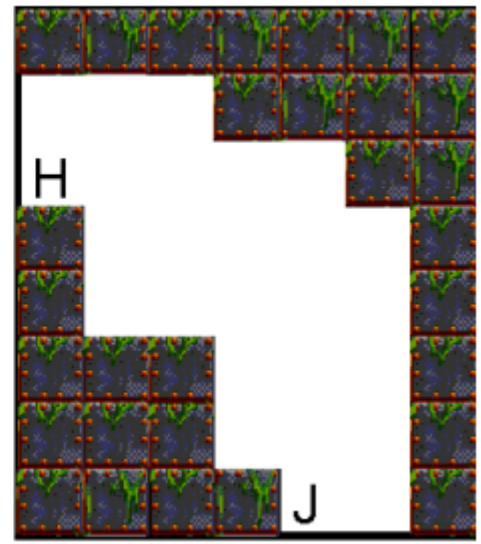
C → B



Crossover

D → E

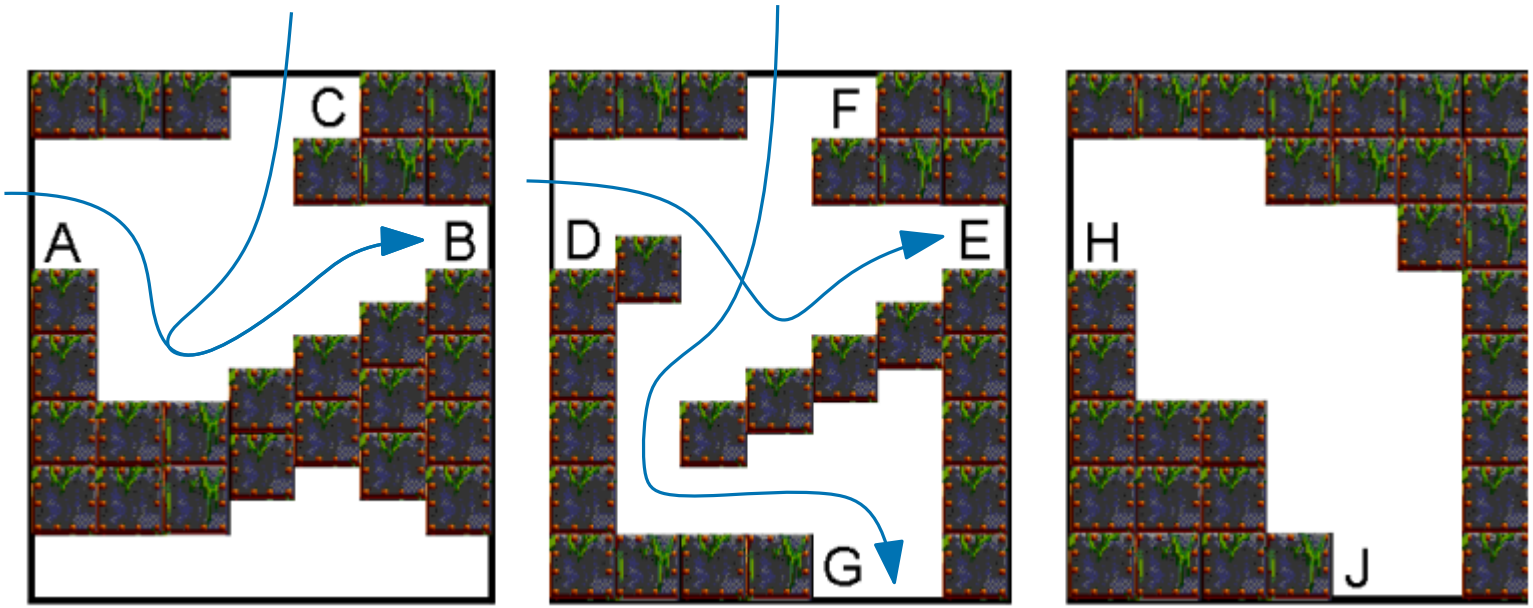
F → G



Corner

H → J

Other bits . . .



Junction

$A \rightarrow B$

$C \rightarrow B$

Crossover

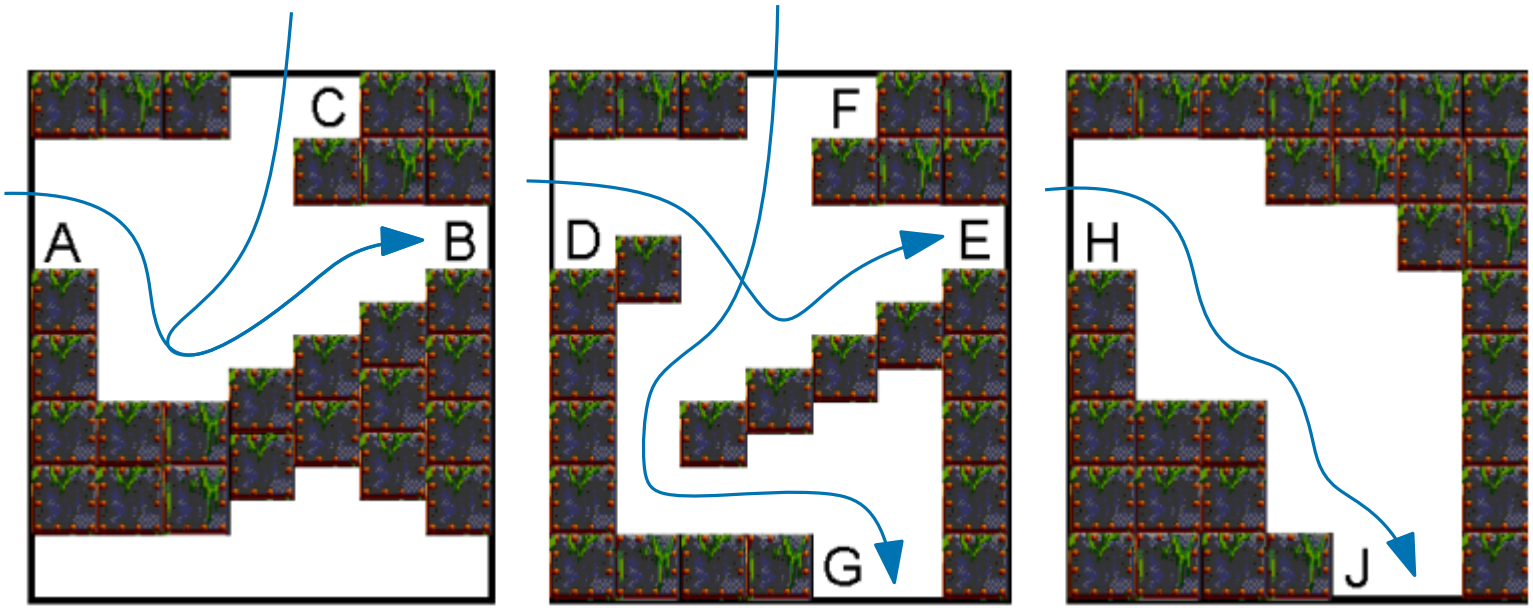
$D \rightarrow E$

$F \rightarrow G$

Corner

$H \rightarrow J$

Other bits . . .



Junction

$A \rightarrow B$

$C \rightarrow B$

Crossover

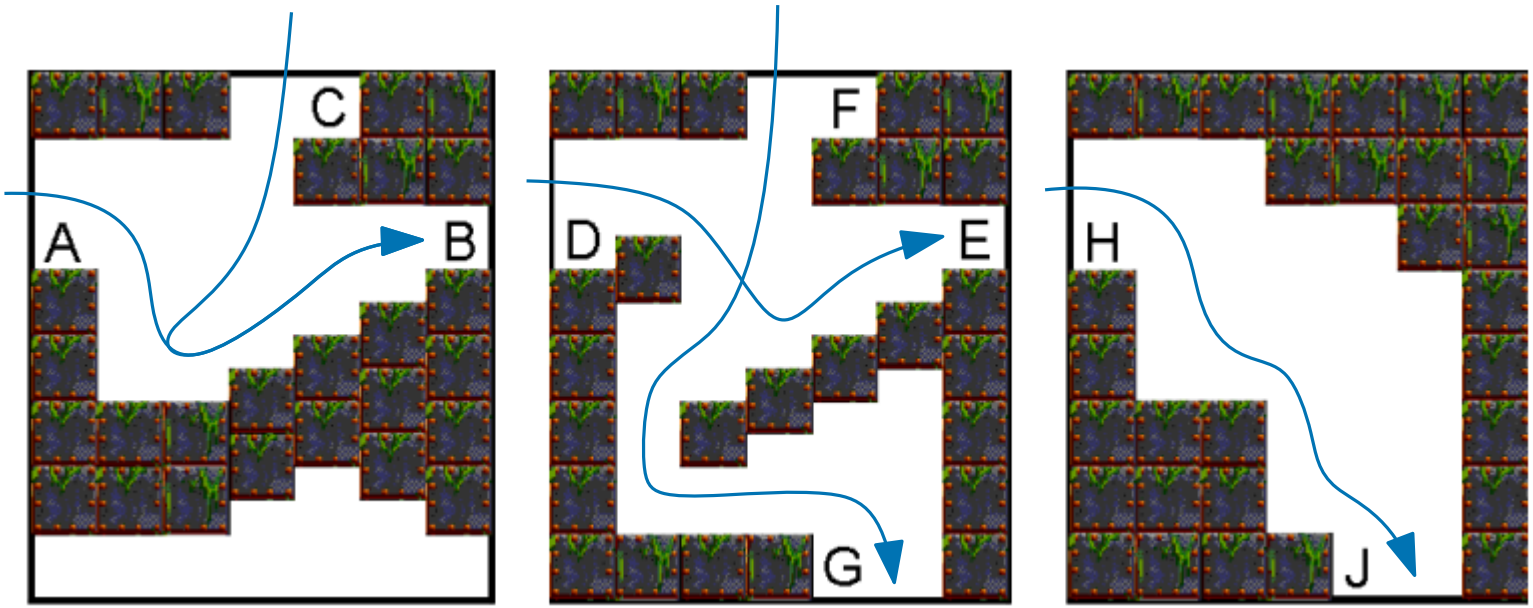
$D \rightarrow E$

$F \rightarrow G$

Corner

$H \rightarrow J$

Other bits . . .



Junction

$A \rightarrow B$

$C \rightarrow B$

Crossover

$D \rightarrow E$

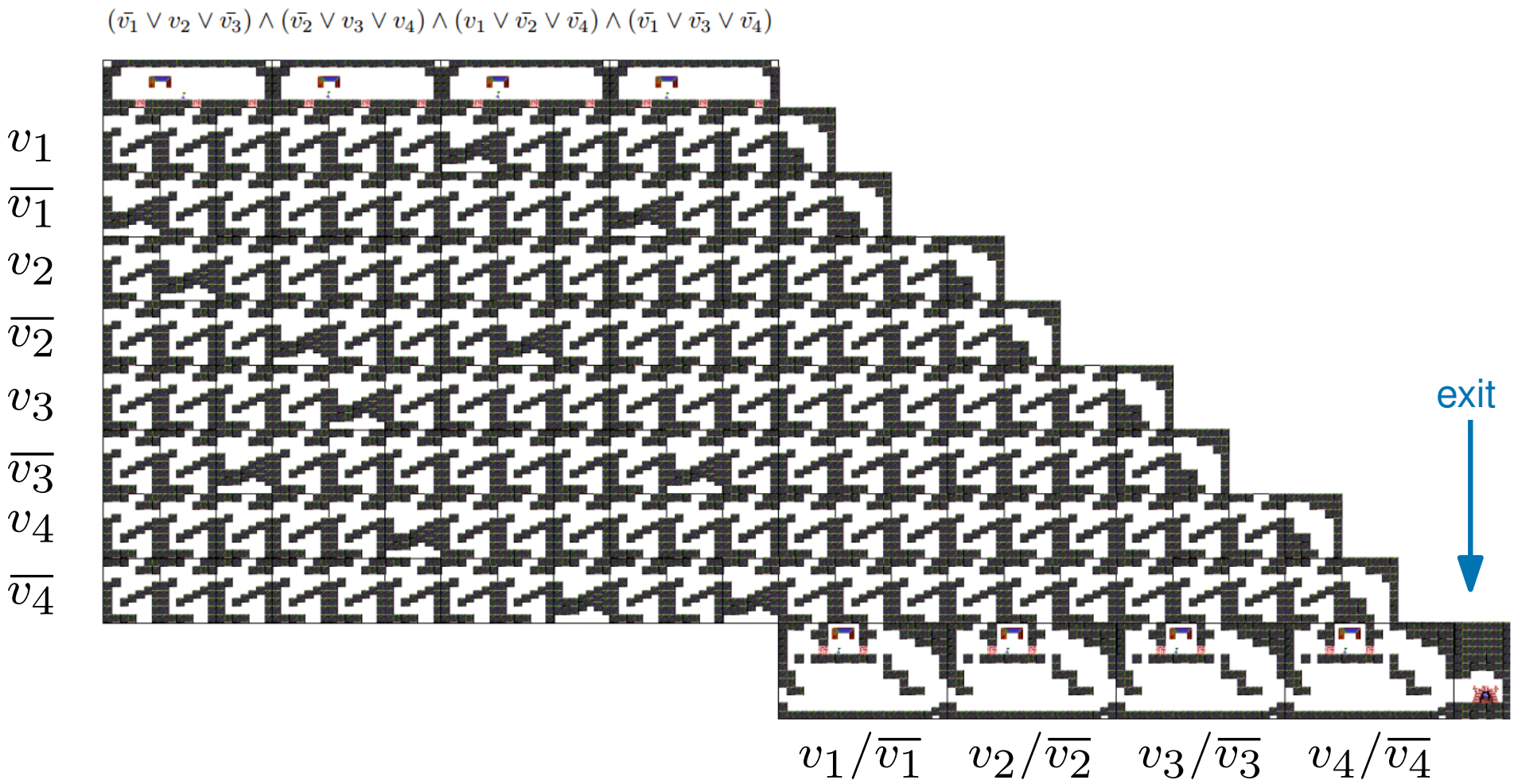
$F \rightarrow G$

Corner

$H \rightarrow J$

. . . without player interaction

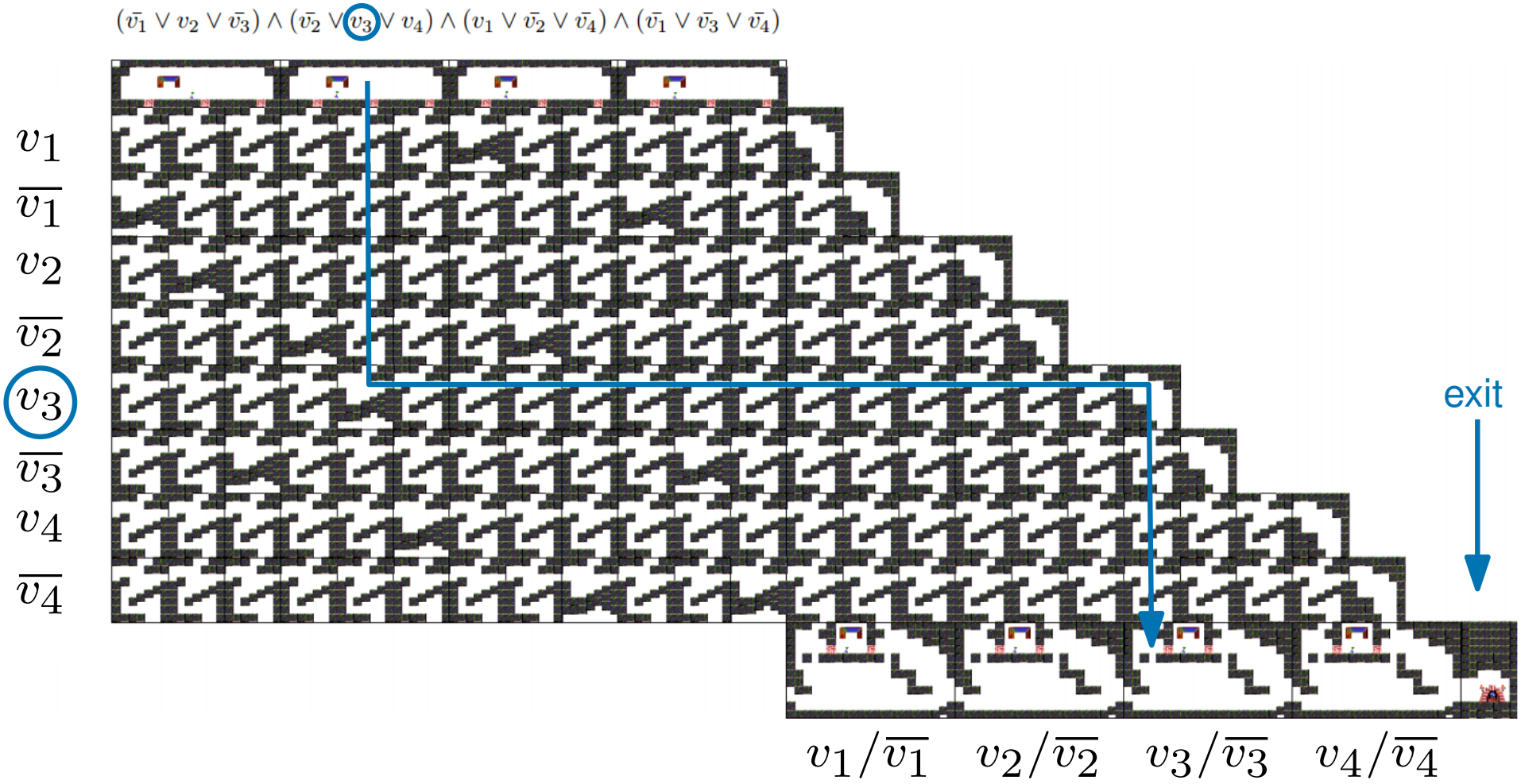
The big picture



This Lemming's level encodes the formula,

$$(\bar{v_1} \vee v_2 \vee \bar{v_3}) \wedge (\bar{v_2} \vee v_3 \vee v_4) \wedge (v_1 \vee \bar{v_2} \vee \bar{v_4}) \wedge (\bar{v_1} \vee \bar{v_3} \vee \bar{v_4})$$

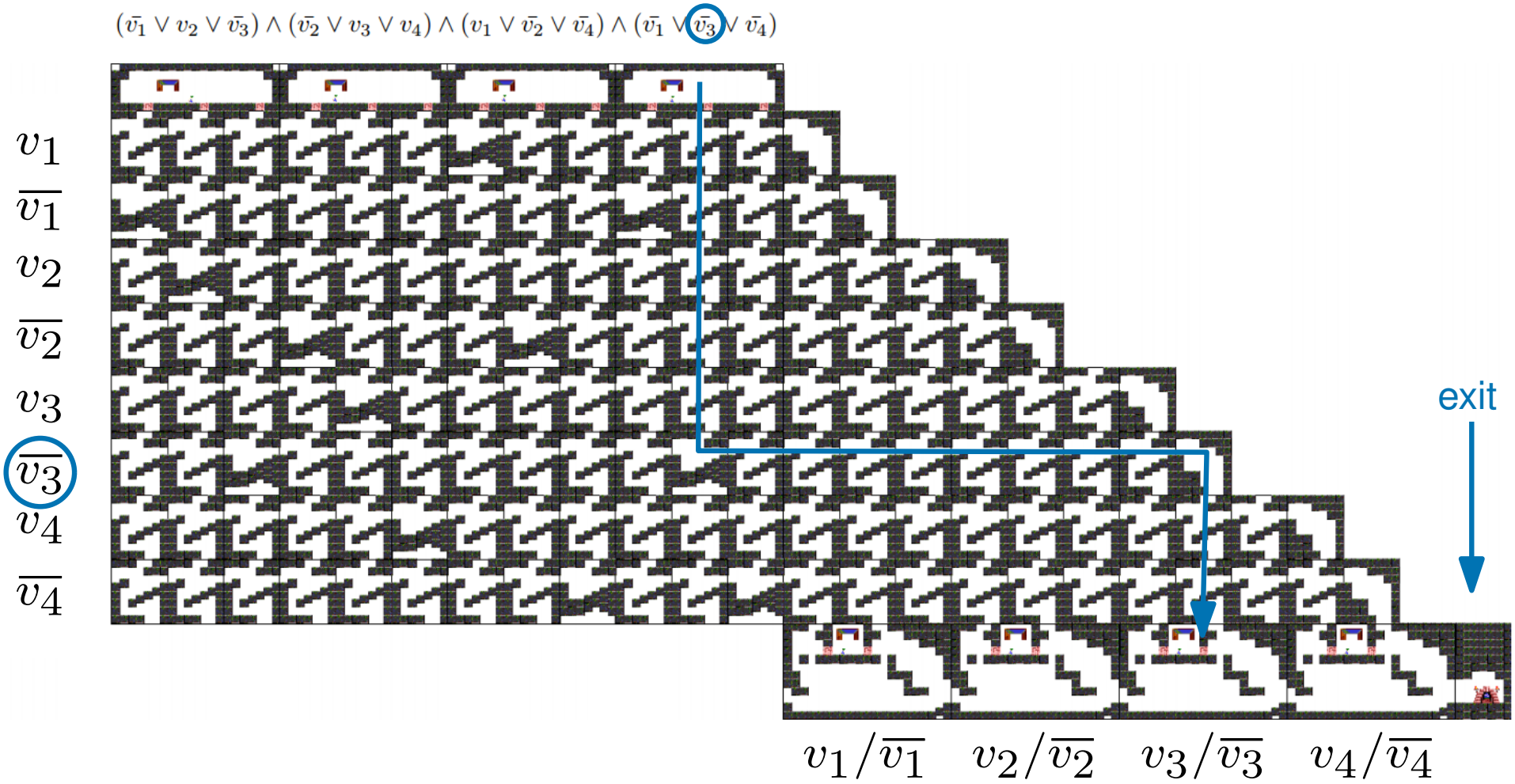
The big picture



This Lemming's level encodes the formula,

$$(\bar{v_1} \vee v_2 \vee \bar{v_3}) \wedge (\bar{v_2} \vee v_3 \vee v_4) \wedge (v_1 \vee \bar{v_2} \vee \bar{v_4}) \wedge (\bar{v_1} \vee \bar{v_3} \vee \bar{v_4})$$

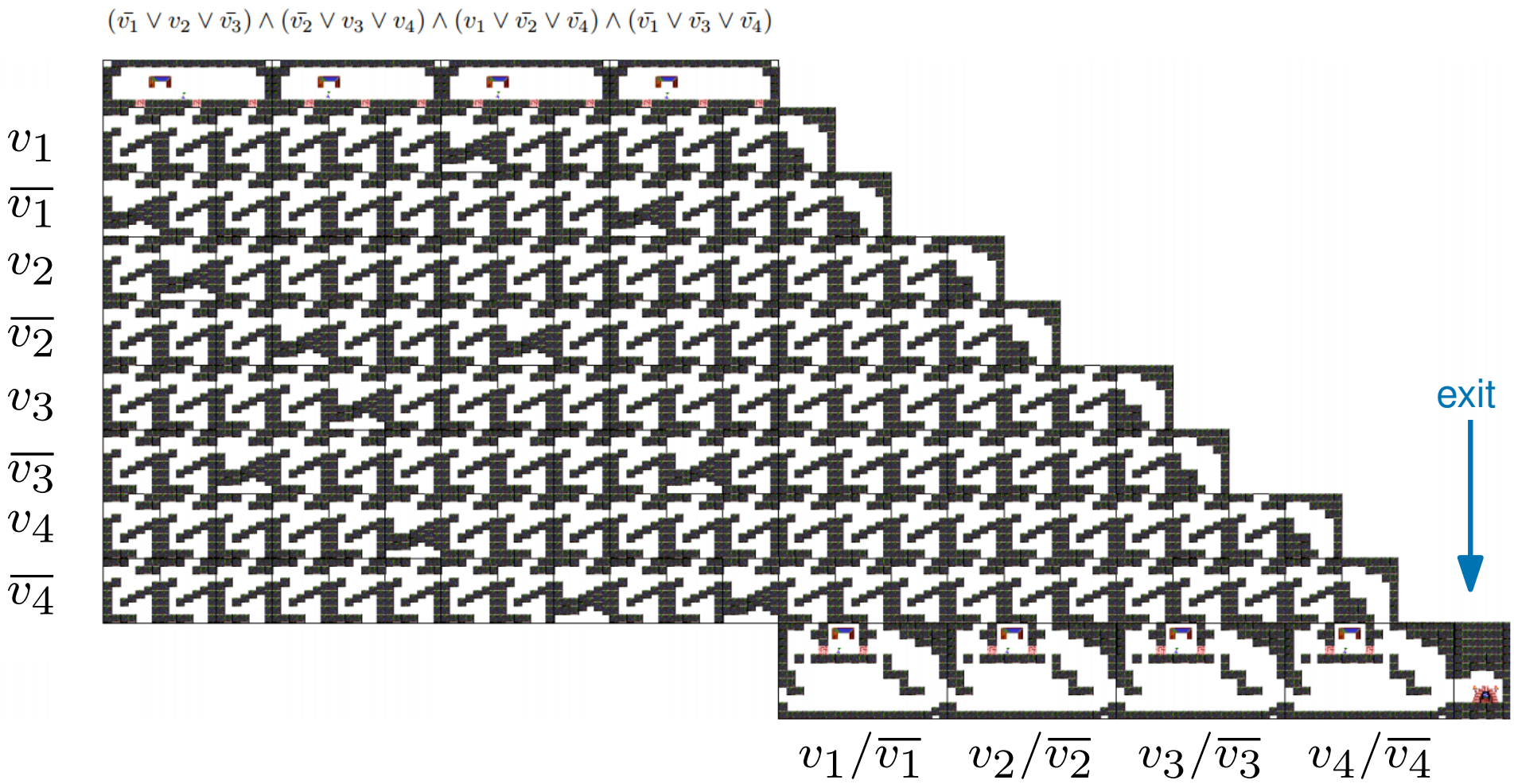
The big picture



This Lemming's level encodes the formula,

$$(\bar{v_1} \vee v_2 \vee \bar{v_3}) \wedge (\bar{v_2} \vee v_3 \vee v_4) \wedge (v_1 \vee \bar{v_2} \vee \bar{v_4}) \wedge (\bar{v_1} \vee \bar{v_3} \vee \bar{v_4})$$

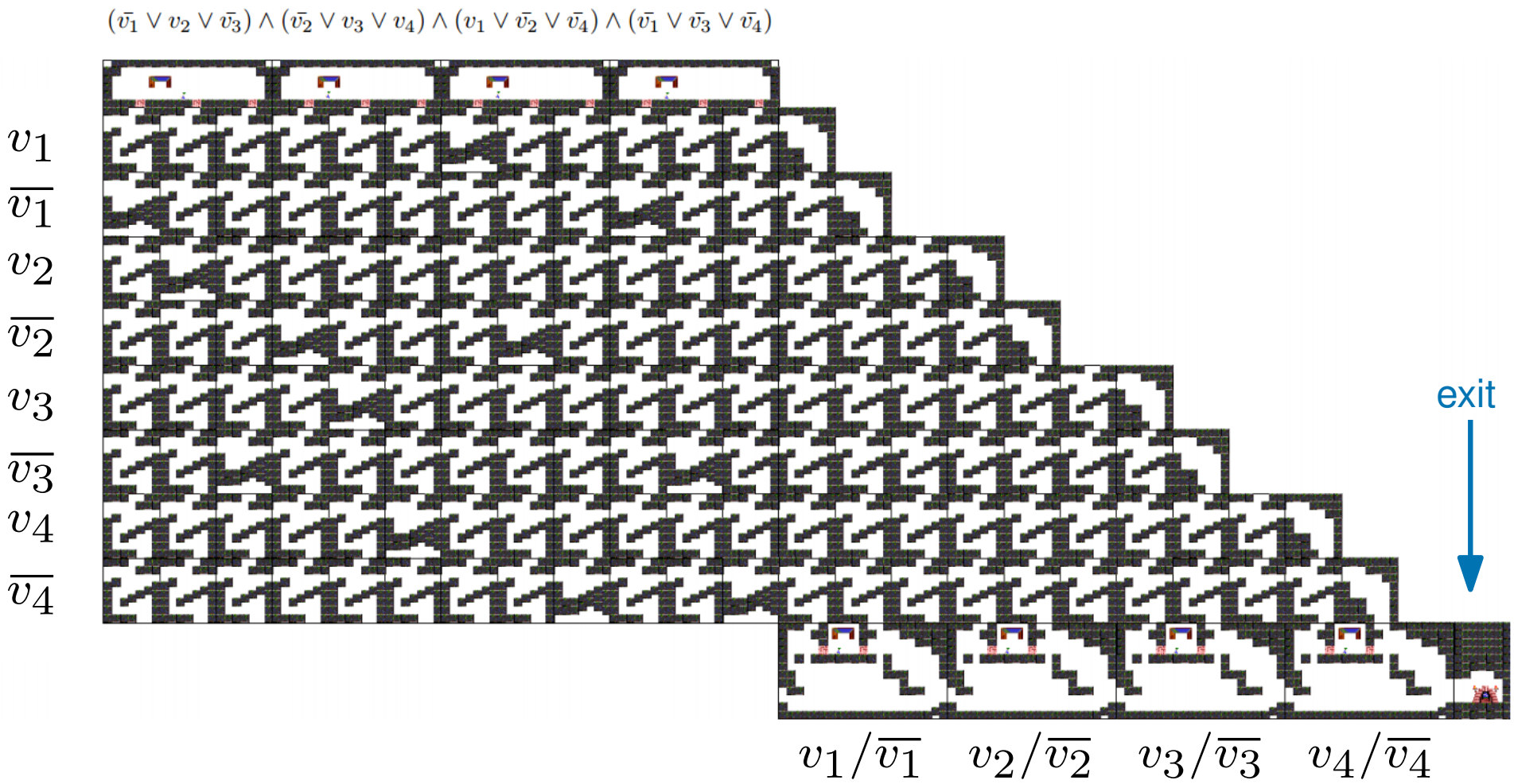
The big picture



This Lemming's level encodes the formula,

$$(\bar{v_1} \vee v_2 \vee \bar{v_3}) \wedge (\bar{v_2} \vee v_3 \vee v_4) \wedge (v_1 \vee \bar{v_2} \vee \bar{v_4}) \wedge (\bar{v_1} \vee \bar{v_3} \vee \bar{v_4})$$

The big picture

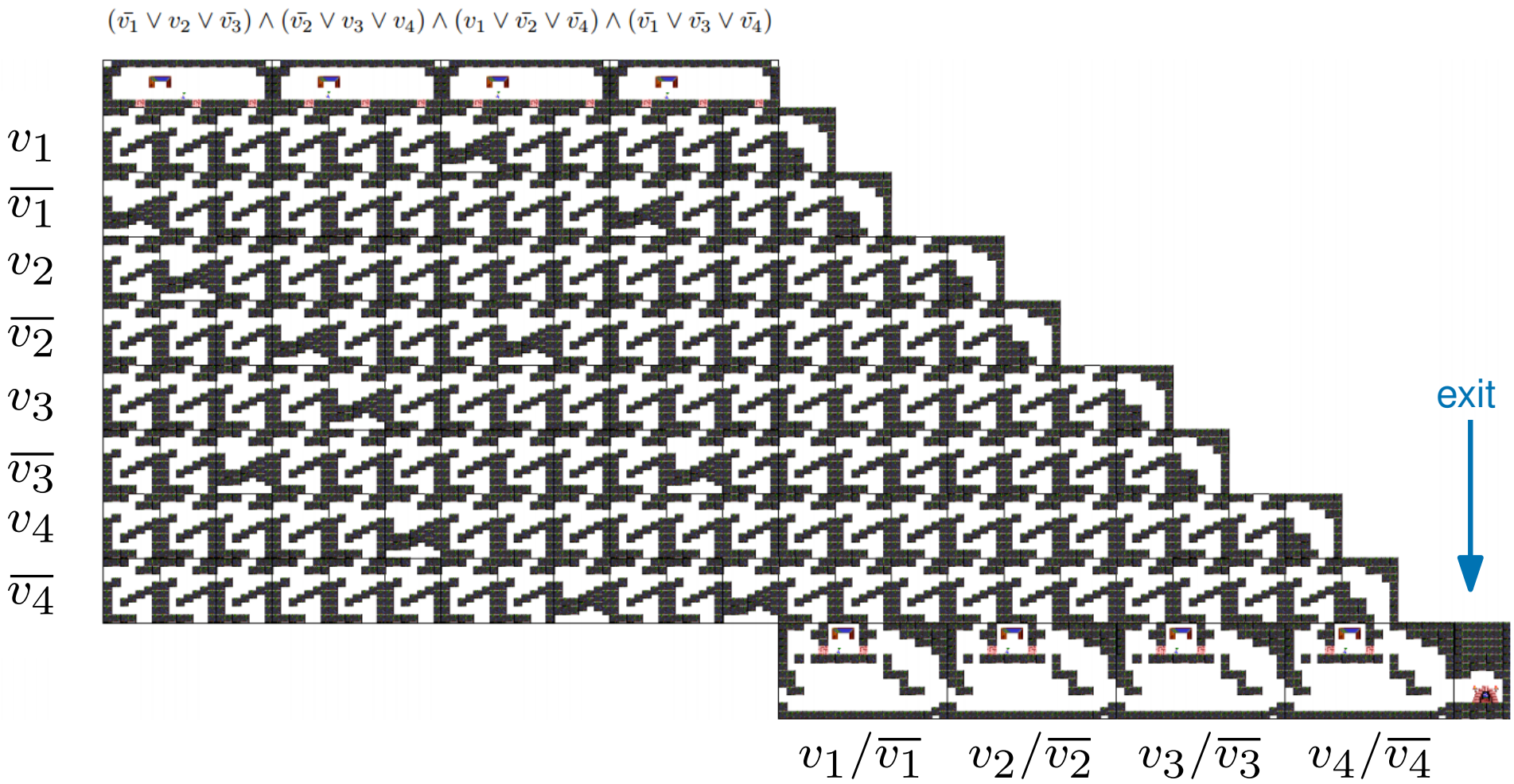


This Lemming's level encodes the formula,

$$(\bar{v_1} \vee v_2 \vee \bar{v_3}) \wedge (\bar{v_2} \vee v_3 \vee v_4) \wedge (v_1 \vee \bar{v_2} \vee \bar{v_4}) \wedge (\bar{v_1} \vee \bar{v_3} \vee \bar{v_4})$$

The formula is satisfiable iff you can save every lemming

The big picture

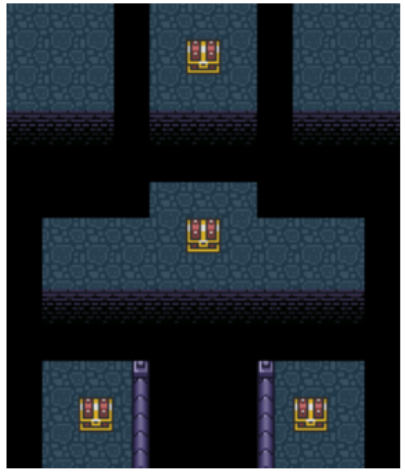


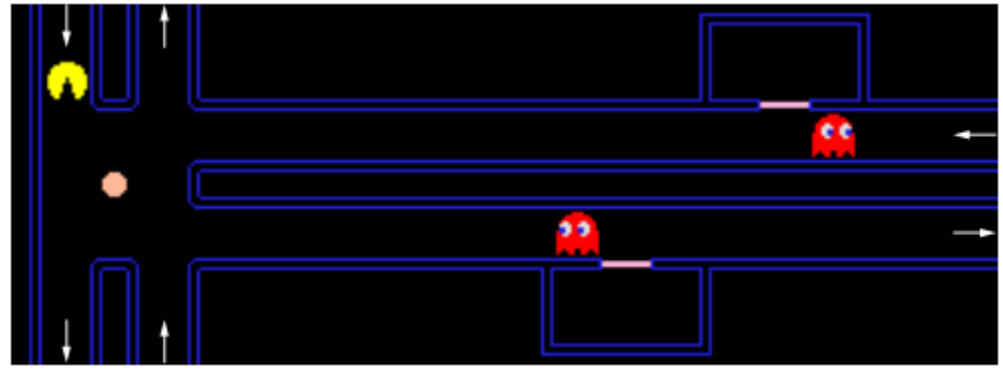
This Lemming's level encodes the formula,

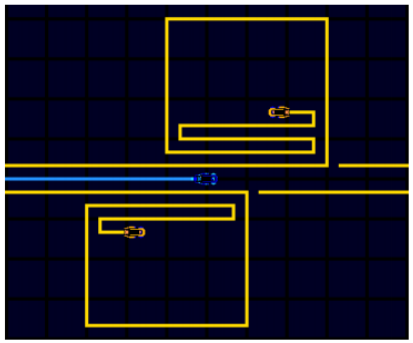
$$(\bar{v_1} \vee v_2 \vee \bar{v_3}) \wedge (\bar{v_2} \vee v_3 \vee v_4) \wedge (v_1 \vee \bar{v_2} \vee \bar{v_4}) \wedge (\bar{v_1} \vee \bar{v_3} \vee \bar{v_4})$$

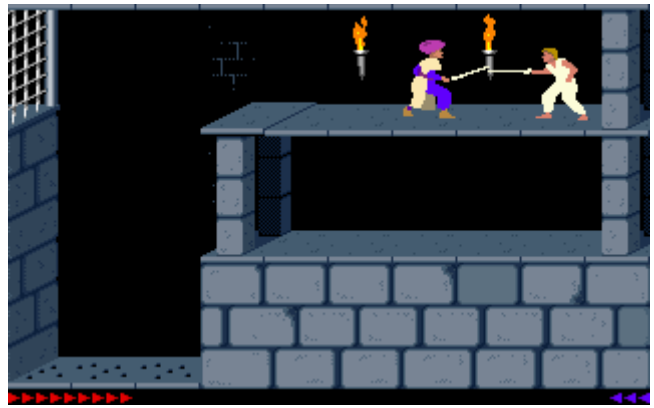
The formula is satisfiable iff you can save every lemming . . . so Lemmings is NP-hard

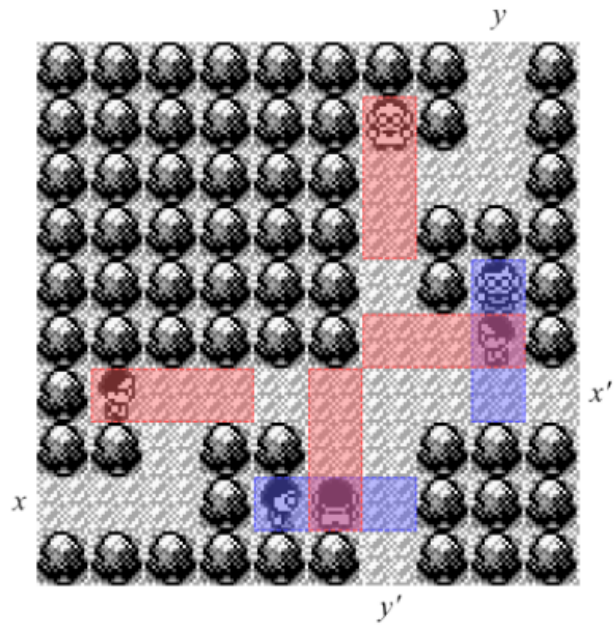
Other results

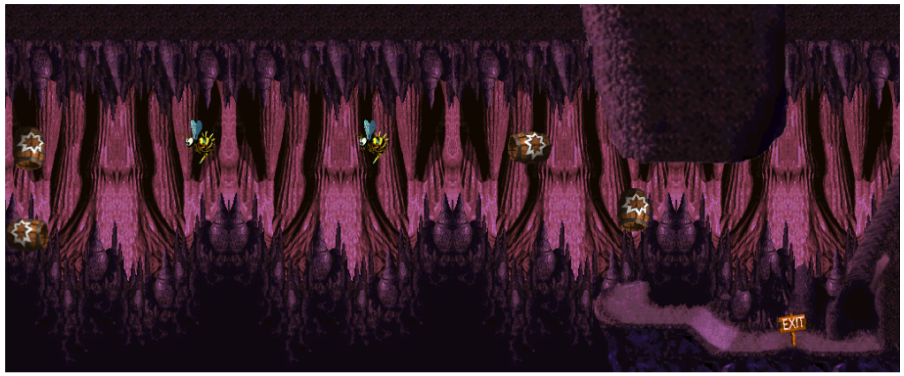




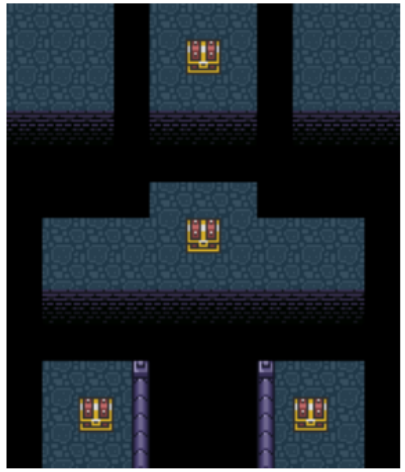


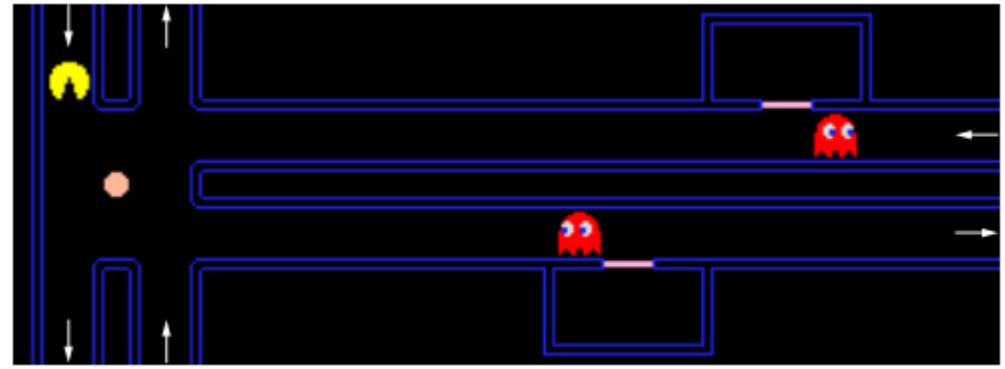


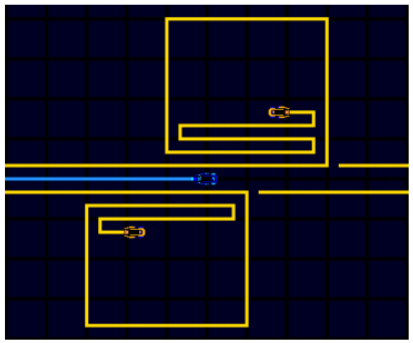


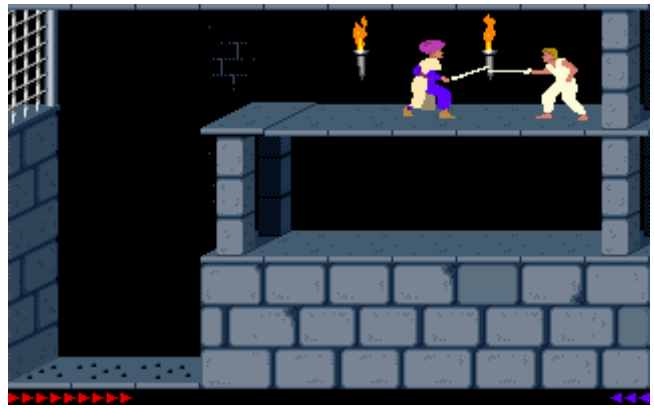


Other results

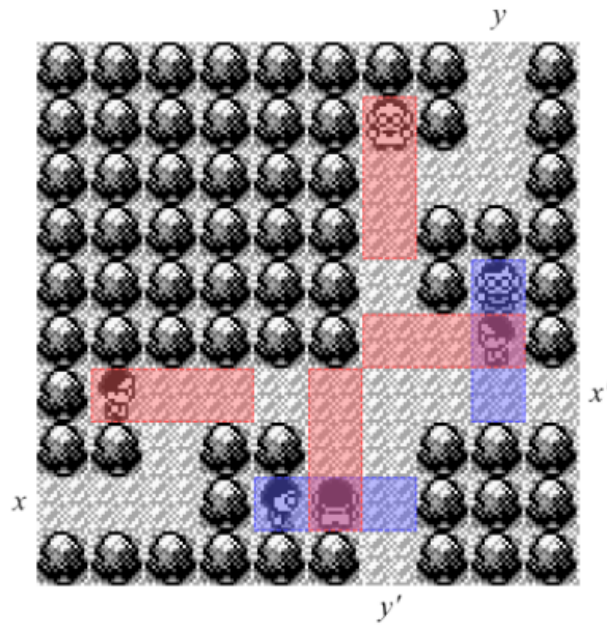


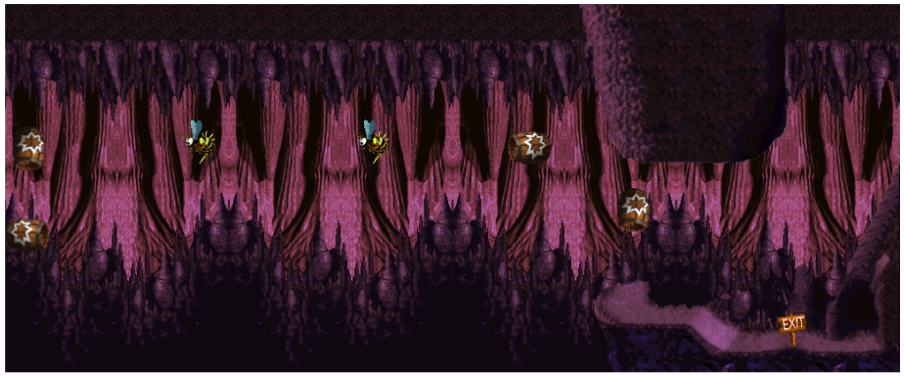






. . . among many others





Bibliography

Classic Nintendo Games are (NP-)Hard

Greg Aloupis, Erik Demaine, Alan Guo - on ArXiv 2012

Computational complexity of two-dimensional platform games

Michal Forišek - FUN with algorithms 2010

Gaming is a hard job, but someone has to do it!

Giovanni Viglietta - FUN with algorithms 2012

The Hardness of the Lemmings Game. or Oh no, more NP-Completeness Proofs

Graham Cormode - FUN with algorithms 2007

And very recently...

Lemmings is PSPACE-Complete

Giovanni Viglietta - FUN with algorithms 2014