

# Complexité CM10

Antonio E. Porreca

[aeporreca.org](http://aeporreca.org)

Le jeu vidéo  
**Super Plombiers Italiens**  
est difficile



Le jeu vidéo  
**Super Plombiers Italiens**  
est **NP-difficile** !



# Références

- Greg Aloupisa, Erik D. Demaine, Alan Guob, Giovanni Viglietta, **Classic Nintendo games are (computationally) hard**, Theoretical Computer Science, [doi.org/10.1016/j.tcs.2015.02.037](https://doi.org/10.1016/j.tcs.2015.02.037) (version gratuite [arxiv.org/abs/1203.1895](https://arxiv.org/abs/1203.1895))
- Cory Chang (aka Undefined Behavior), **What Makes Mario NP-Hard? (Polynomial Reductions)**, [youtu.be/oS8m9fSk-Wk](https://youtu.be/oS8m9fSk-Wk)



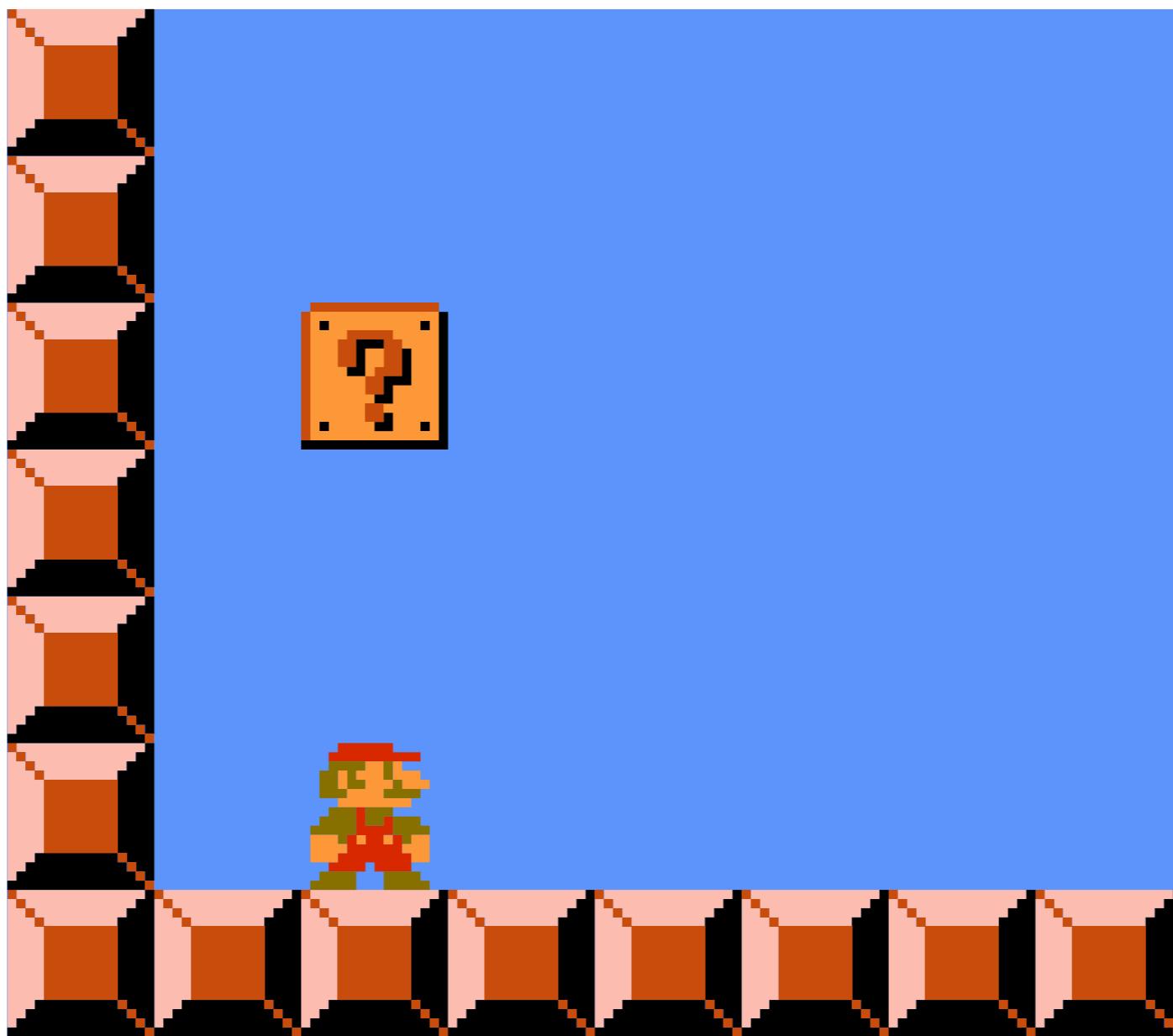
# Définition du problème

- Entrée : le plan d'un niveau de Super Plombiers Italiens
- Question : est-il possible d'atteindre la fin du niveau ?

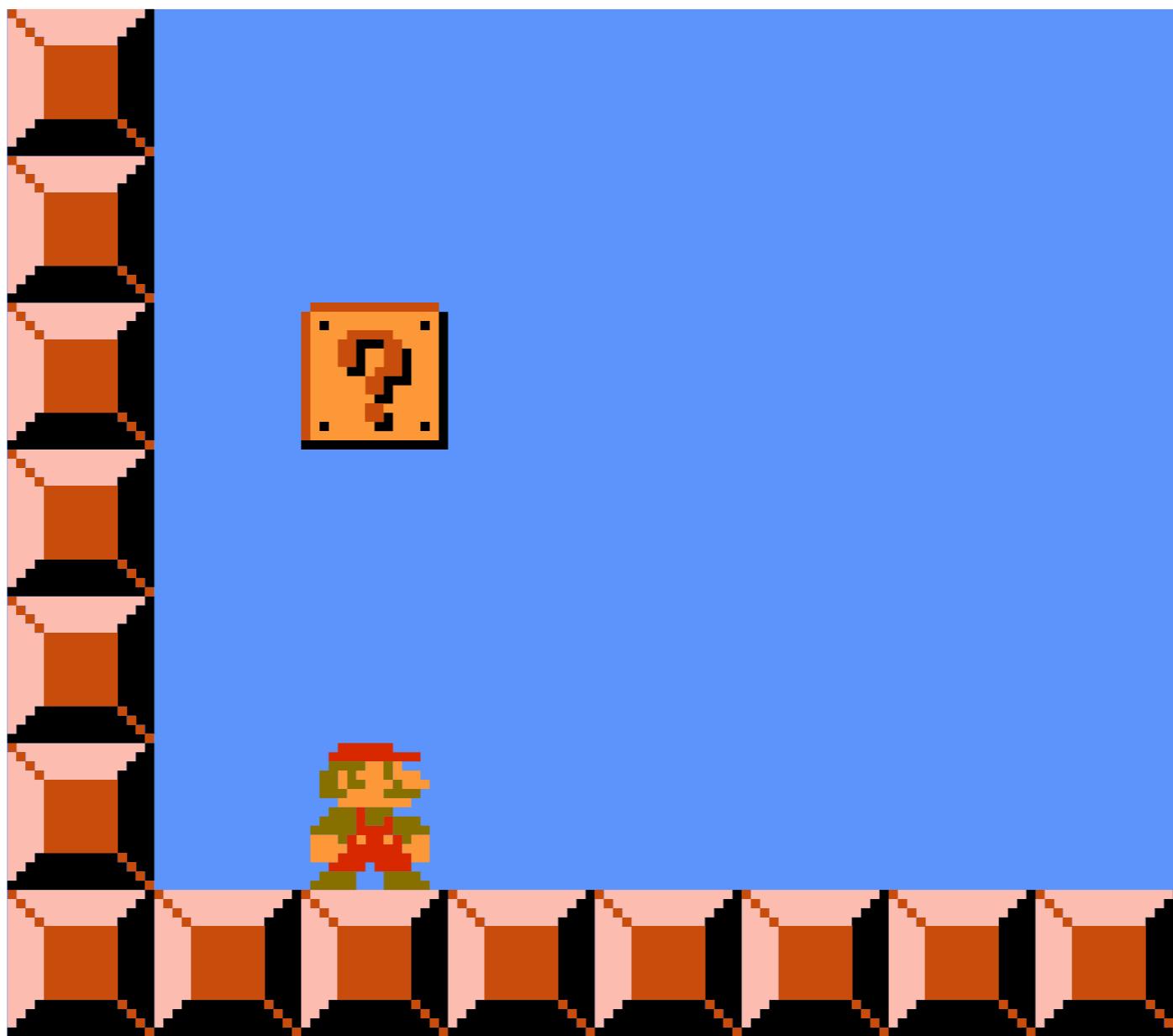
**3SAT** <



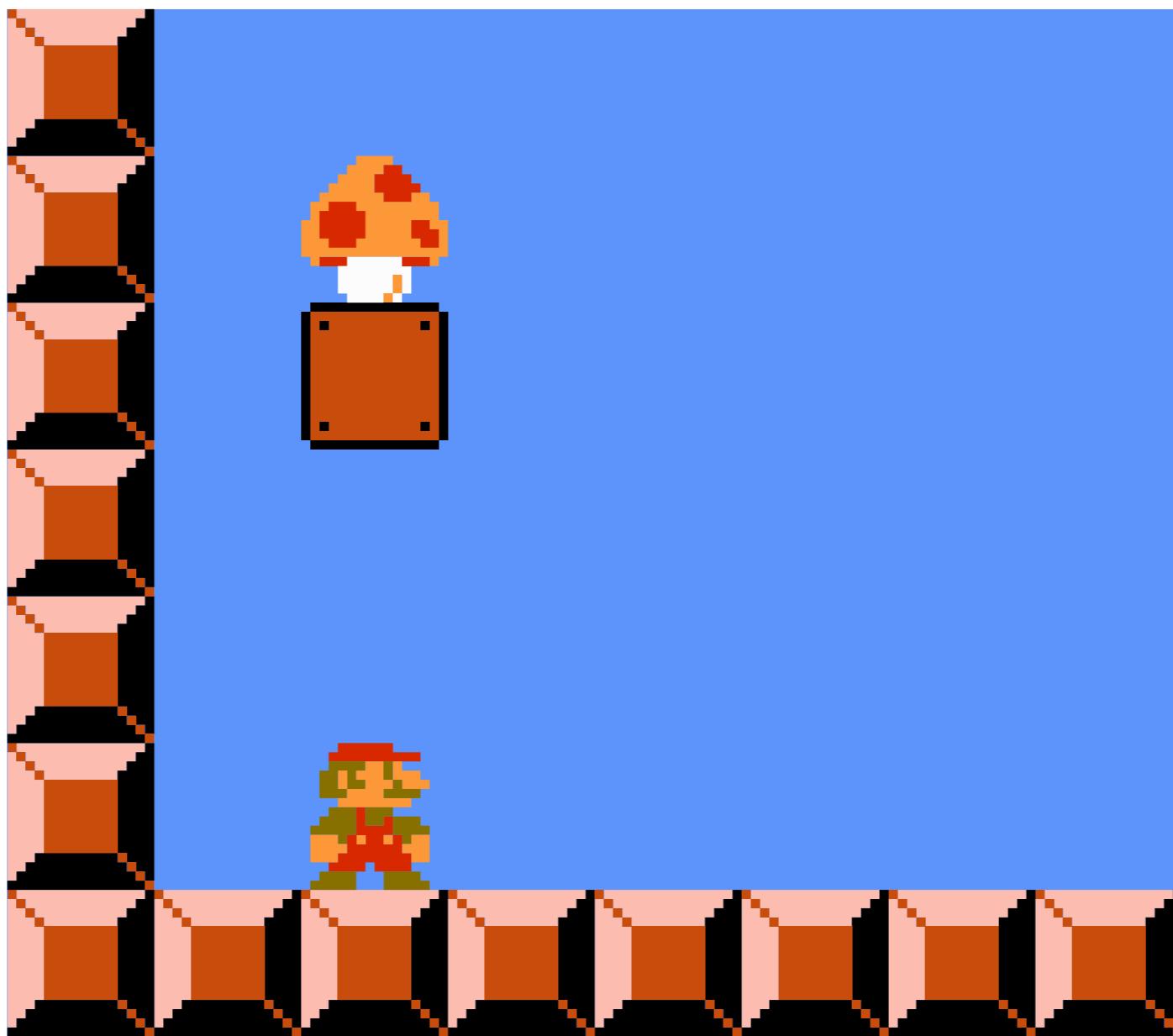
# Gadget de départ



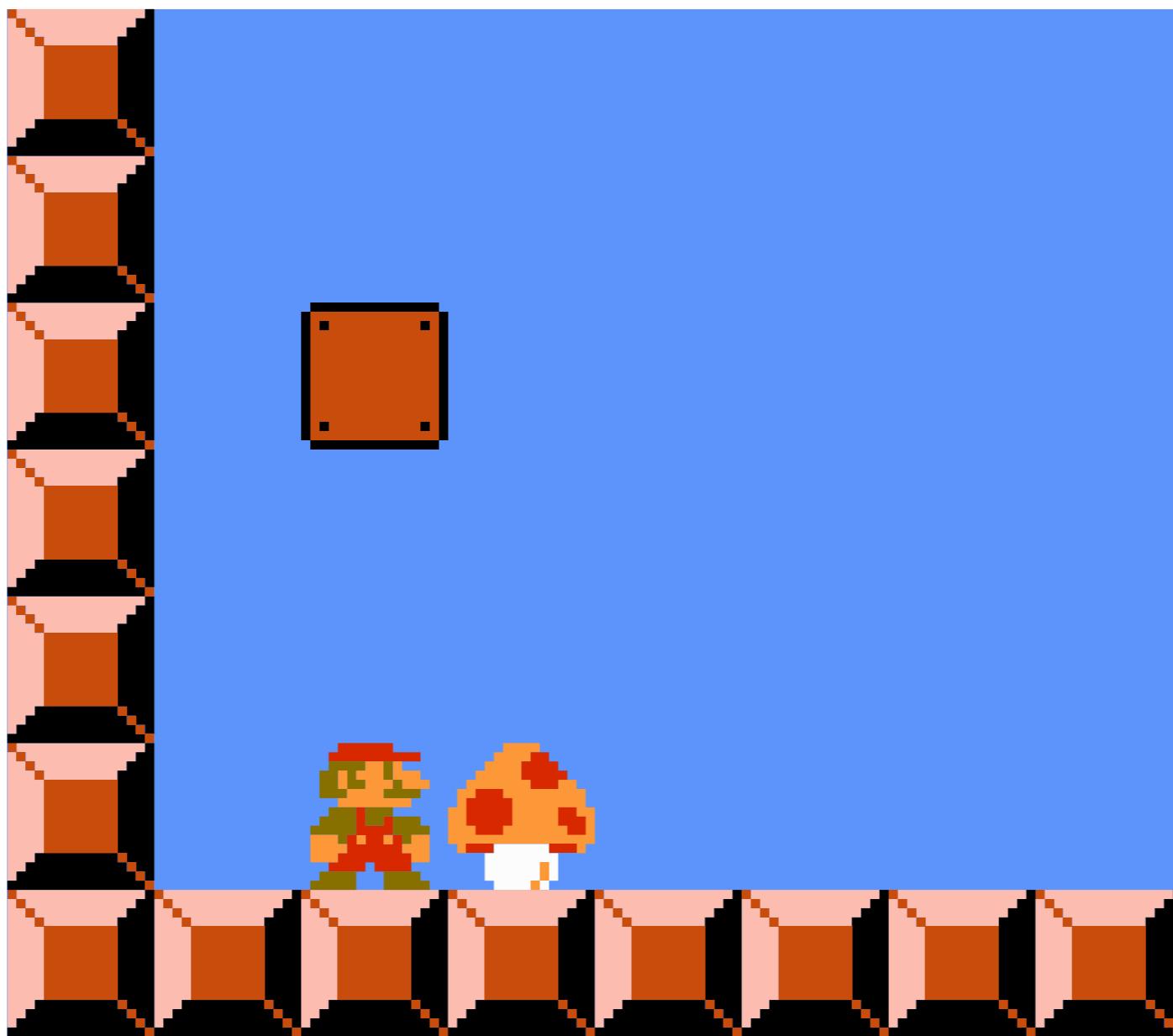
# Gadget de départ



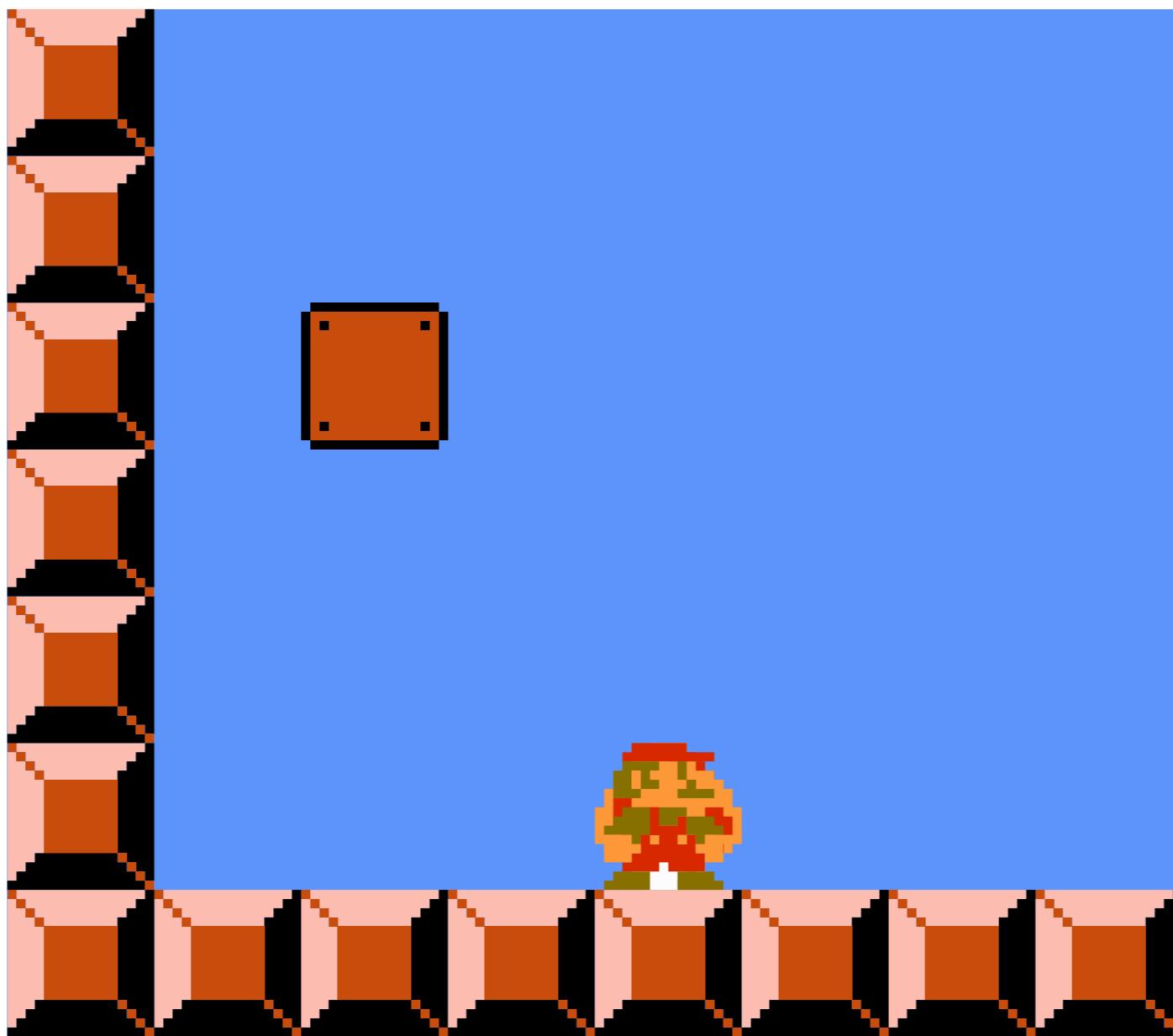
# Gadget de départ



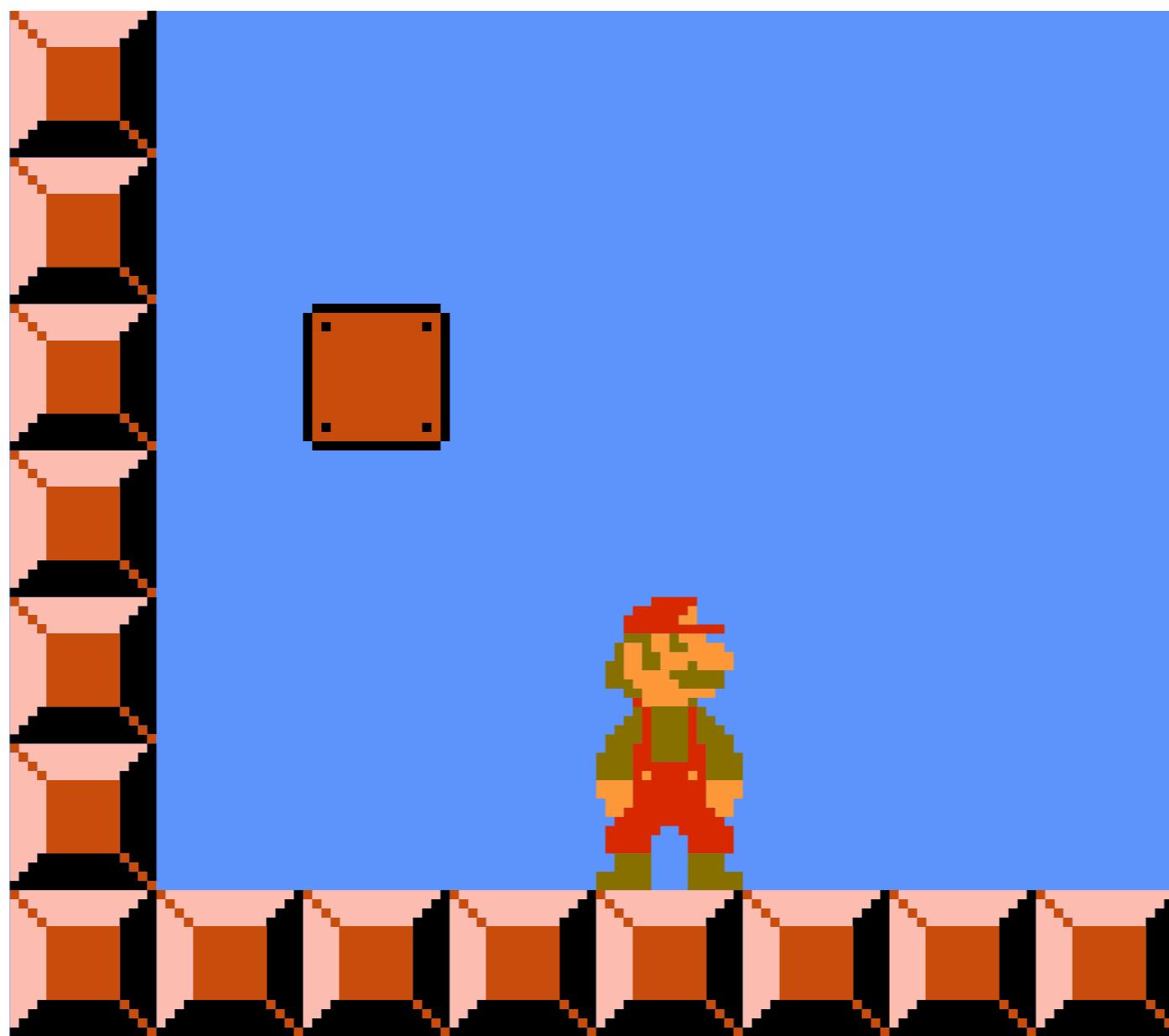
# Gadget de départ



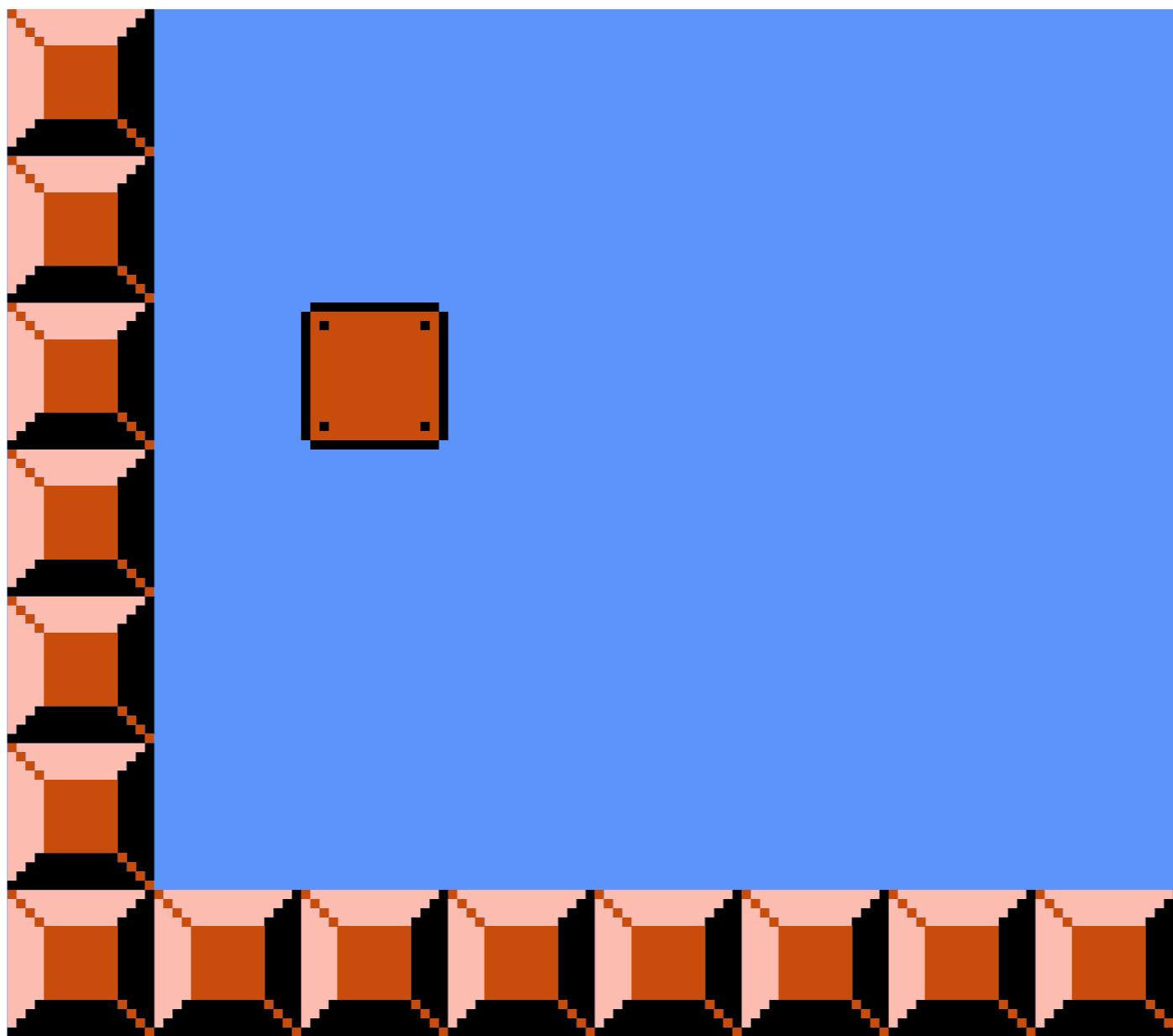
# Gadget de départ



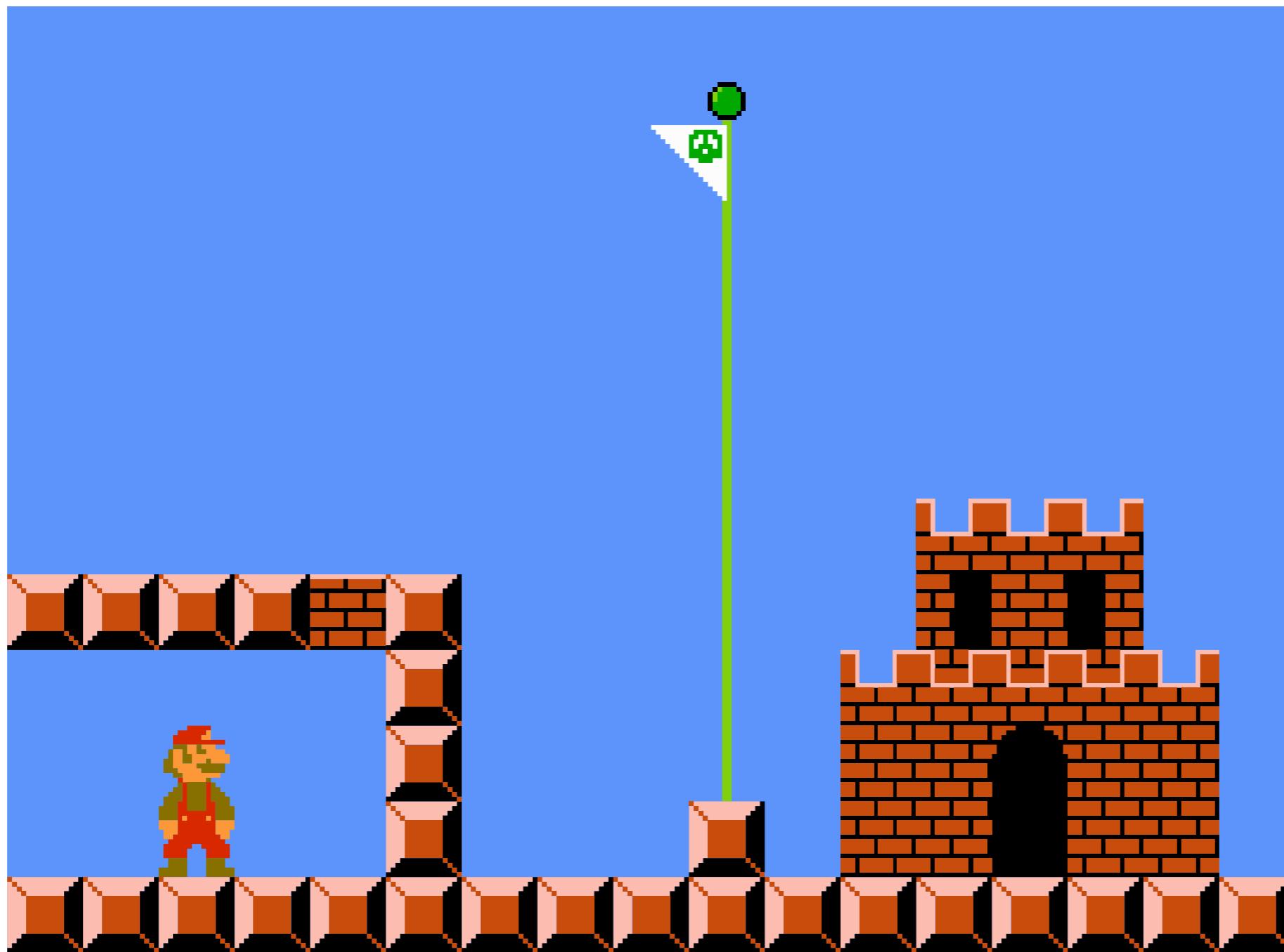
# Gadget de départ



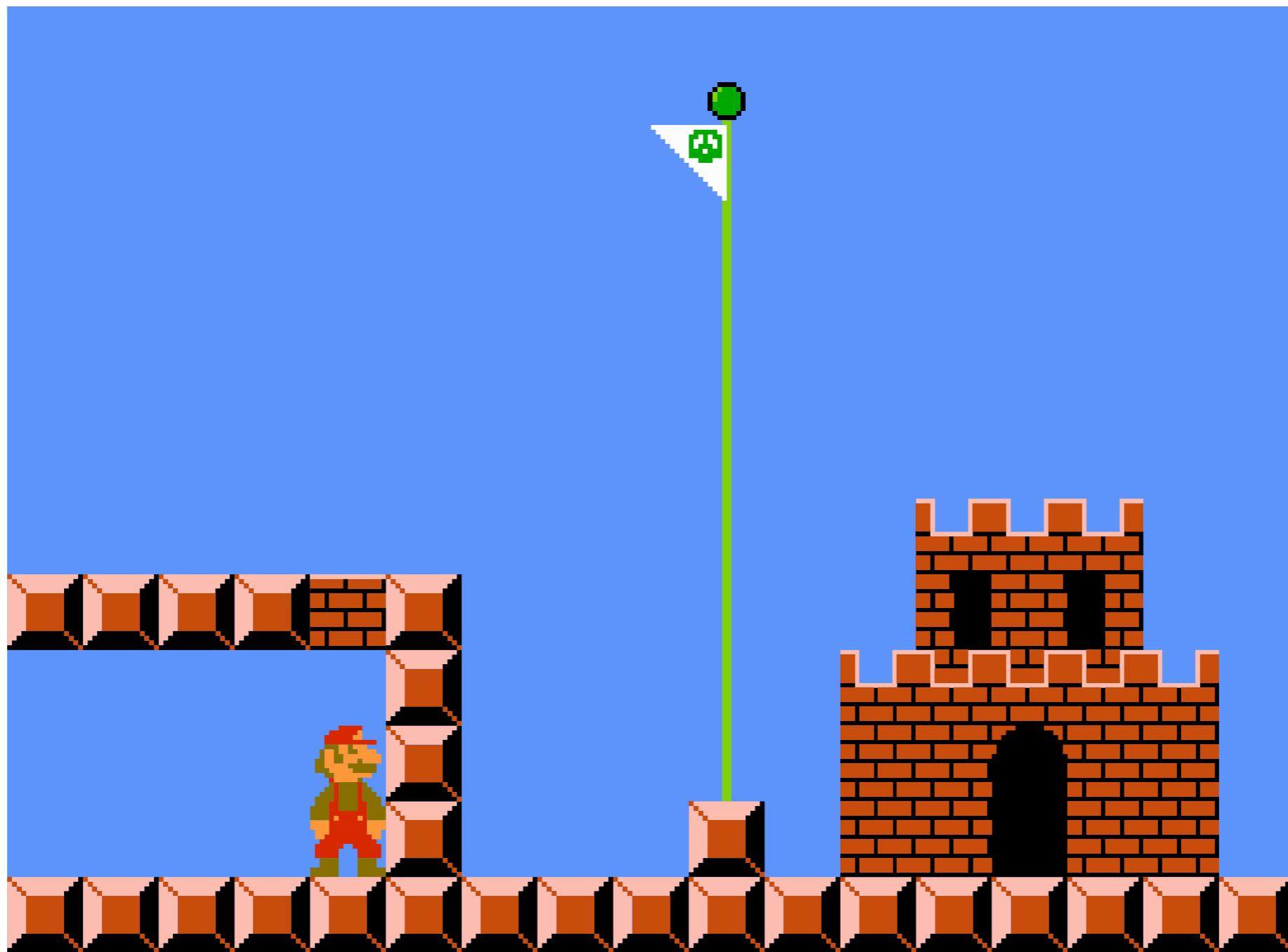
# Gadget de départ



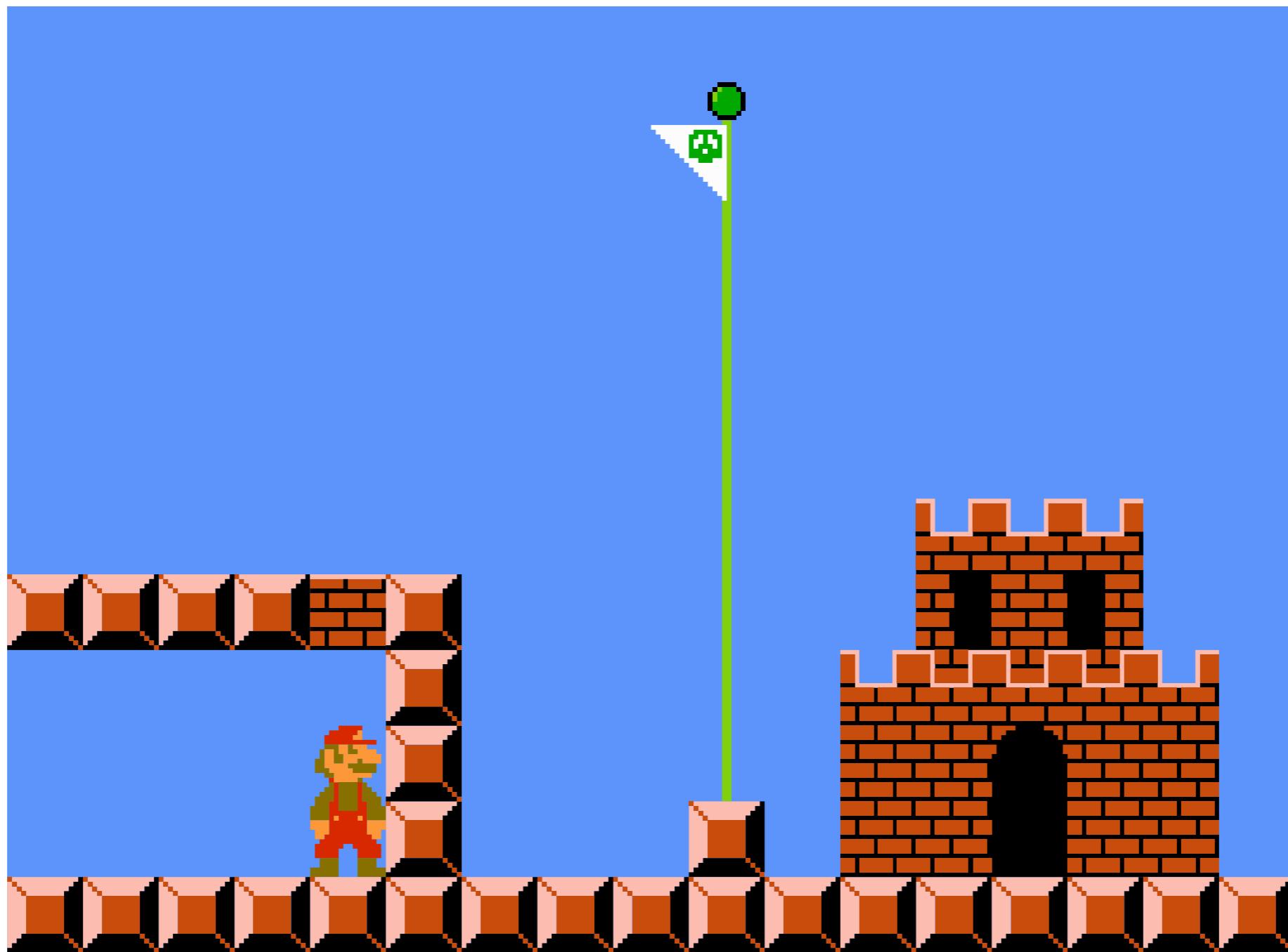
# Gadget d'arrivée



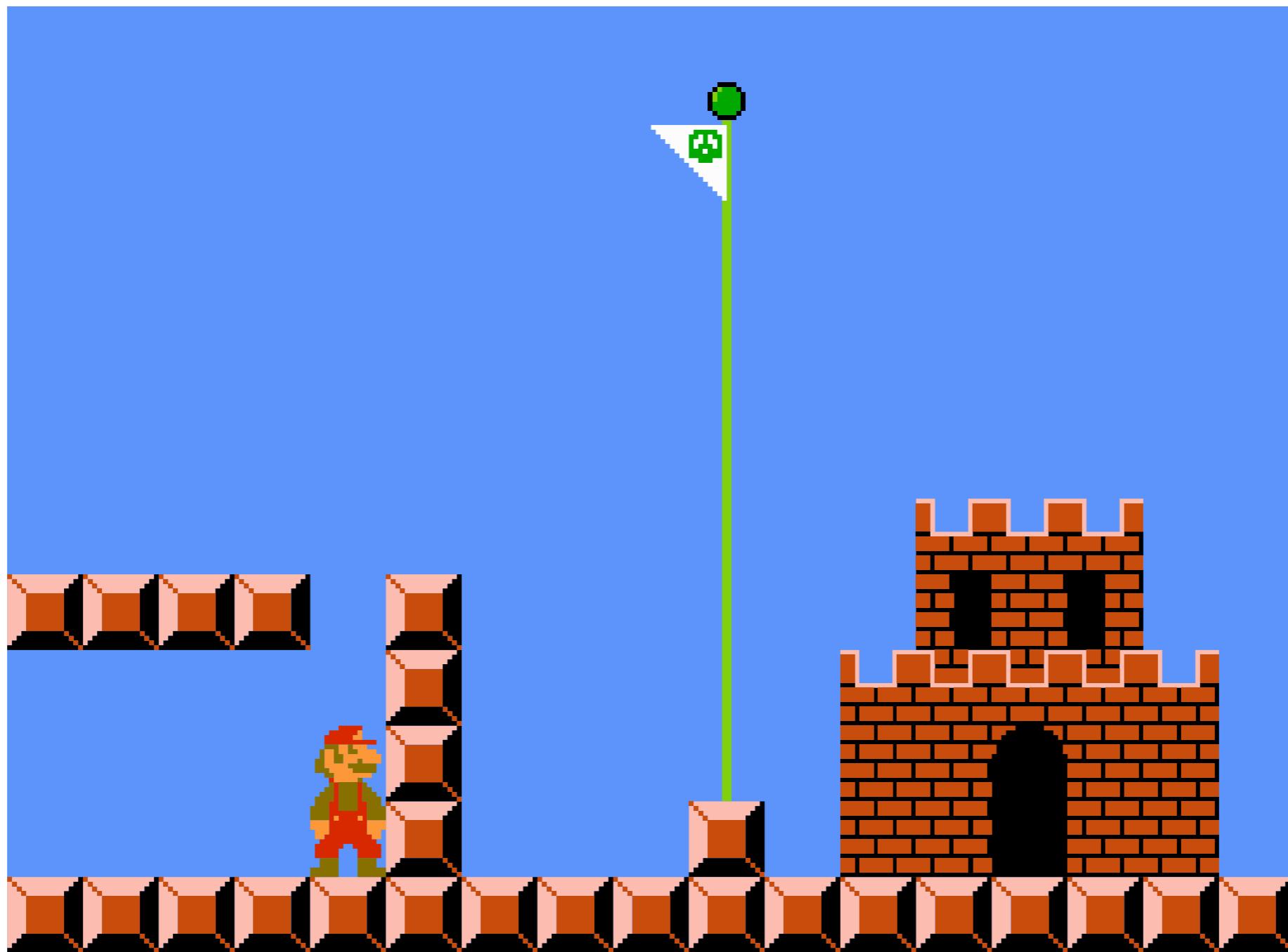
# Gadget d'arrivée



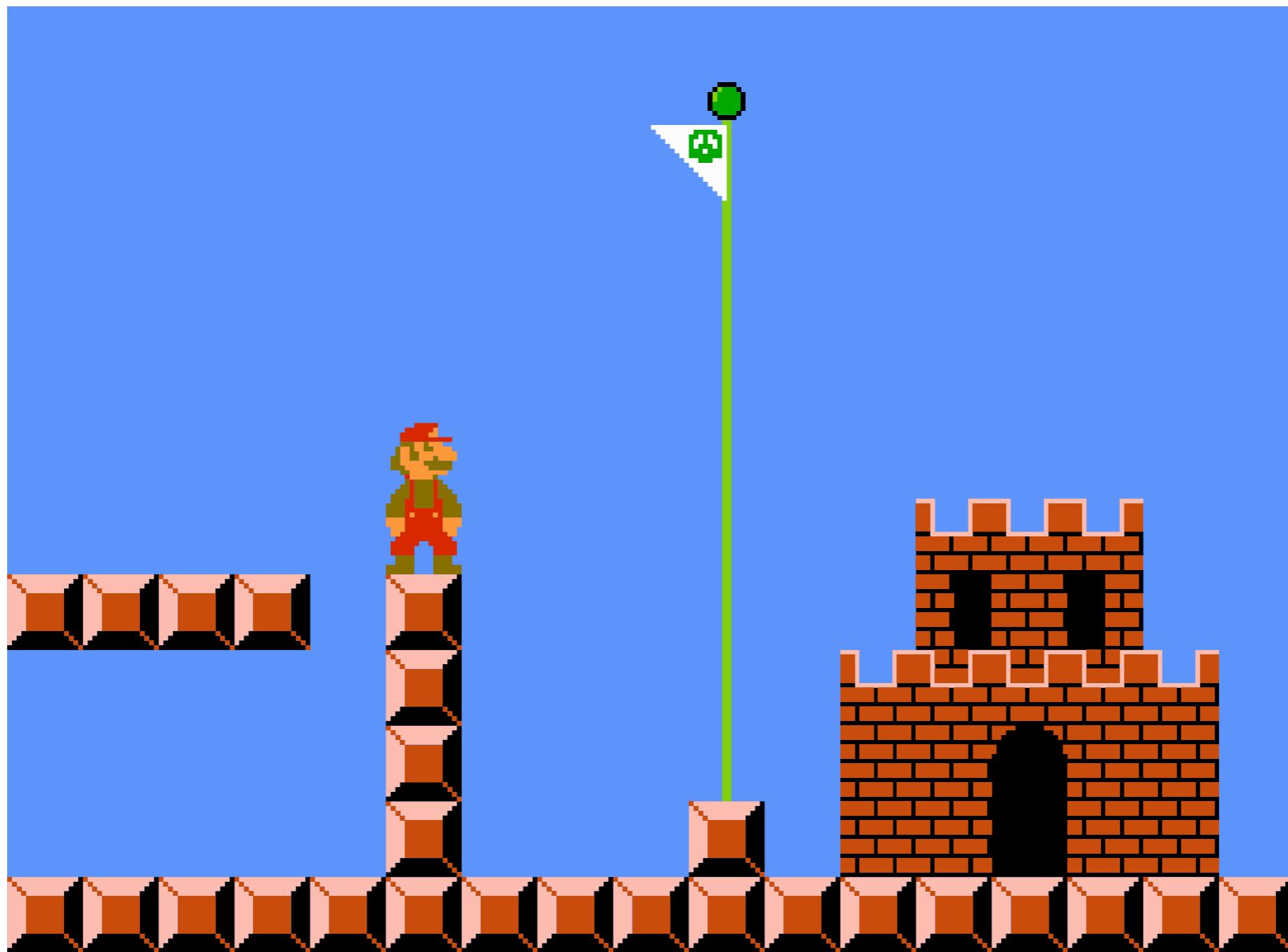
# Gadget d'arrivée



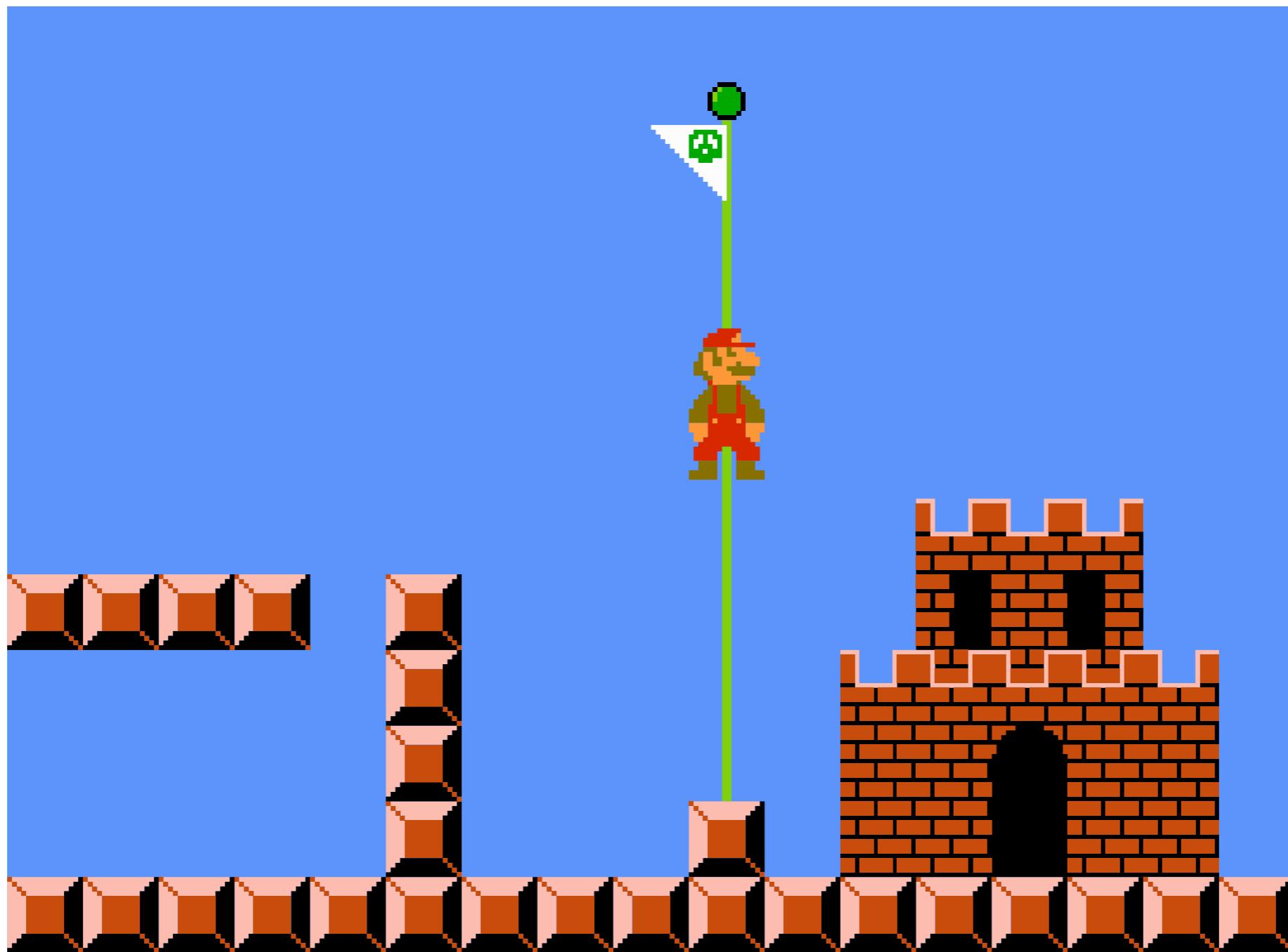
# Gadget d'arrivée



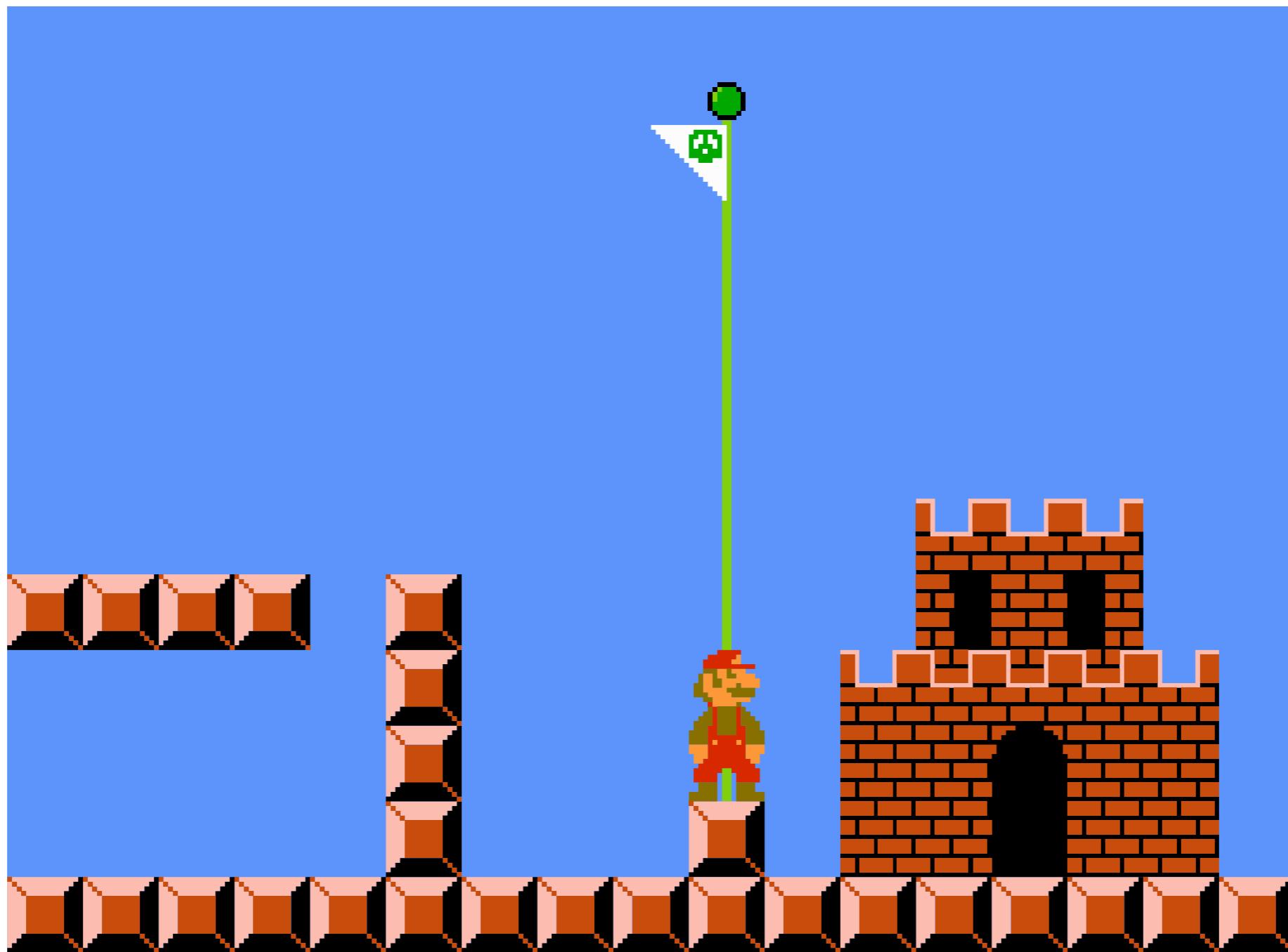
# Gadget d'arrivée



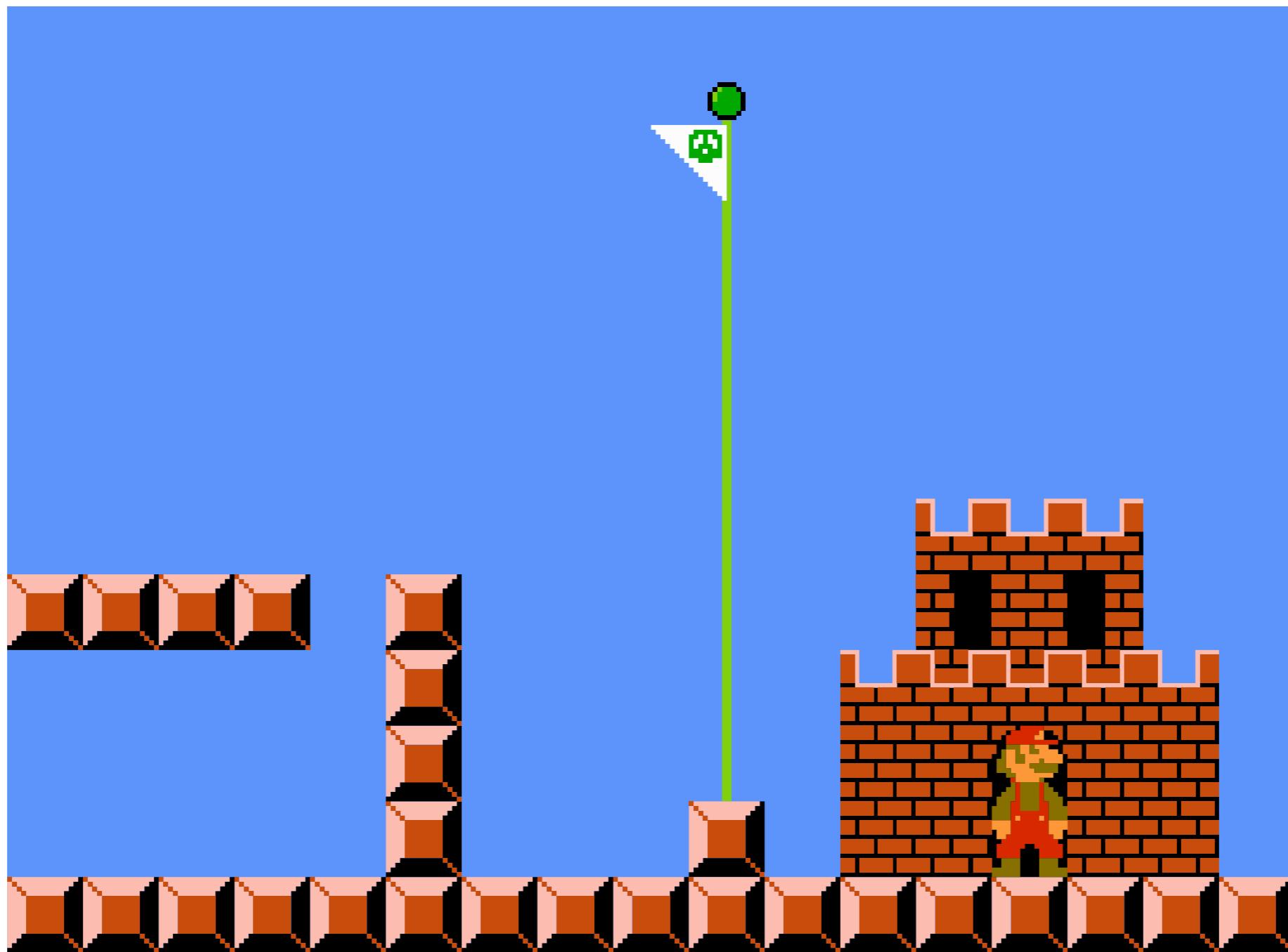
# Gadget d'arrivée



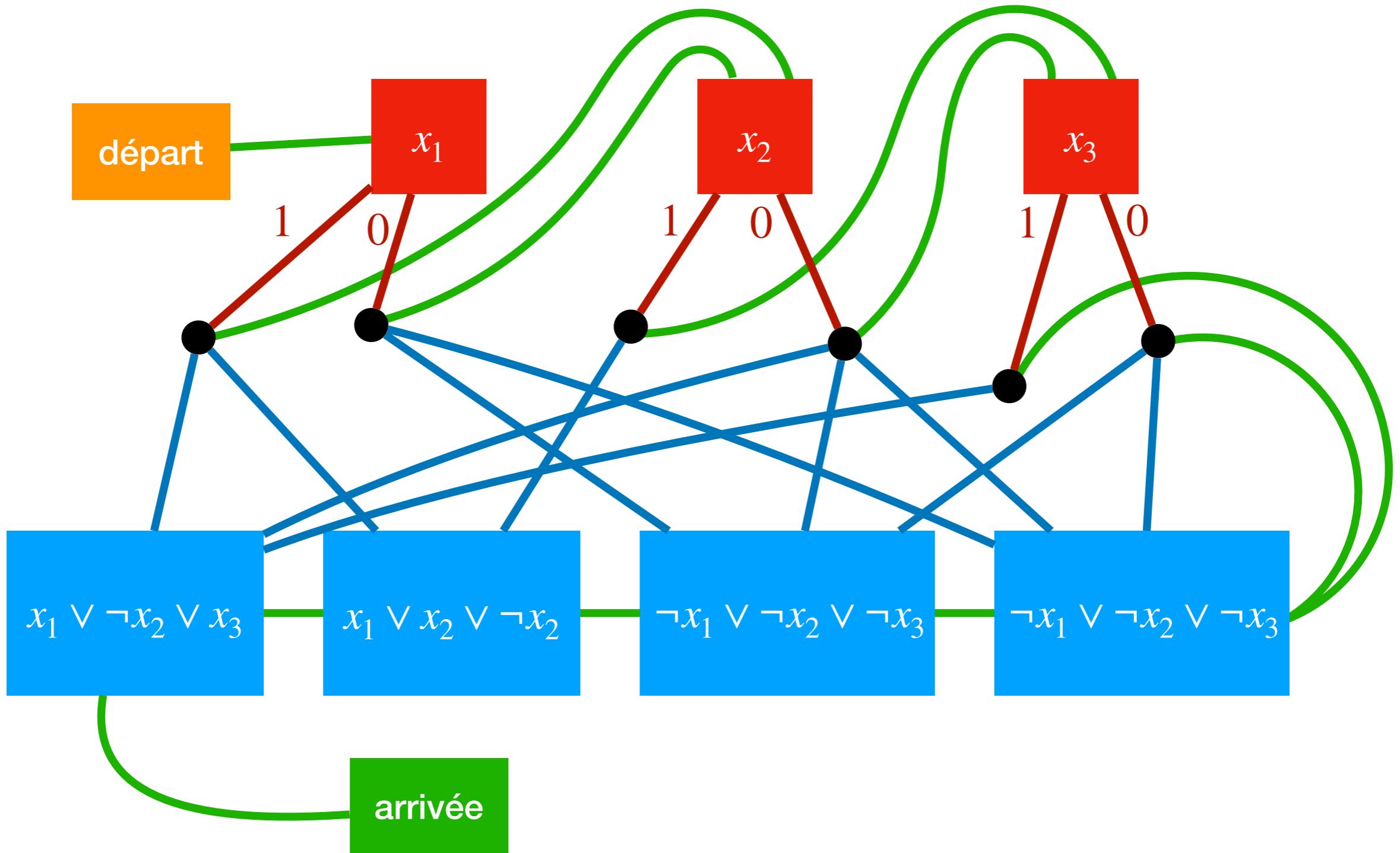
# Gadget d'arrivée



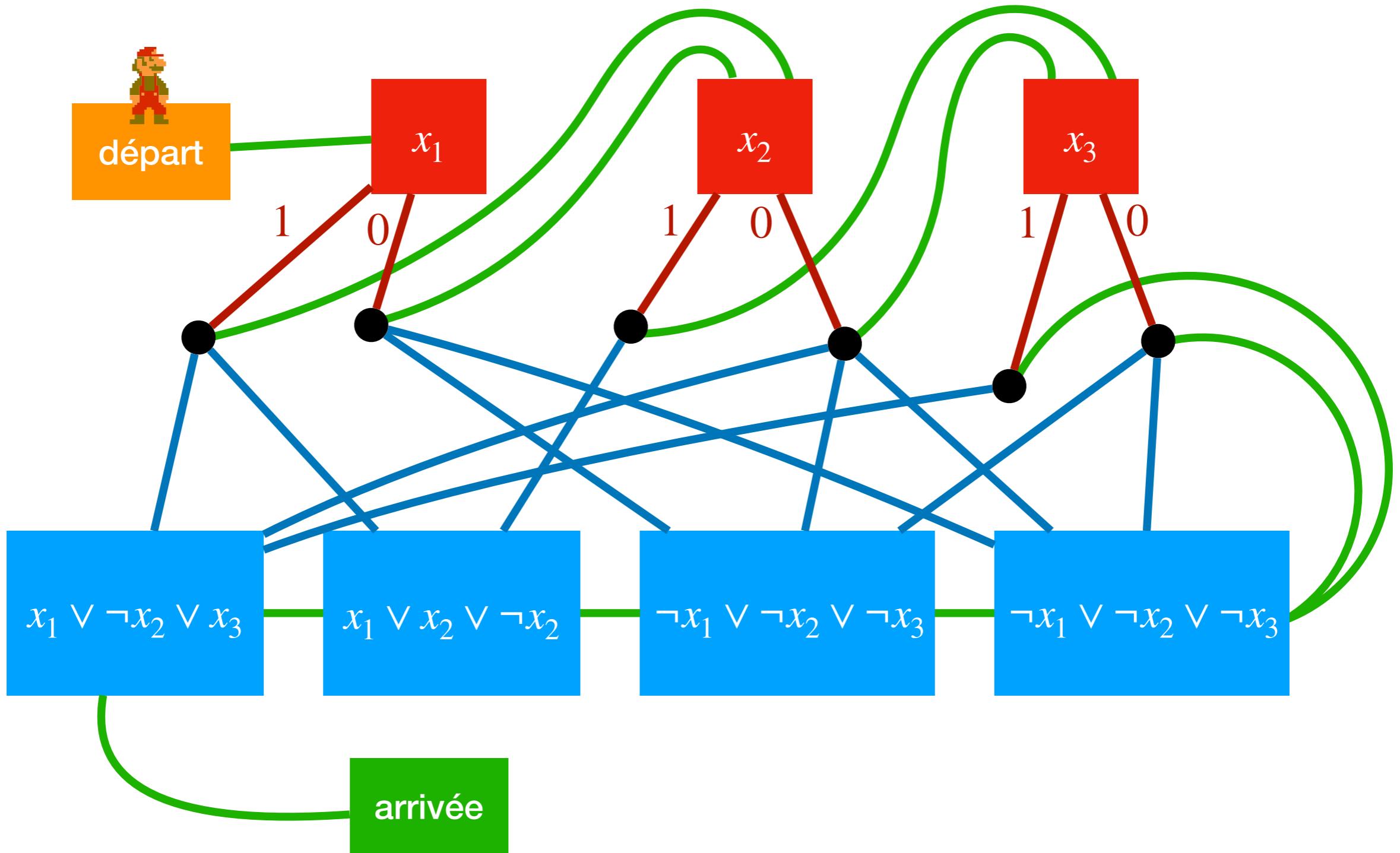
# Gadget d'arrivée



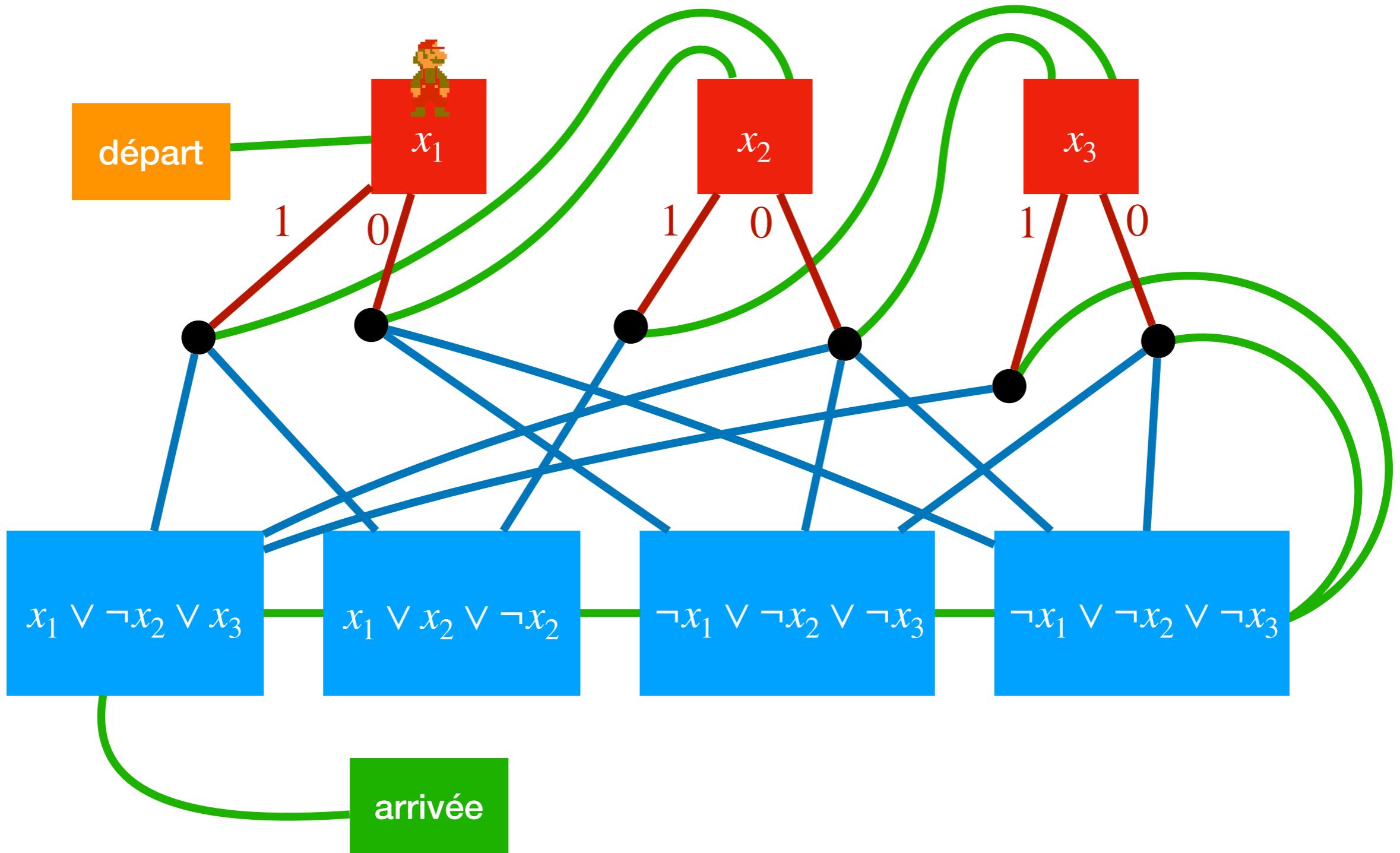
# Parcours du plombier



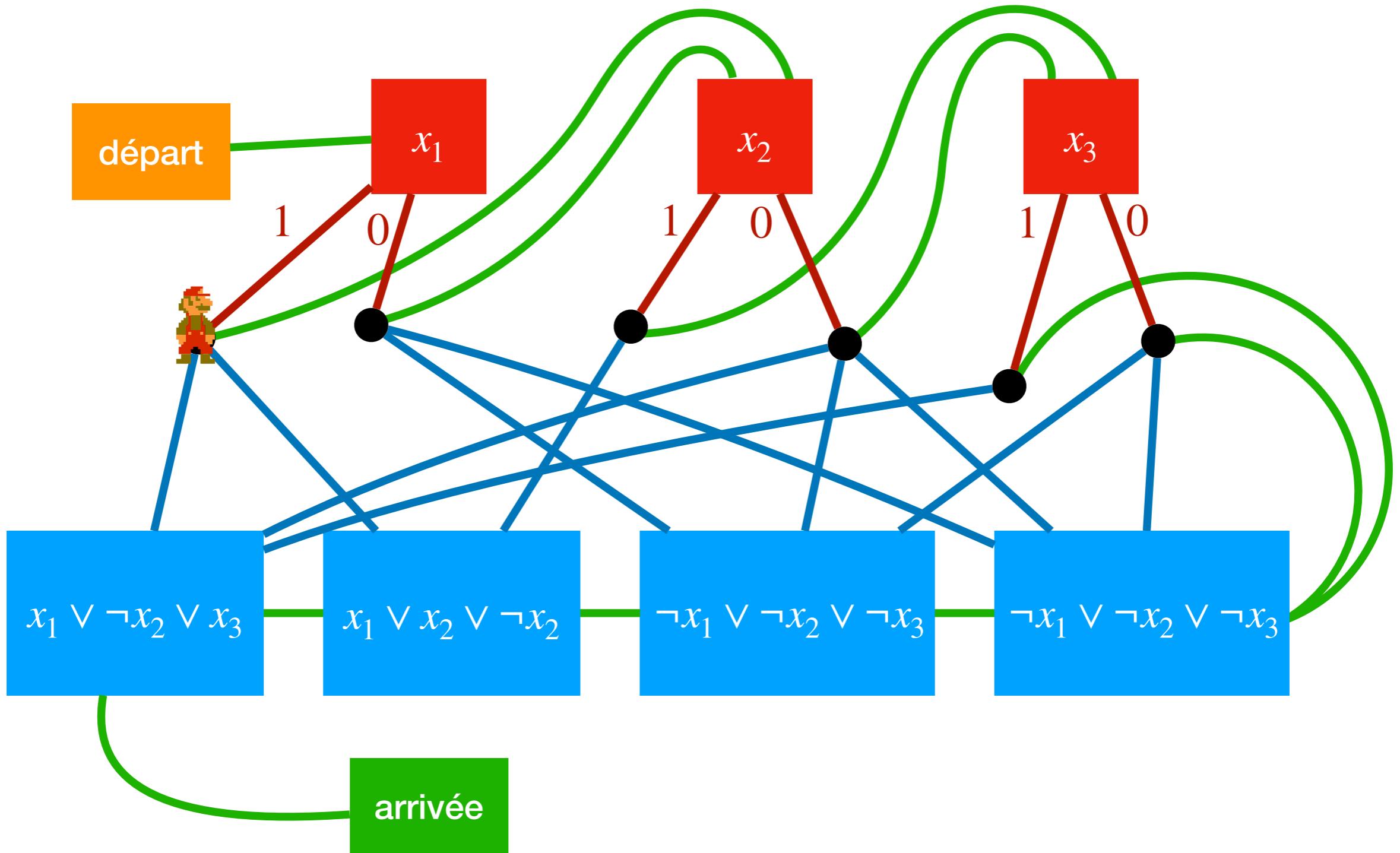
# Parcours du plombier



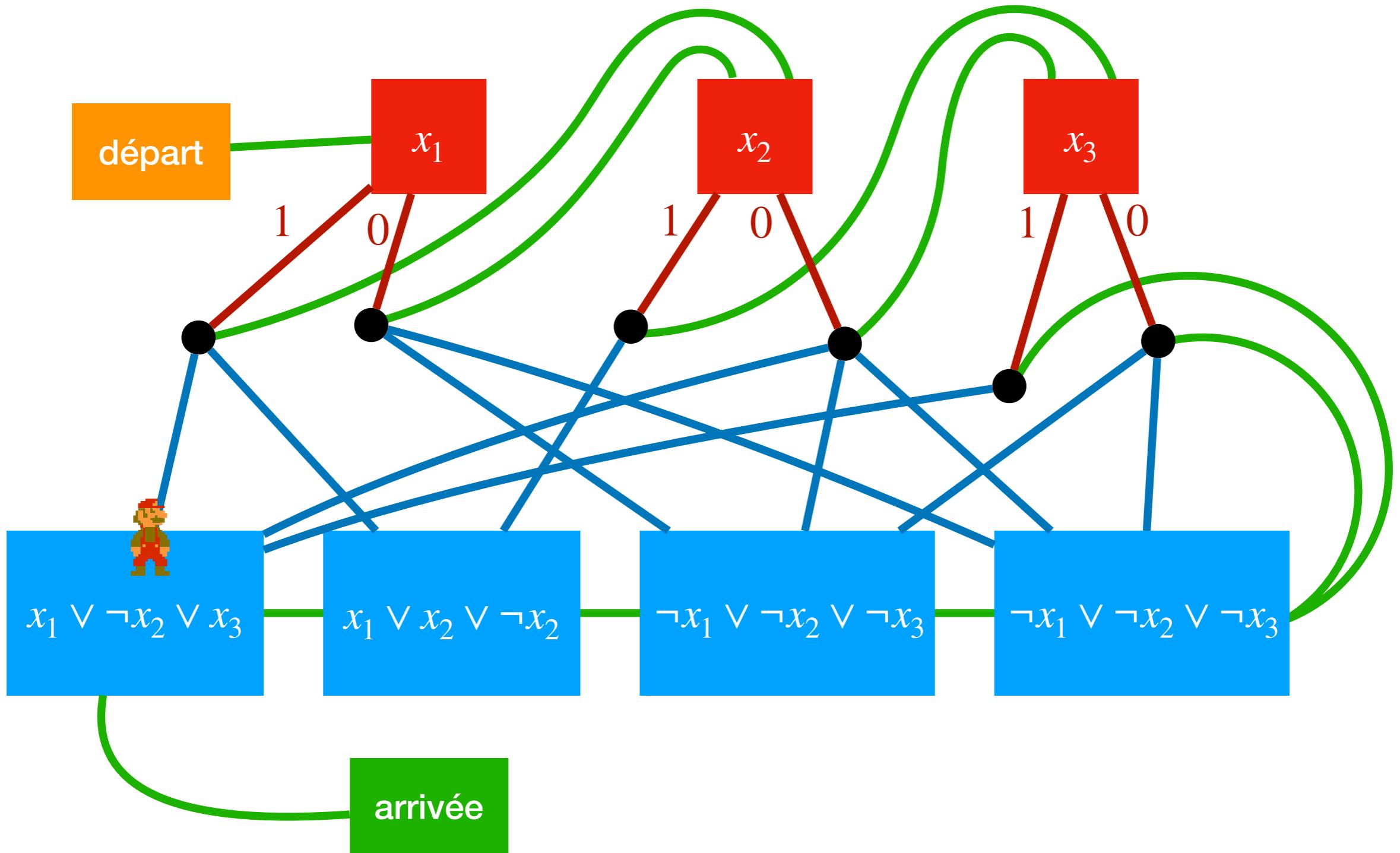
# Parcours du plombier



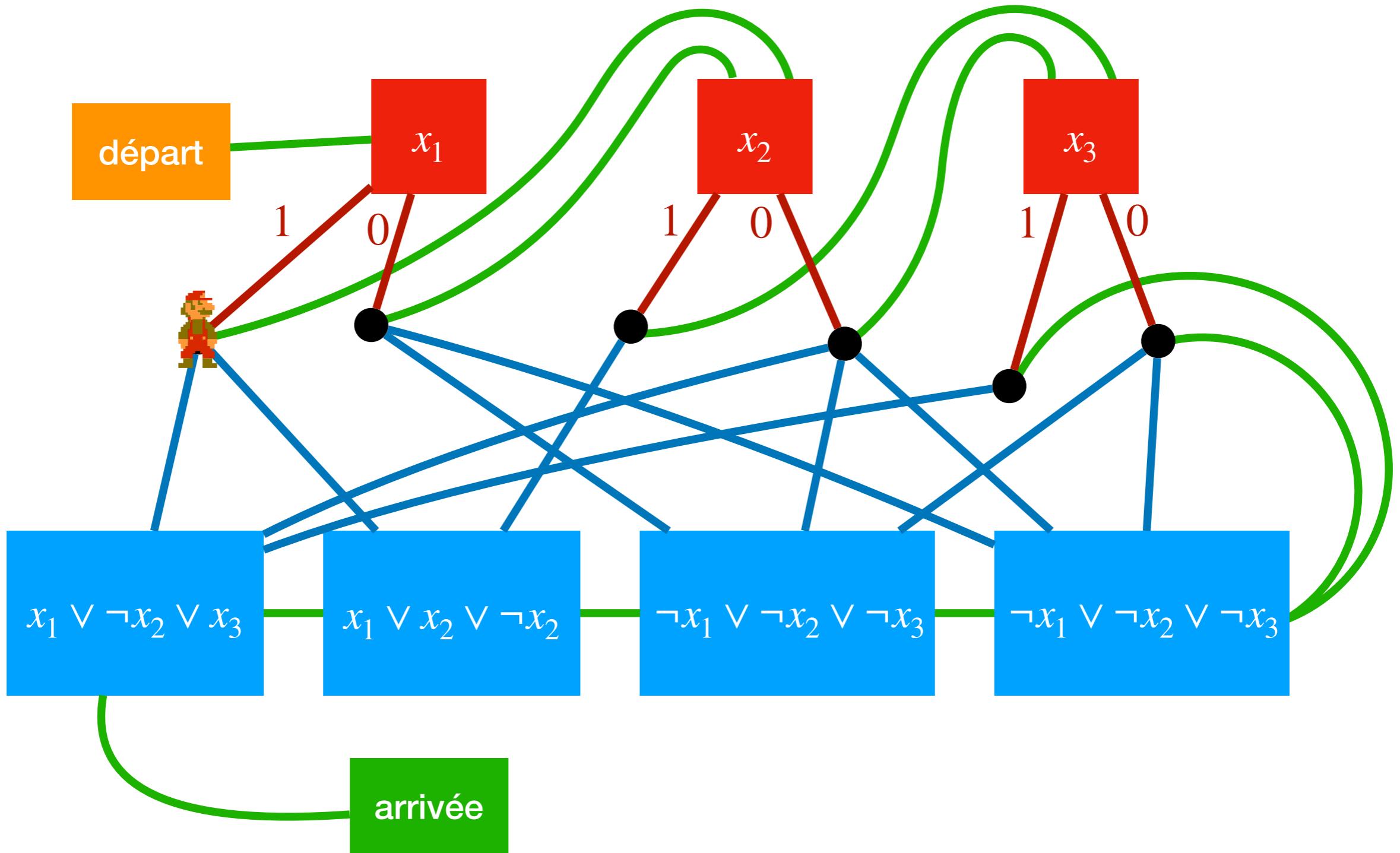
# Parcours du plombier



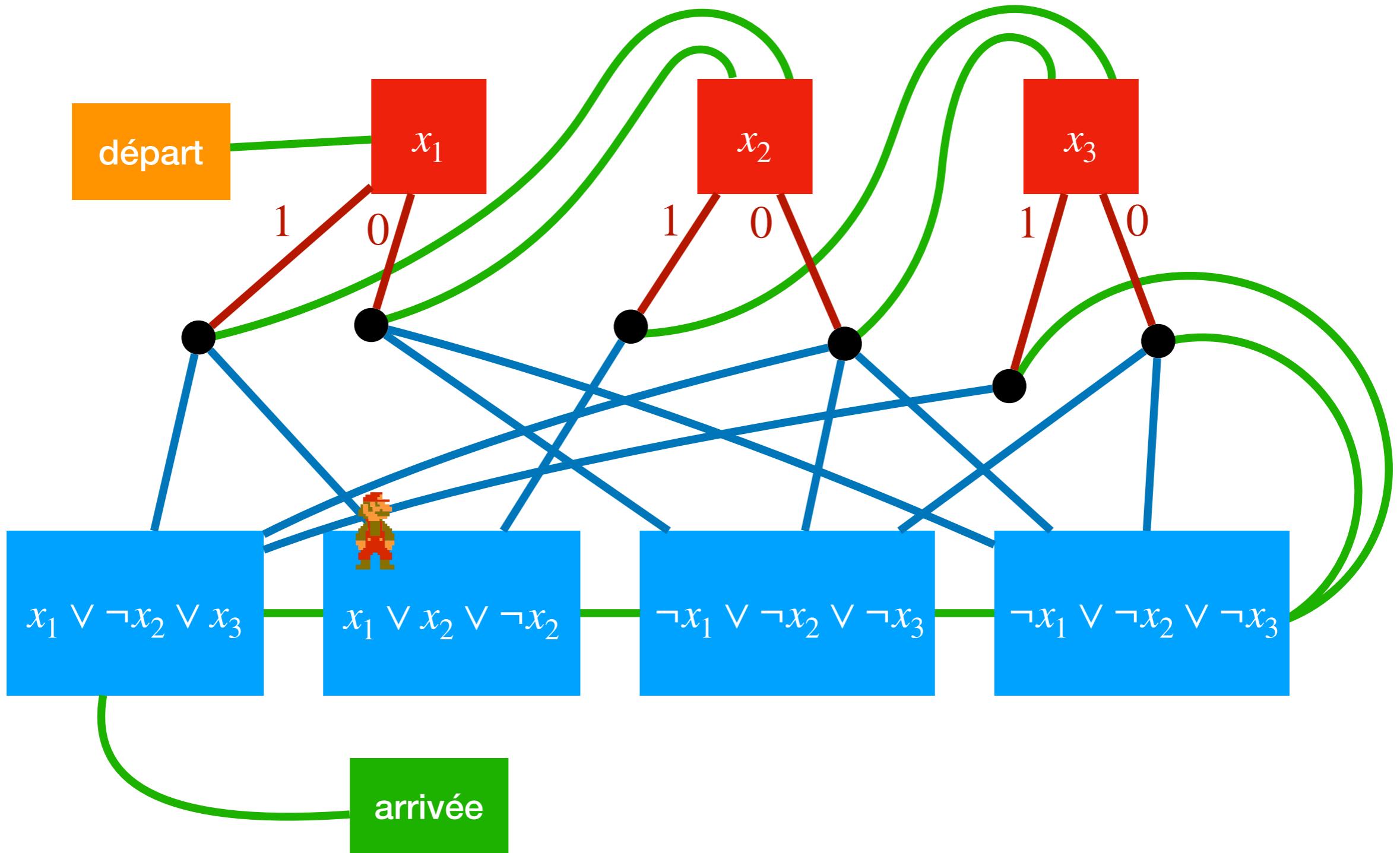
# Parcours du plombier



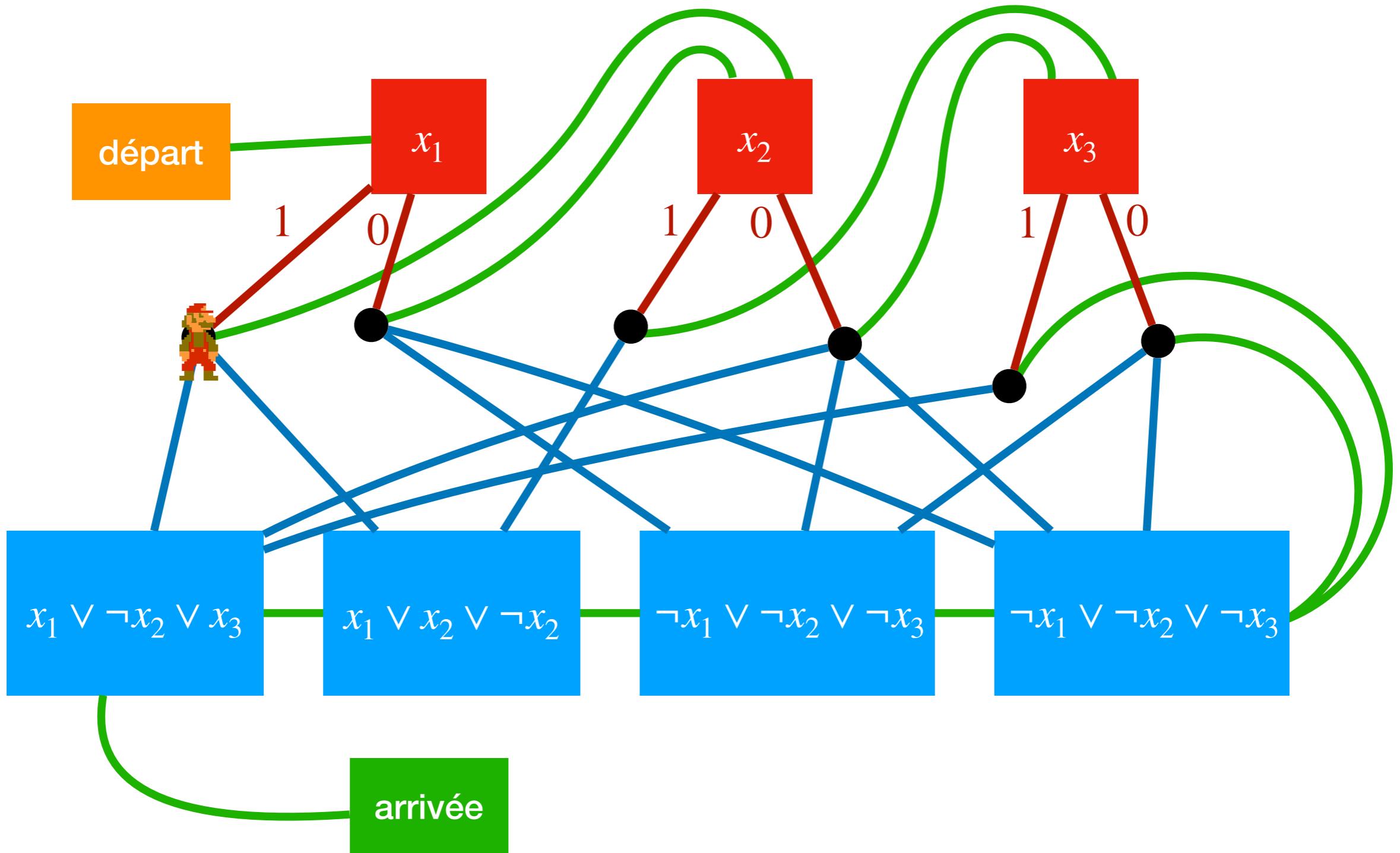
# Parcours du plombier



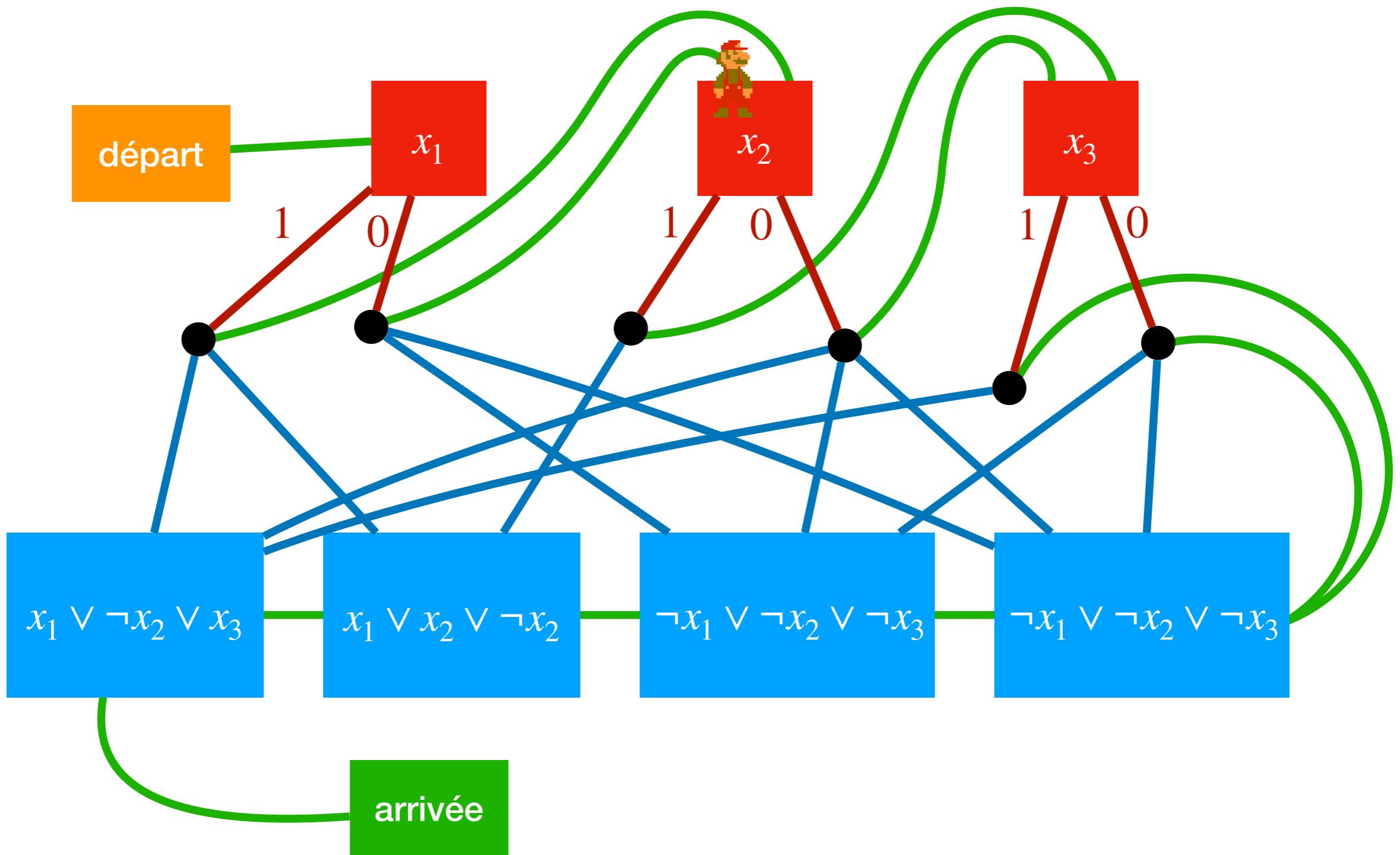
# Parcours du plombier



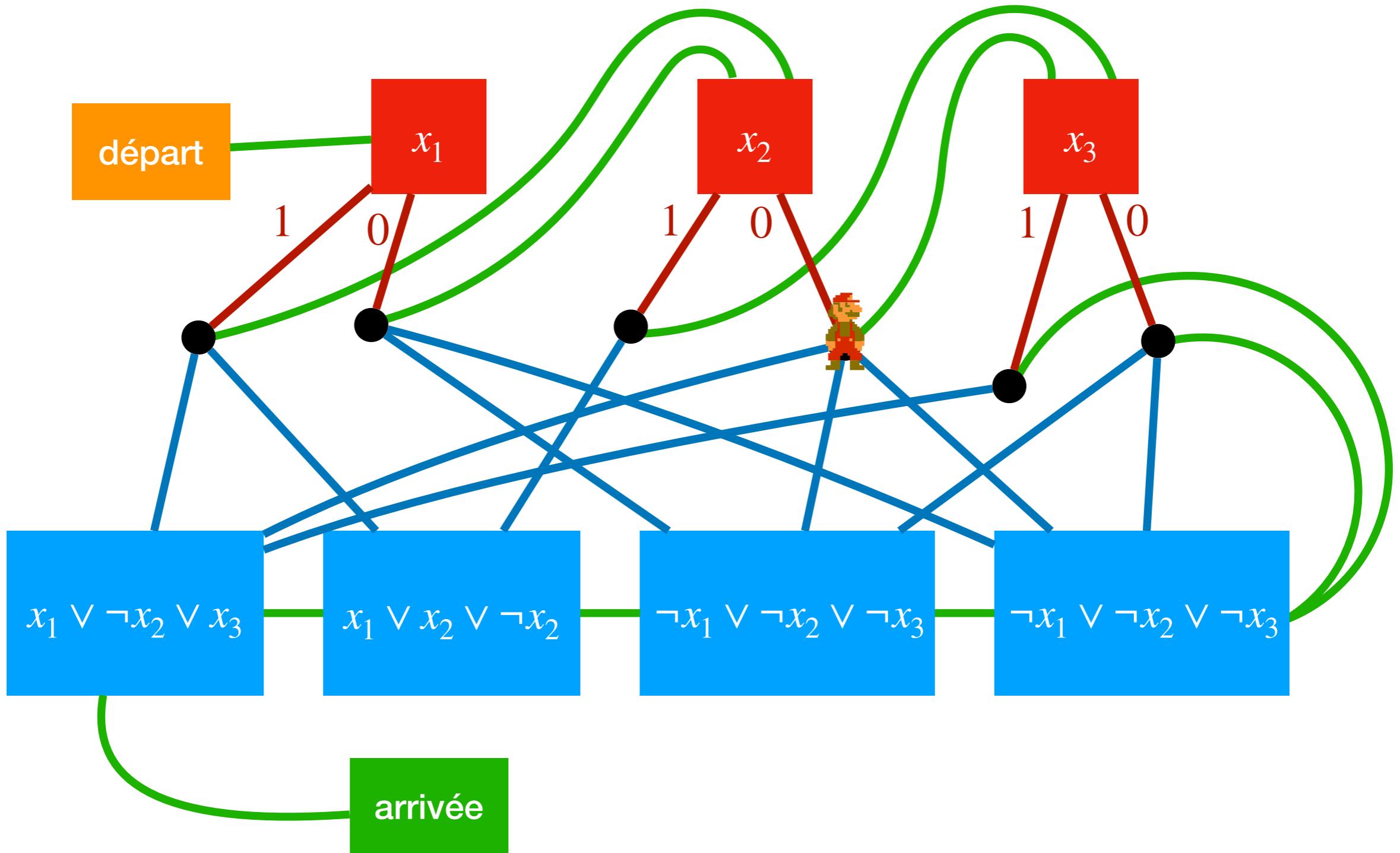
# Parcours du plombier



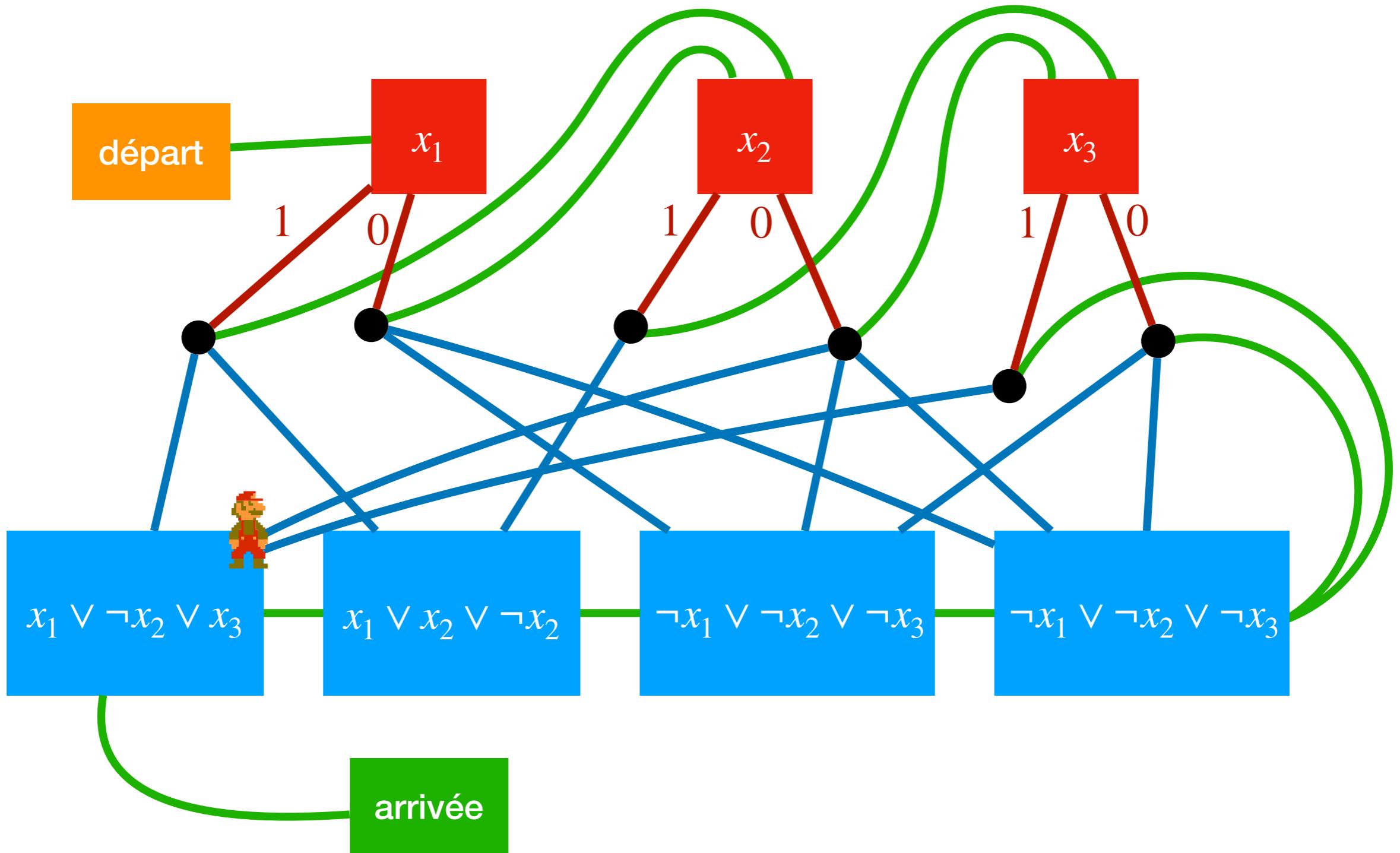
# Parcours du plombier



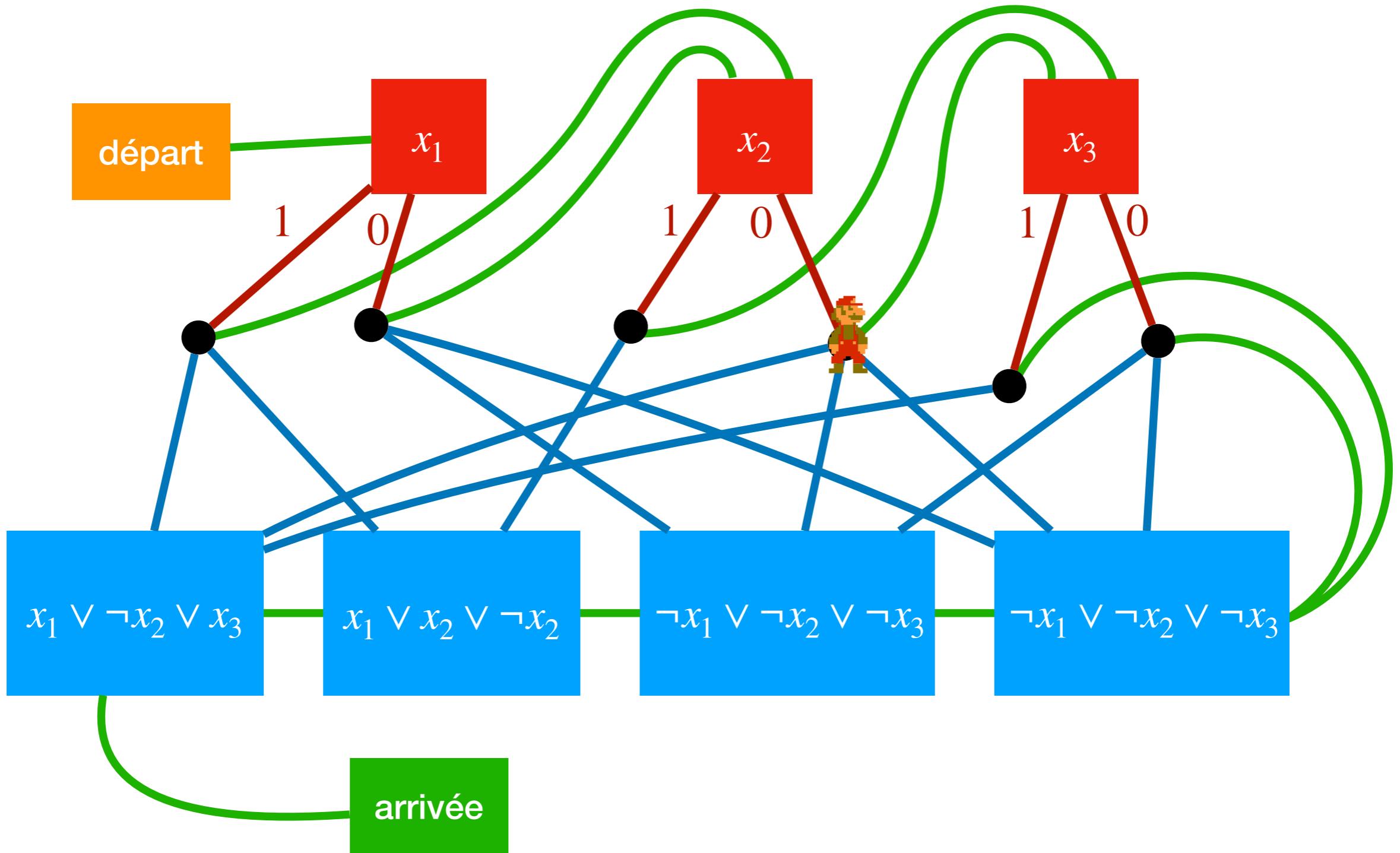
# Parcours du plombier



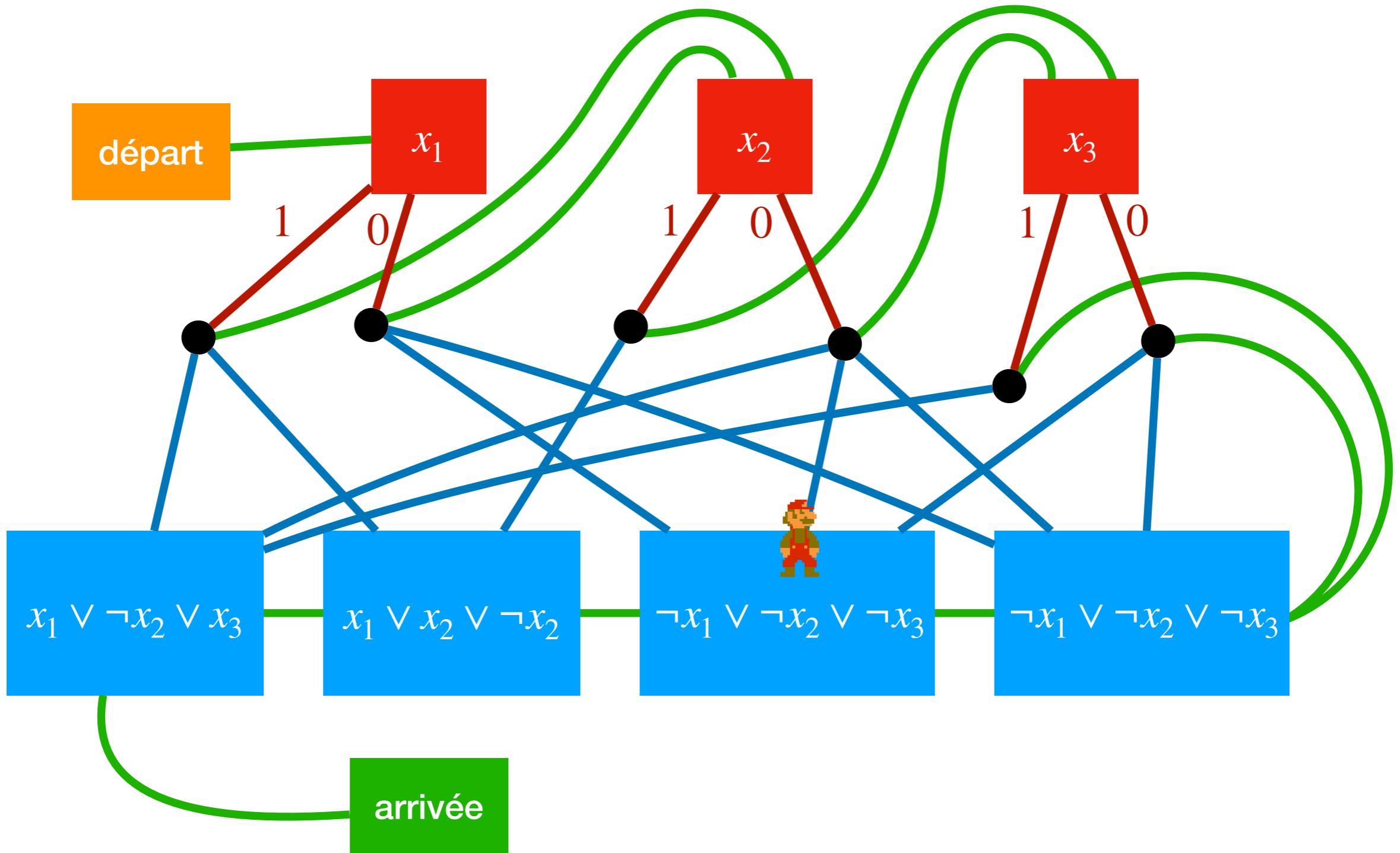
# Parcours du plombier



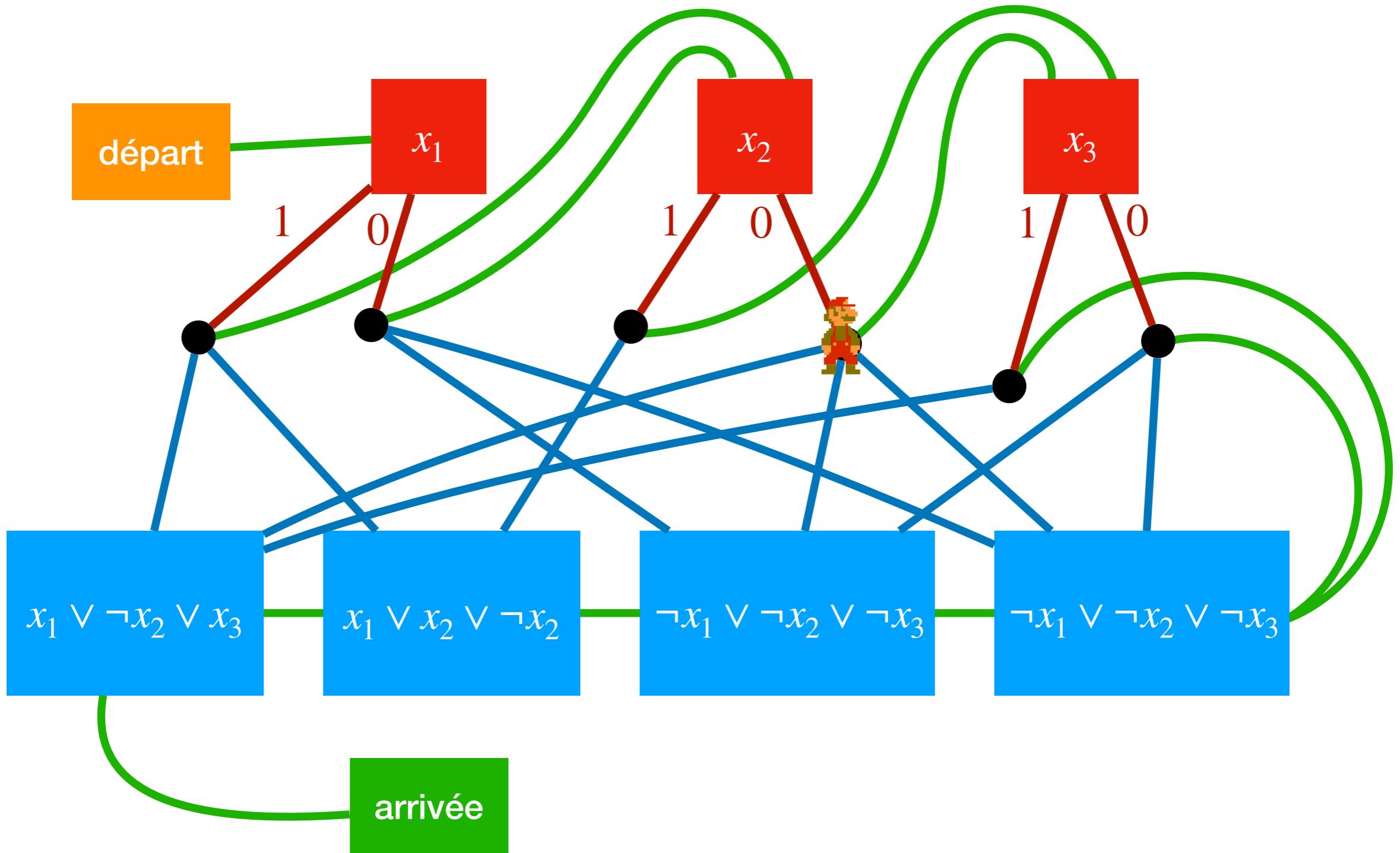
# Parcours du plombier



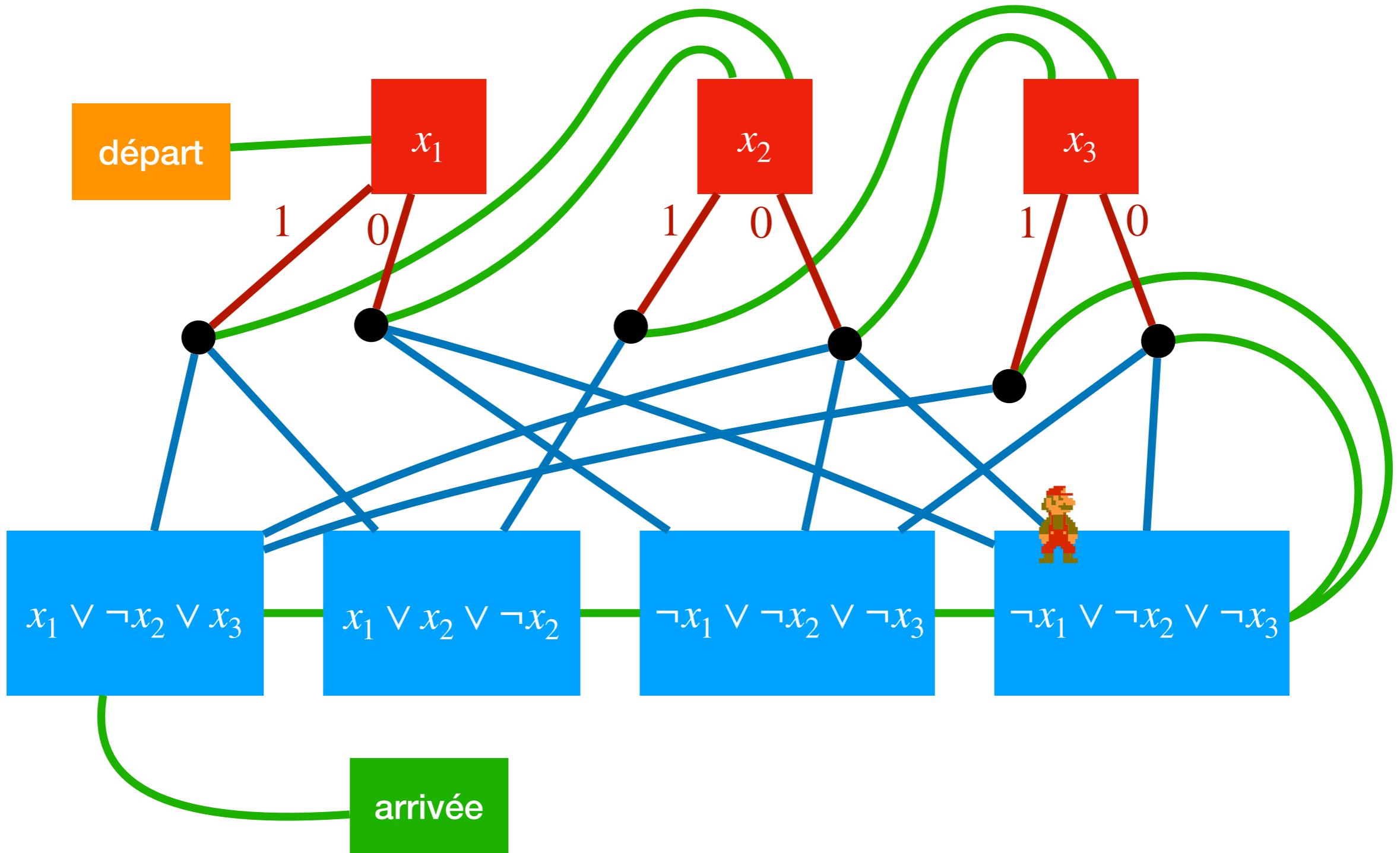
# Parcours du plombier



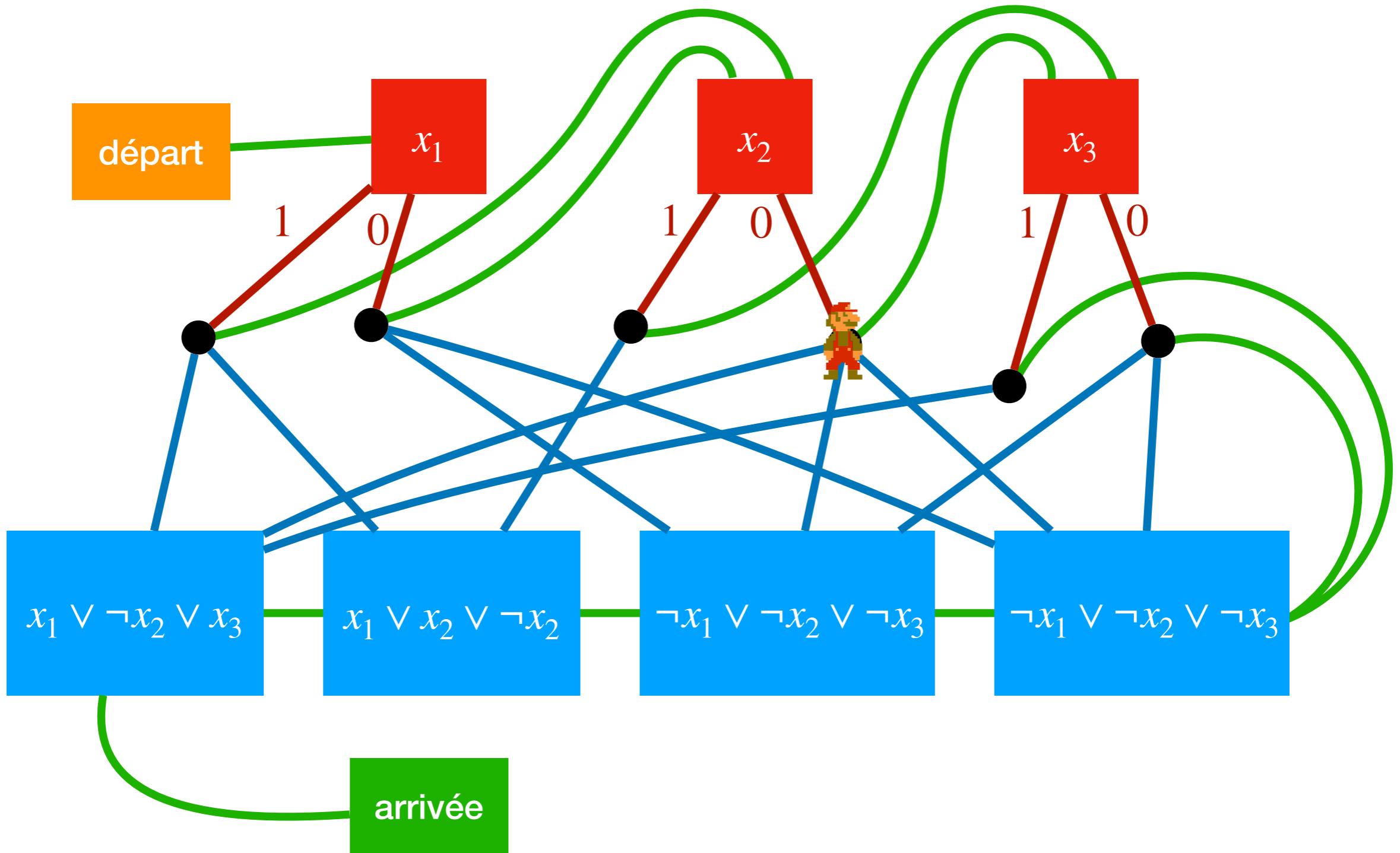
# Parcours du plombier



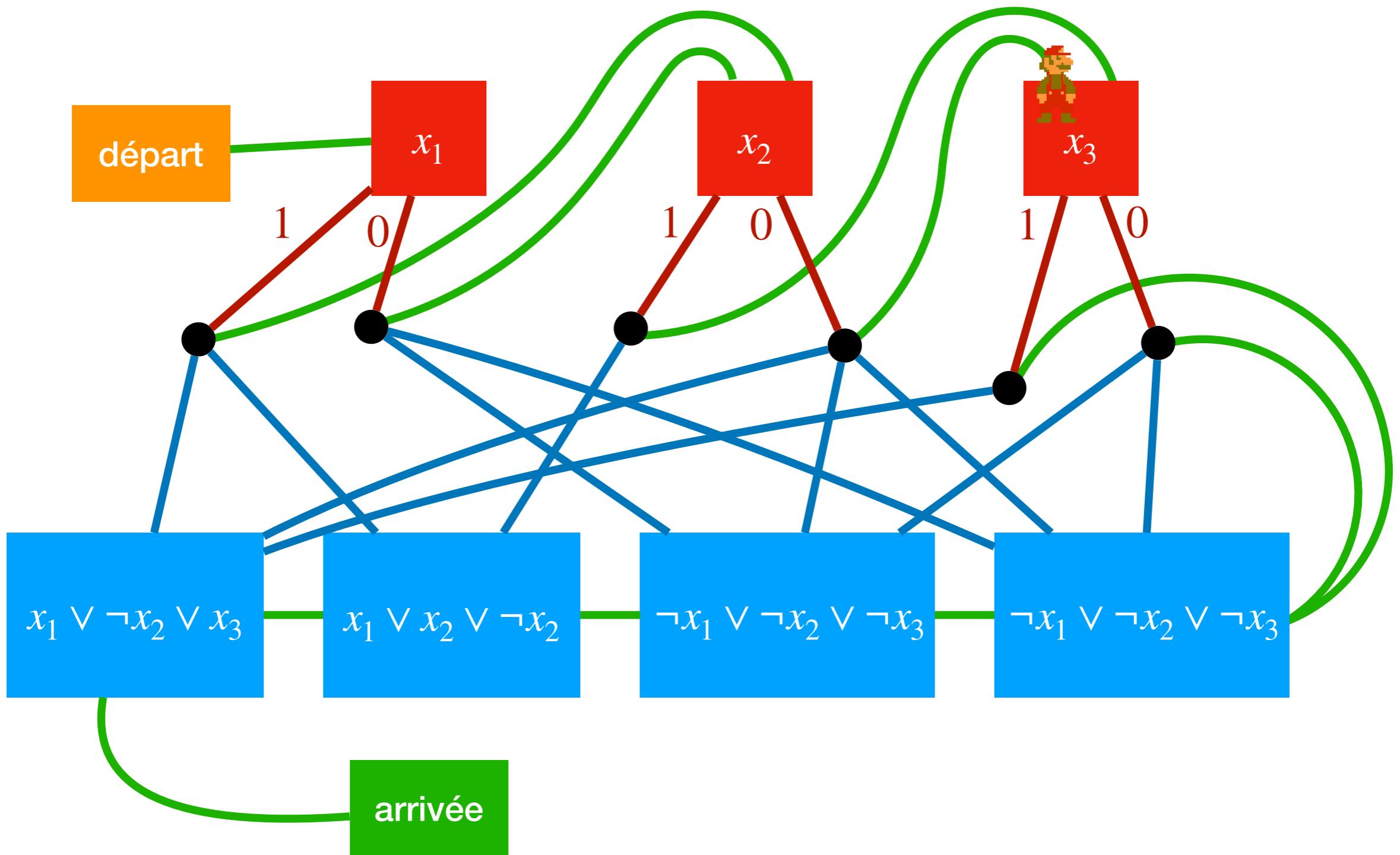
# Parcours du plombier



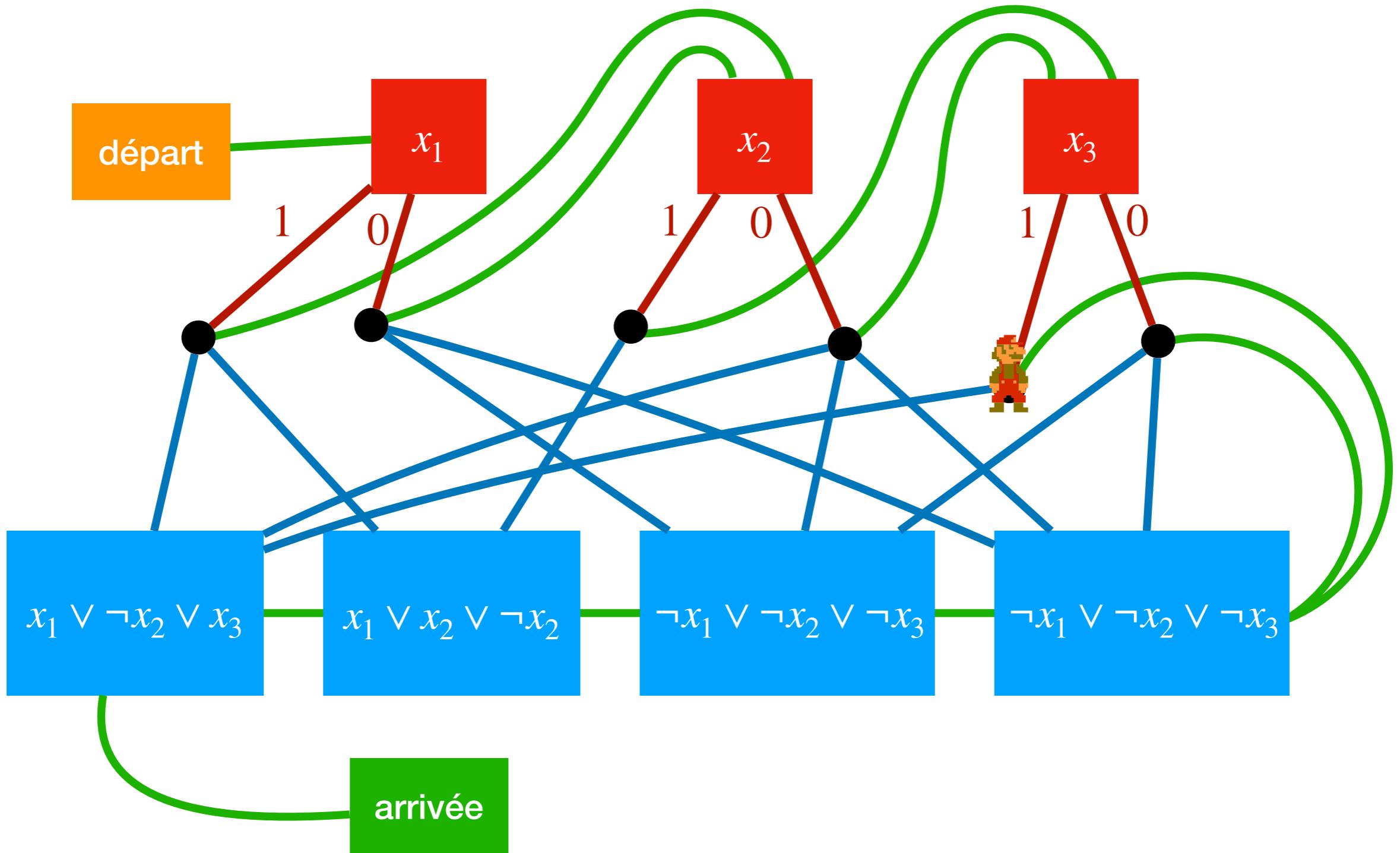
# Parcours du plombier



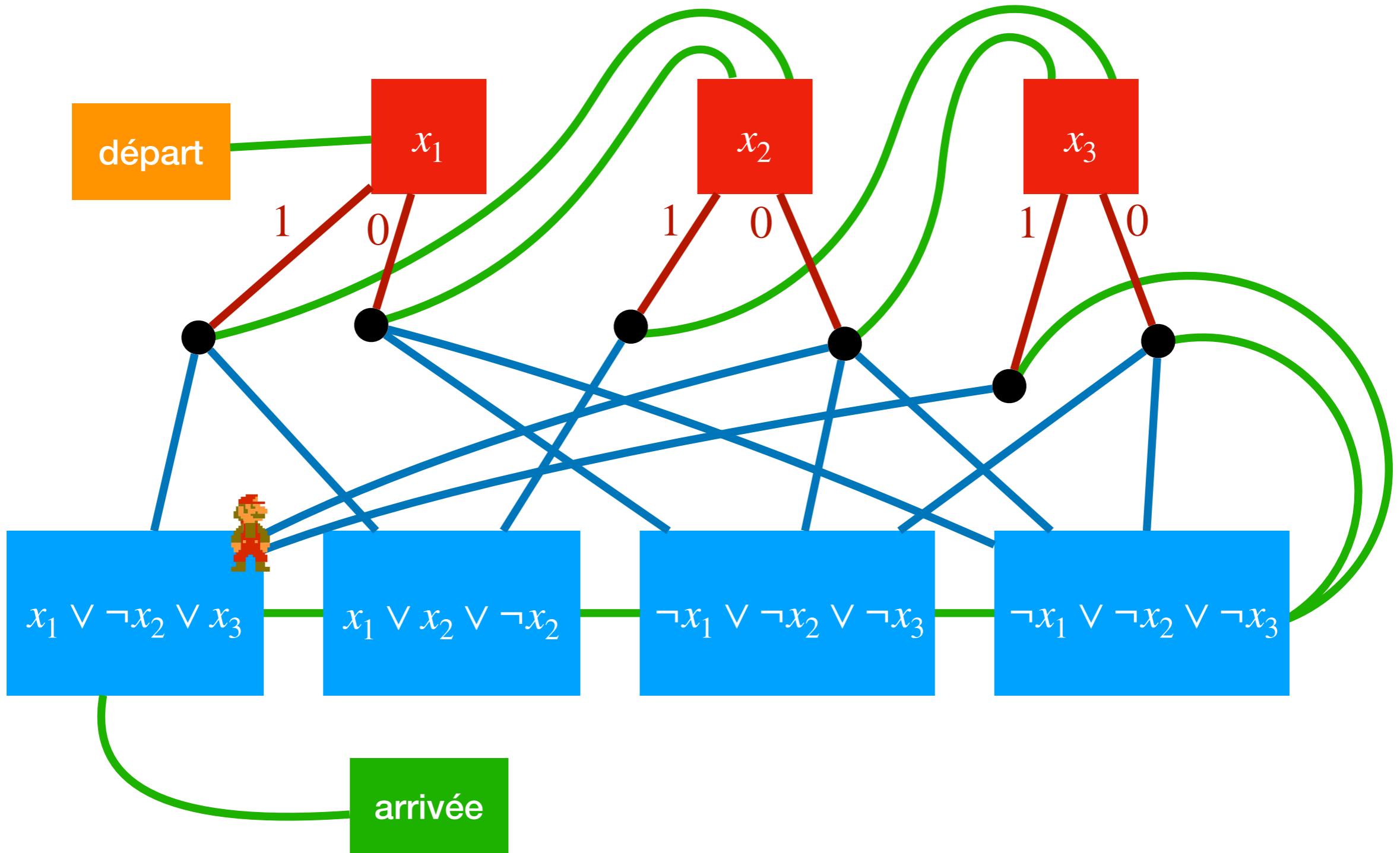
# Parcours du plombier



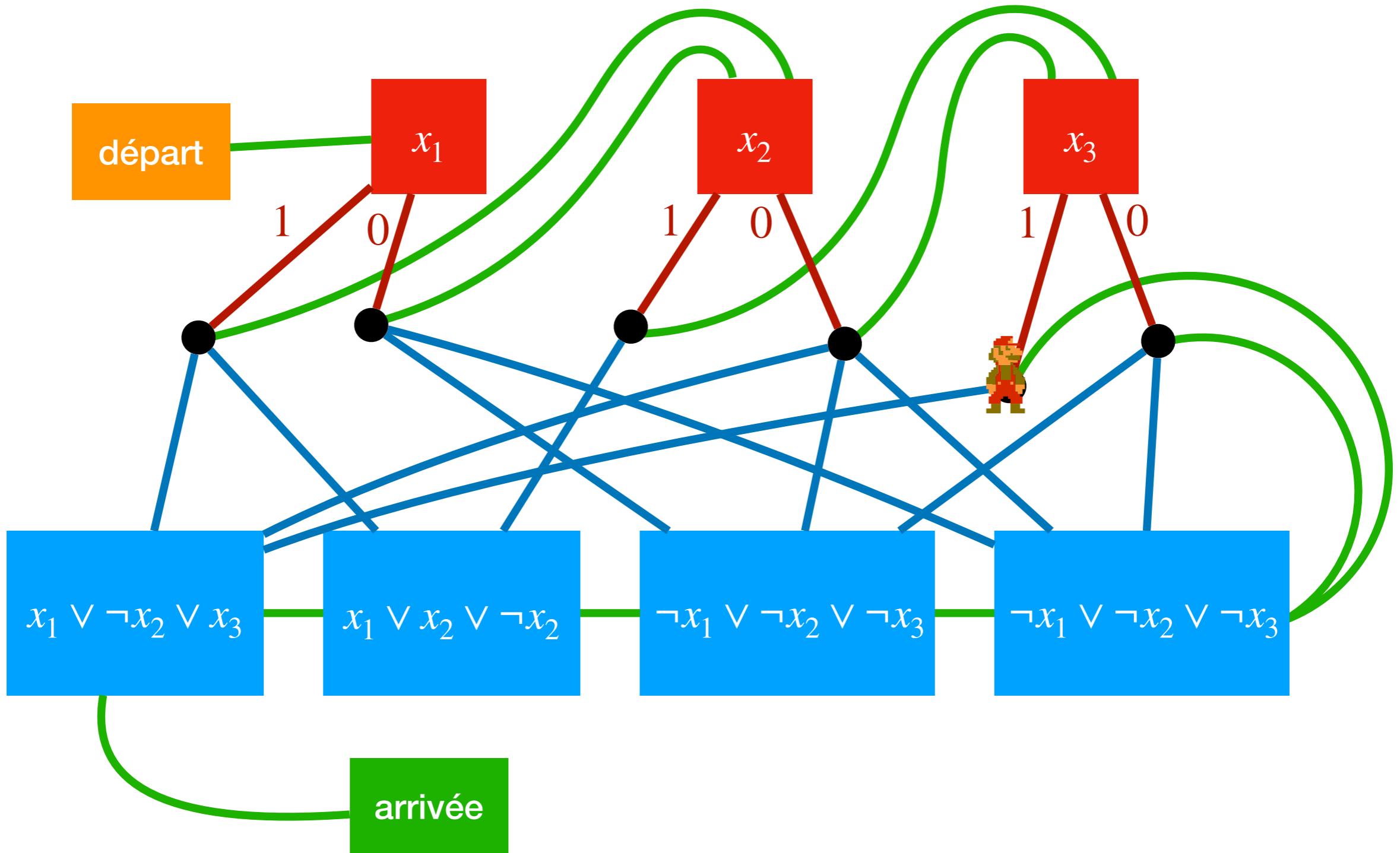
# Parcours du plombier



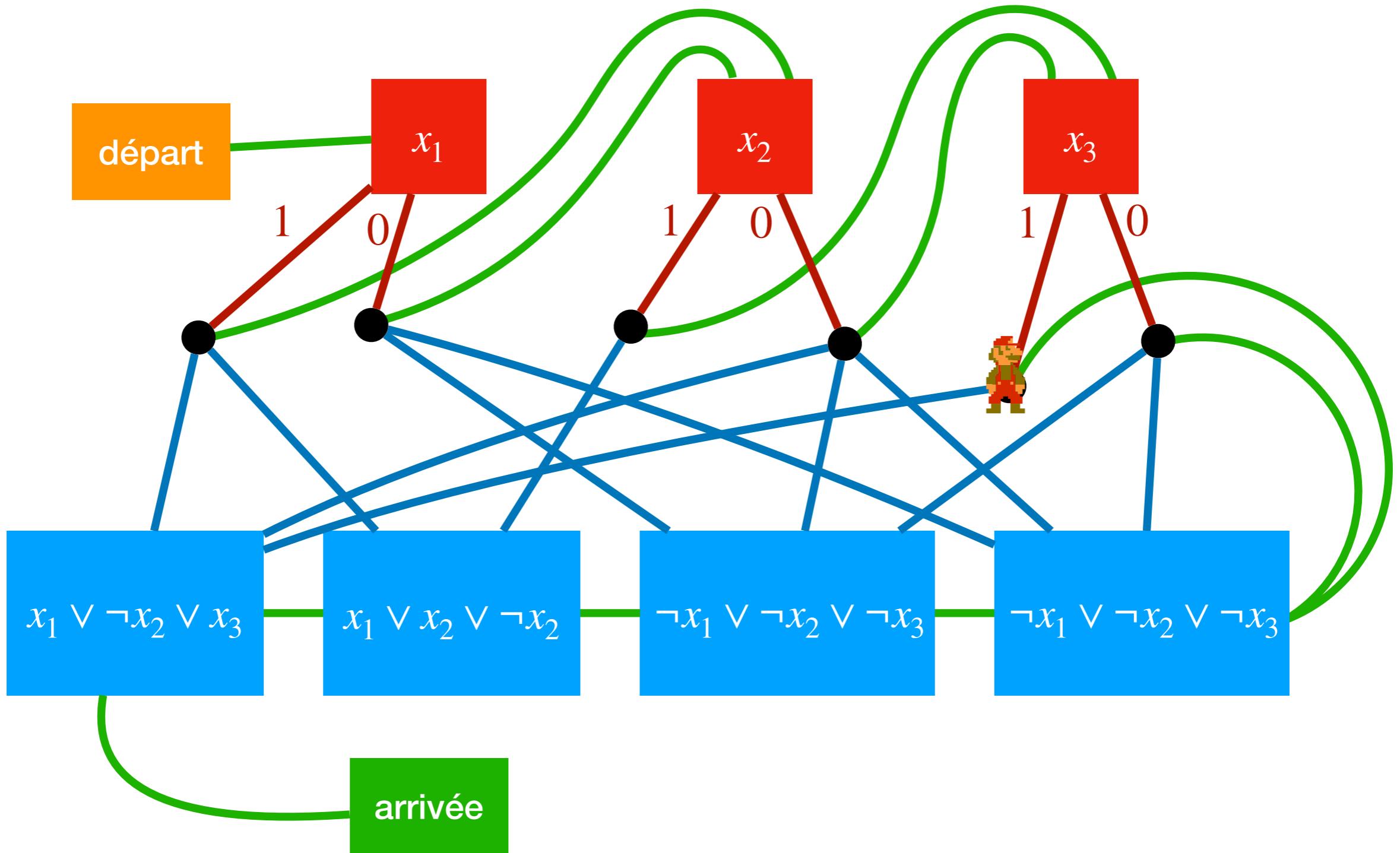
# Parcours du plombier



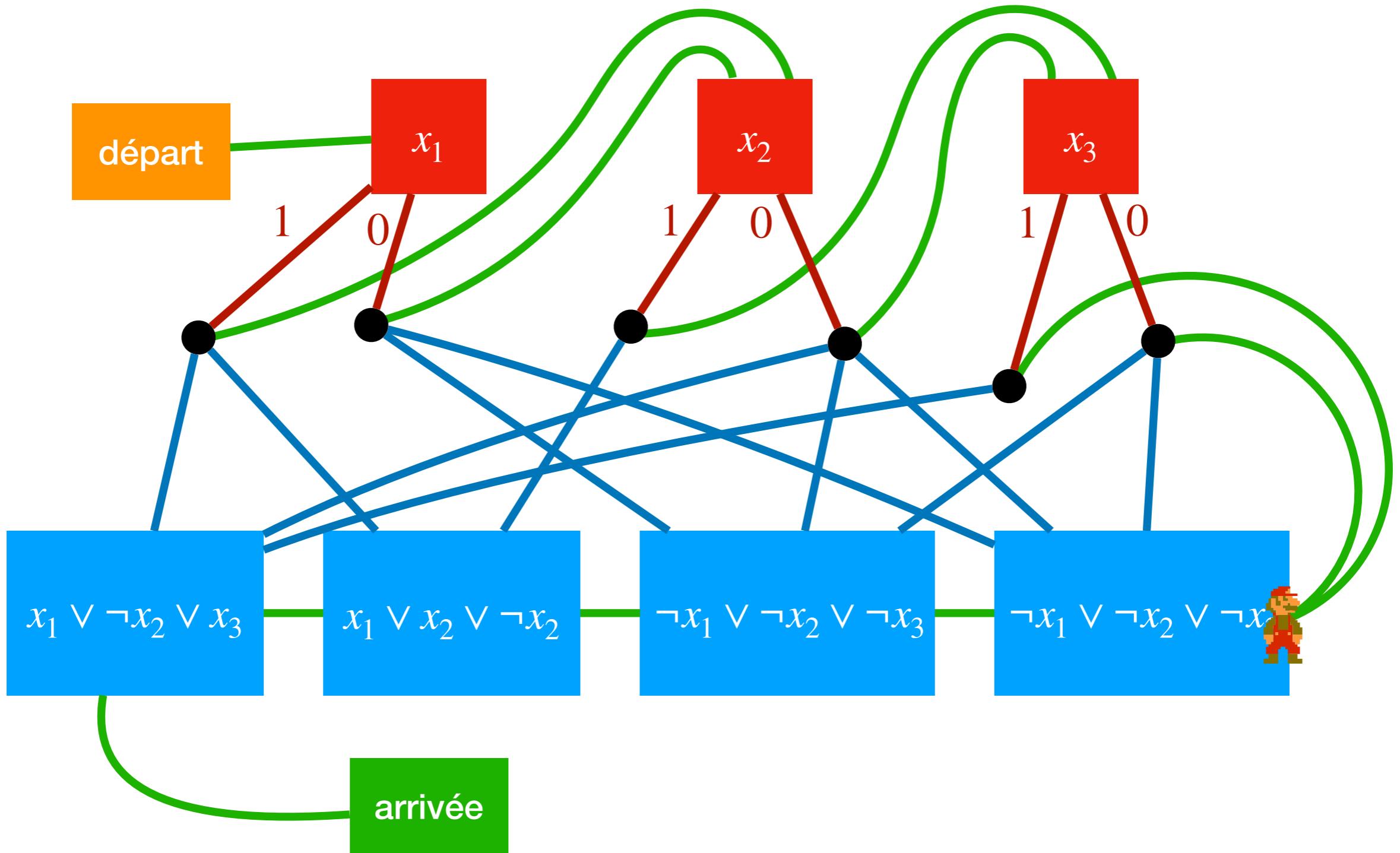
# Parcours du plombier



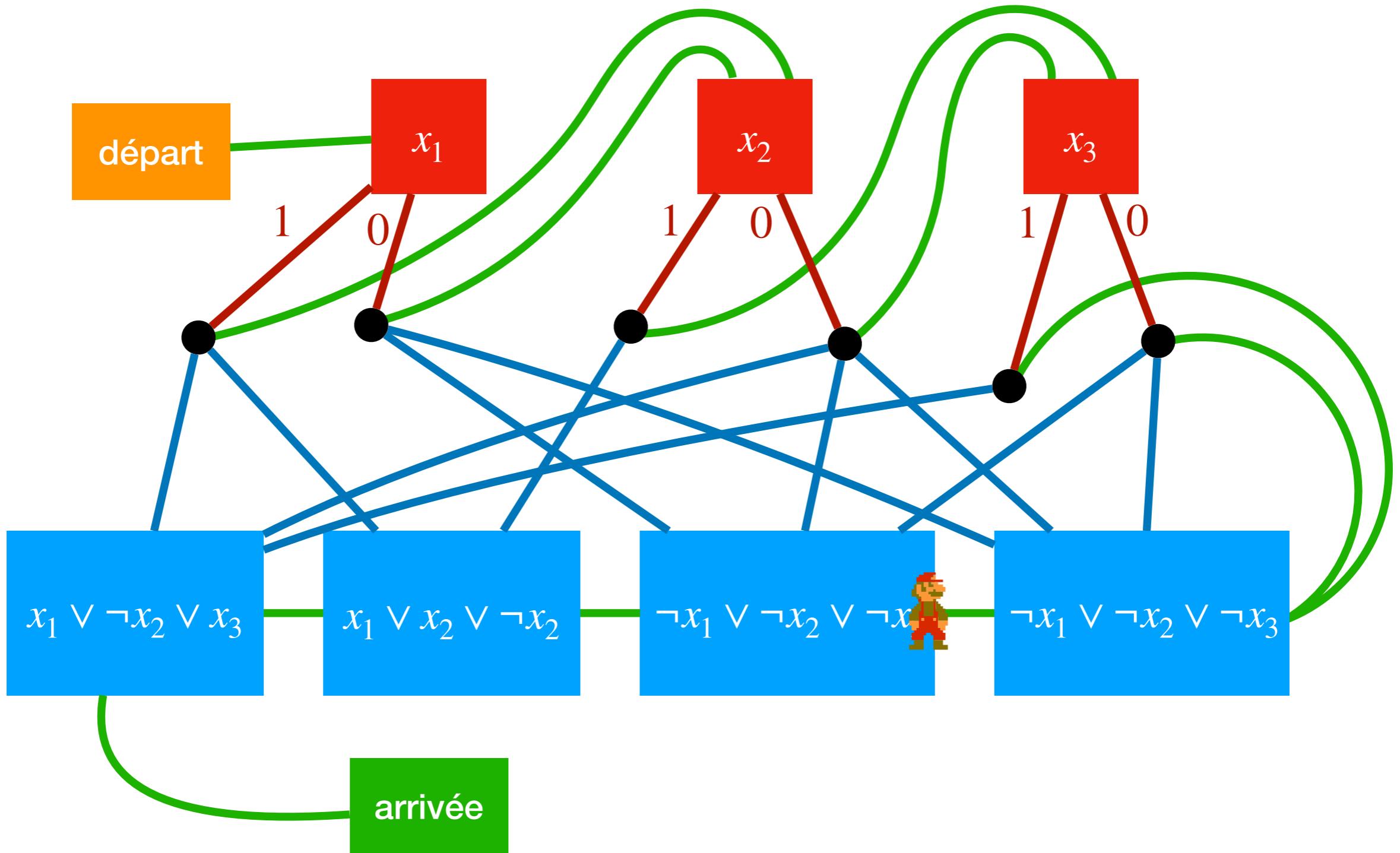
# Parcours du plombier



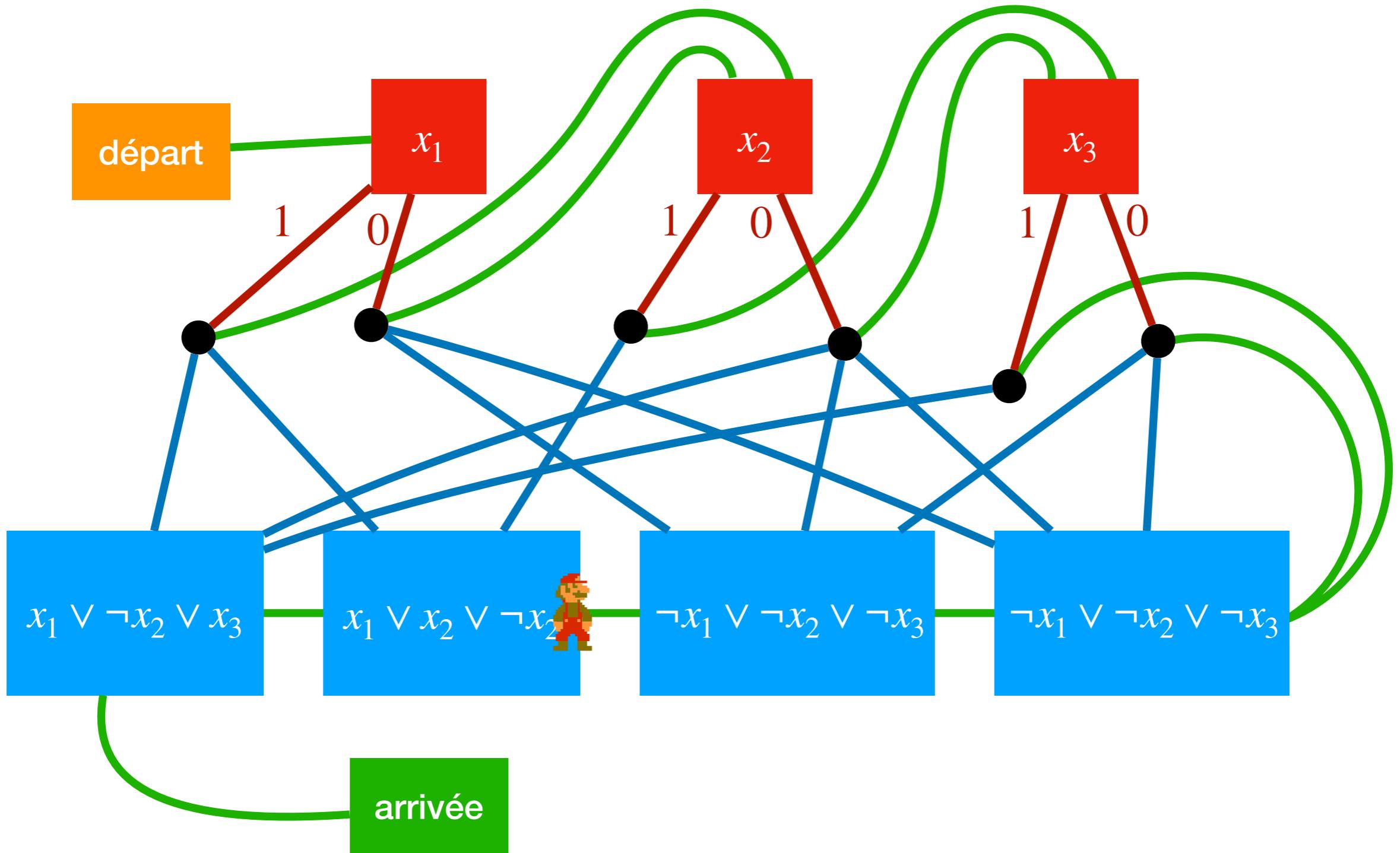
# Parcours du plombier



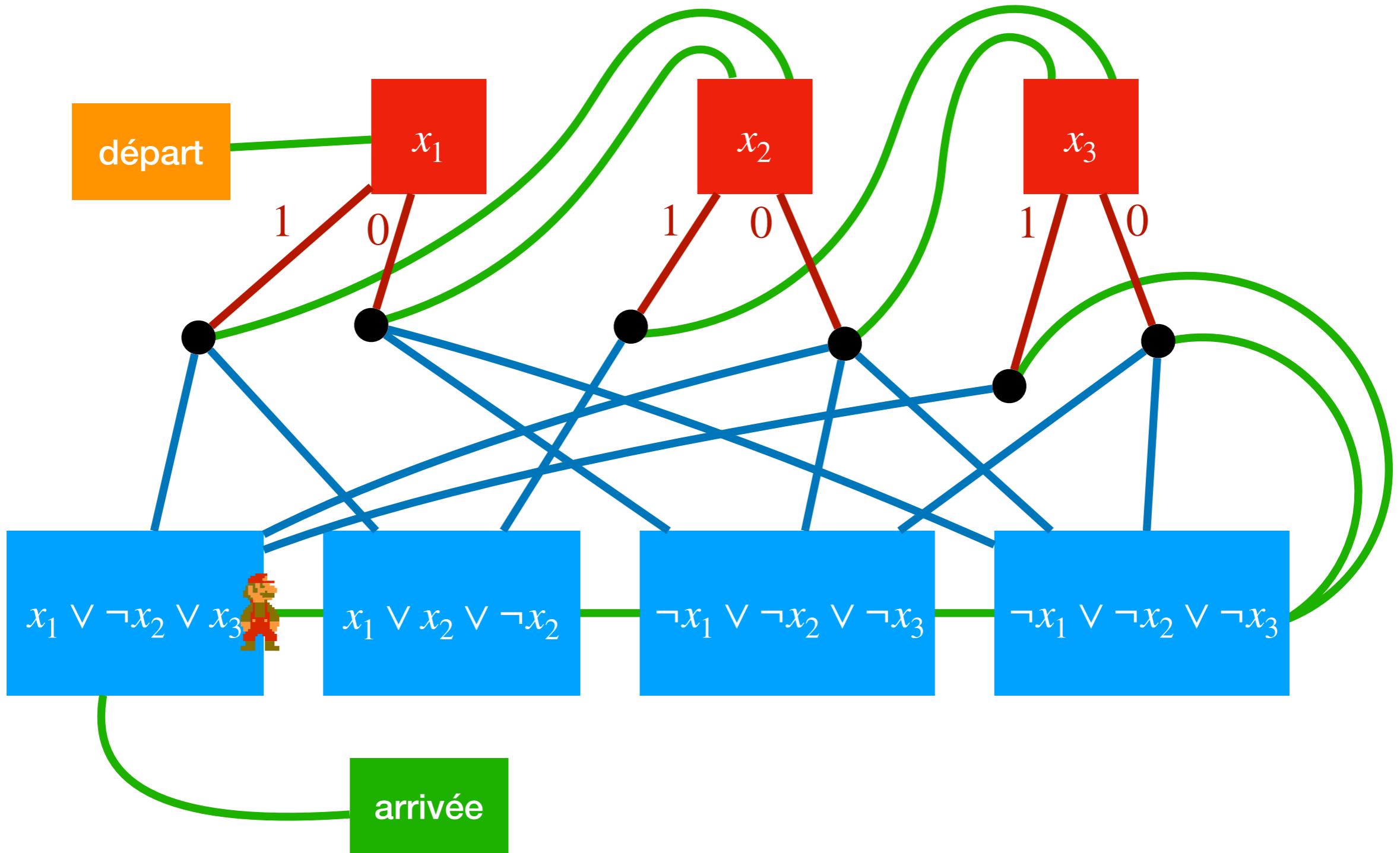
# Parcours du plombier



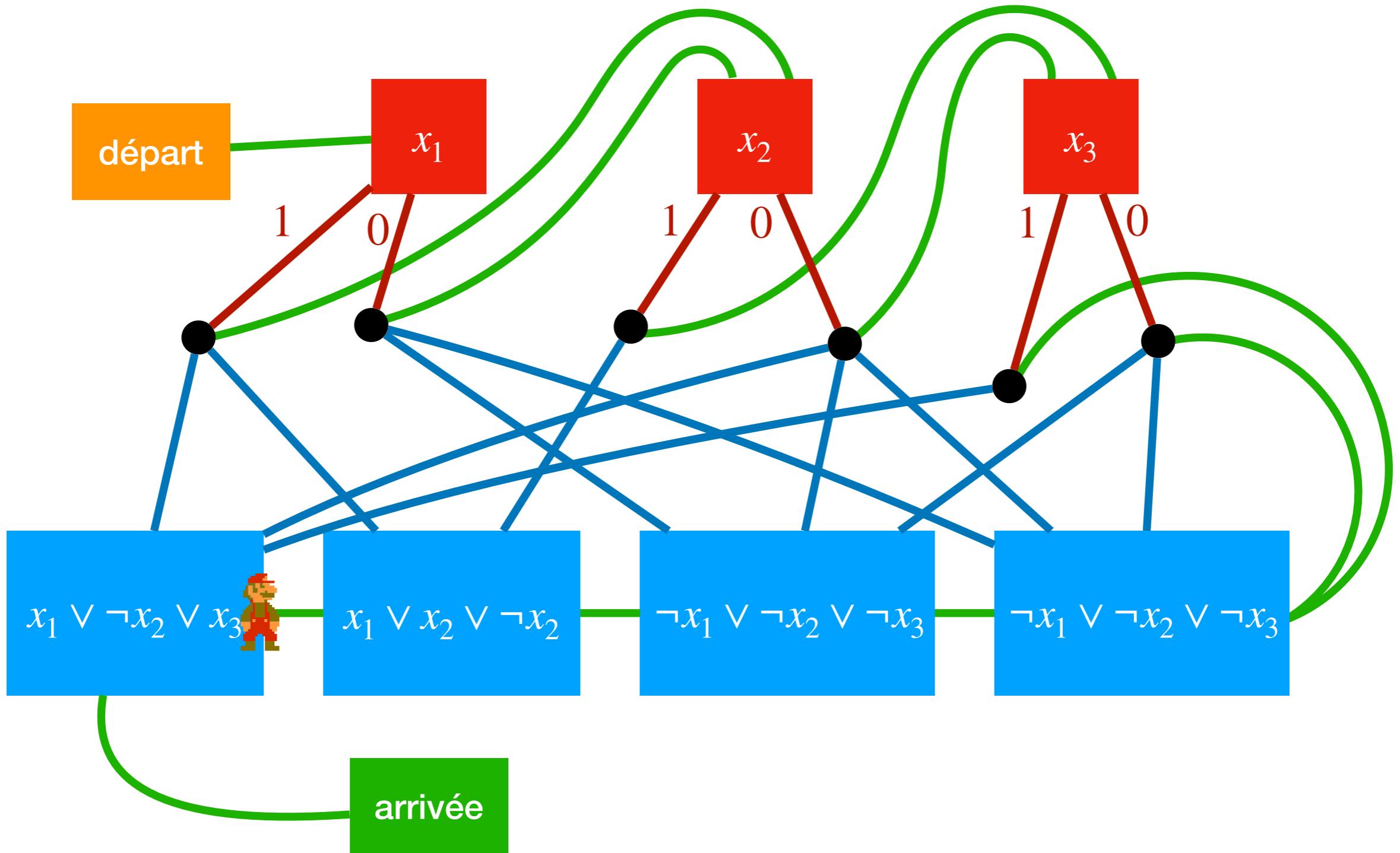
# Parcours du plombier



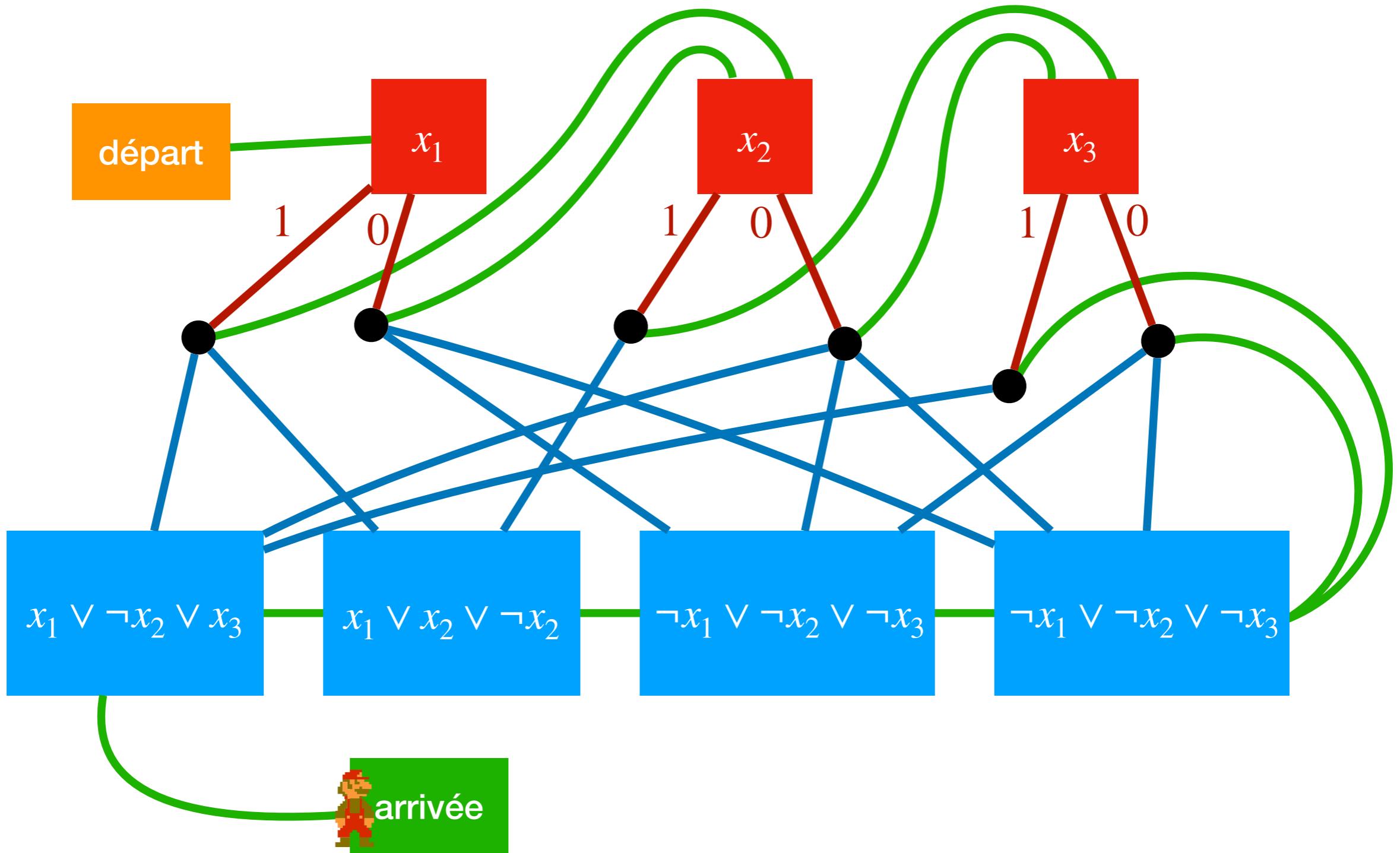
# Parcours du plombier



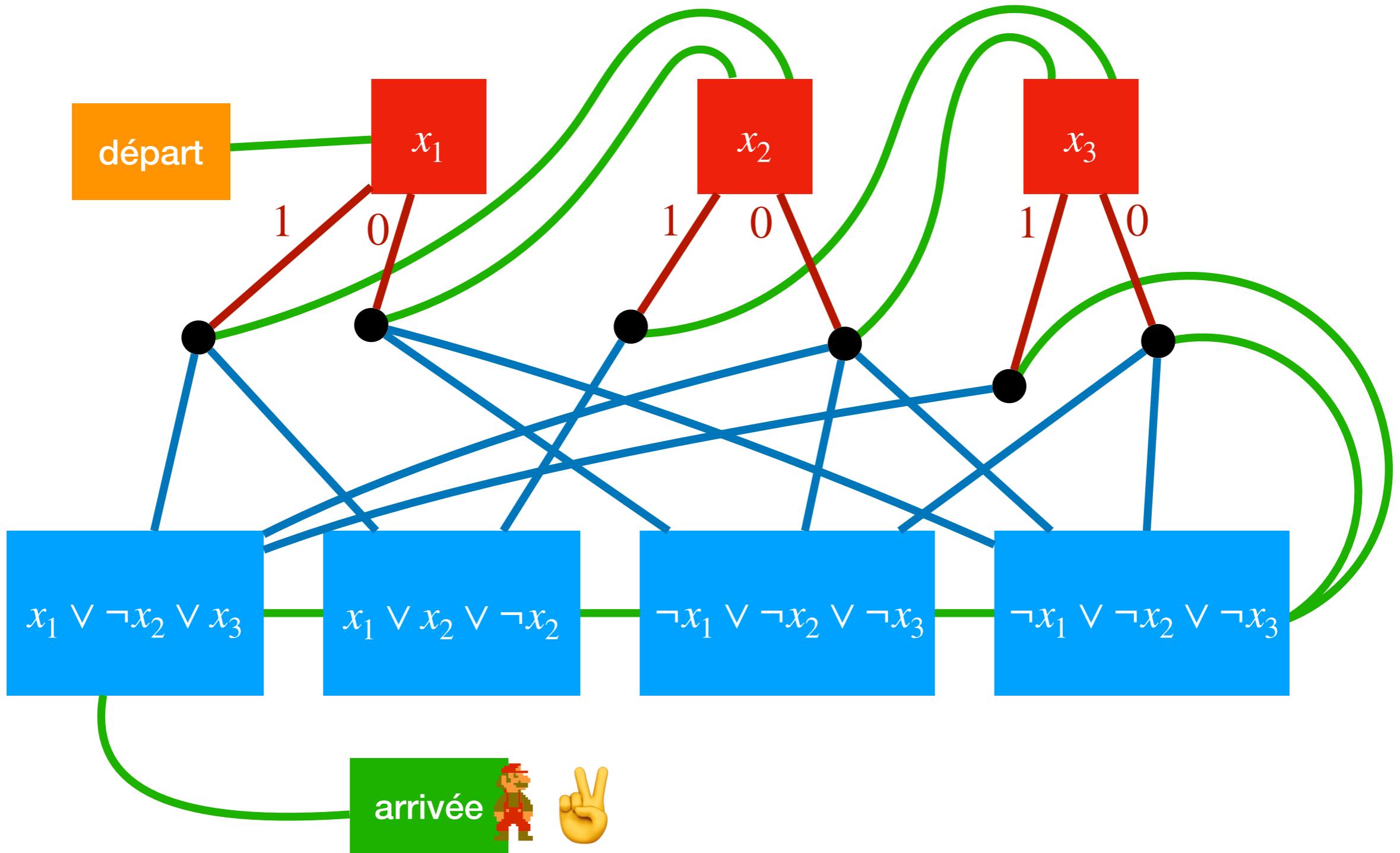
# Parcours du plombier



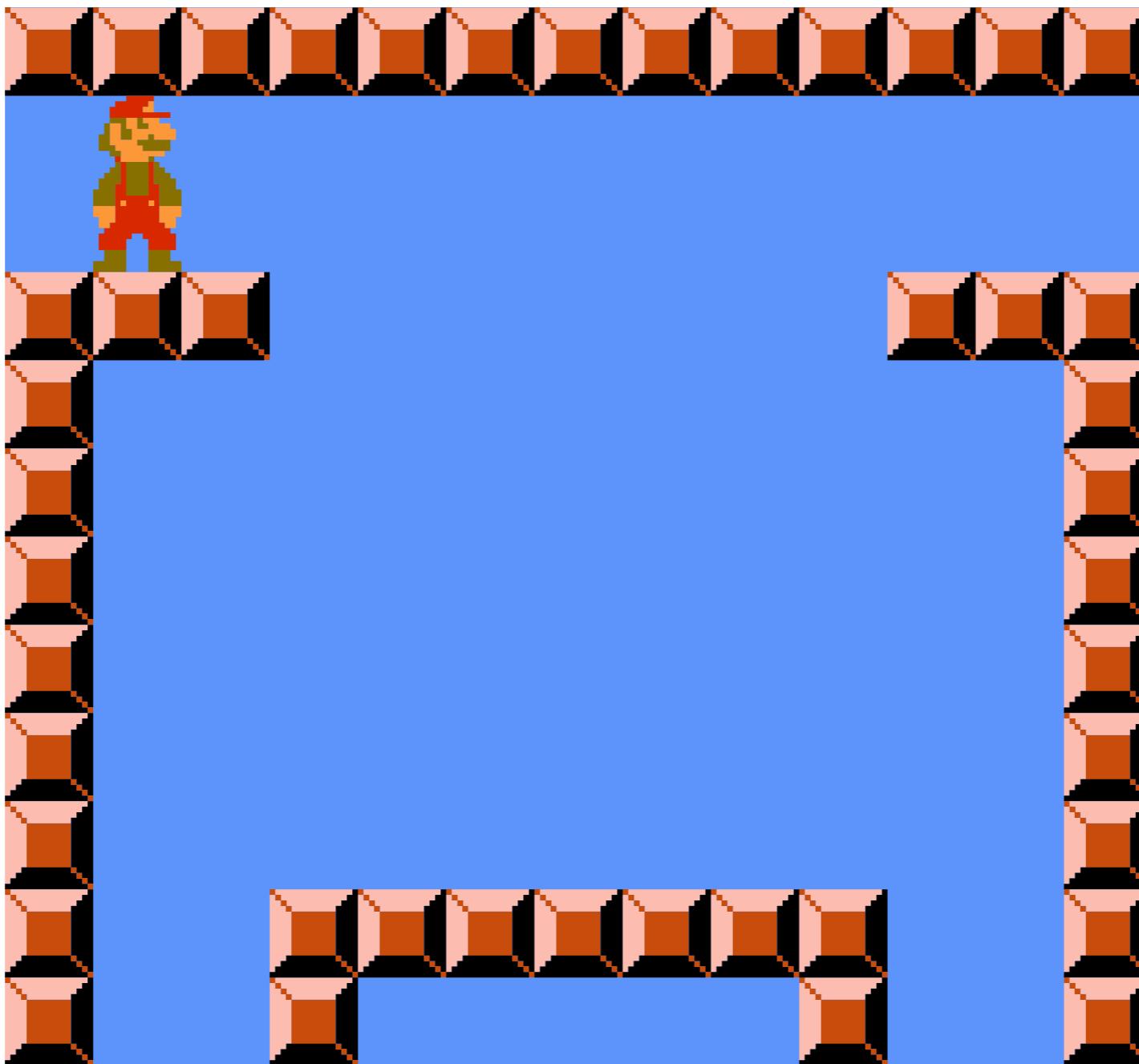
# Parcours du plombier



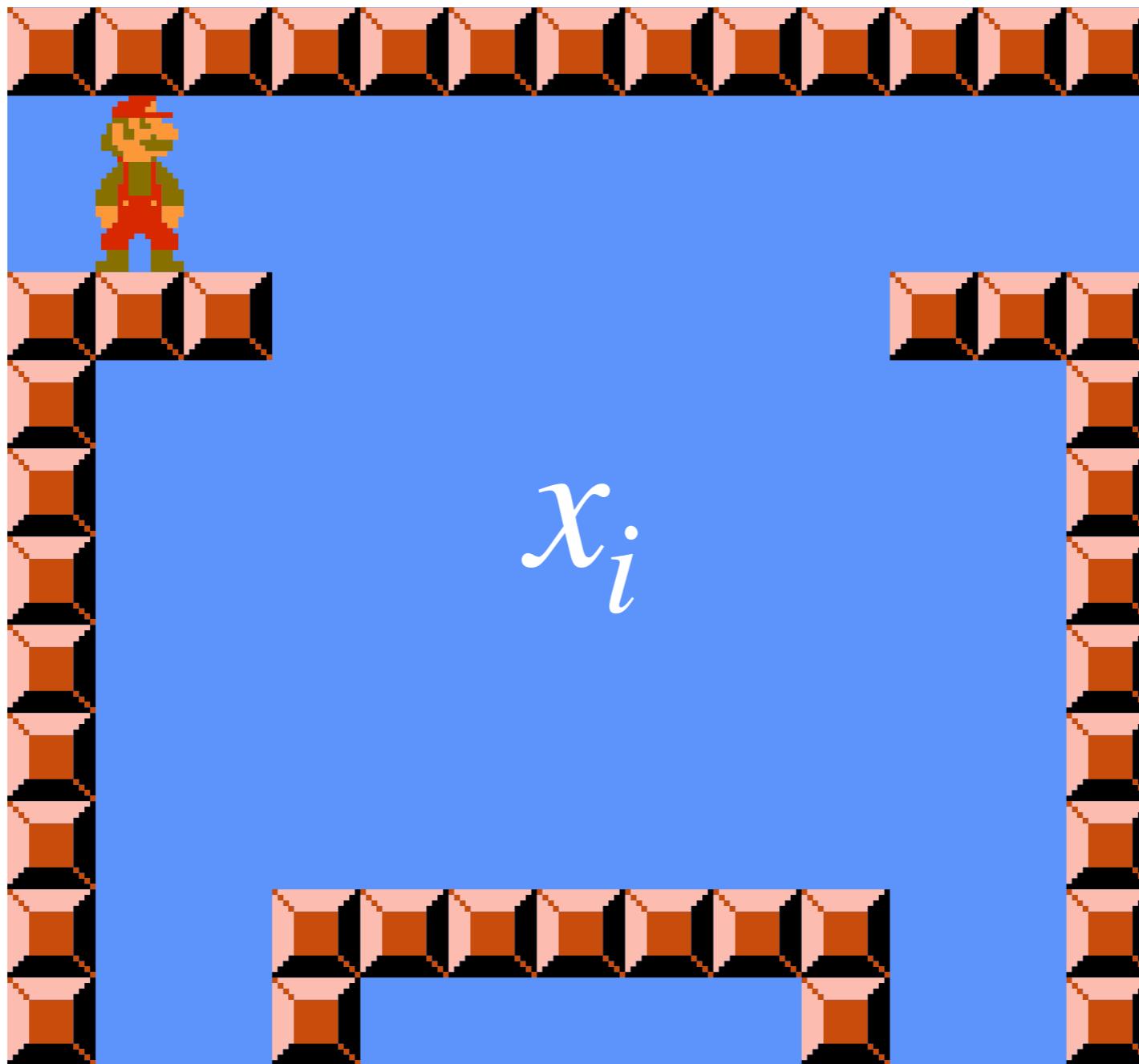
# Parcours du plombier



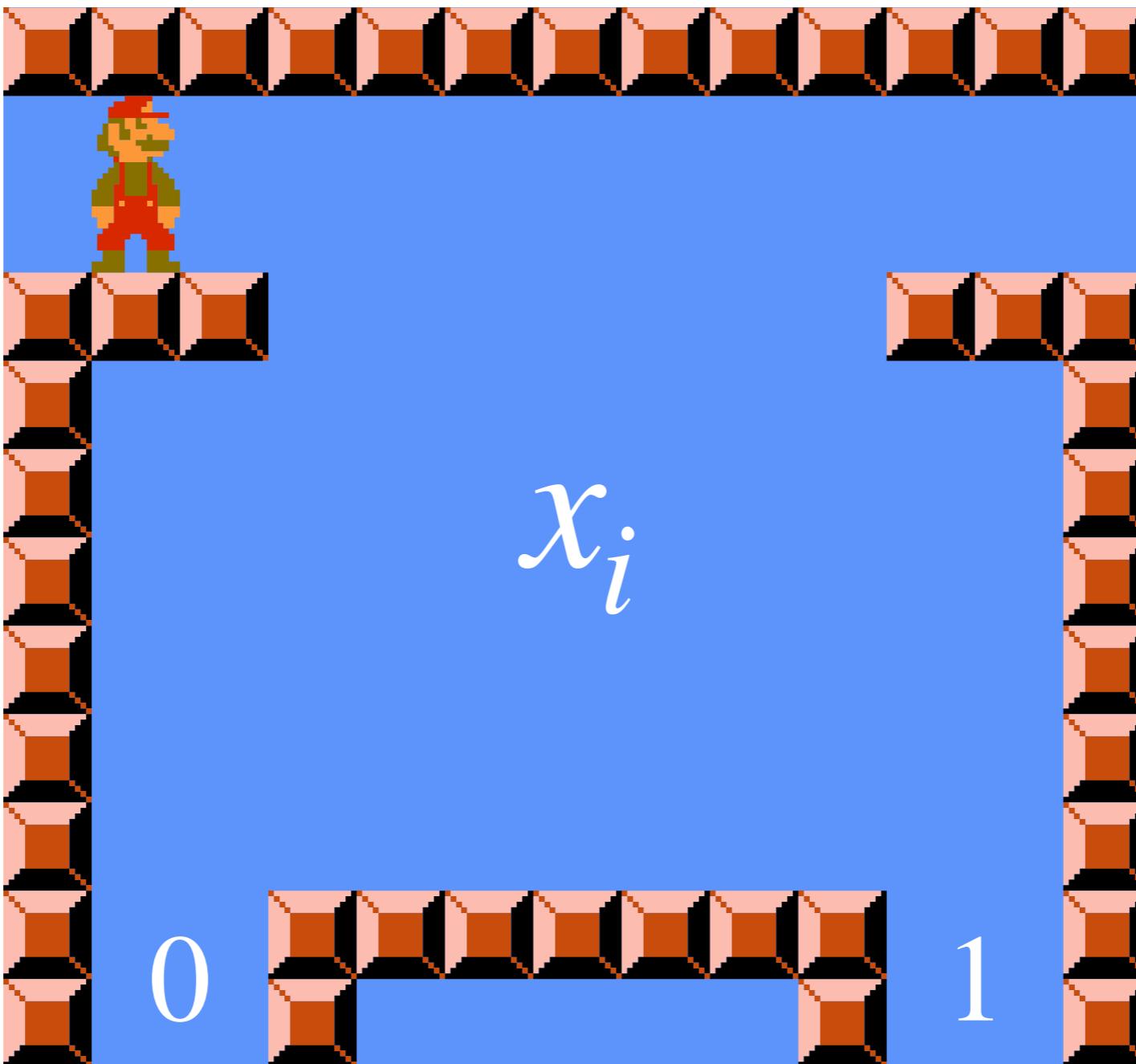
# Gadget pour les variables



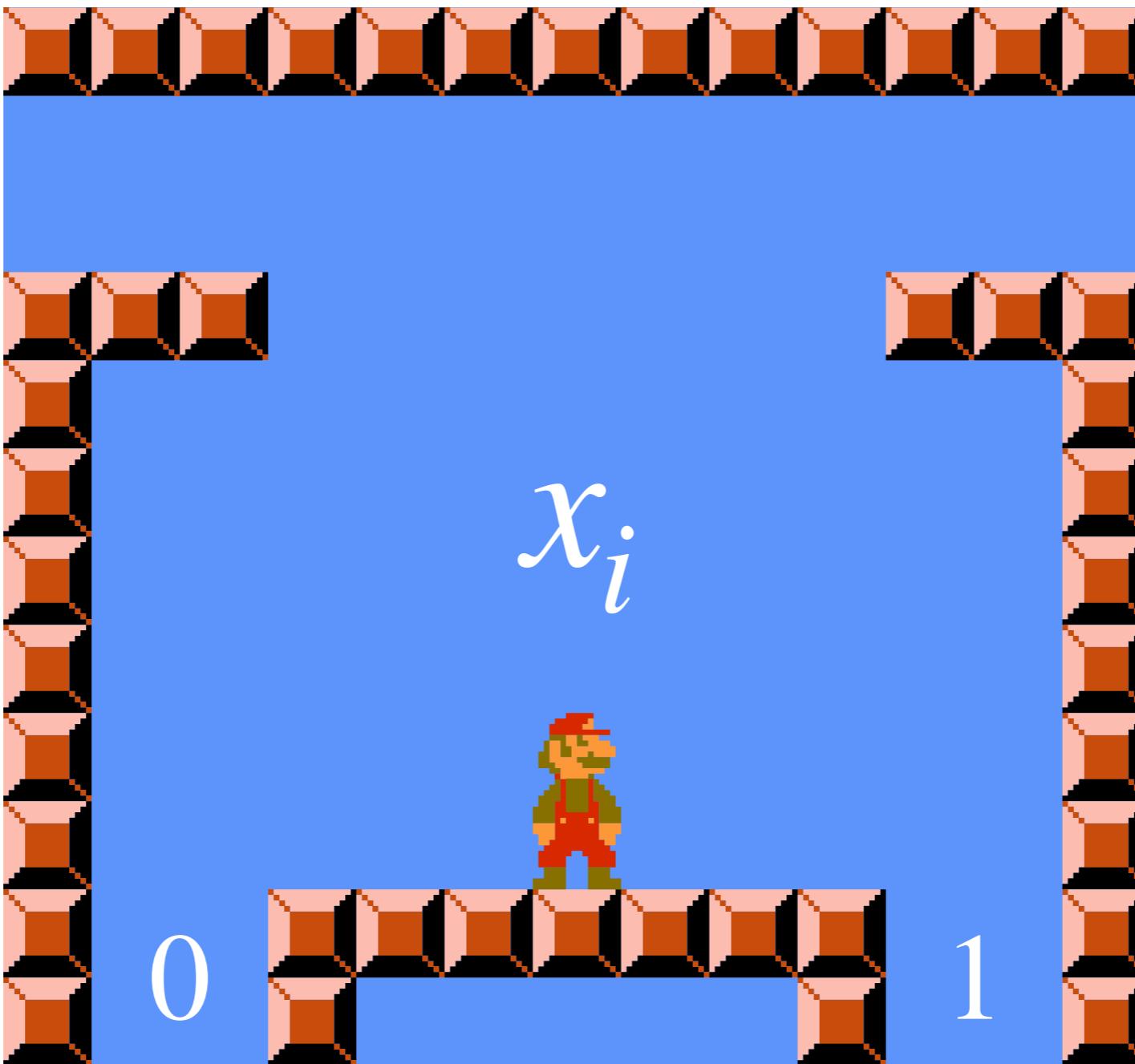
# Gadget pour les variables



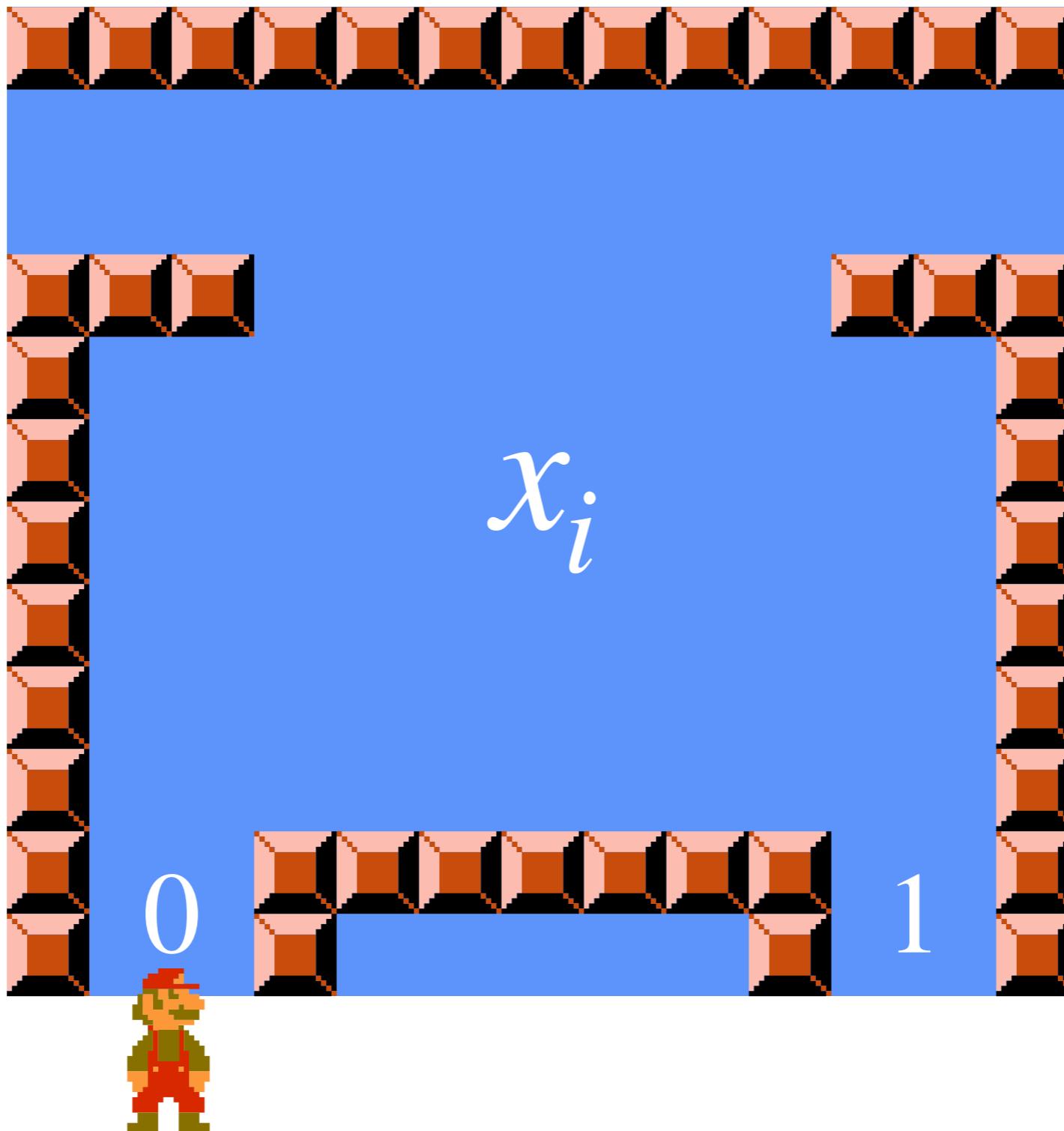
# Gadget pour les variables



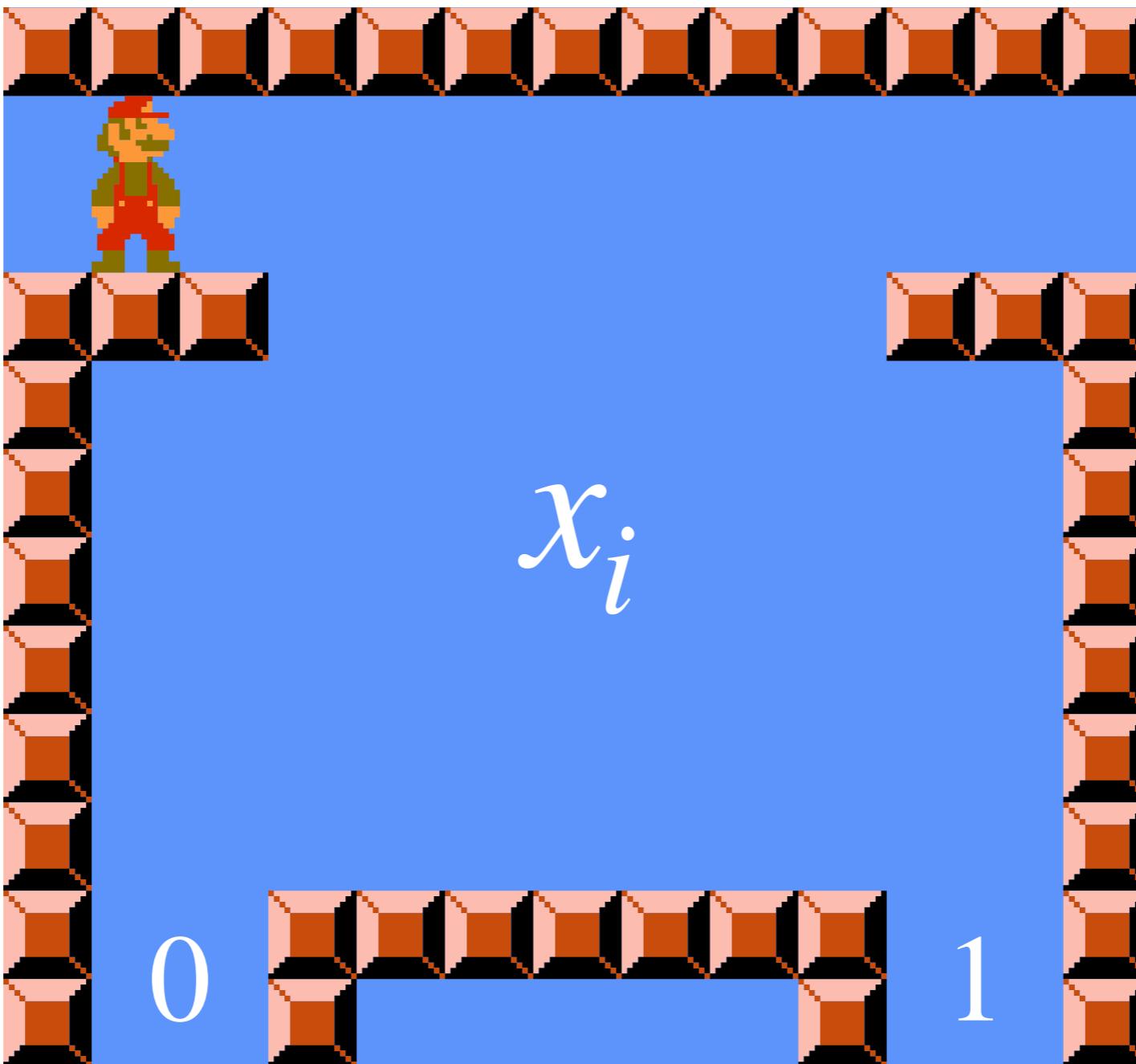
# Gadget pour les variables



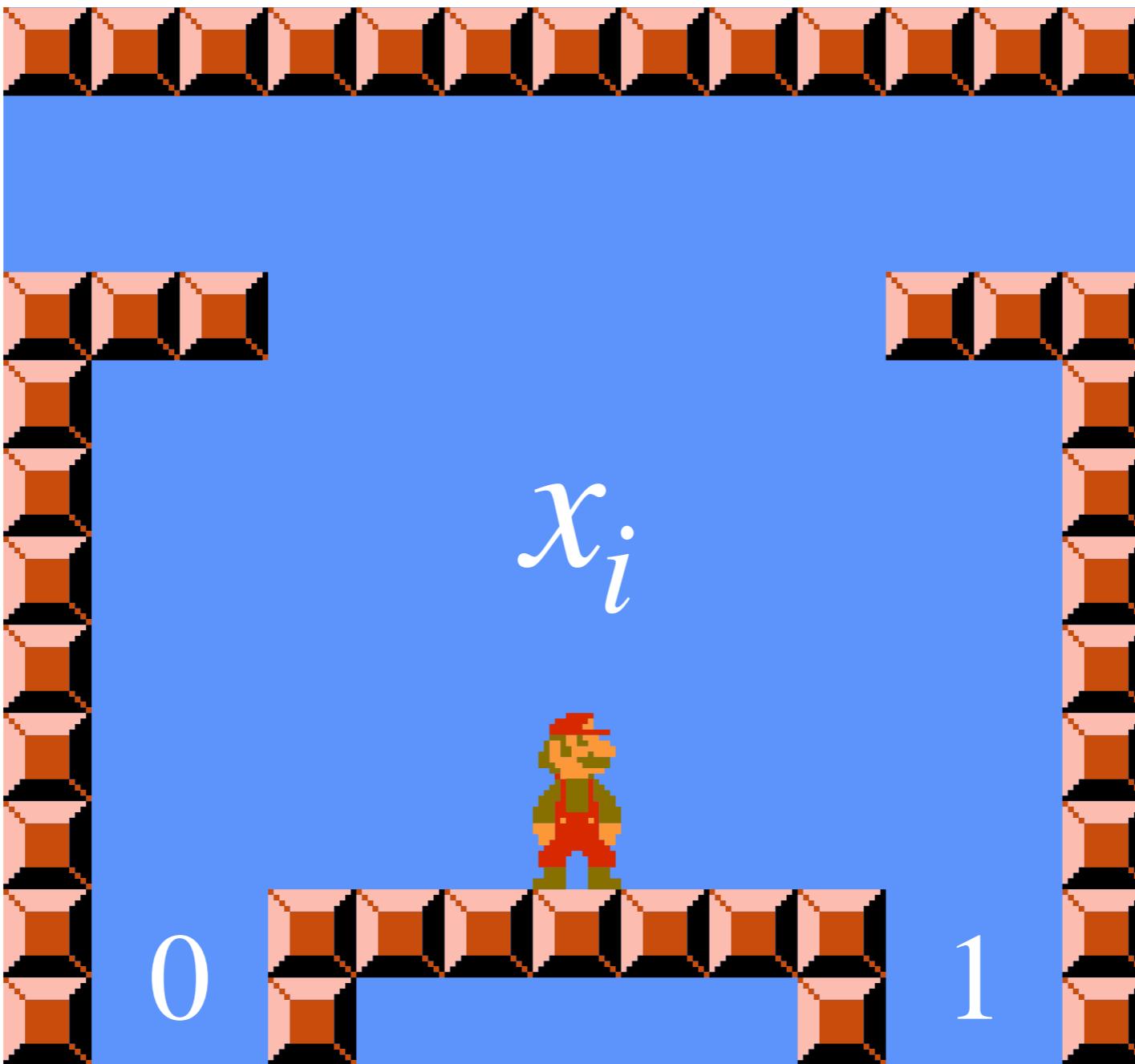
# Gadget pour les variables



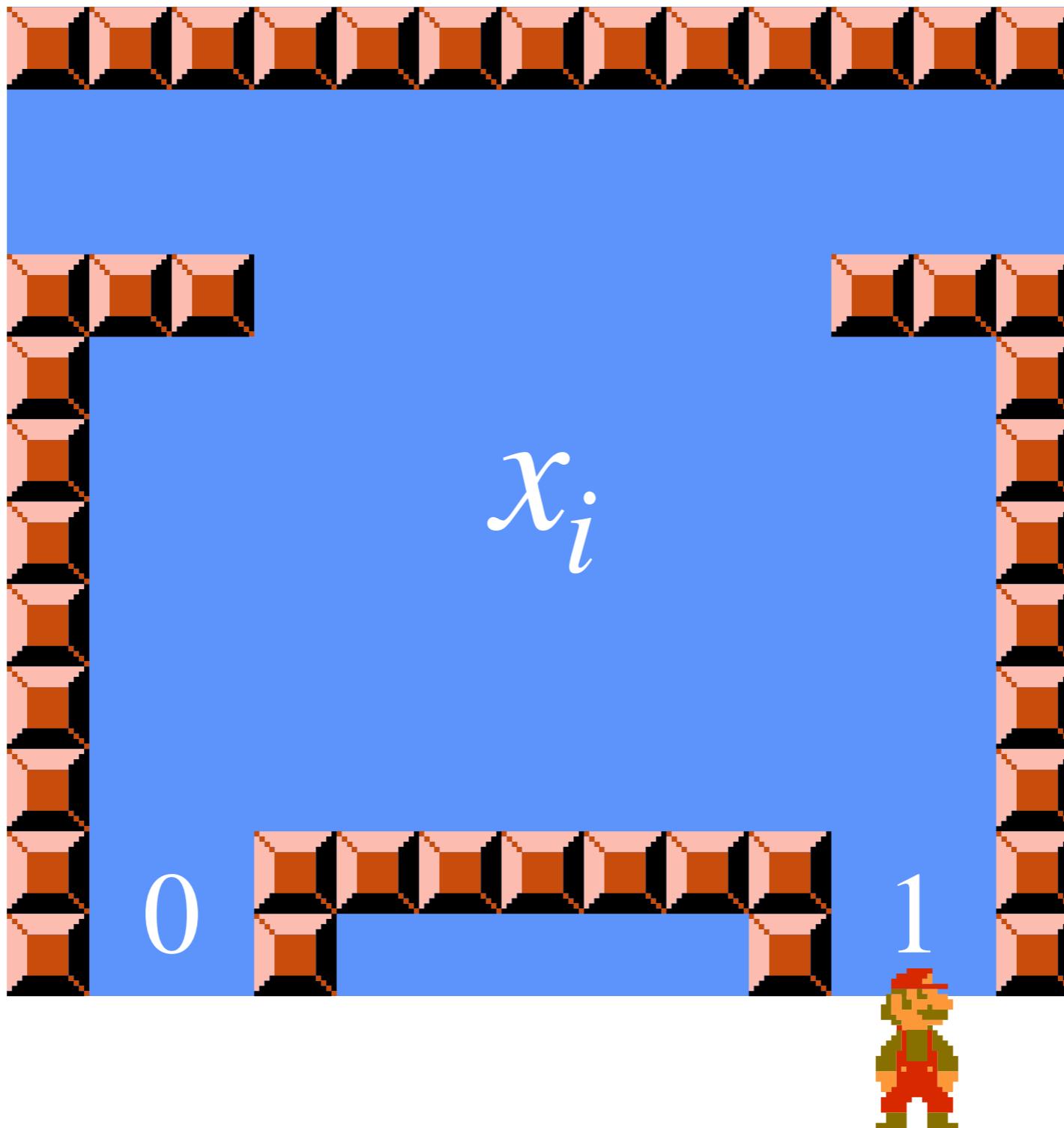
# Gadget pour les variables



# Gadget pour les variables

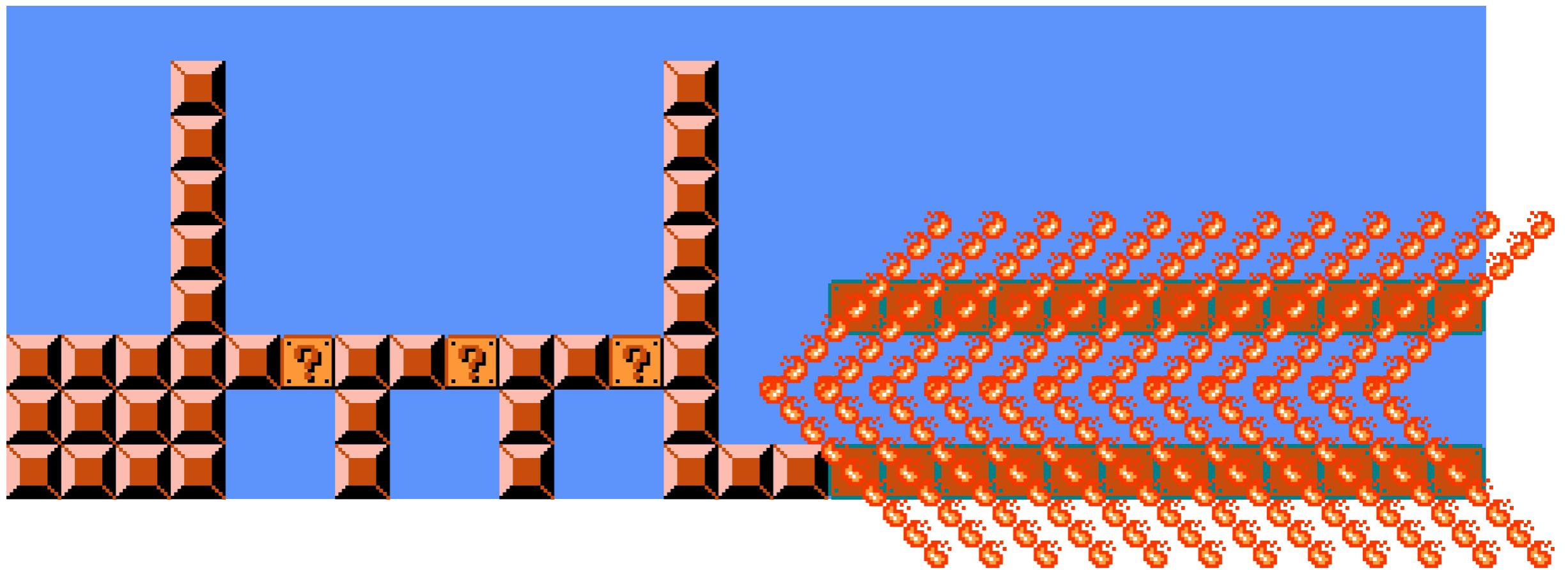


# Gadget pour les variables



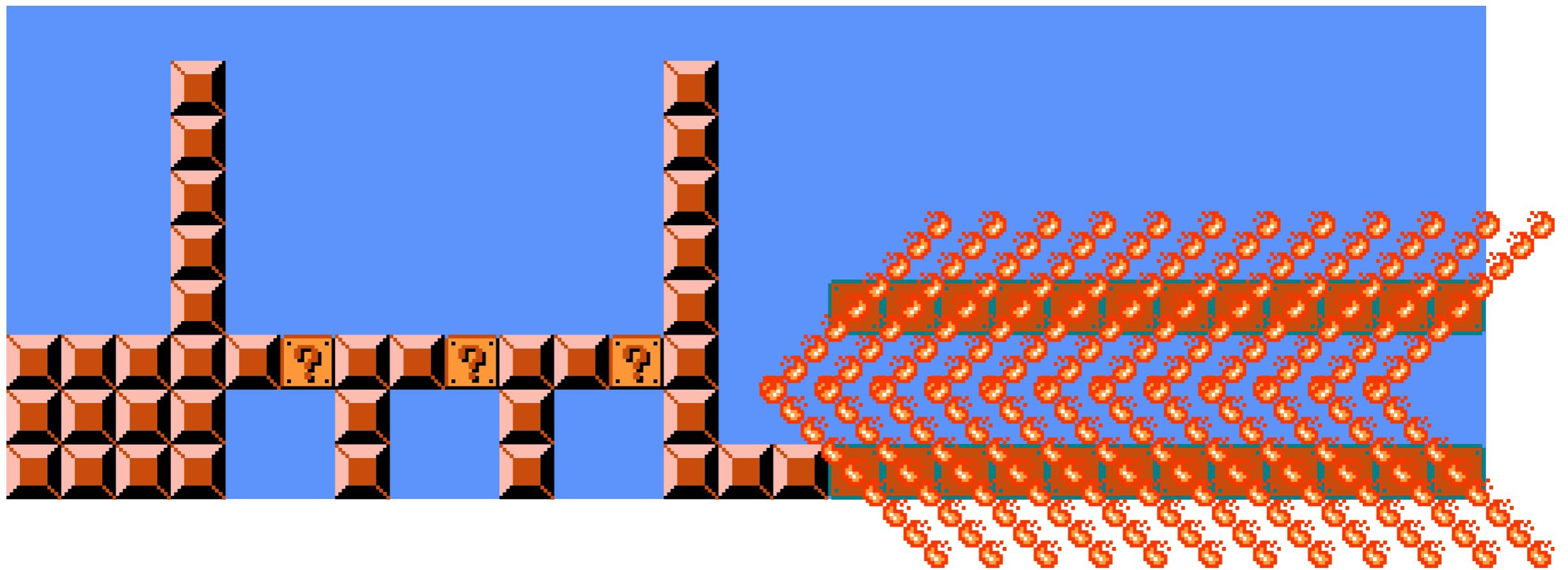
# Gadget pour les clauses

$$x_1 \vee \neg x_3 \vee x_6$$



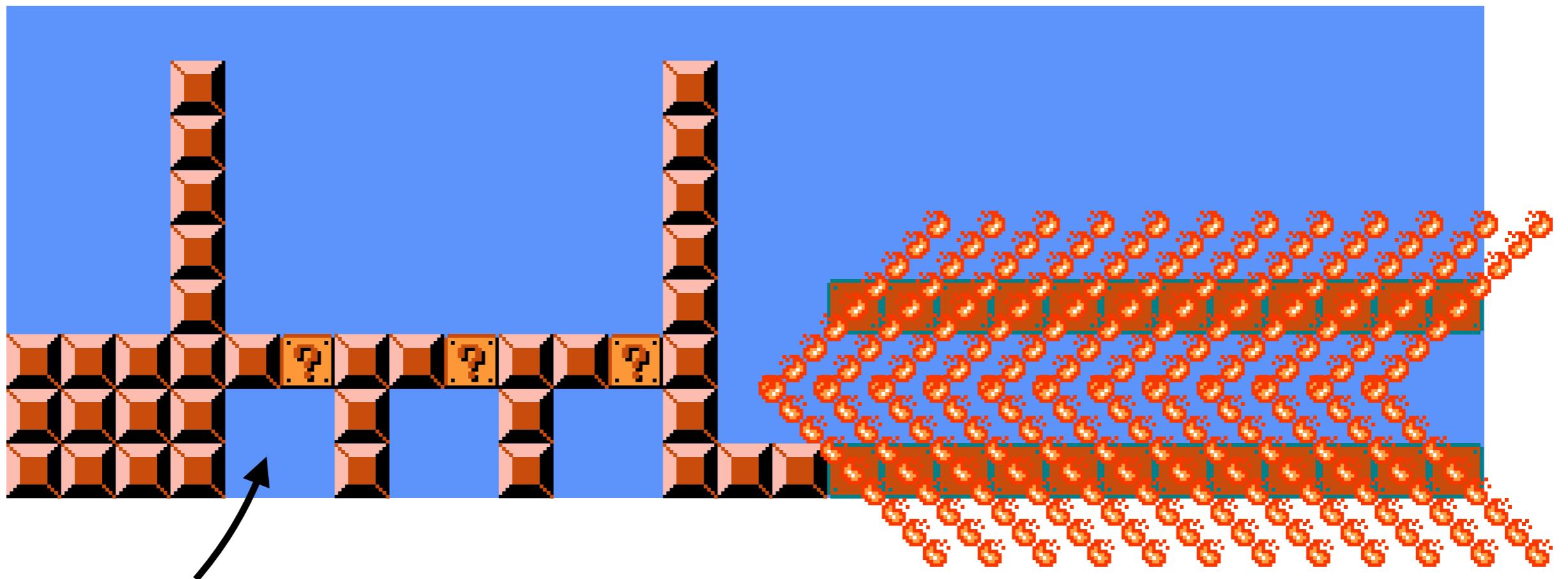
# Gadget pour les clauses

$$x_1 \vee \neg x_3 \vee x_6$$



# Gadget pour les clauses

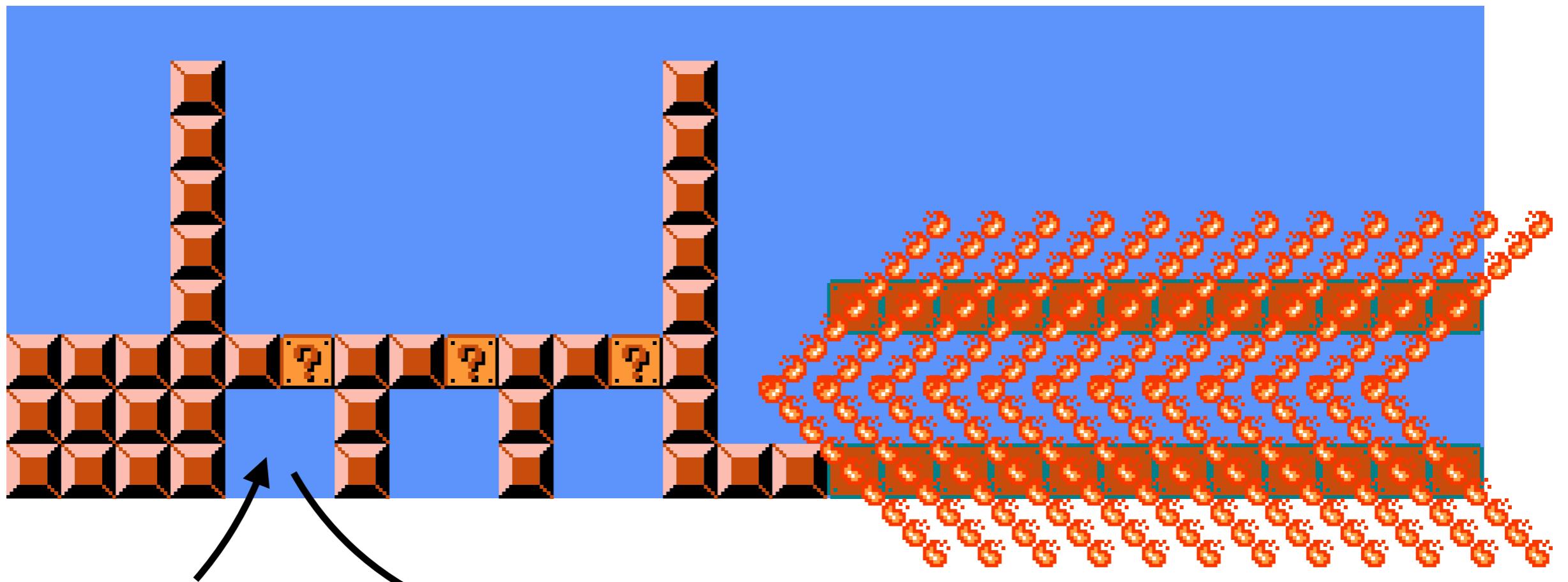
$$x_1 \vee \neg x_3 \vee x_6$$



de la sortie  $x_1 = 1$   
du gadget pour  $x_1$

# Gadget pour les clauses

$$x_1 \vee \neg x_3 \vee x_6$$

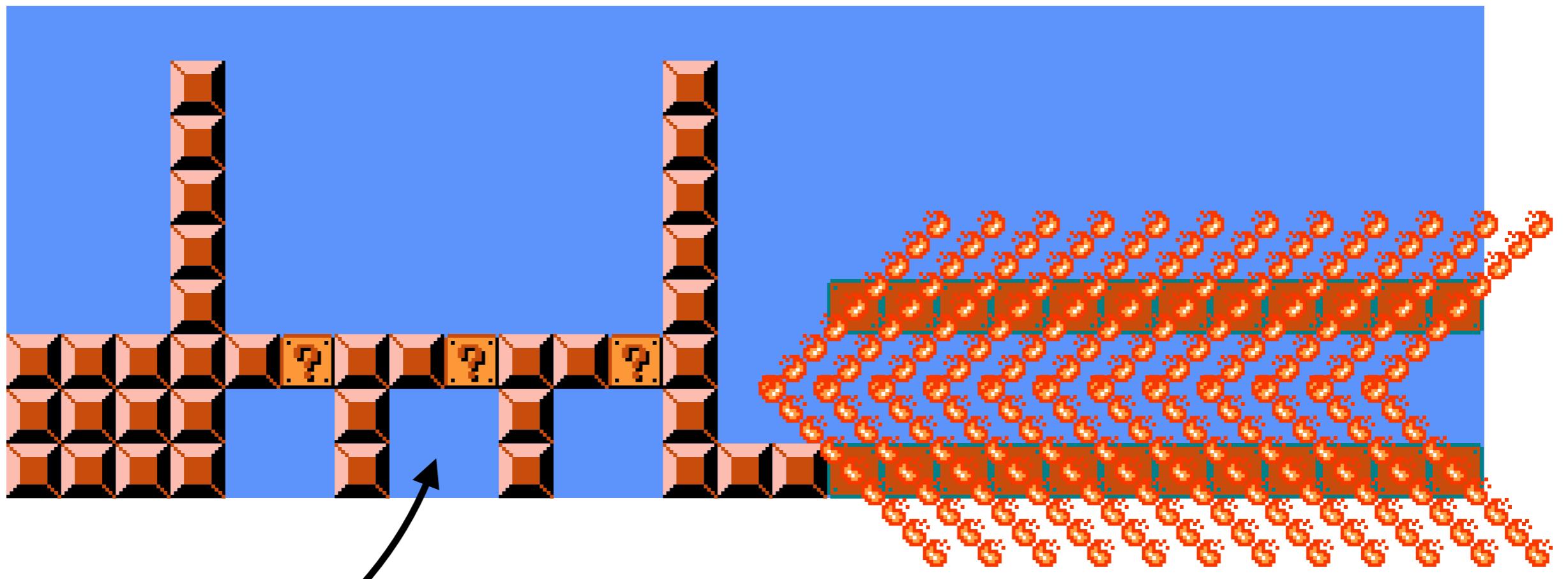


de la sortie  $x_1 = 1$   
du gadget pour  $x_1$

à l'entrée du gadget  
pour  $x_2$  (variable suivante)

# Gadget pour les clauses

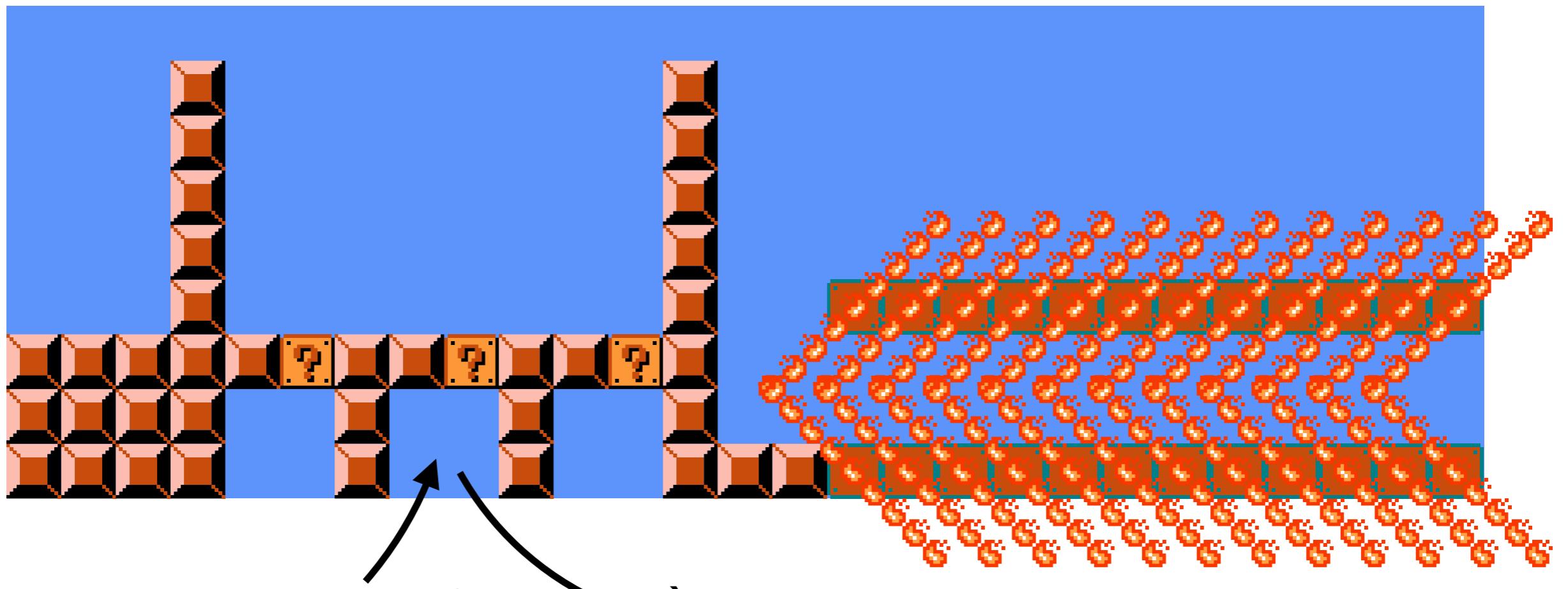
$$x_1 \vee \neg x_3 \vee x_6$$



de la sortie  $x_3 = 0$   
du gadget pour  $x_3$

# Gadget pour les clauses

$$x_1 \vee \neg x_3 \vee x_6$$

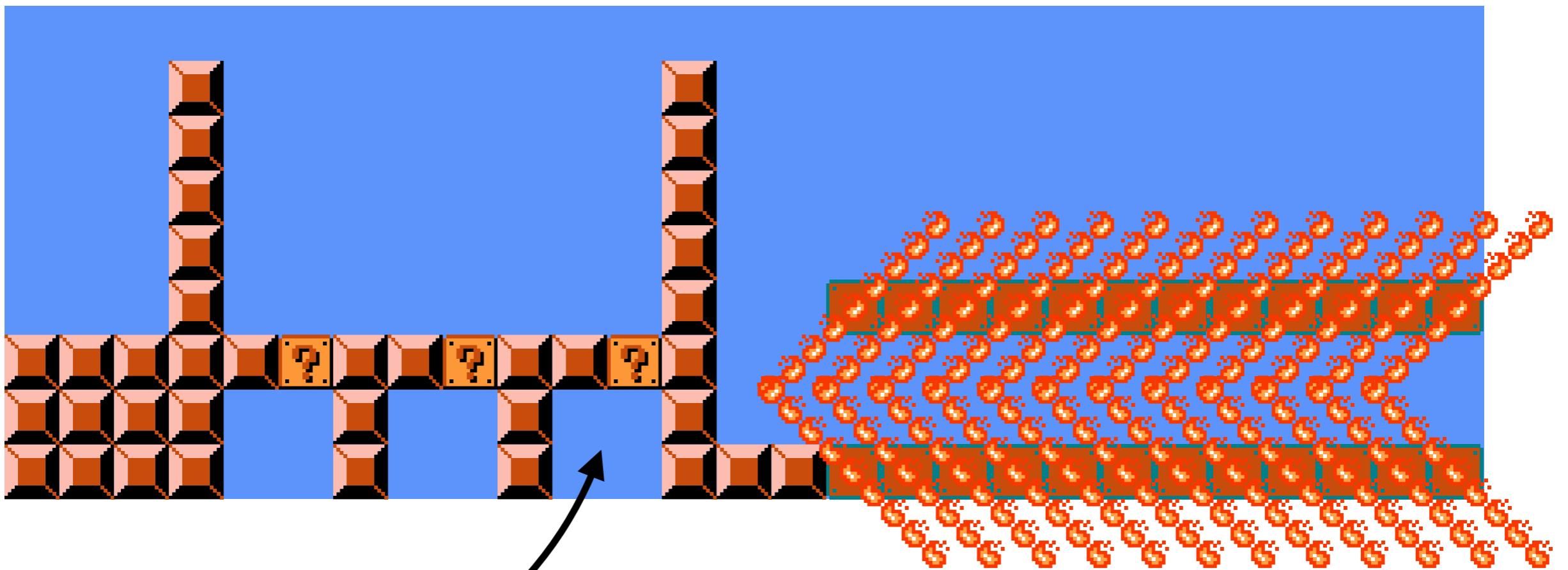


de la sortie  $x_3 = 0$   
du gadget pour  $x_3$

à l'entrée du gadget  
pour  $x_4$  (variable suivante)

# Gadget pour les clauses

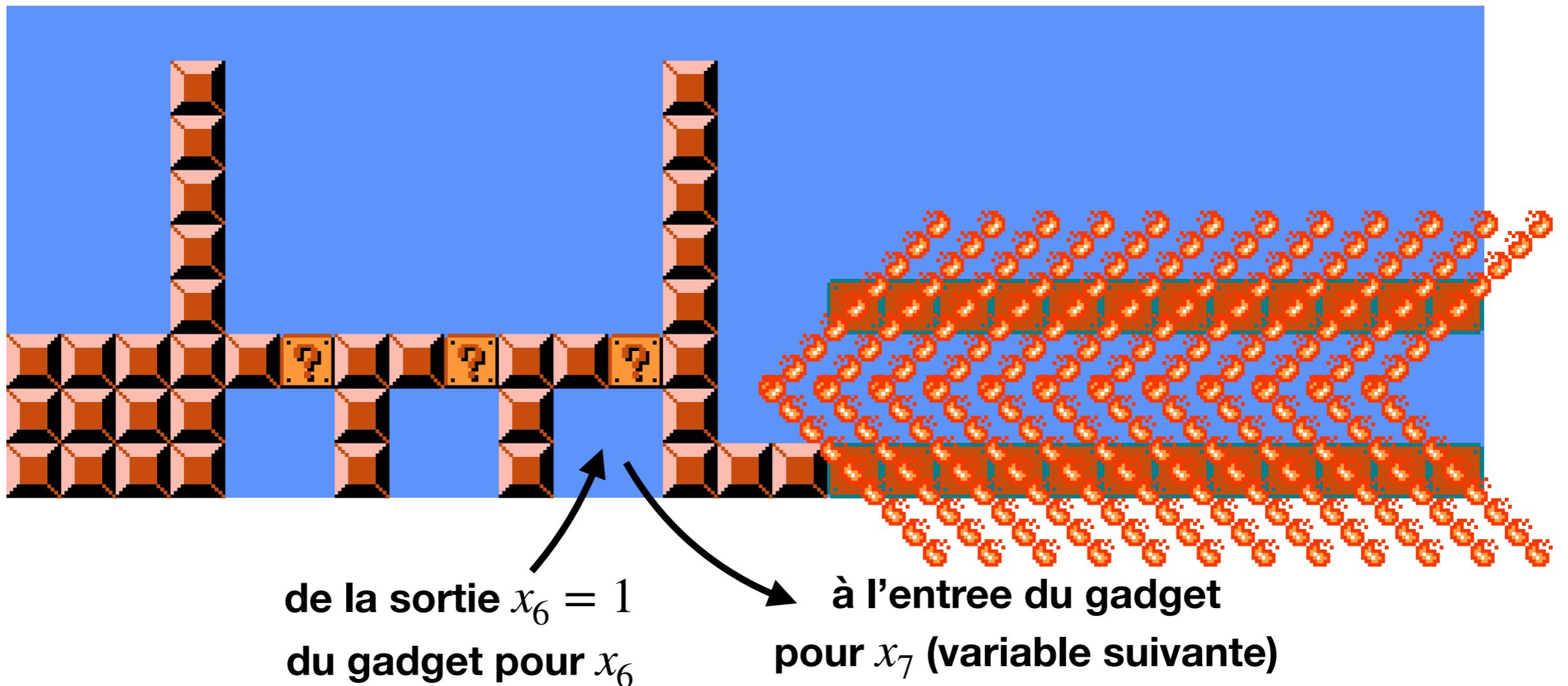
$$x_1 \vee \neg x_3 \vee x_6$$



de la sortie  $x_6 = 1$   
du gadget pour  $x_6$

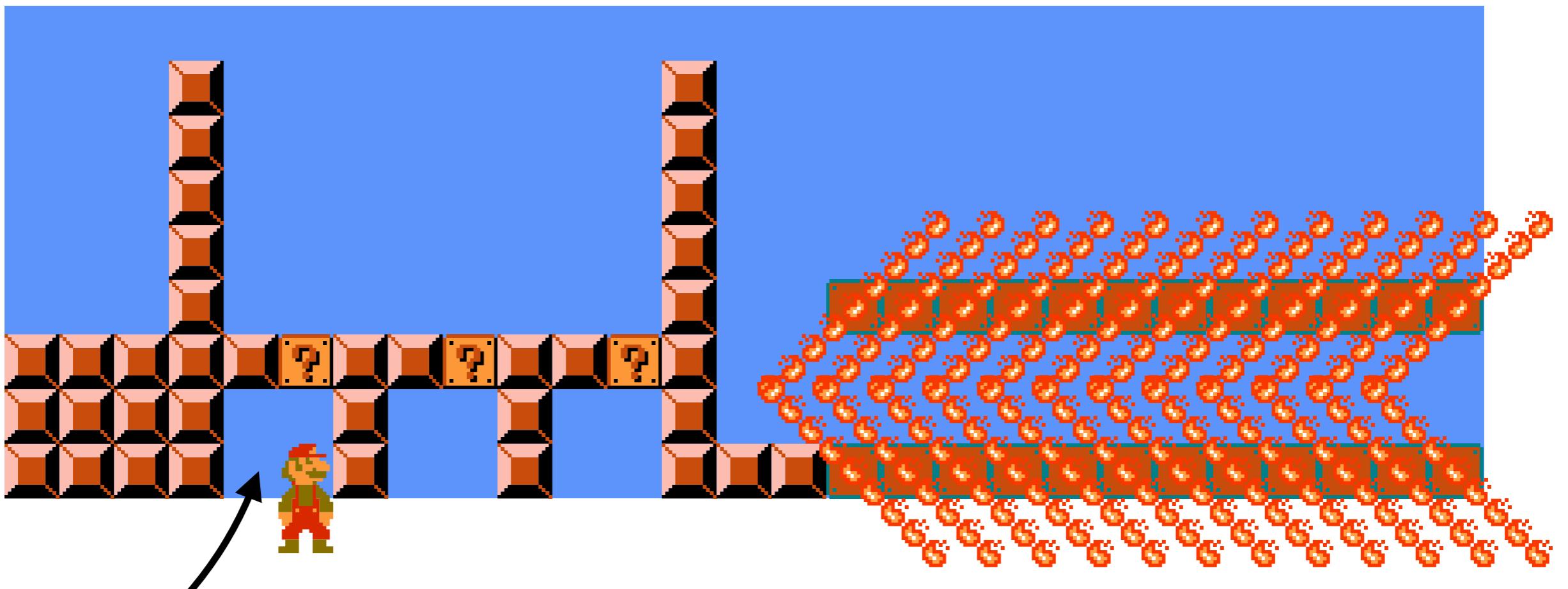
# Gadget pour les clauses

$$x_1 \vee \neg x_3 \vee x_6$$



# Gadget pour les clauses

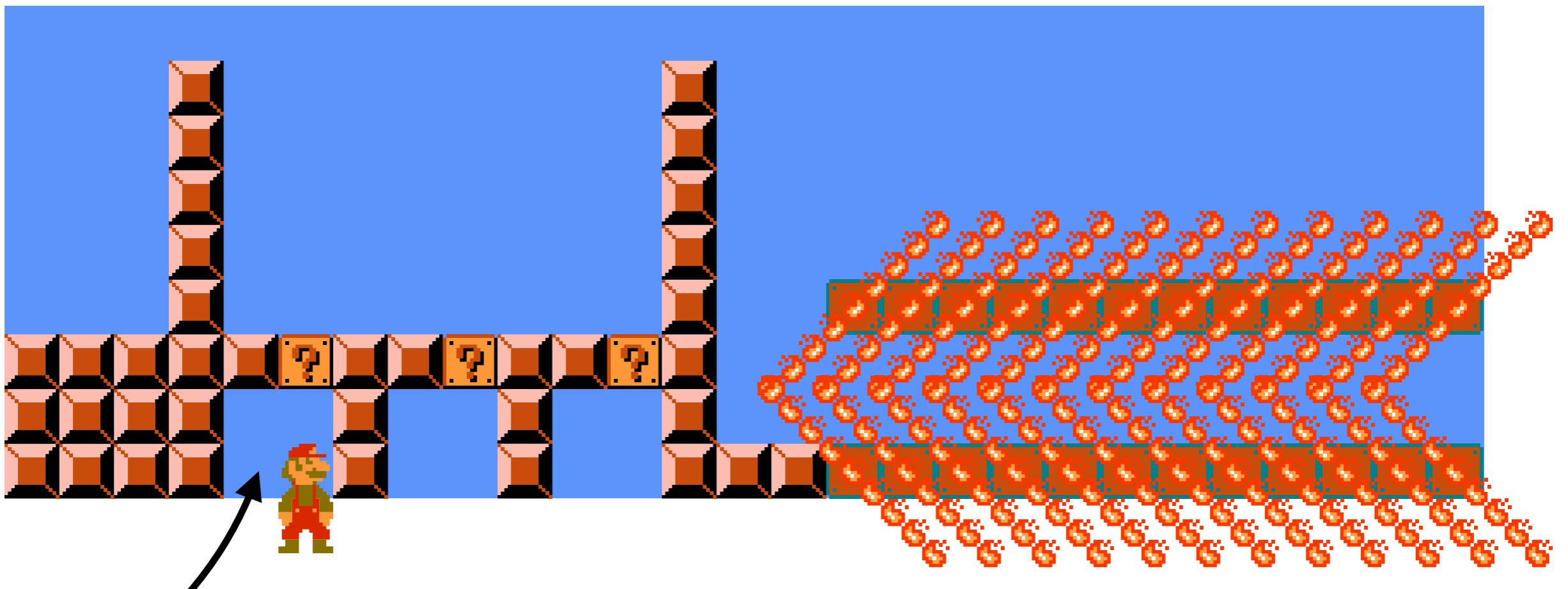
$$x_1 \vee \neg x_3 \vee x_6$$



de la sortie  $x_1 = 1$   
du gadget pour  $x_1$

# Gadget pour les clauses

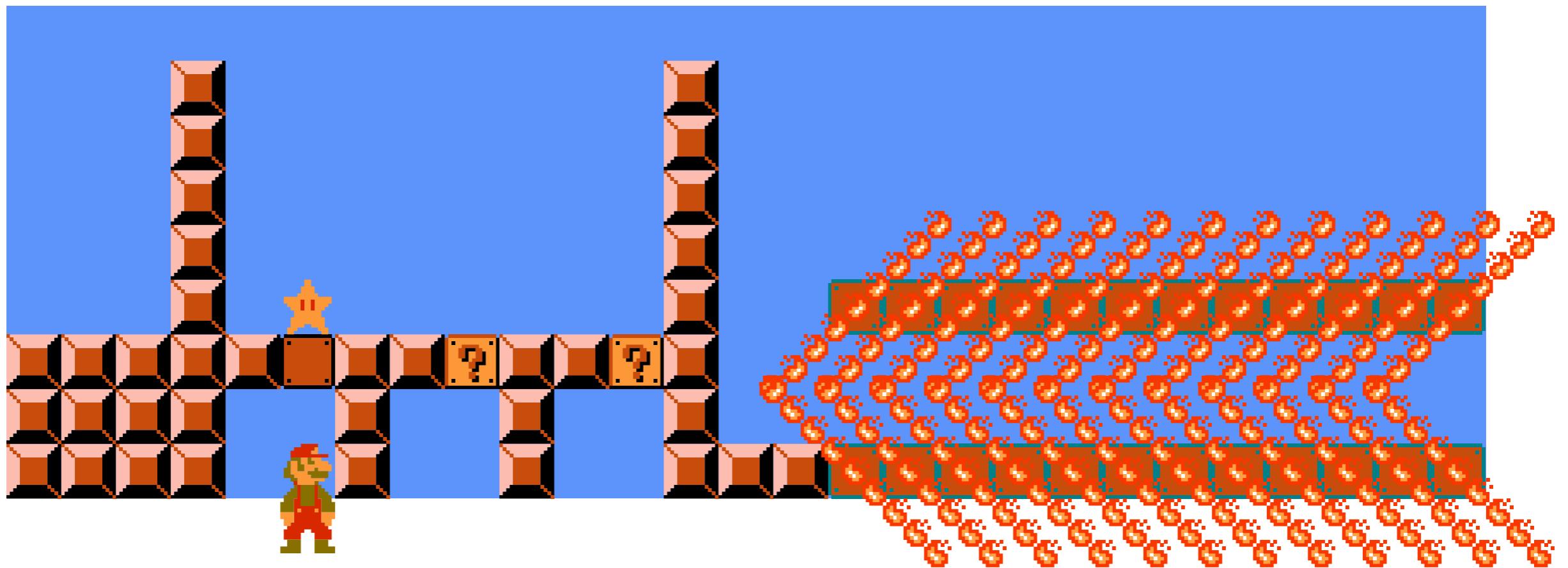
$$x_1 \vee \neg x_3 \vee x_6$$



de la sortie  $x_1 = 1$   
du gadget pour  $x_1$

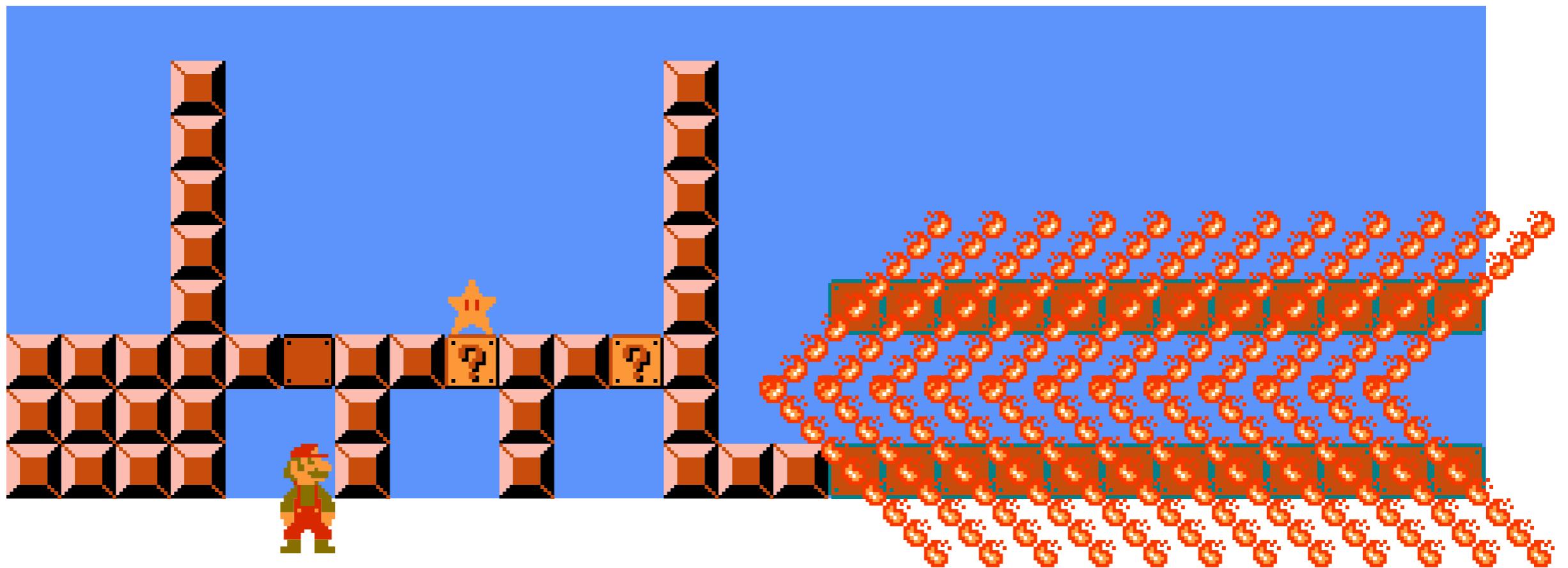
# Gadget pour les clauses

$$x_1 \vee \neg x_3 \vee x_6$$



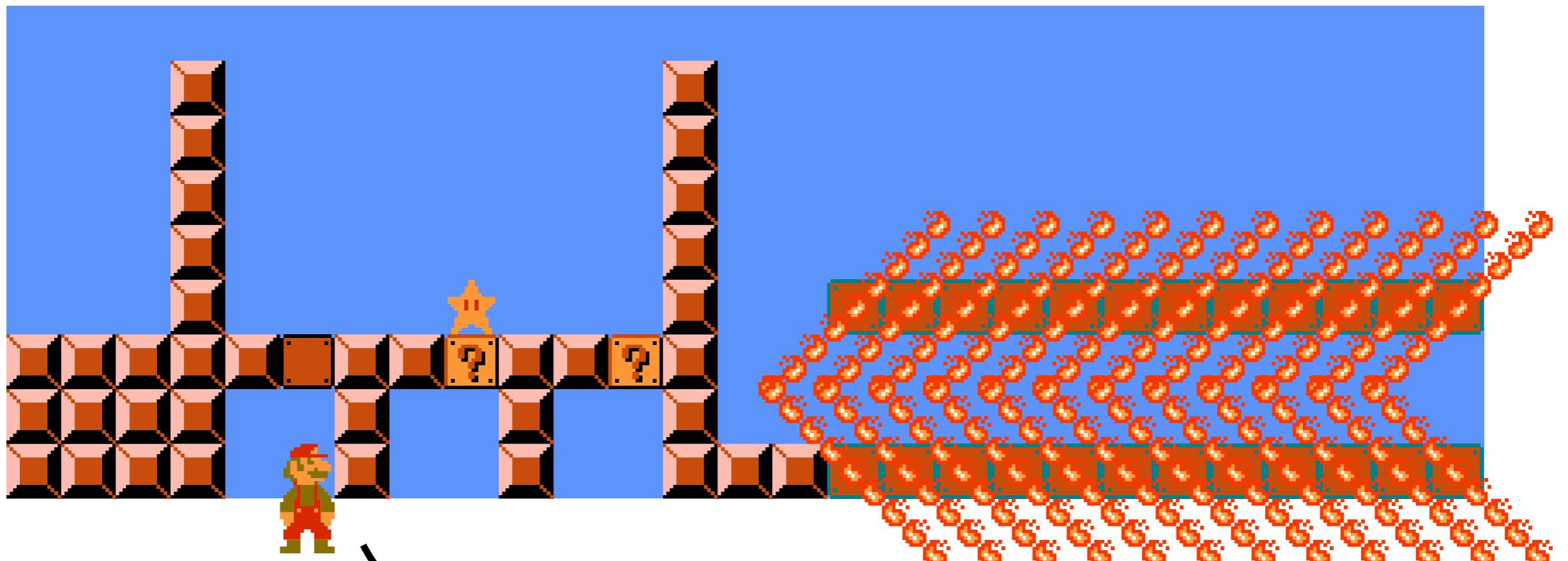
# Gadget pour les clauses

$$x_1 \vee \neg x_3 \vee x_6$$



# Gadget pour les clauses

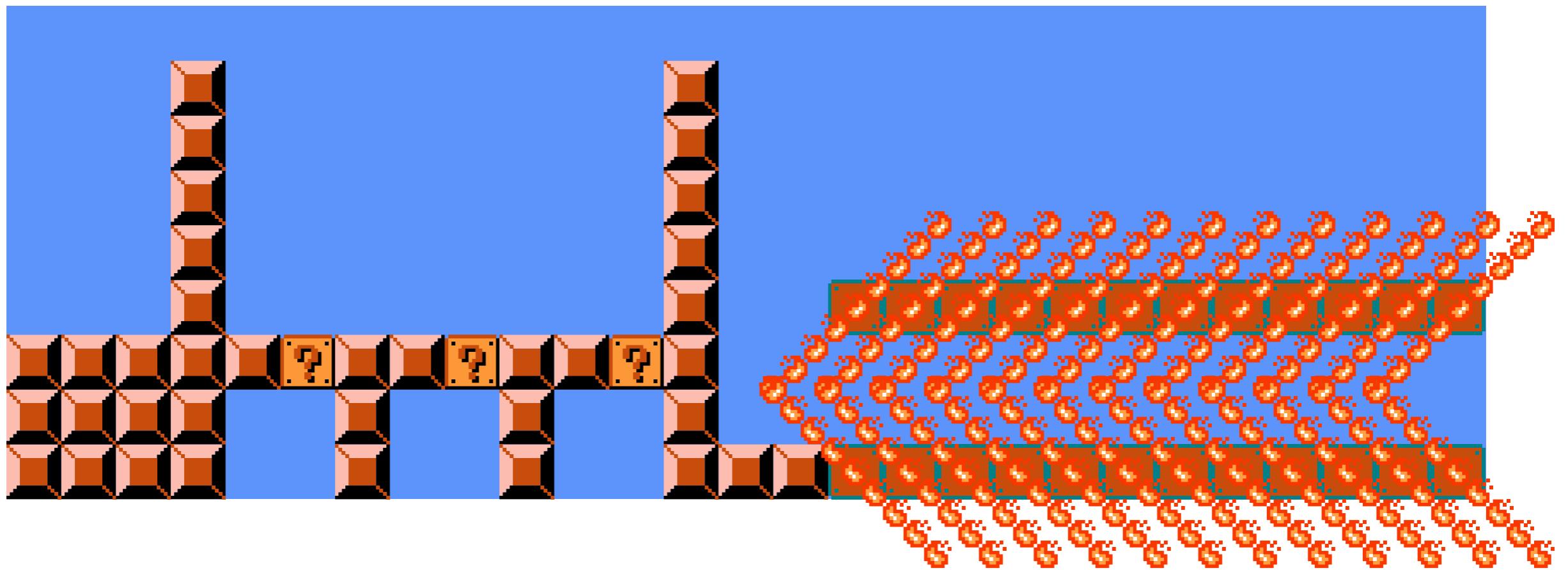
$$x_1 \vee \neg x_3 \vee x_6$$



à l'entrée du gadget  
pour  $x_2$  (variable suivante)

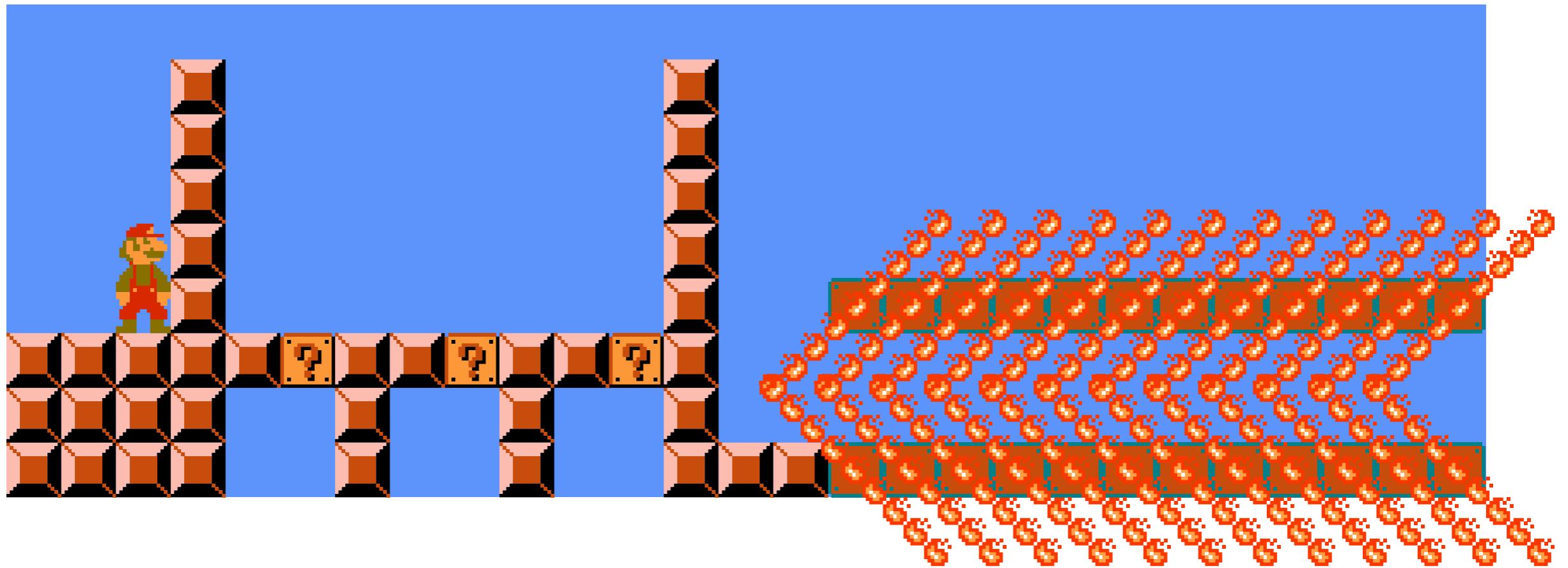
# Évaluation des clauses

$$x_1 \vee \neg x_3 \vee x_6 = 0$$



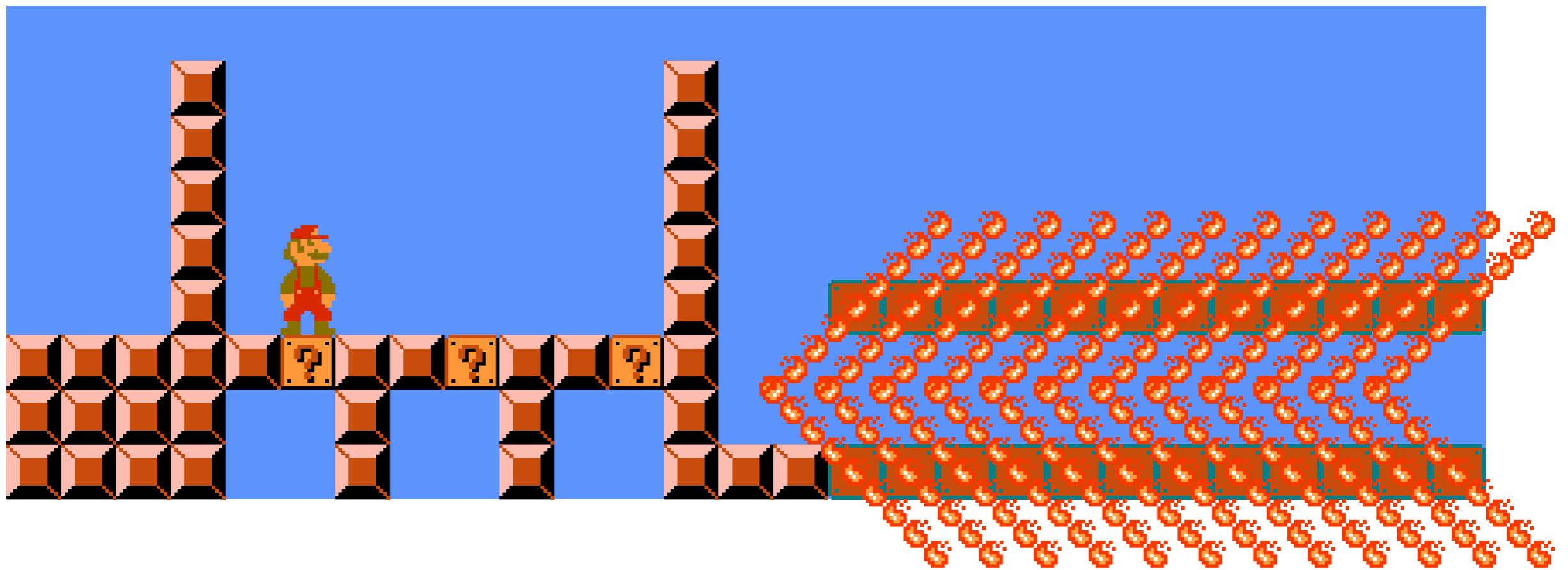
# Évaluation des clauses

$$x_1 \vee \neg x_3 \vee x_6 = 0$$



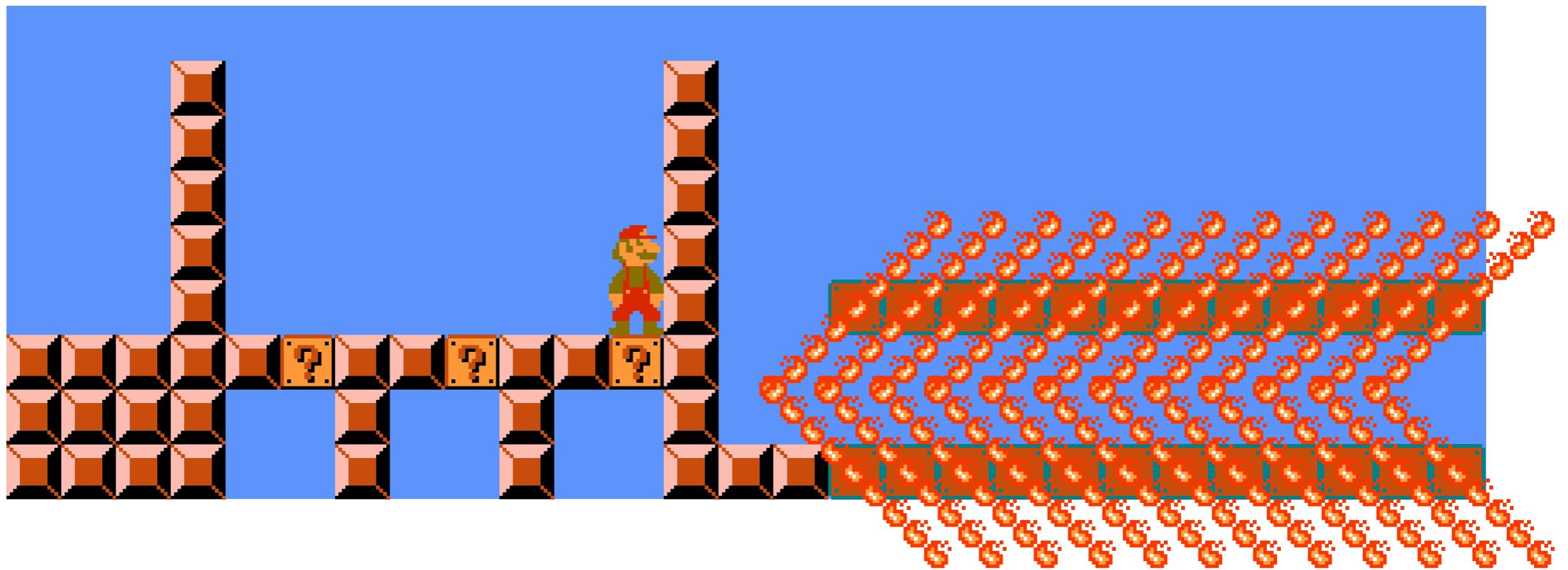
# Évaluation des clauses

$$x_1 \vee \neg x_3 \vee x_6 = 0$$



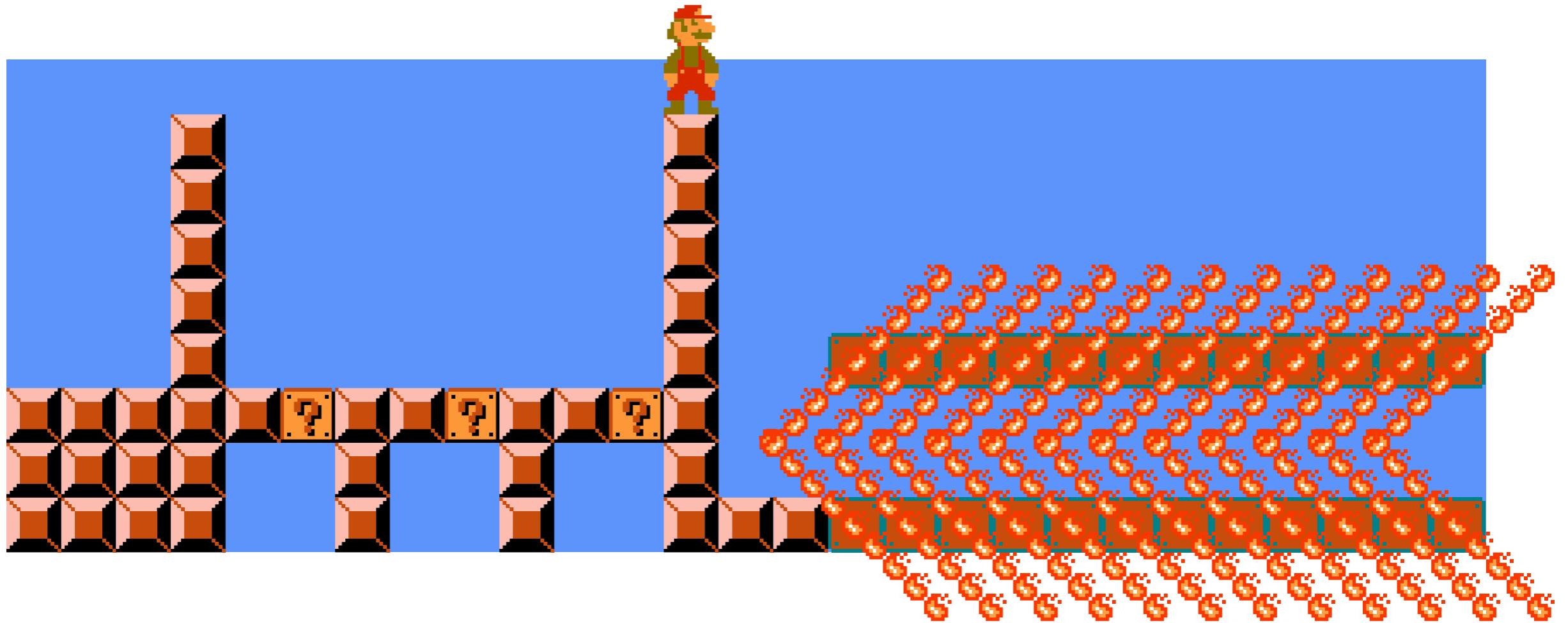
# Évaluation des clauses

$$x_1 \vee \neg x_3 \vee x_6 = 0$$



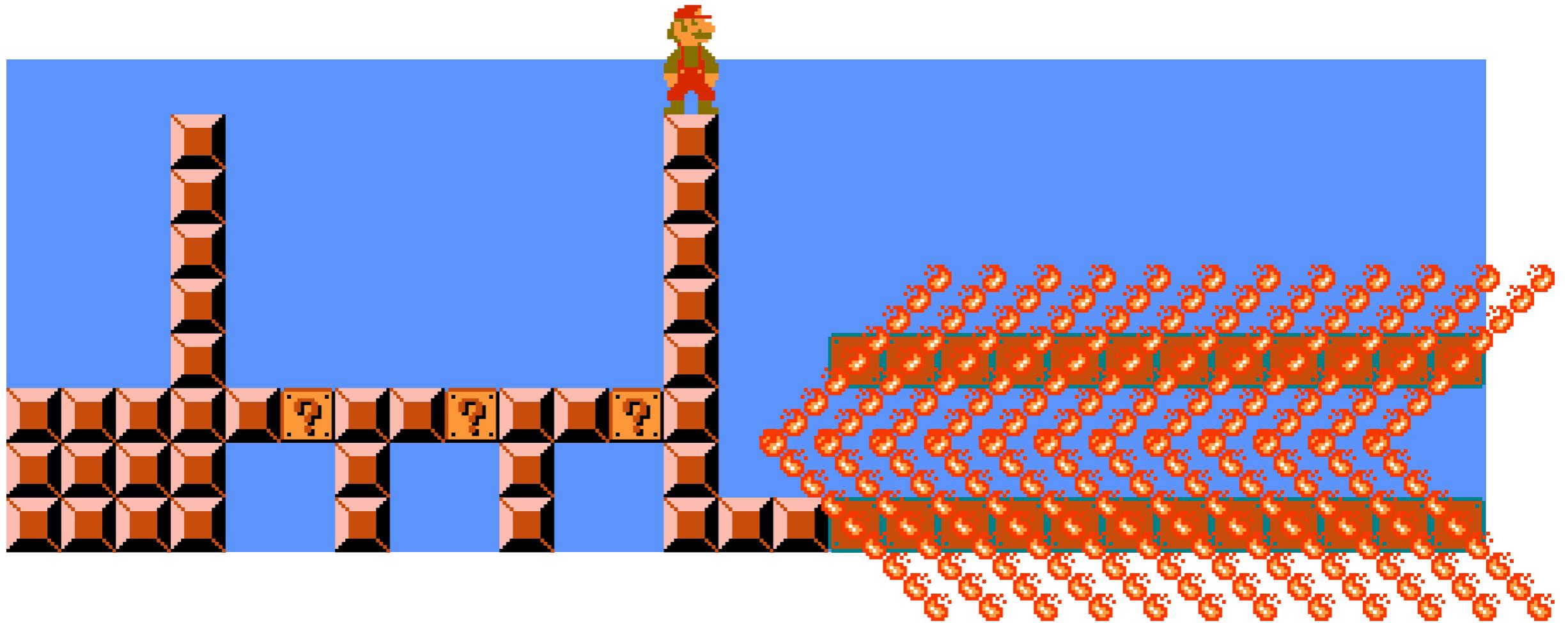
# Évaluation des clauses

$$x_1 \vee \neg x_3 \vee x_6 = 0$$



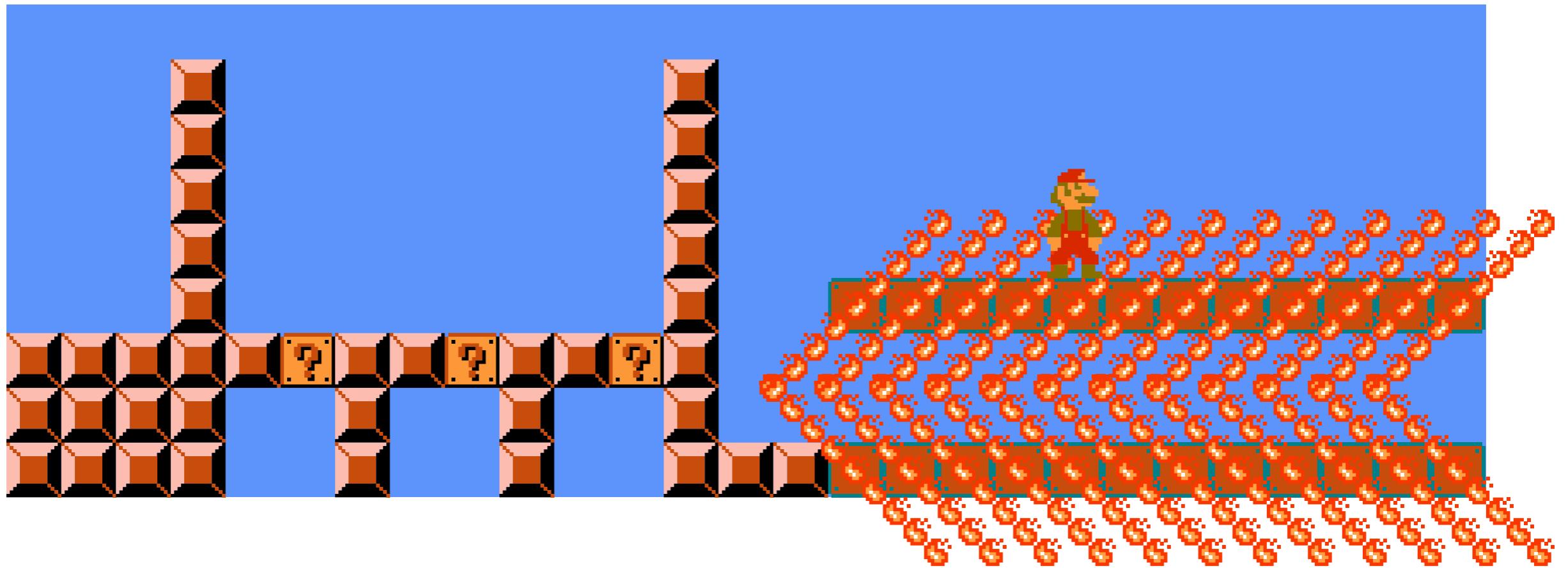
# Évaluation des clauses

$$x_1 \vee \neg x_3 \vee x_6 = 0$$



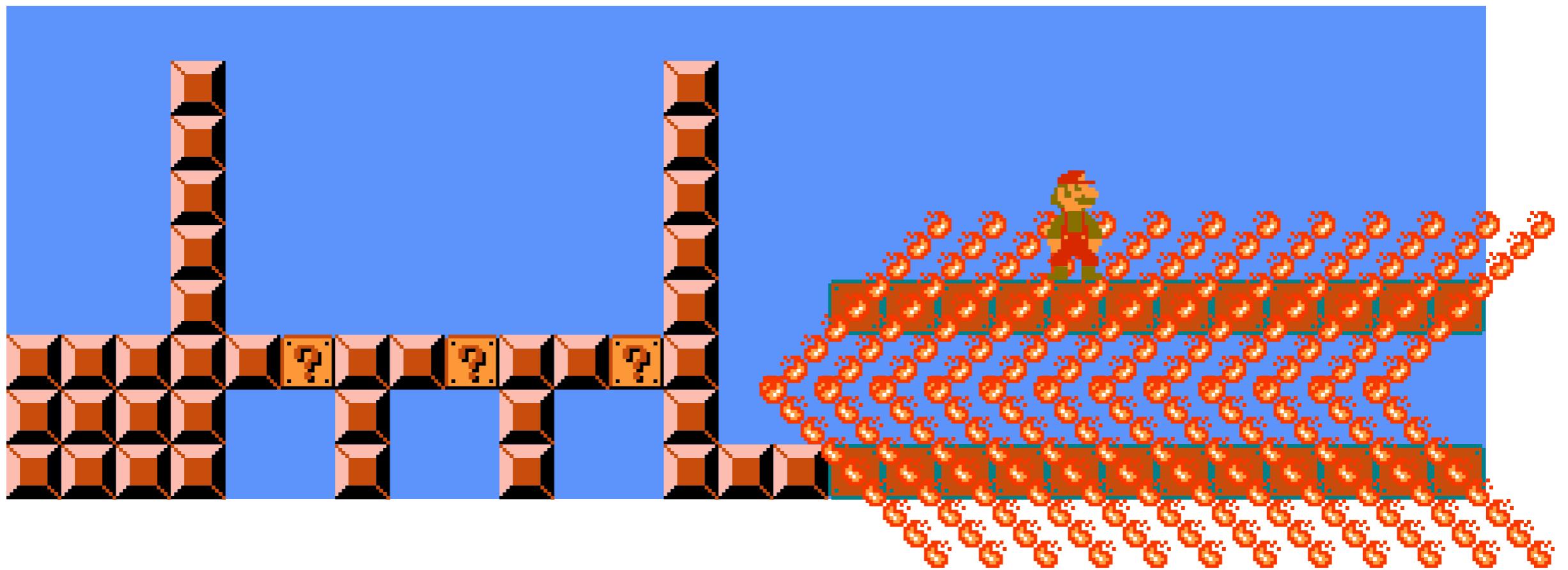
# Évaluation des clauses

$$x_1 \vee \neg x_3 \vee x_6 = 0$$



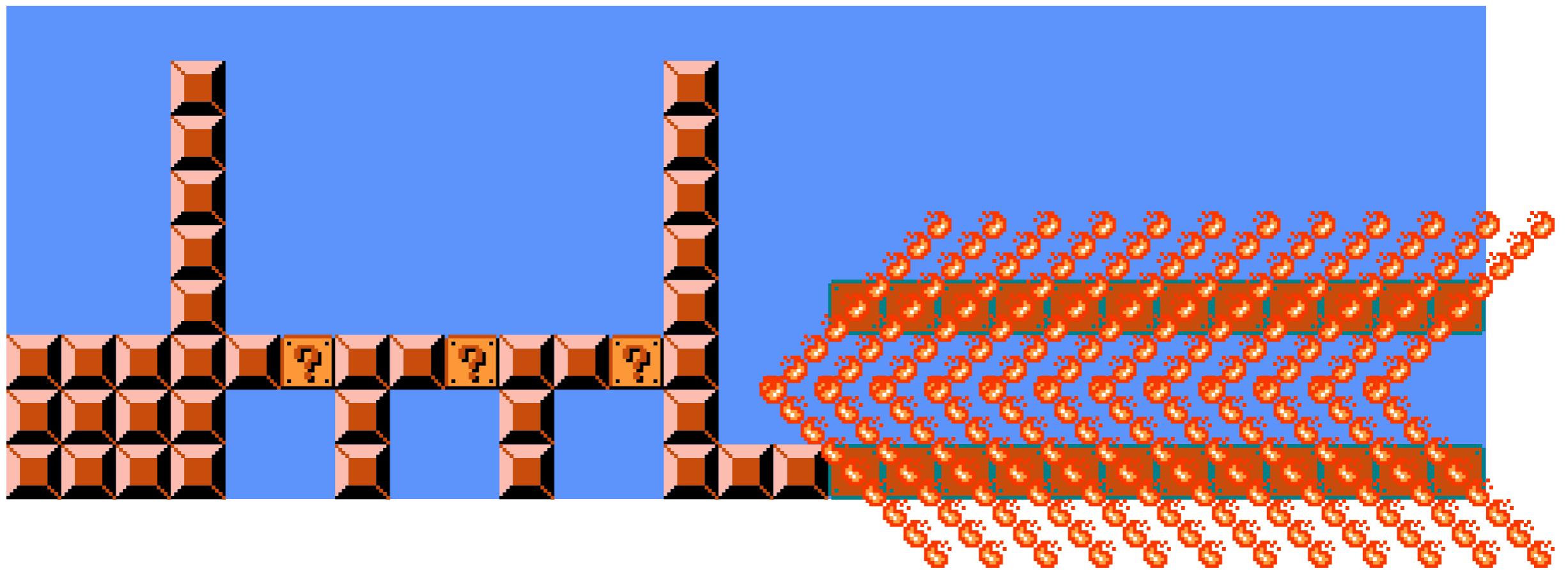
# Évaluation des clauses

$$x_1 \vee \neg x_3 \vee x_6 = 0$$



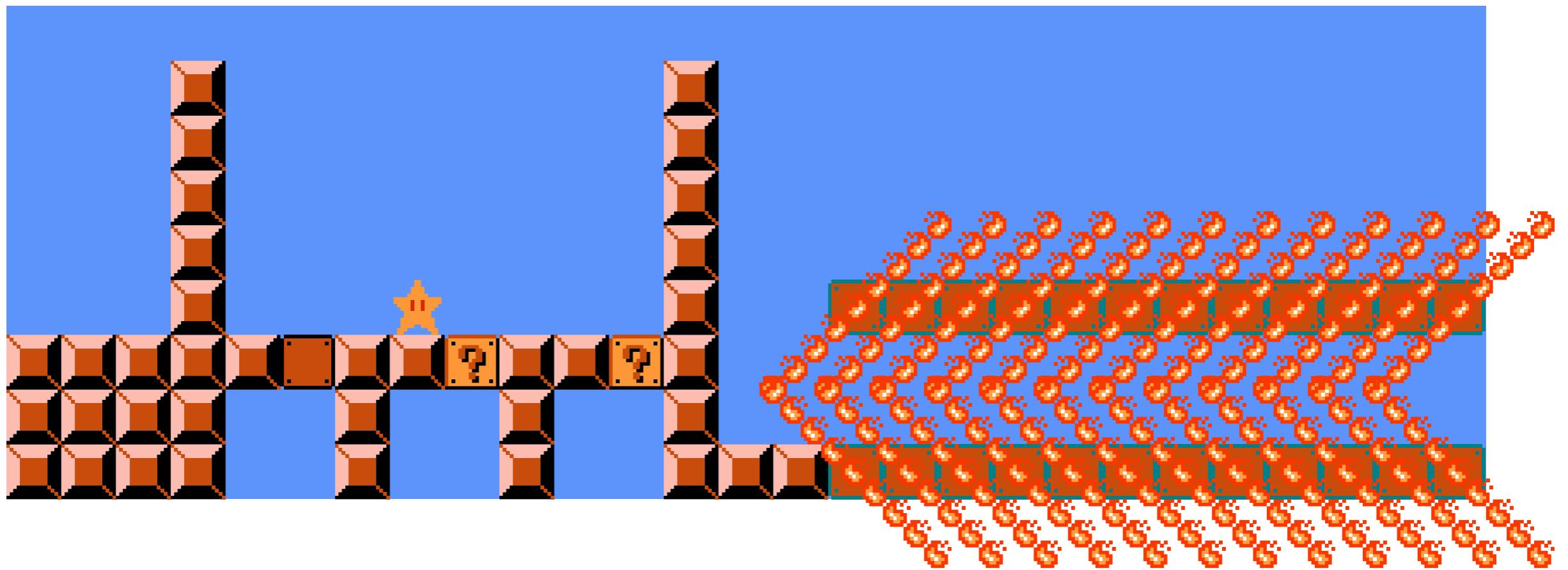
# Évaluation des clauses

$$x_1 \vee \neg x_3 \vee x_6 = 0$$



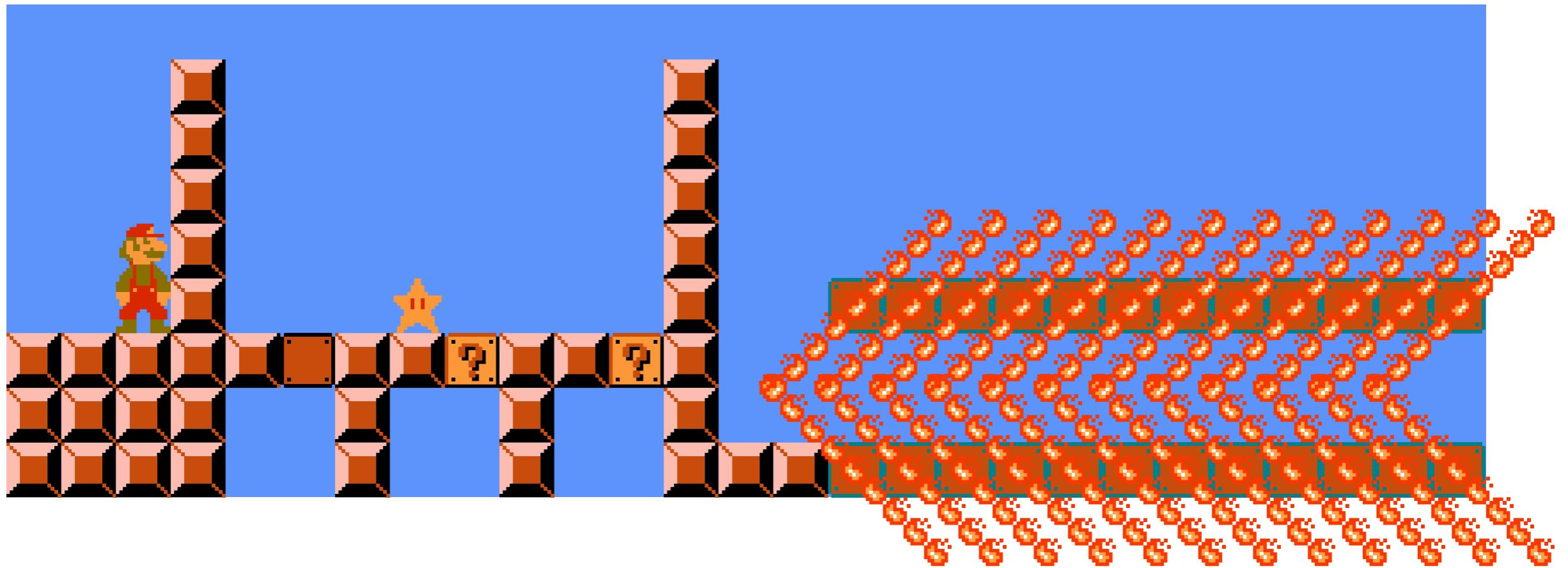
# Évaluation des clauses

$$x_1 \vee \neg x_3 \vee x_6 = 1$$



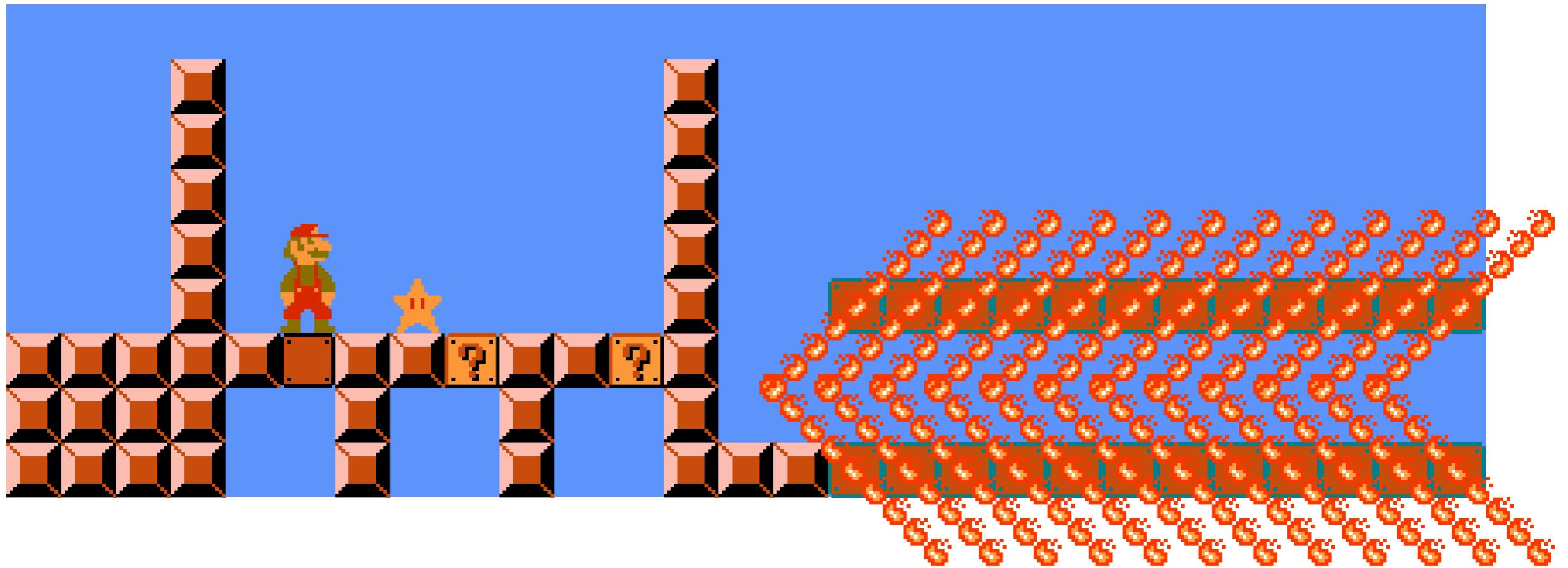
# Évaluation des clauses

$$x_1 \vee \neg x_3 \vee x_6 = 1$$



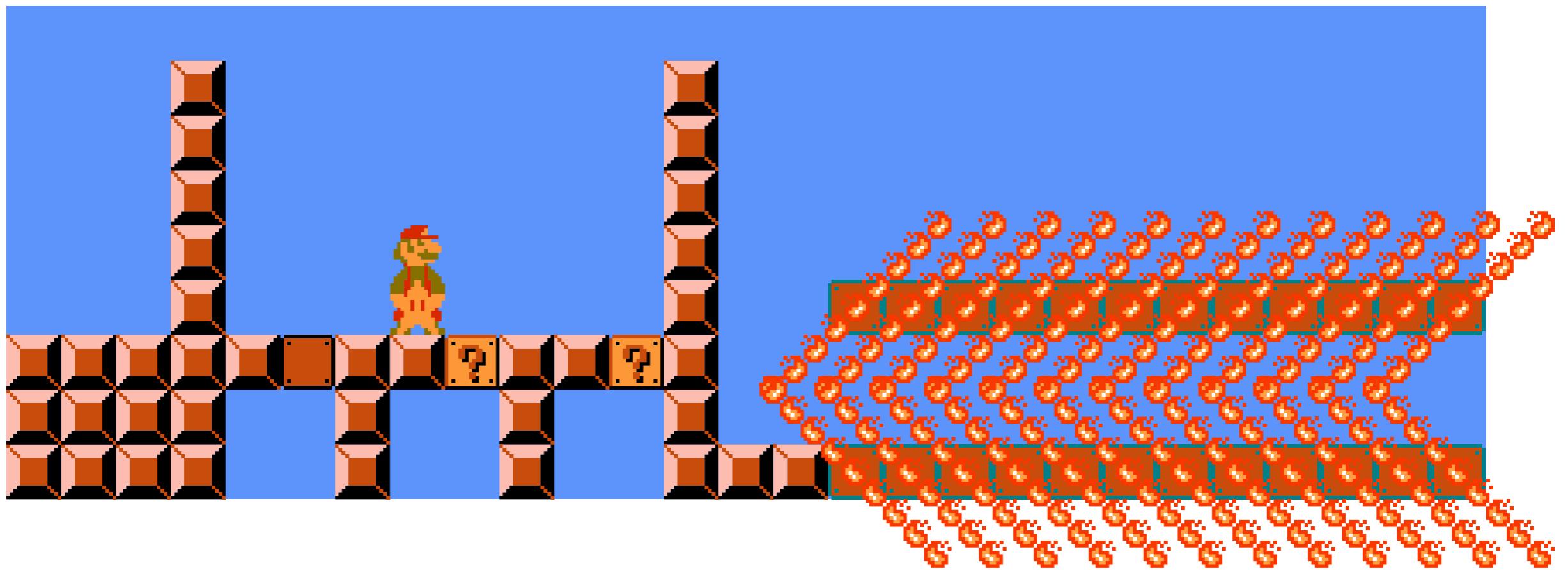
# Évaluation des clauses

$$x_1 \vee \neg x_3 \vee x_6 = 1$$



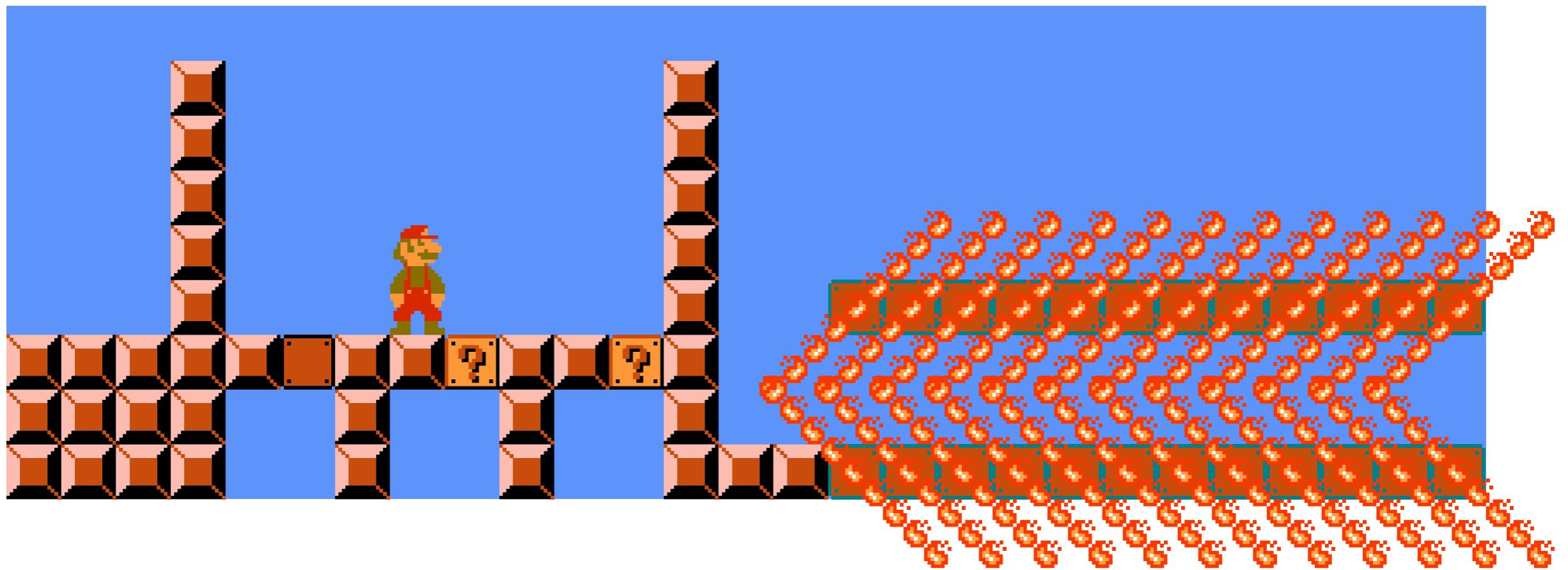
# Évaluation des clauses

$$x_1 \vee \neg x_3 \vee x_6 = 1$$



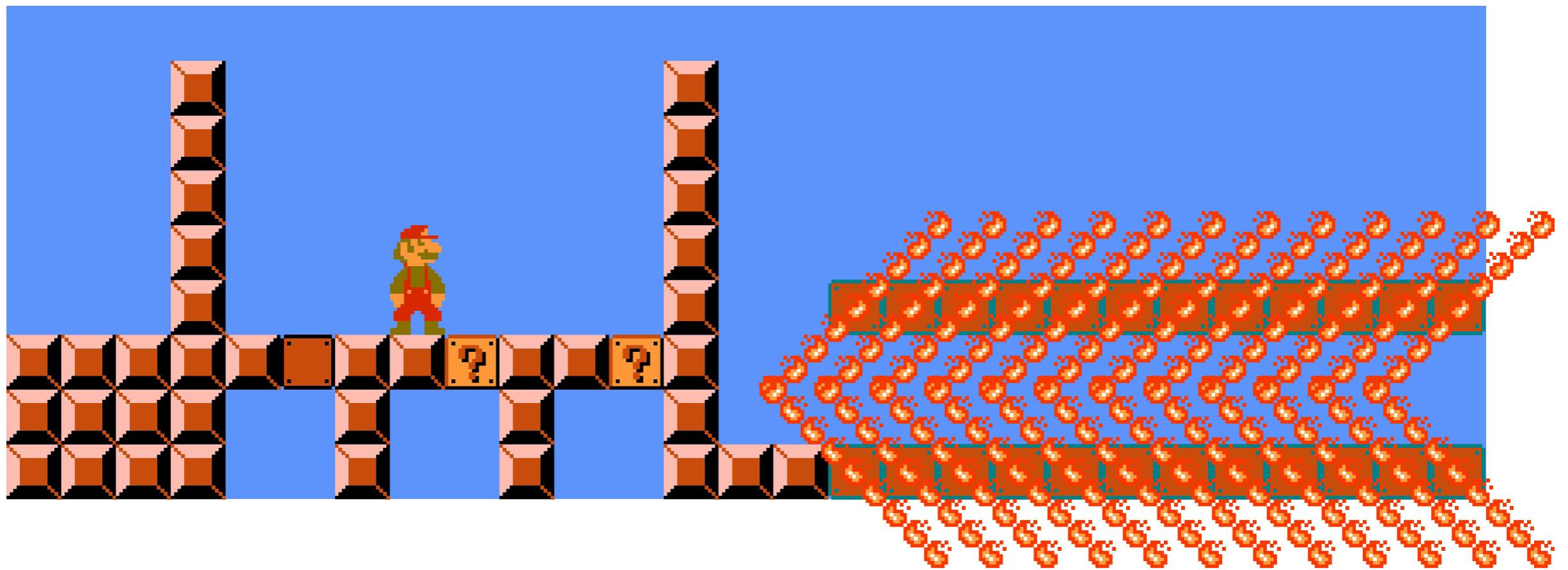
# Évaluation des clauses

$$x_1 \vee \neg x_3 \vee x_6 = 1$$



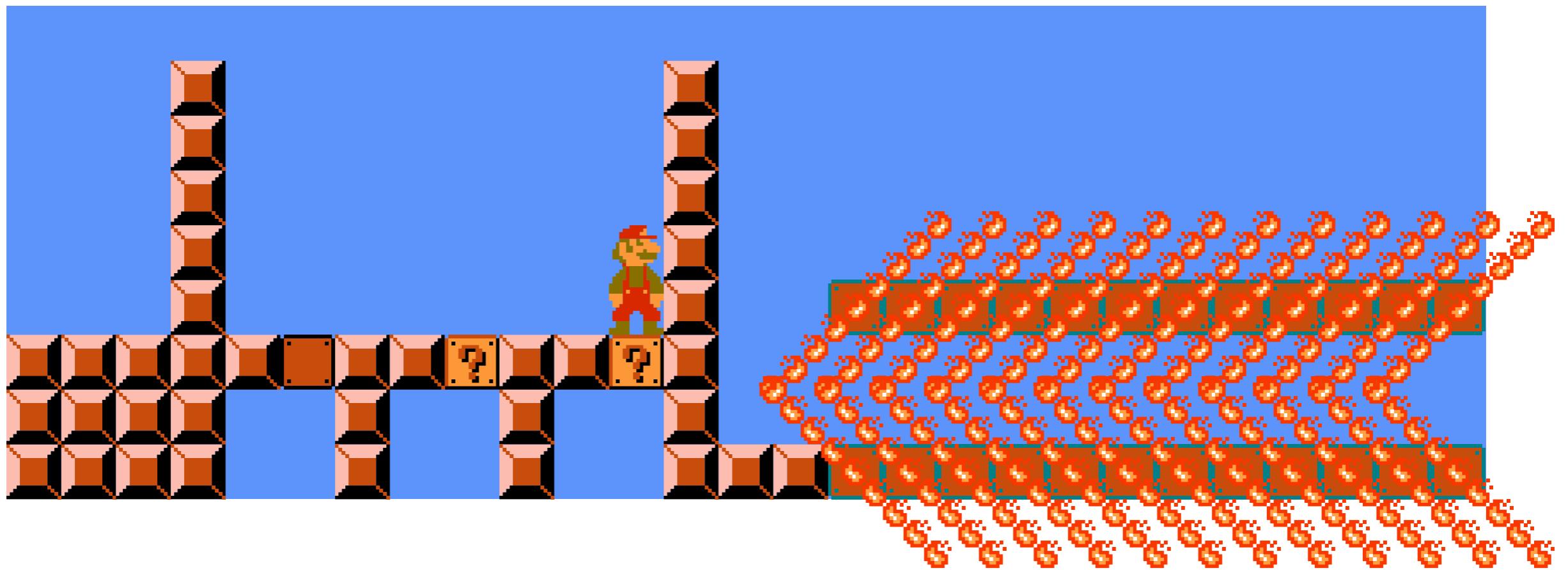
# Évaluation des clauses

$$x_1 \vee \neg x_3 \vee x_6 = 1$$



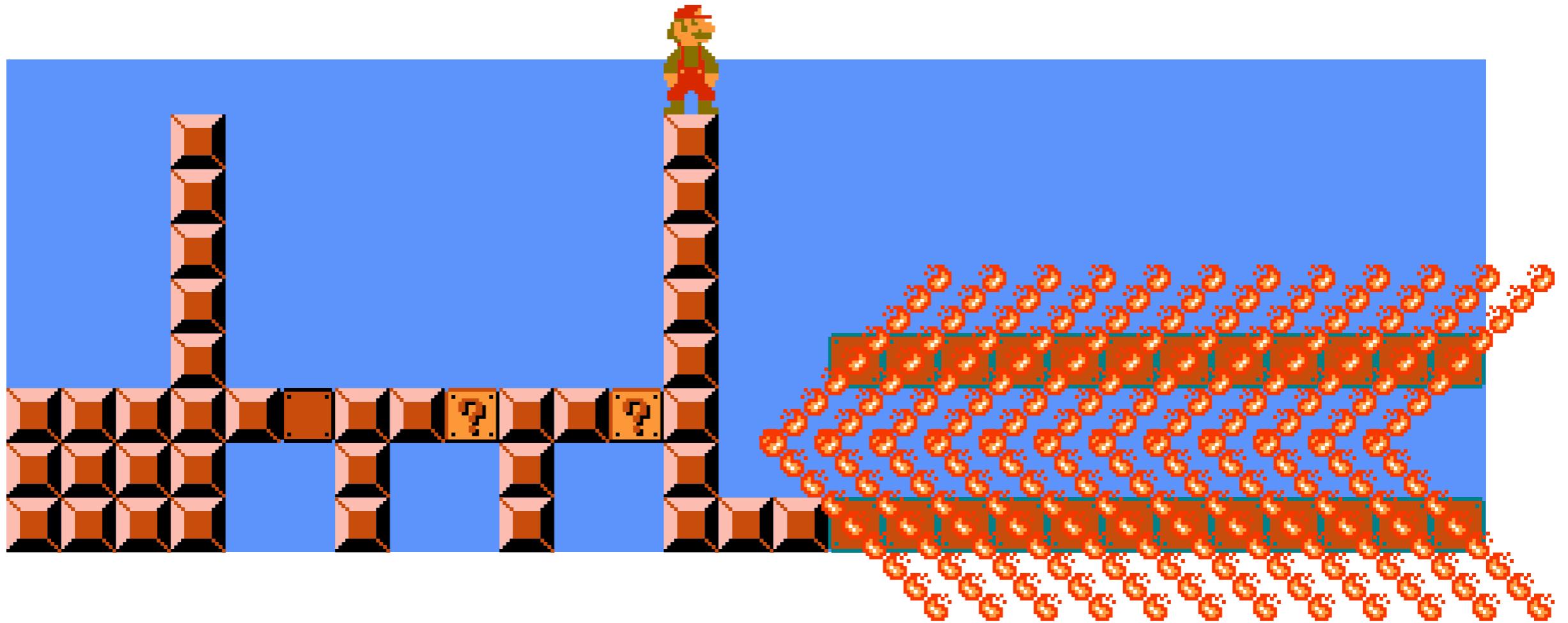
# Évaluation des clauses

$$x_1 \vee \neg x_3 \vee x_6 = 1$$



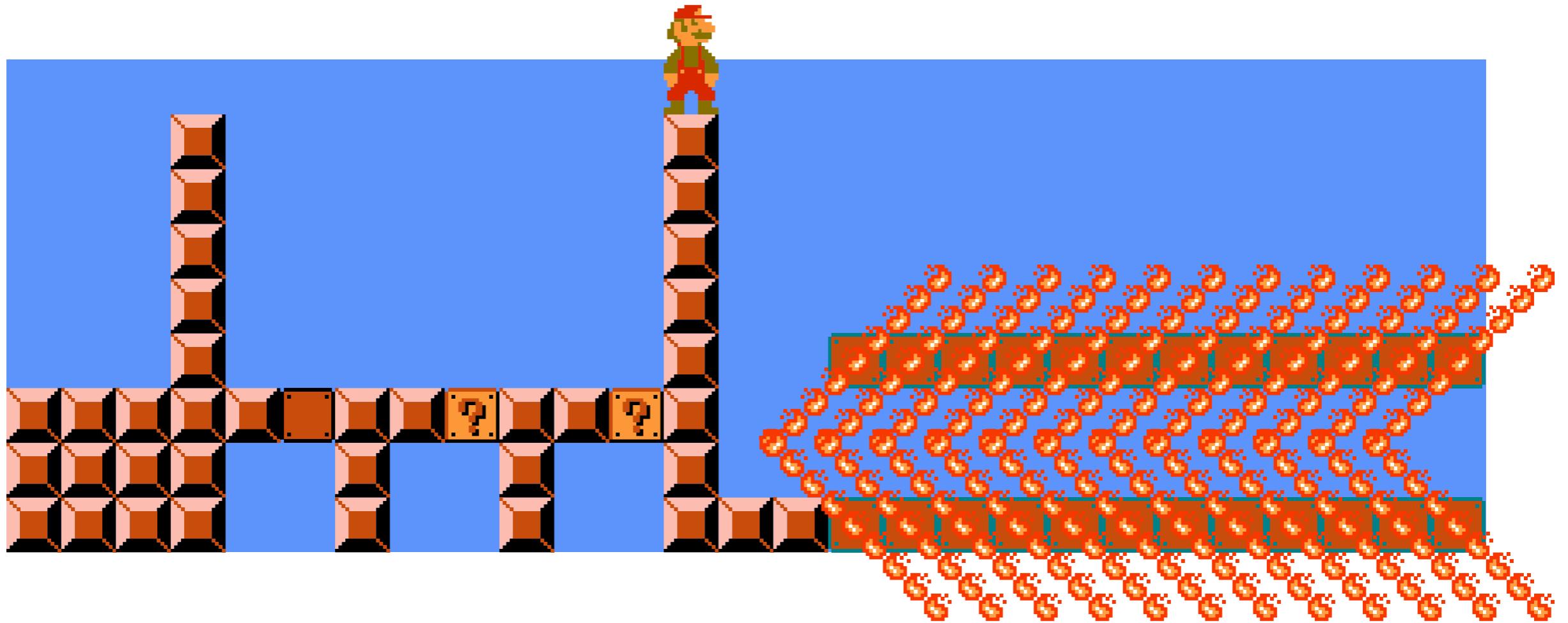
# Évaluation des clauses

$$x_1 \vee \neg x_3 \vee x_6 = 1$$



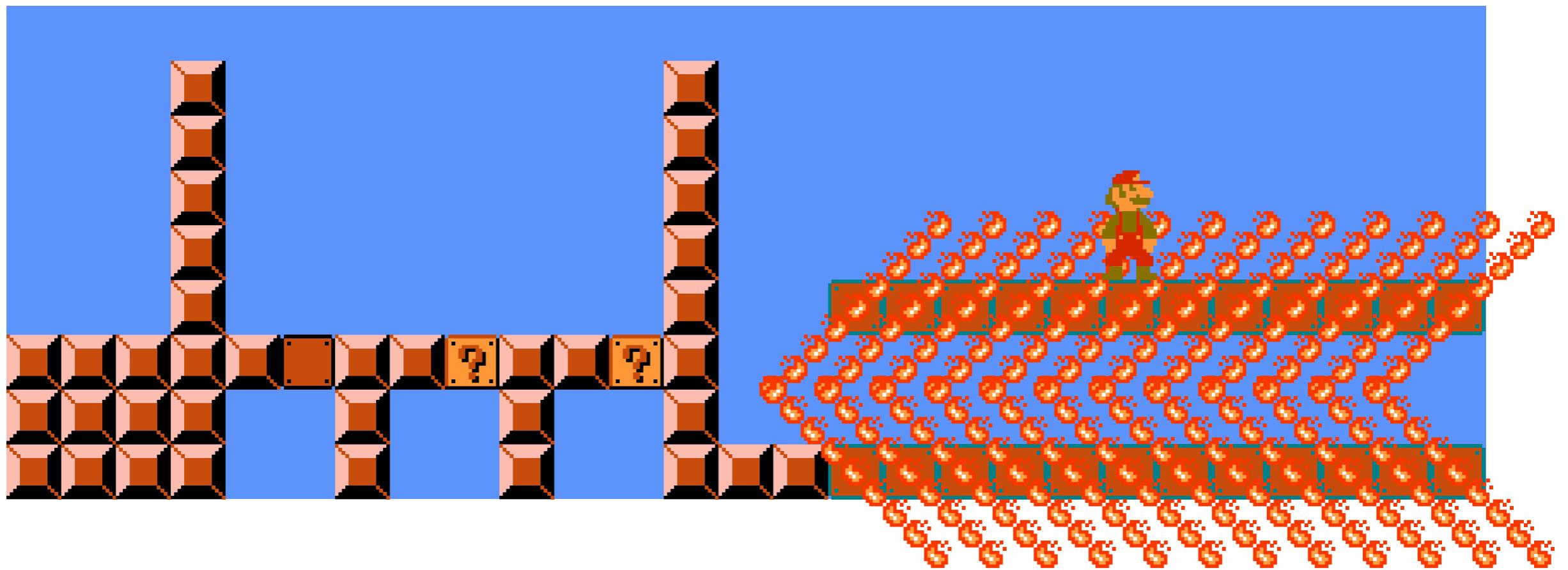
# Évaluation des clauses

$$x_1 \vee \neg x_3 \vee x_6 = 1$$



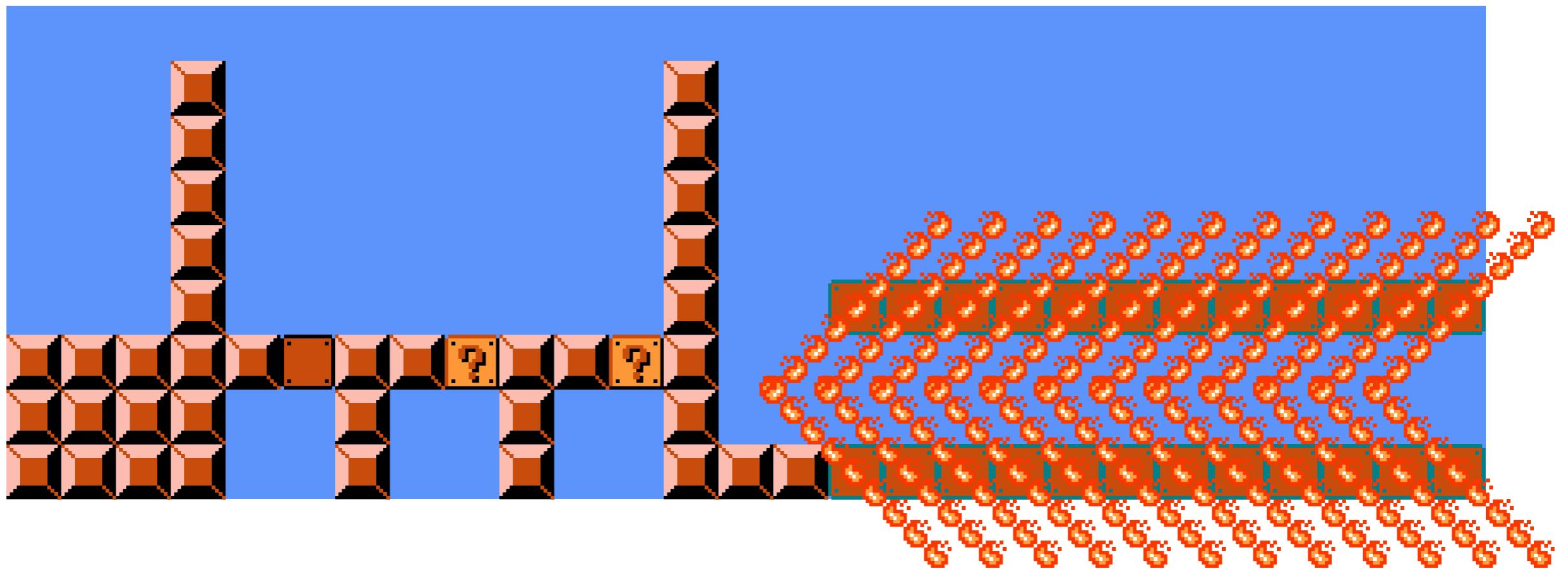
# Évaluation des clauses

$$x_1 \vee \neg x_3 \vee x_6 = 1$$

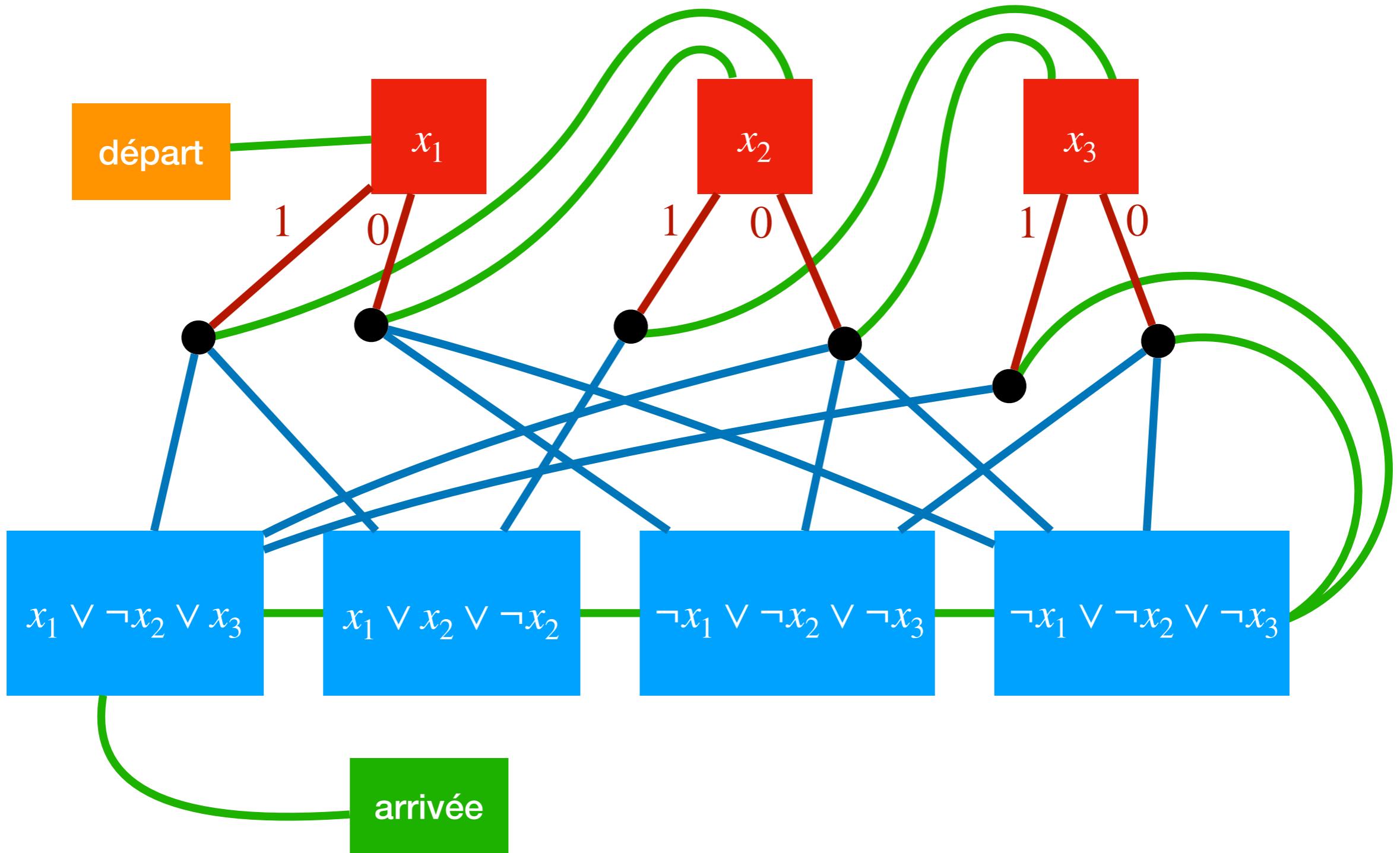


# Évaluation des clauses

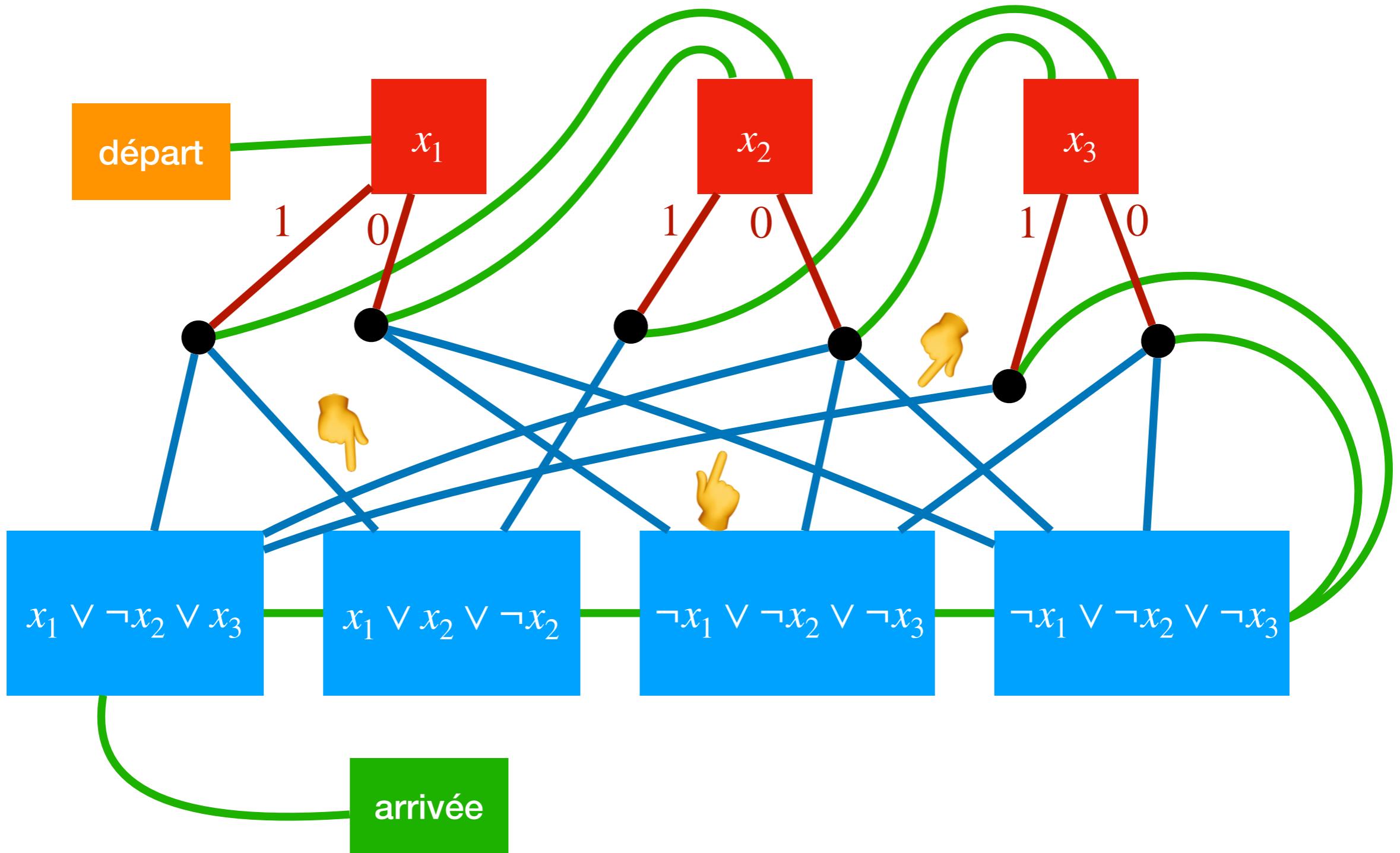
$$x_1 \vee \neg x_3 \vee x_6 = 1$$



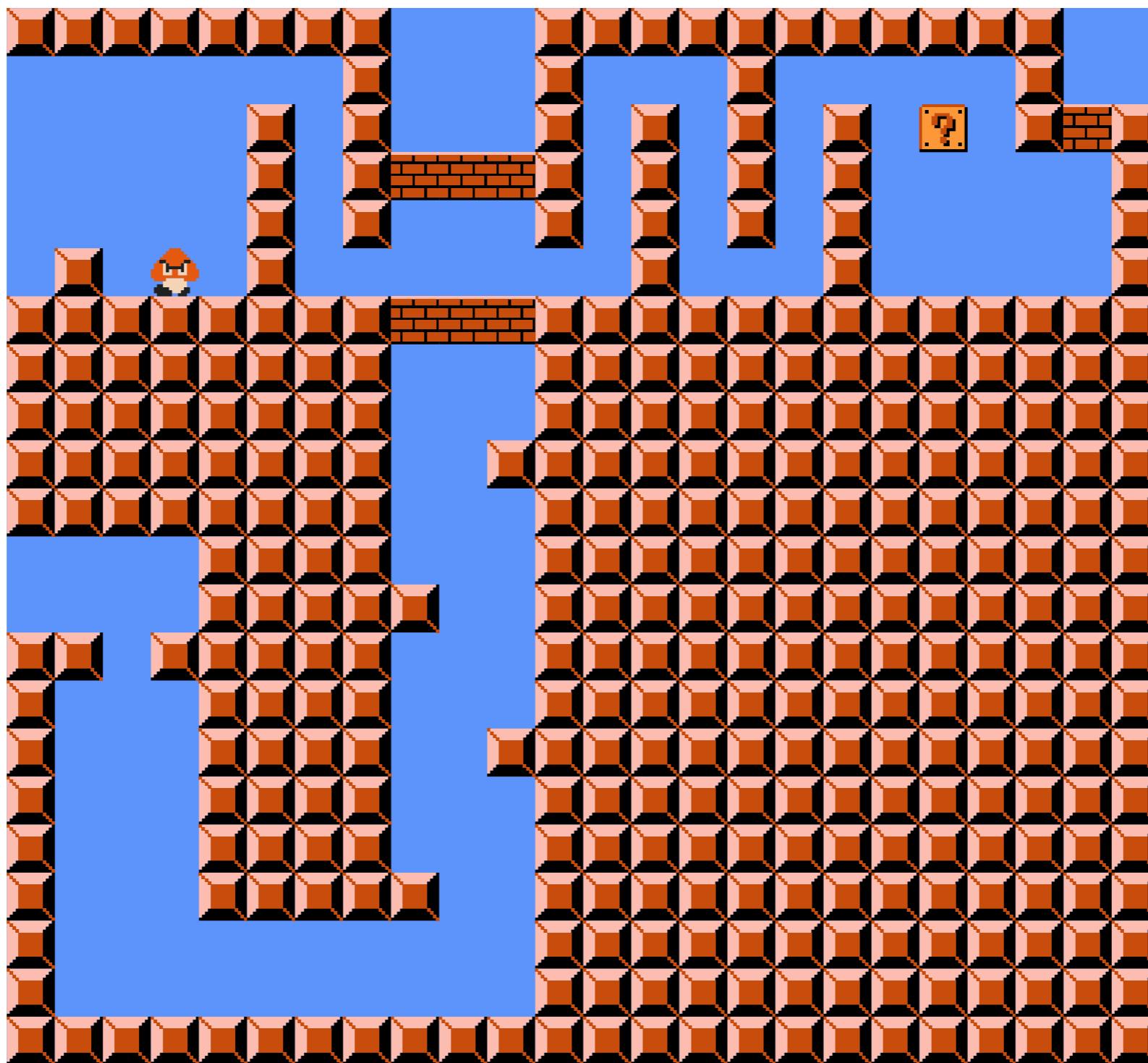
# Et les croisements ?



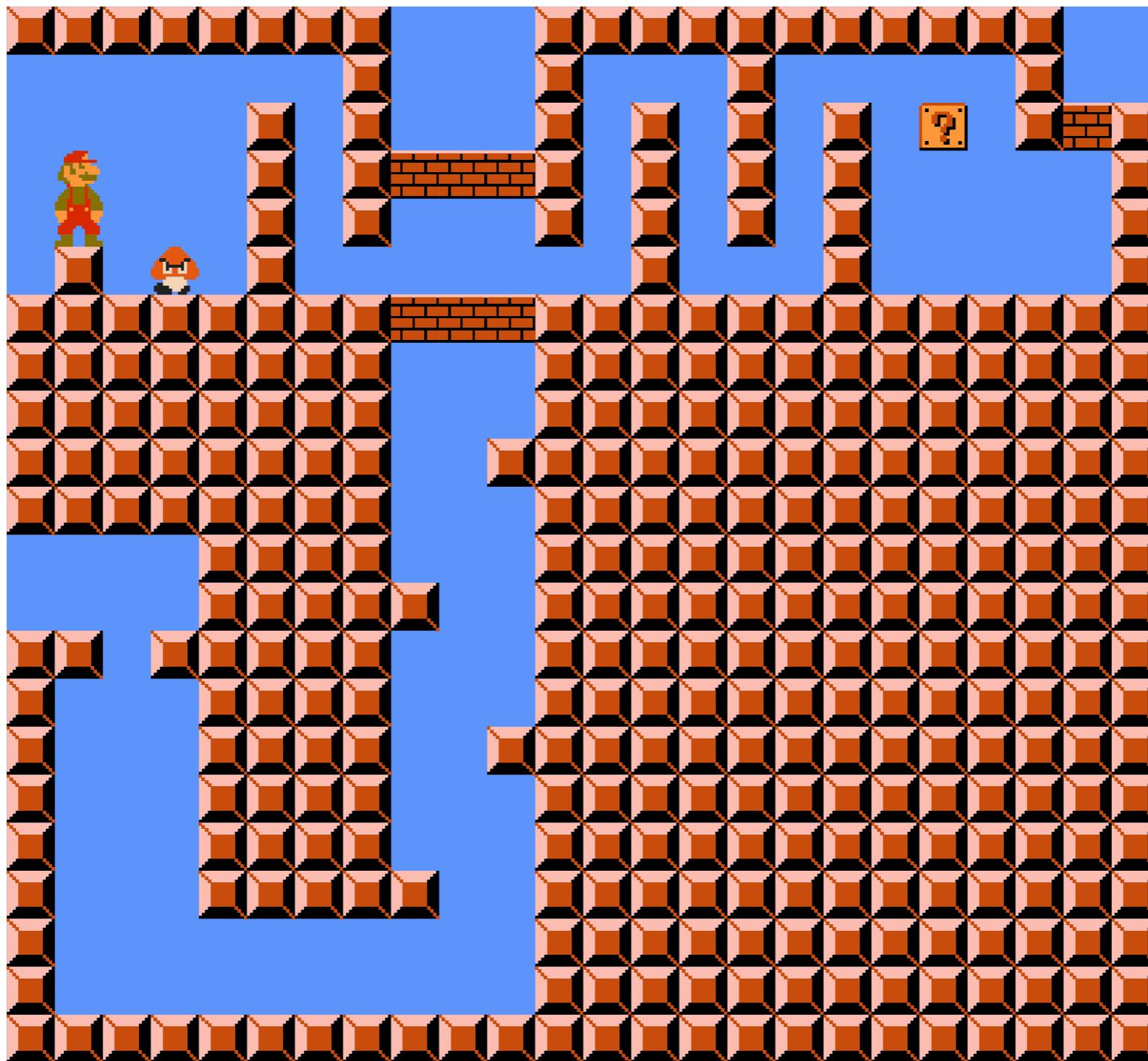
# Et les croisements ?



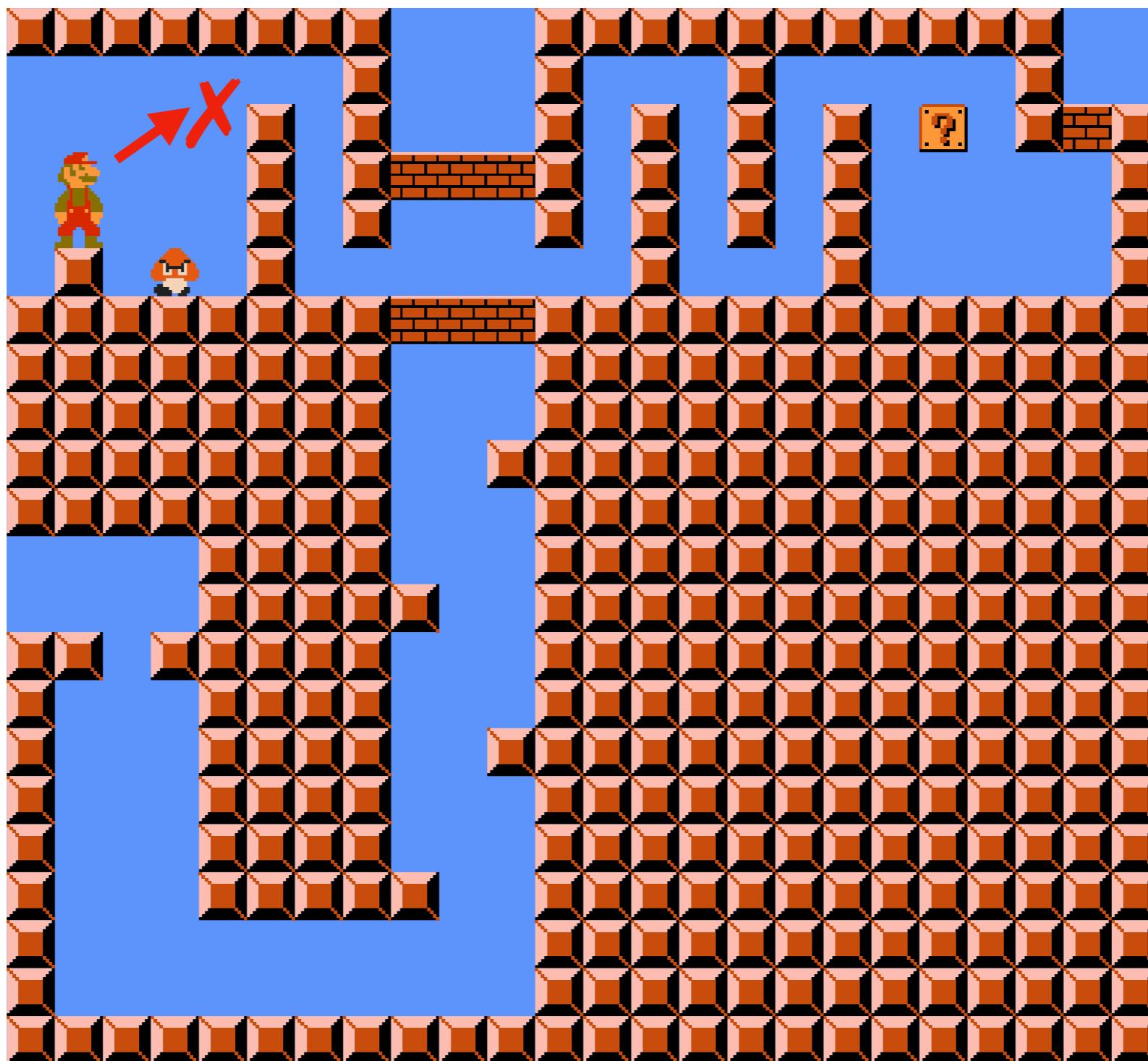
# Gadget pour les croisements



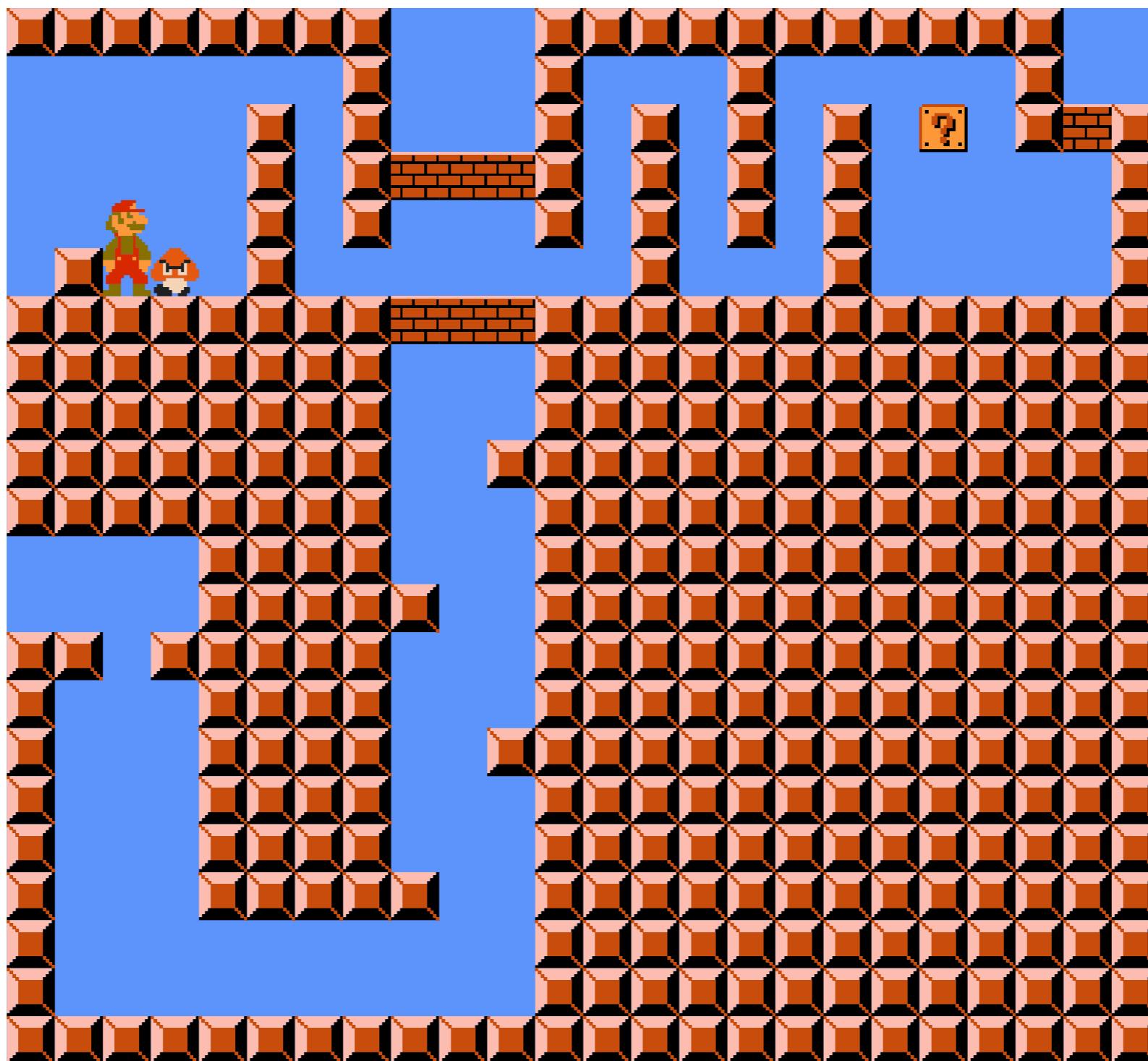
# Gadget pour les croisements



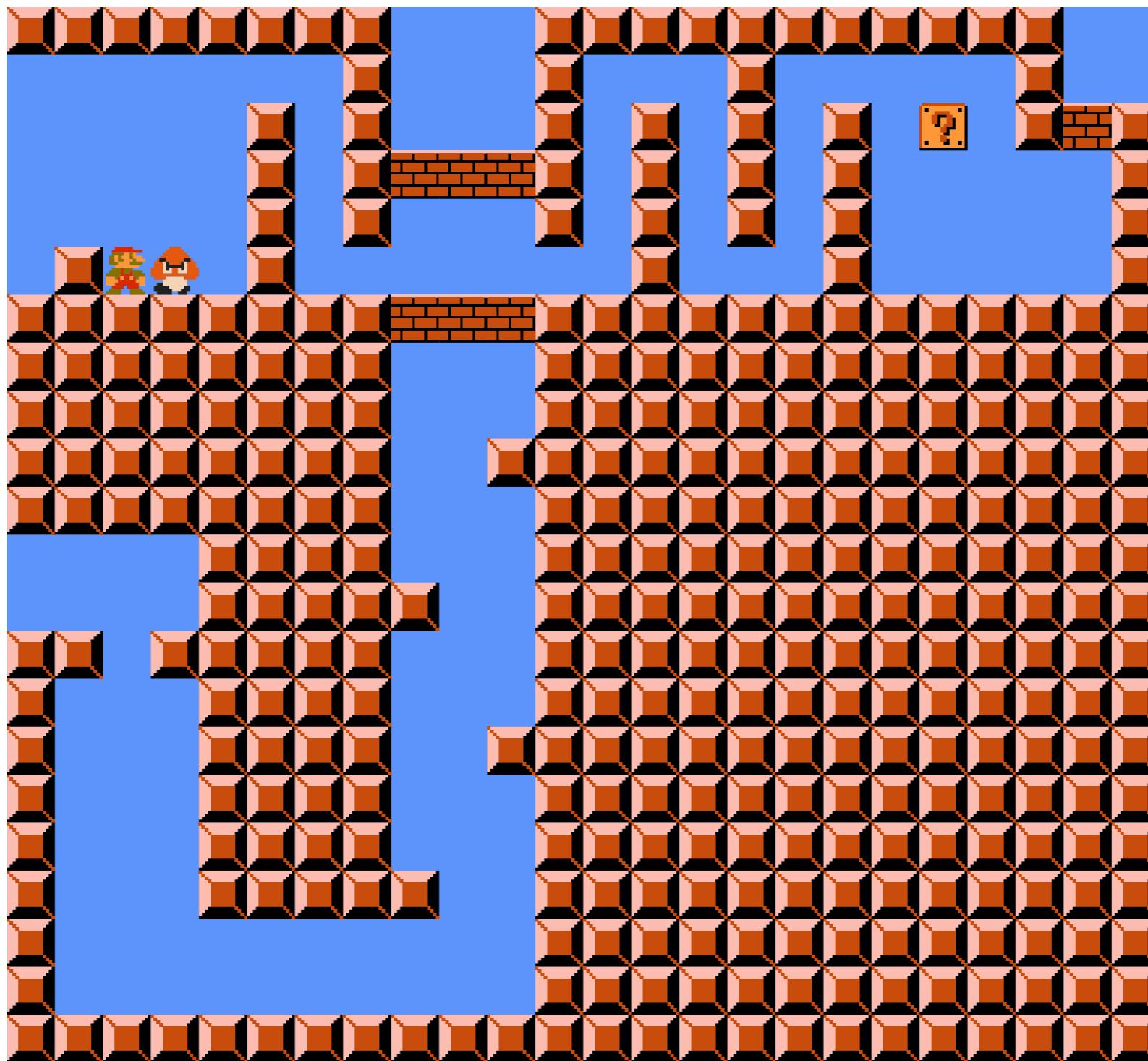
# Gadget pour les croisements



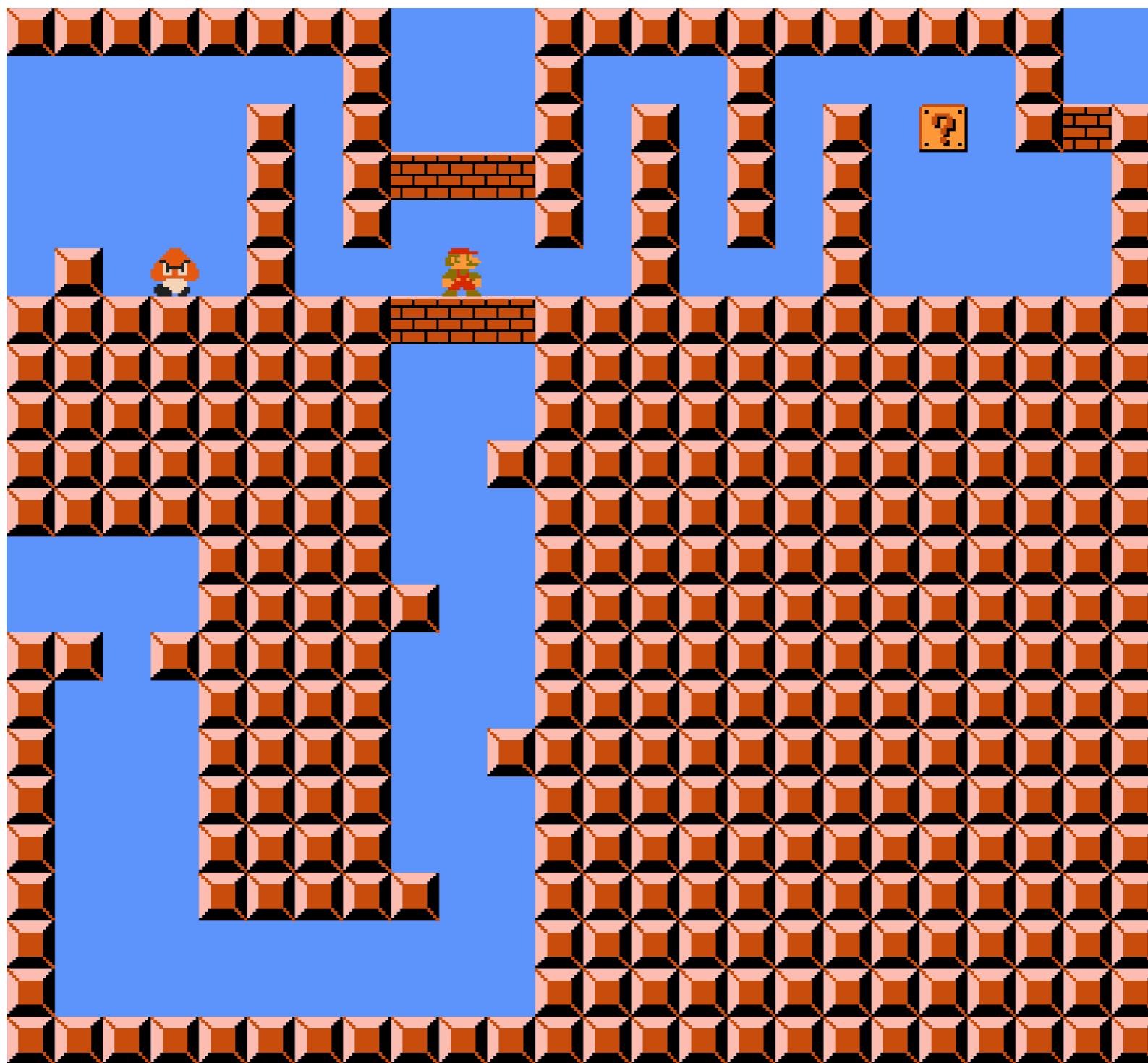
# Gadget pour les croisements



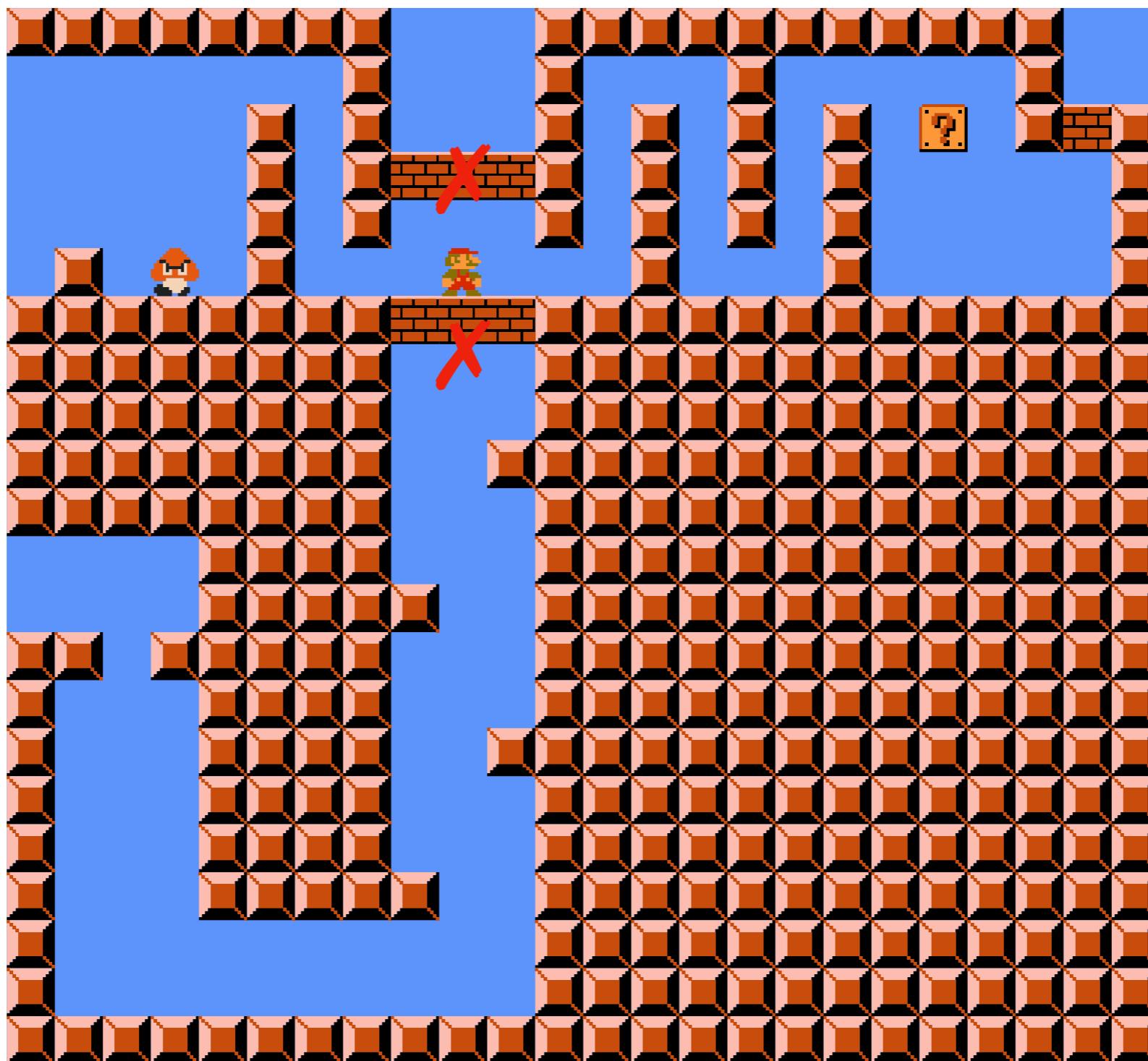
# Gadget pour les croisements



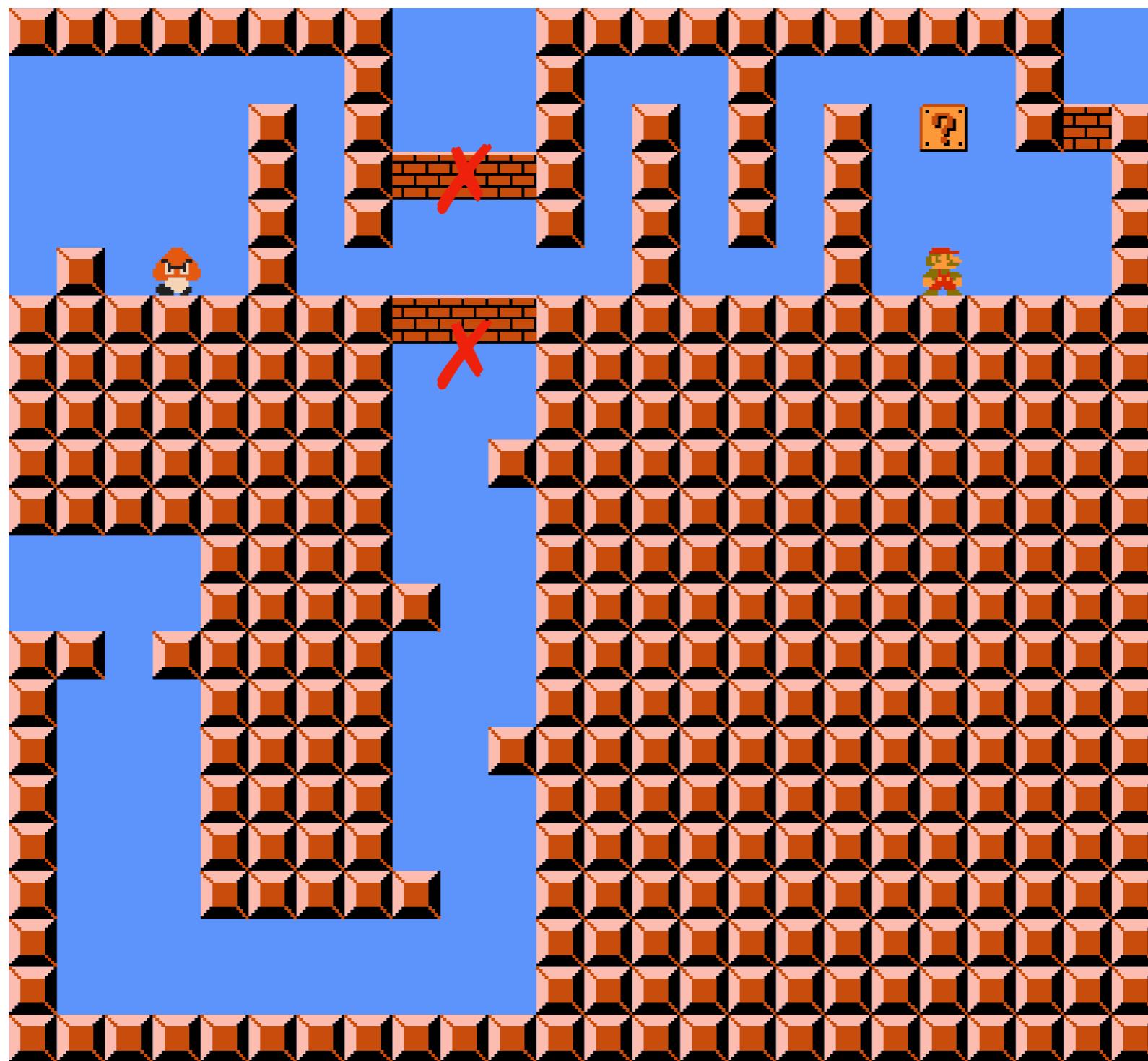
# Gadget pour les croisements



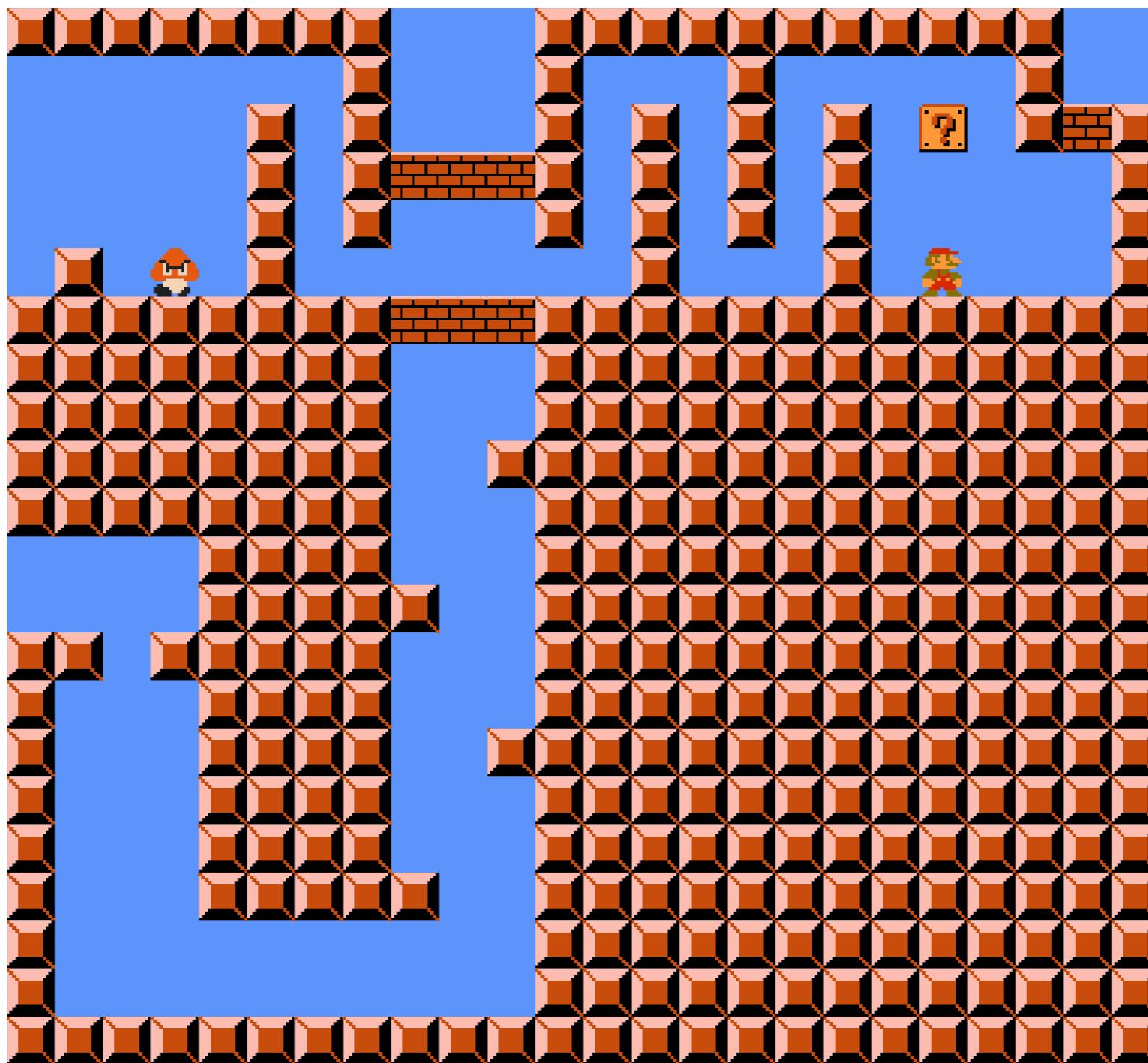
# Gadget pour les croisements



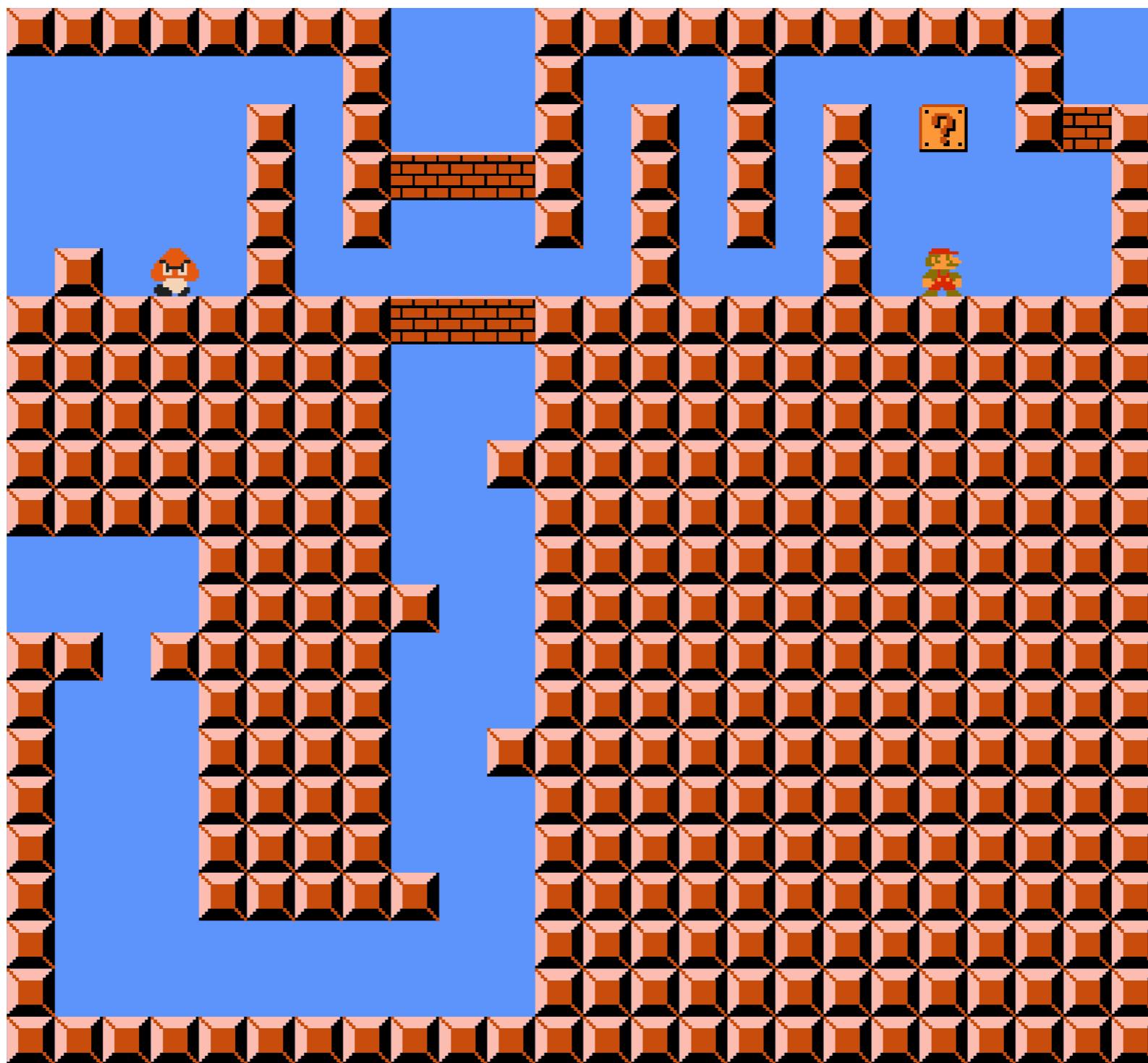
# Gadget pour les croisements



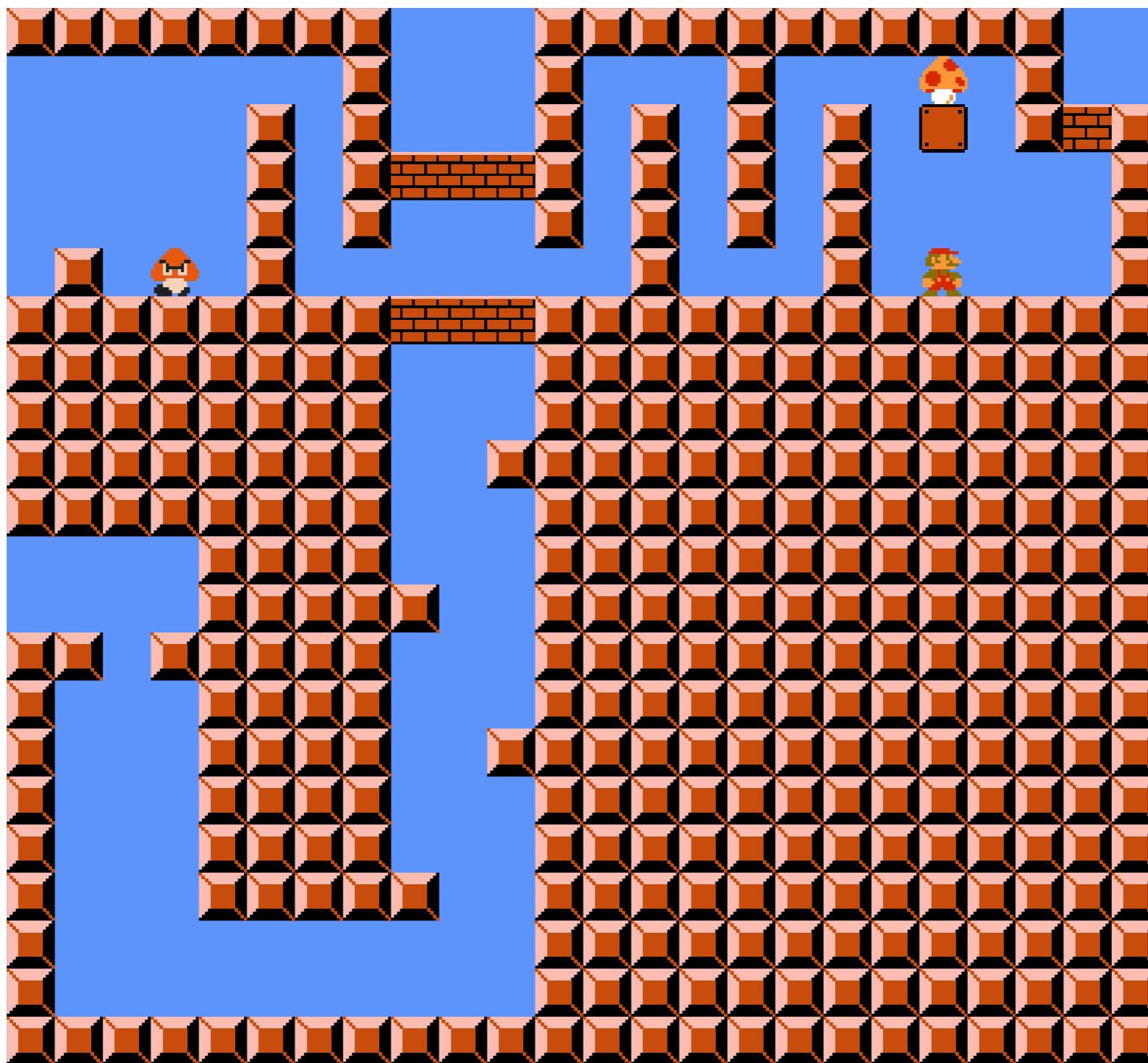
# Gadget pour les croisements



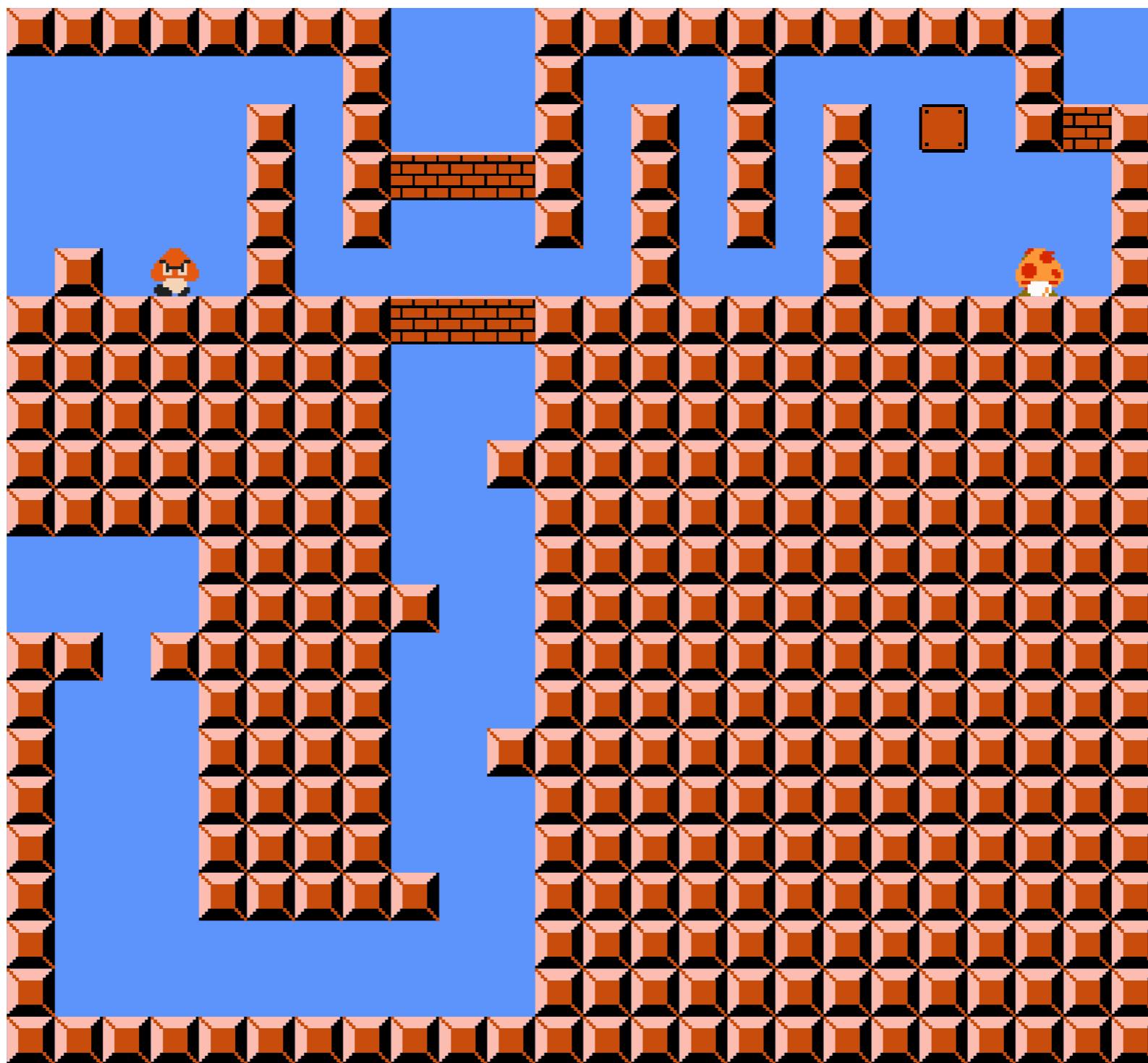
# Gadget pour les croisements



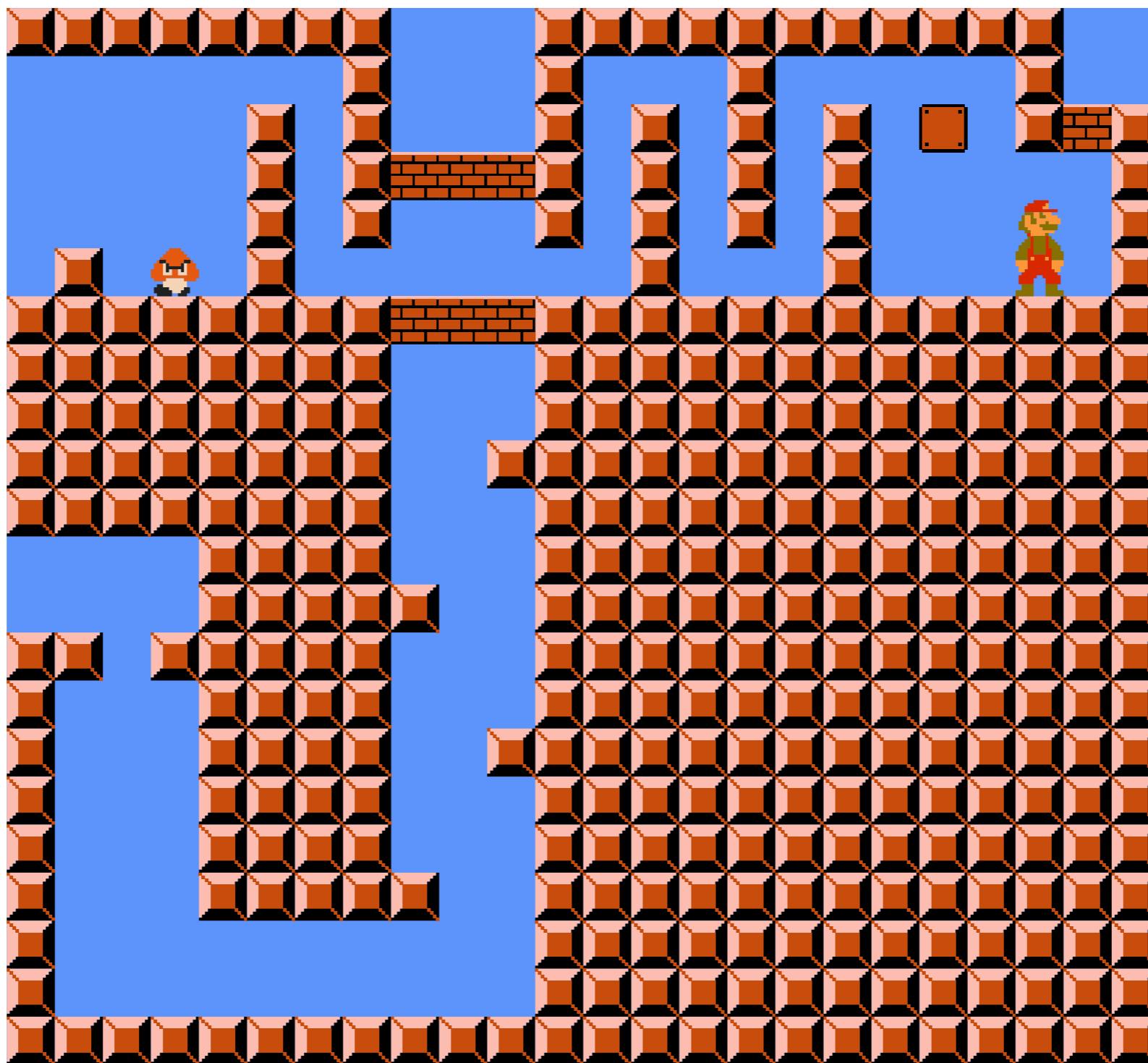
# Gadget pour les croisements



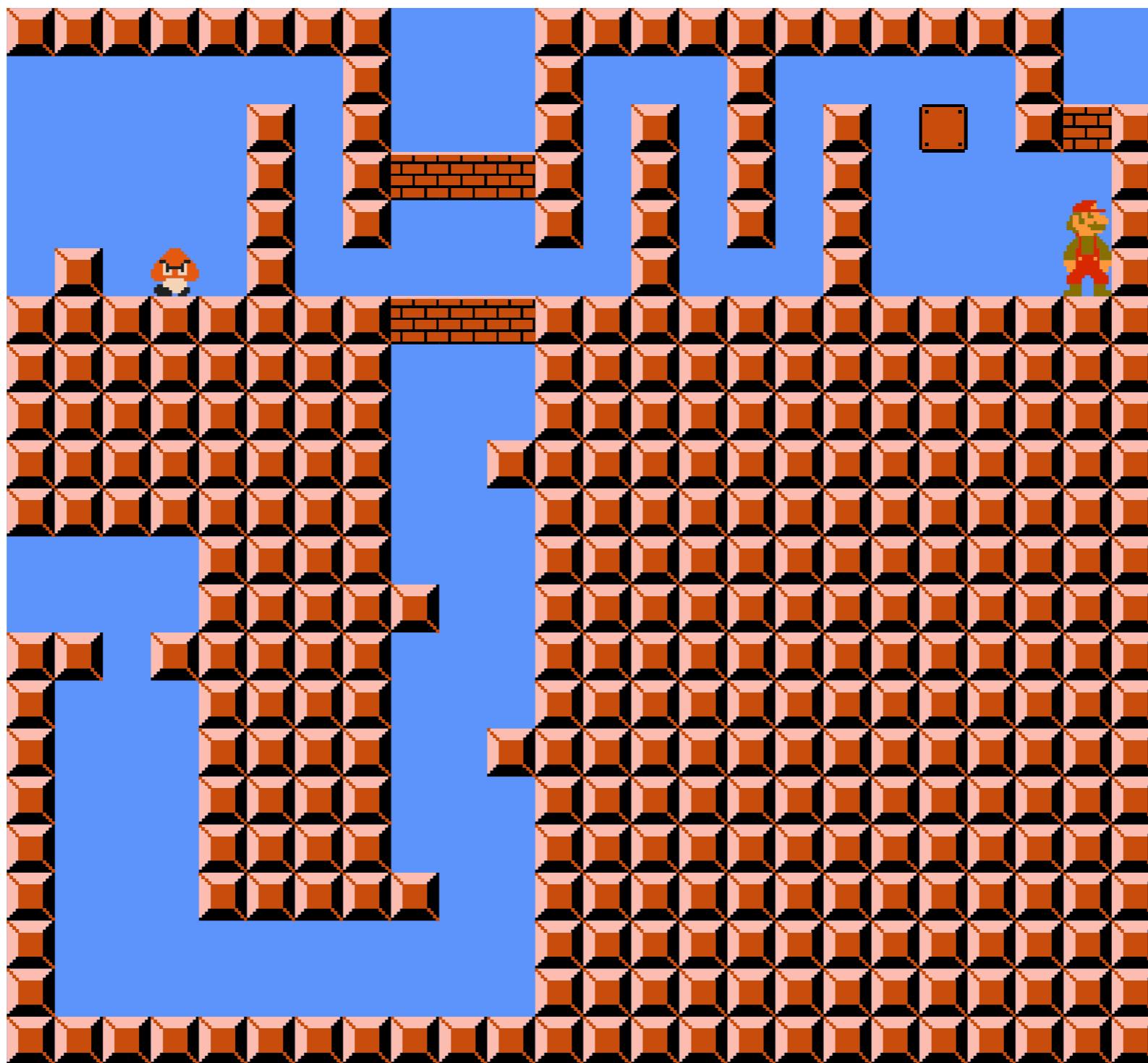
# Gadget pour les croisements



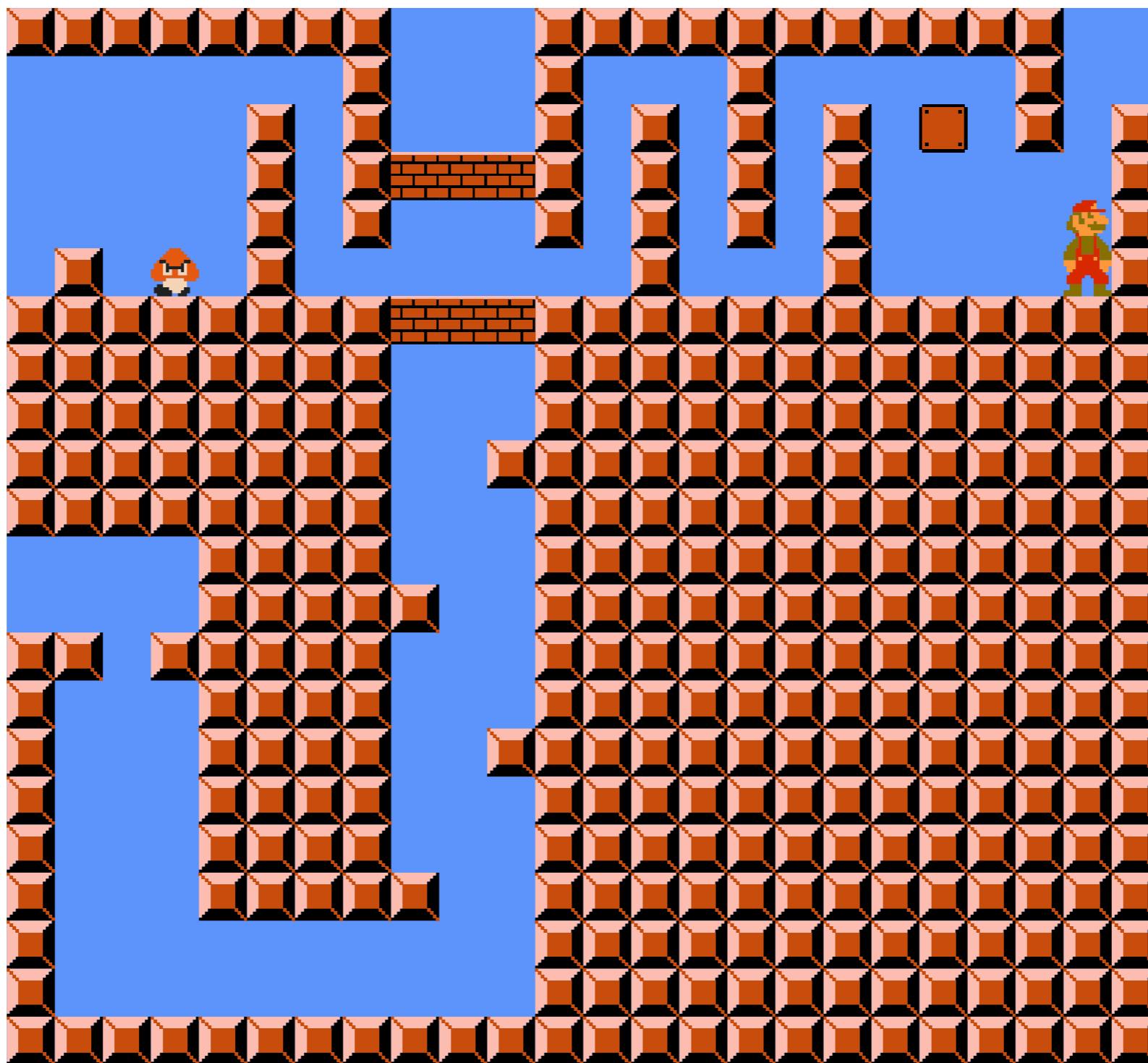
# Gadget pour les croisements



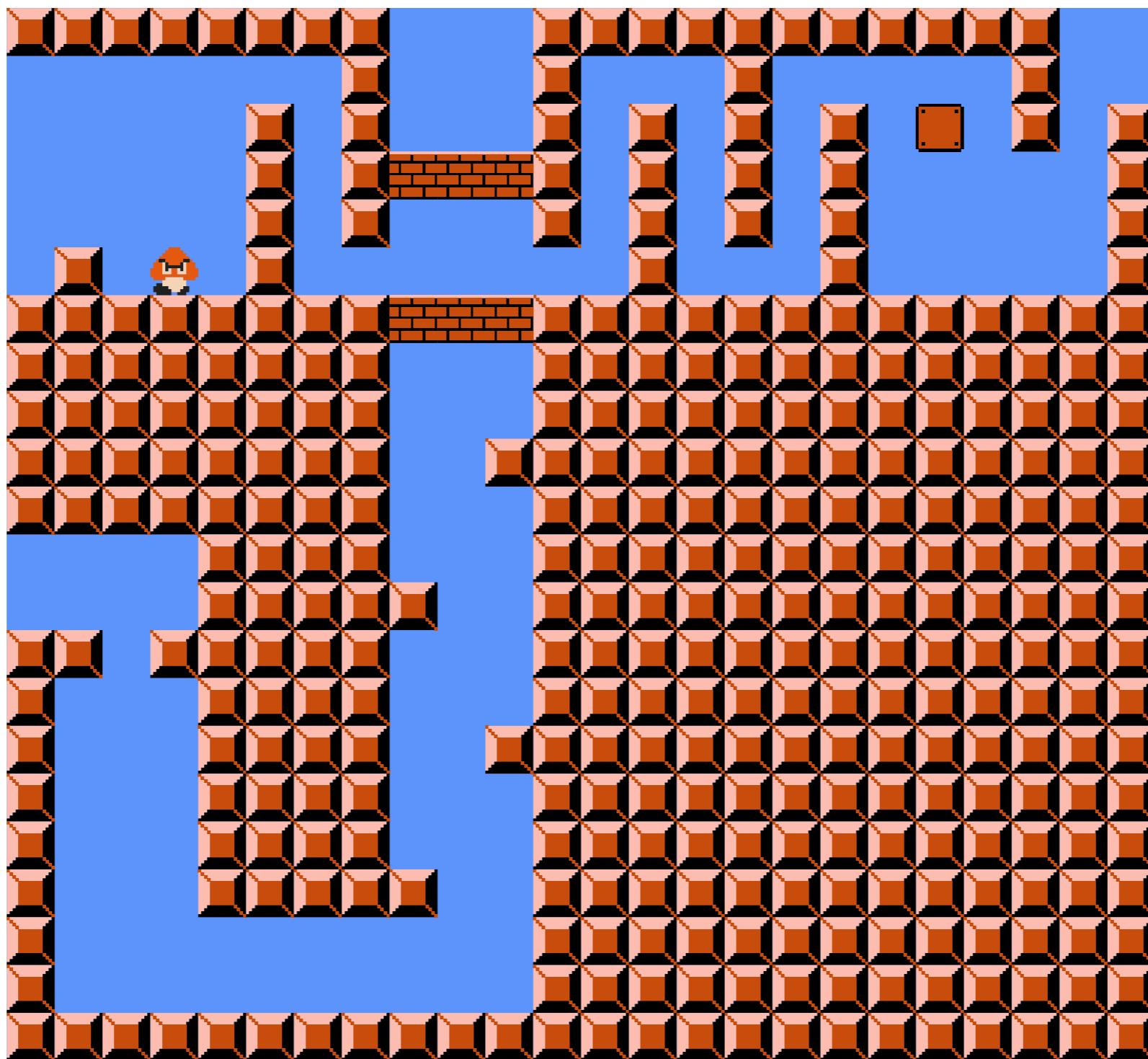
# Gadget pour les croisements



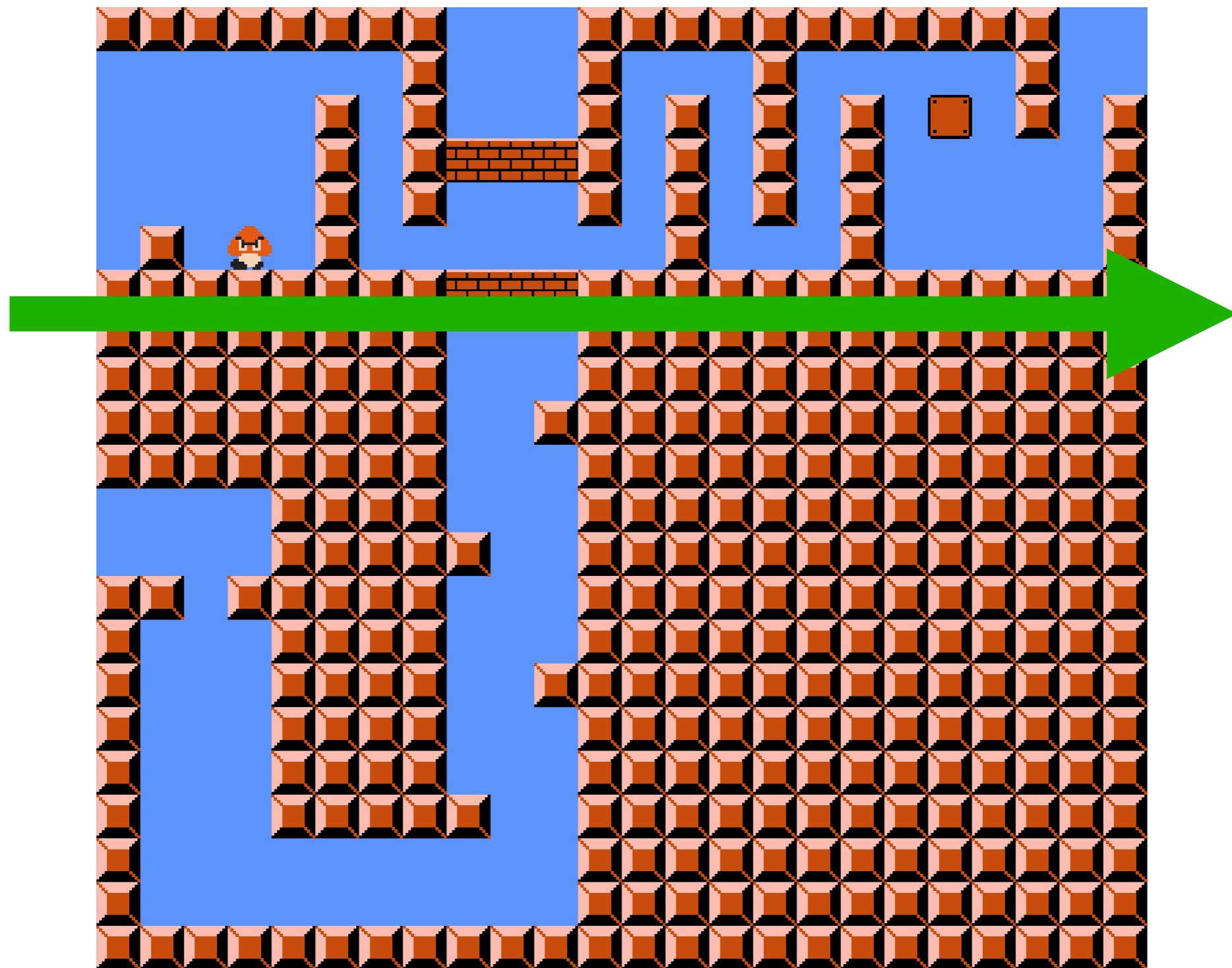
# Gadget pour les croisements



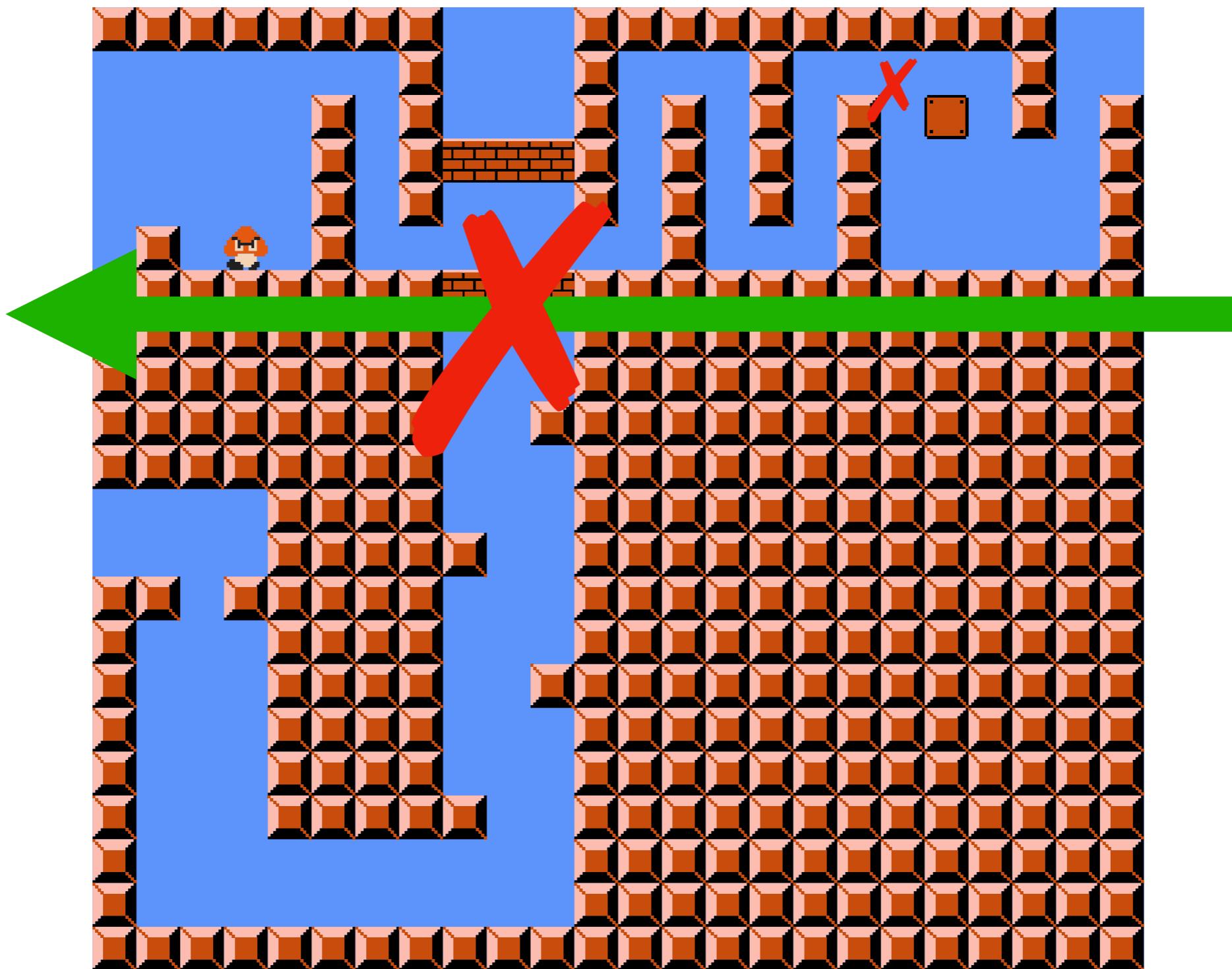
# Gadget pour les croisements



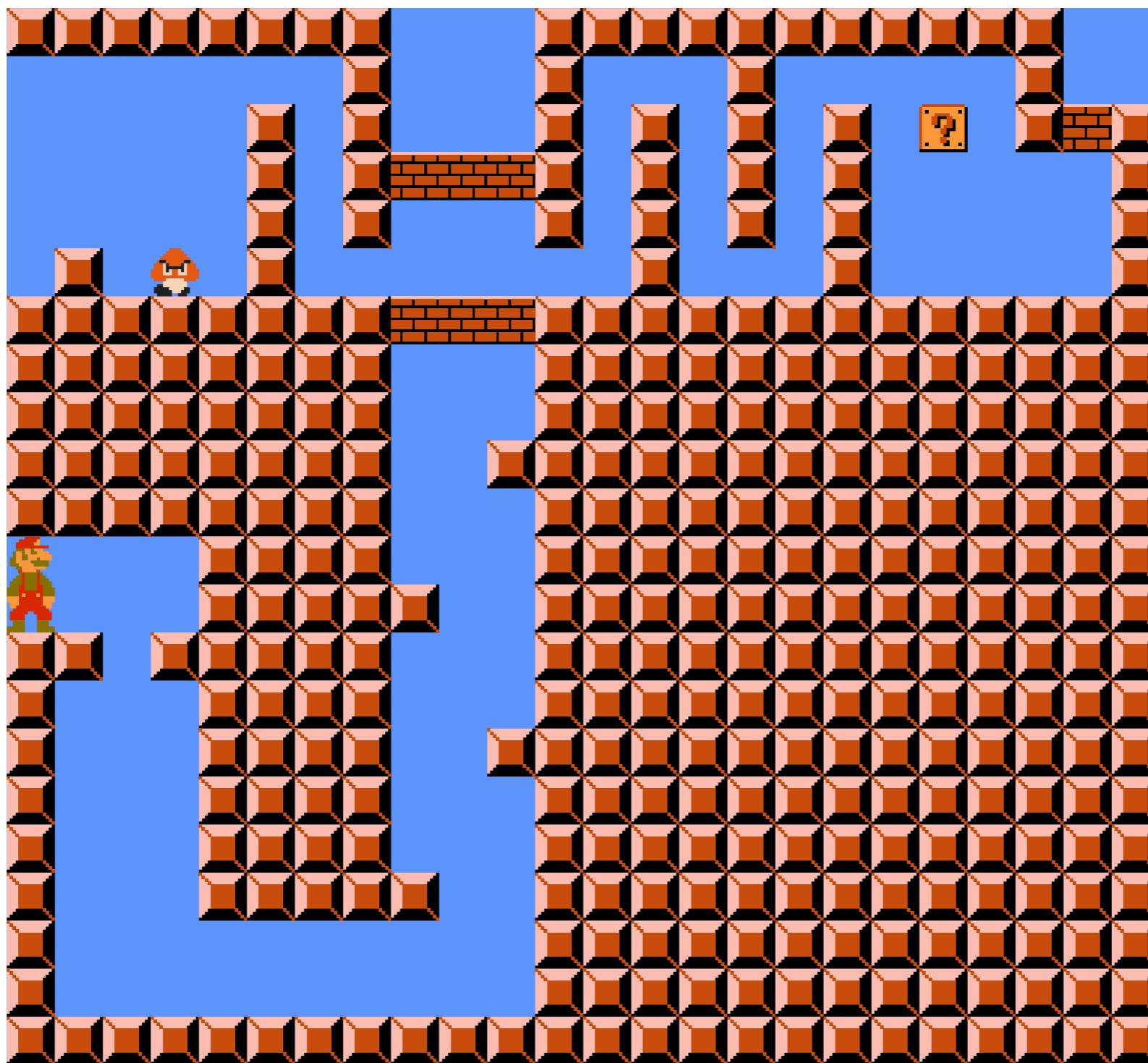
# Gadget pour les croisements



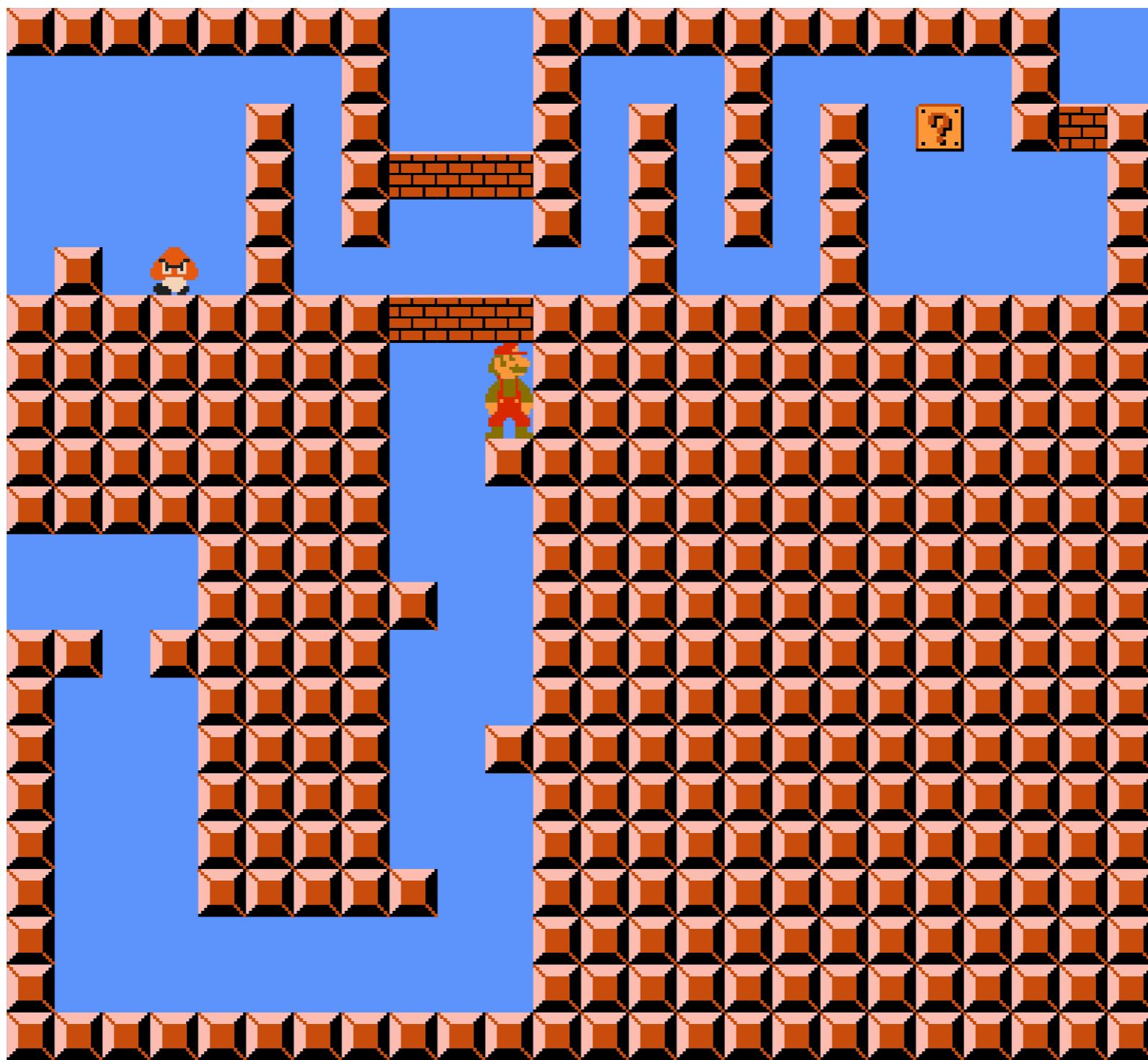
# Gadget pour les croisements



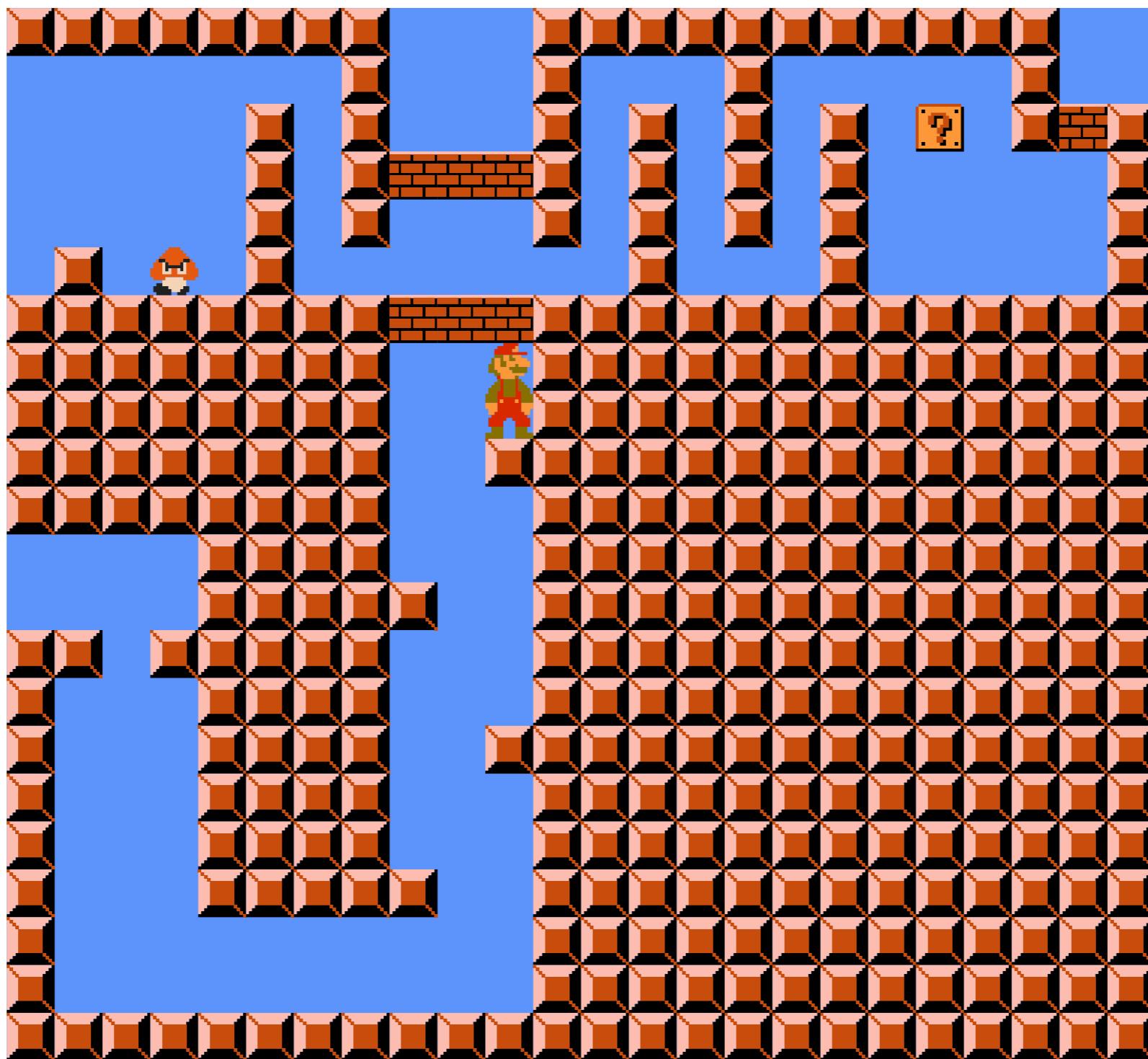
# Gadget pour les croisements



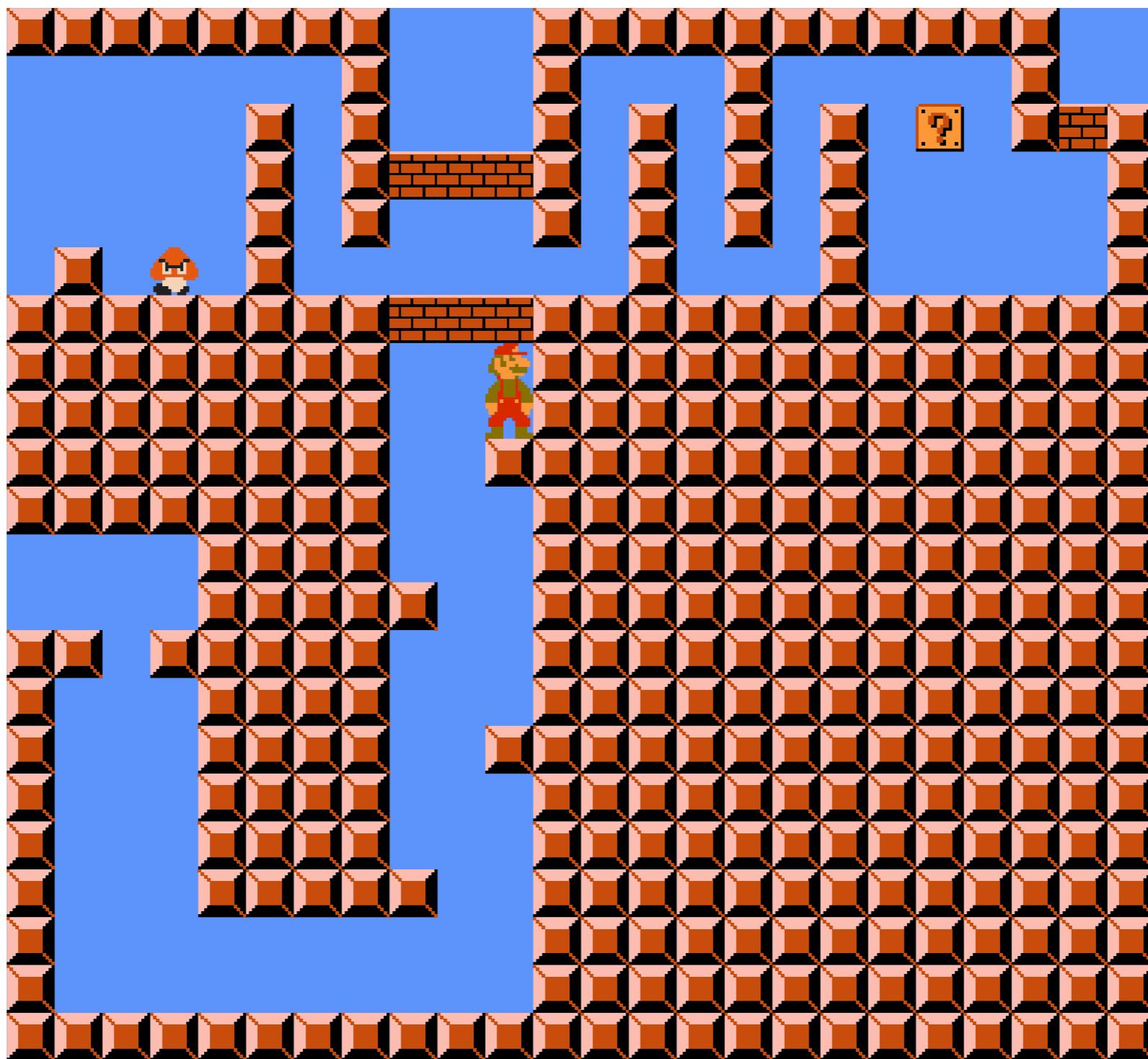
# Gadget pour les croisements



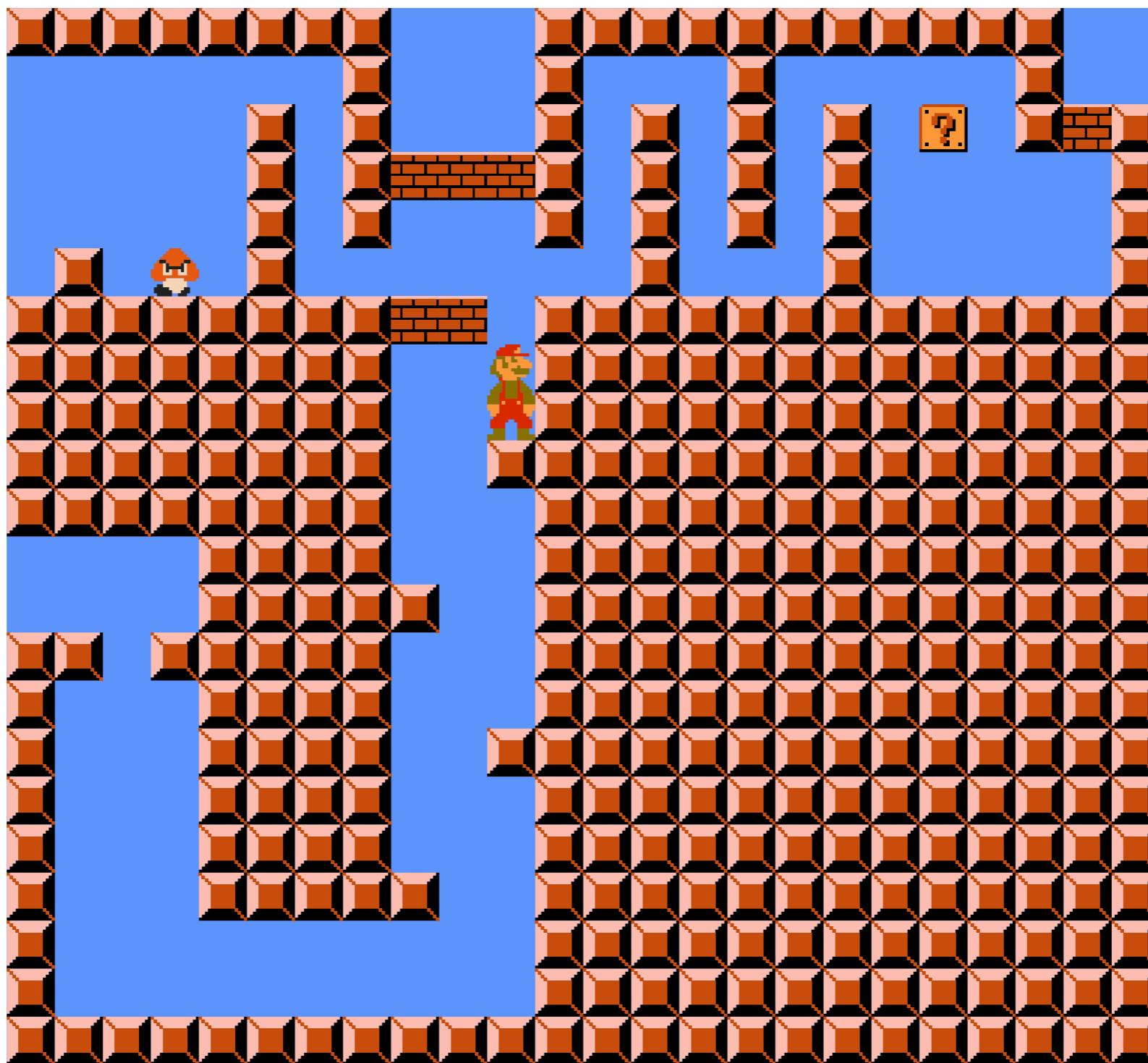
# Gadget pour les croisements



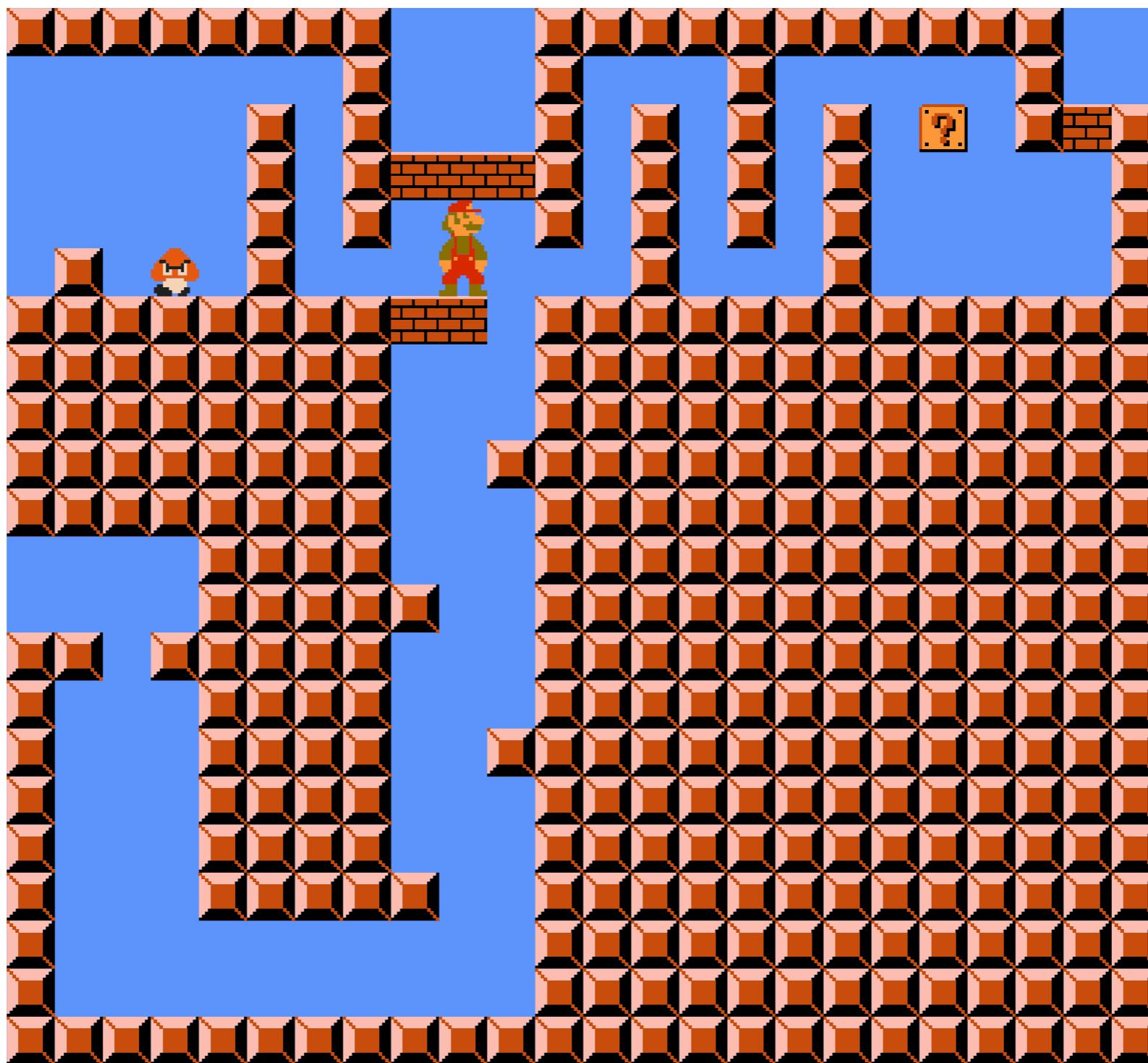
# Gadget pour les croisements



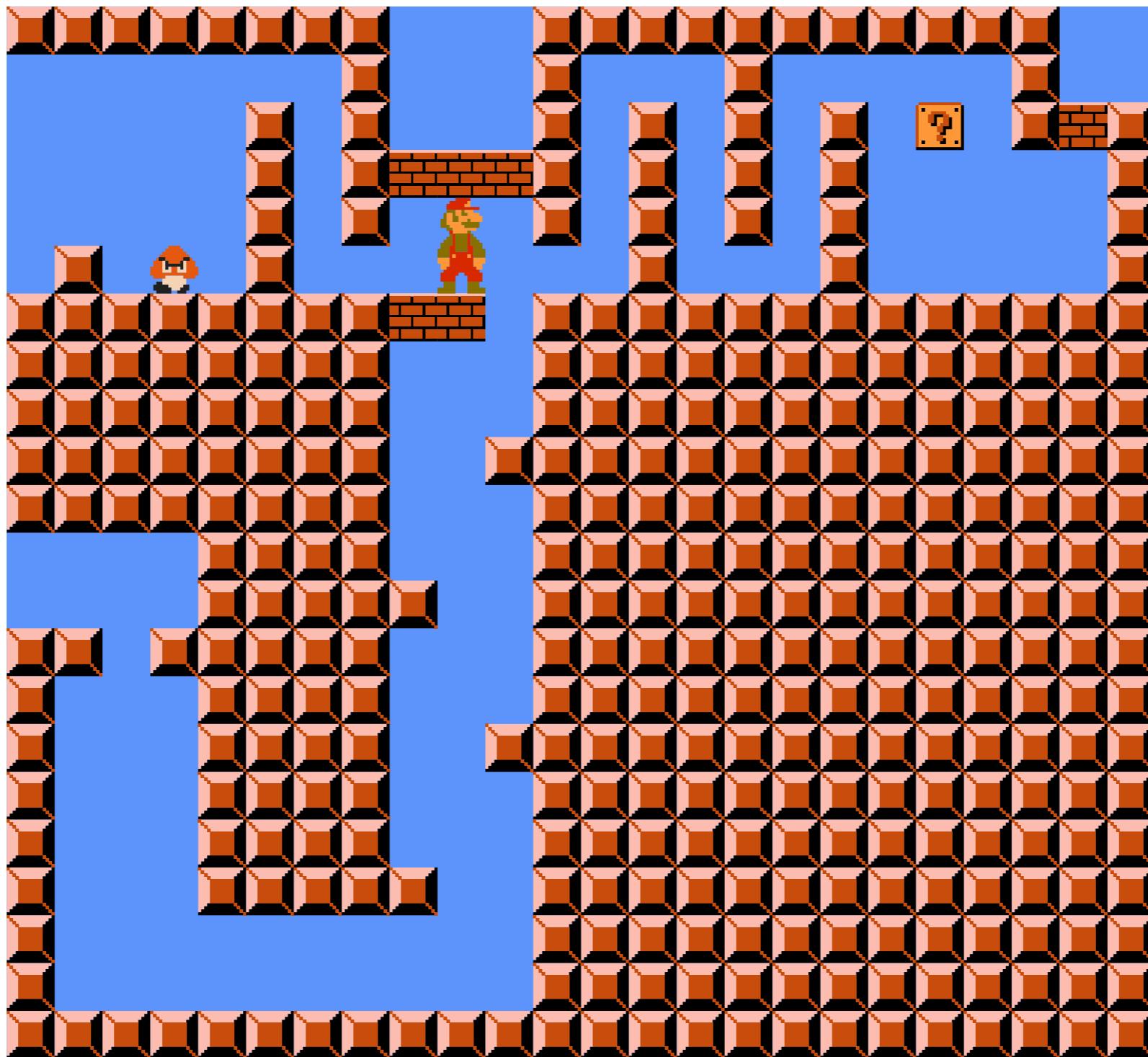
# Gadget pour les croisements



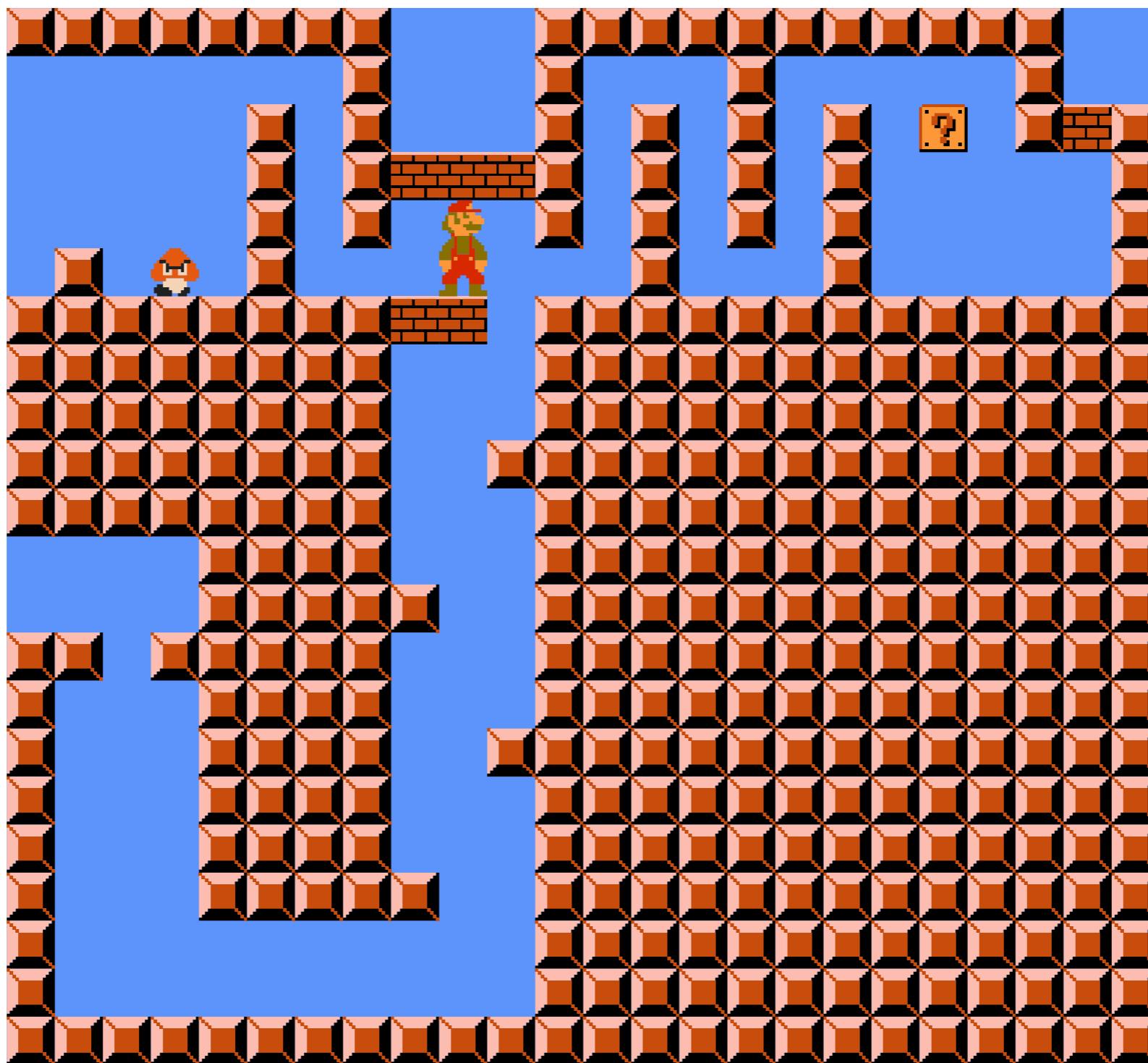
# Gadget pour les croisements



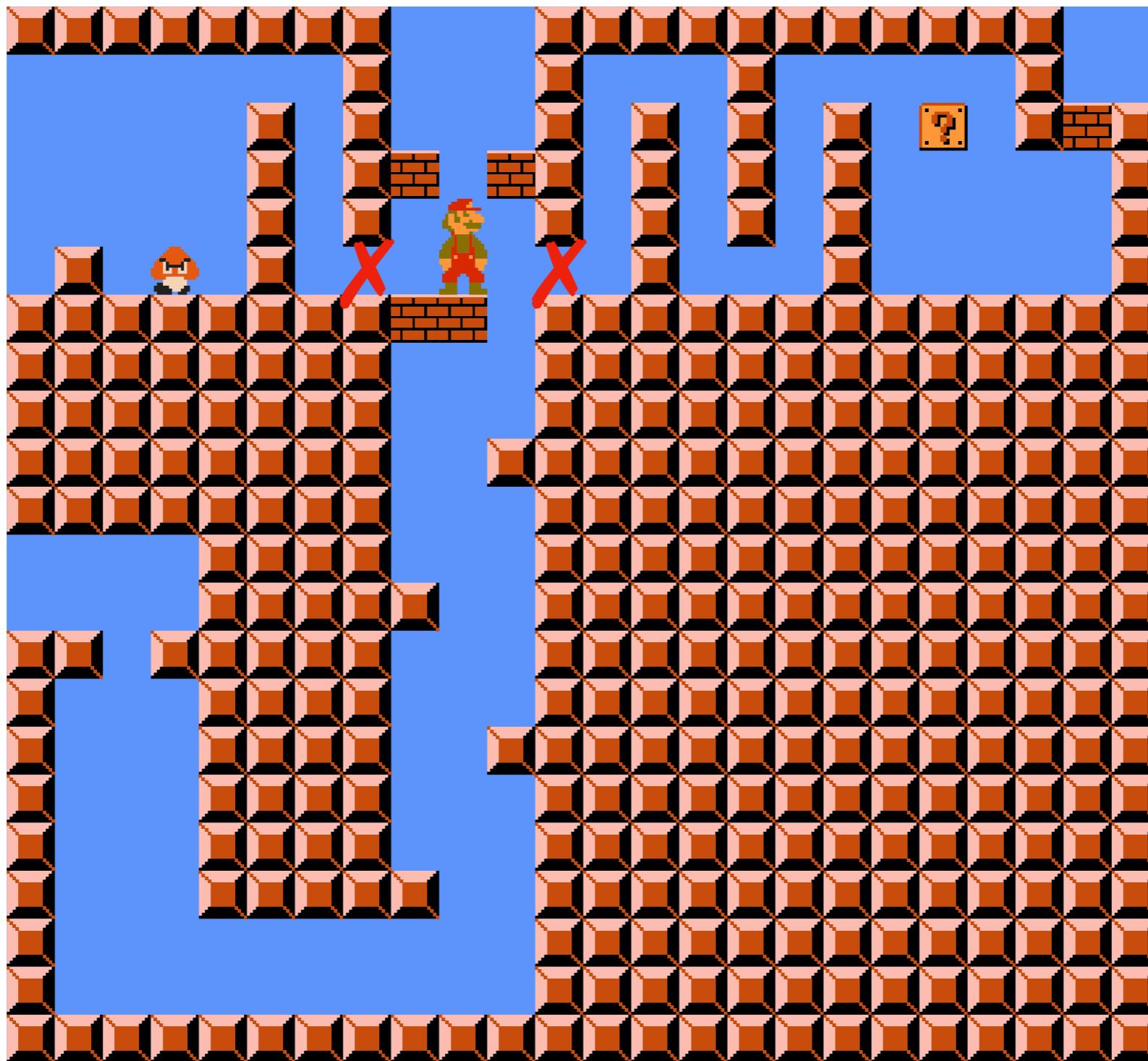
# Gadget pour les croisements



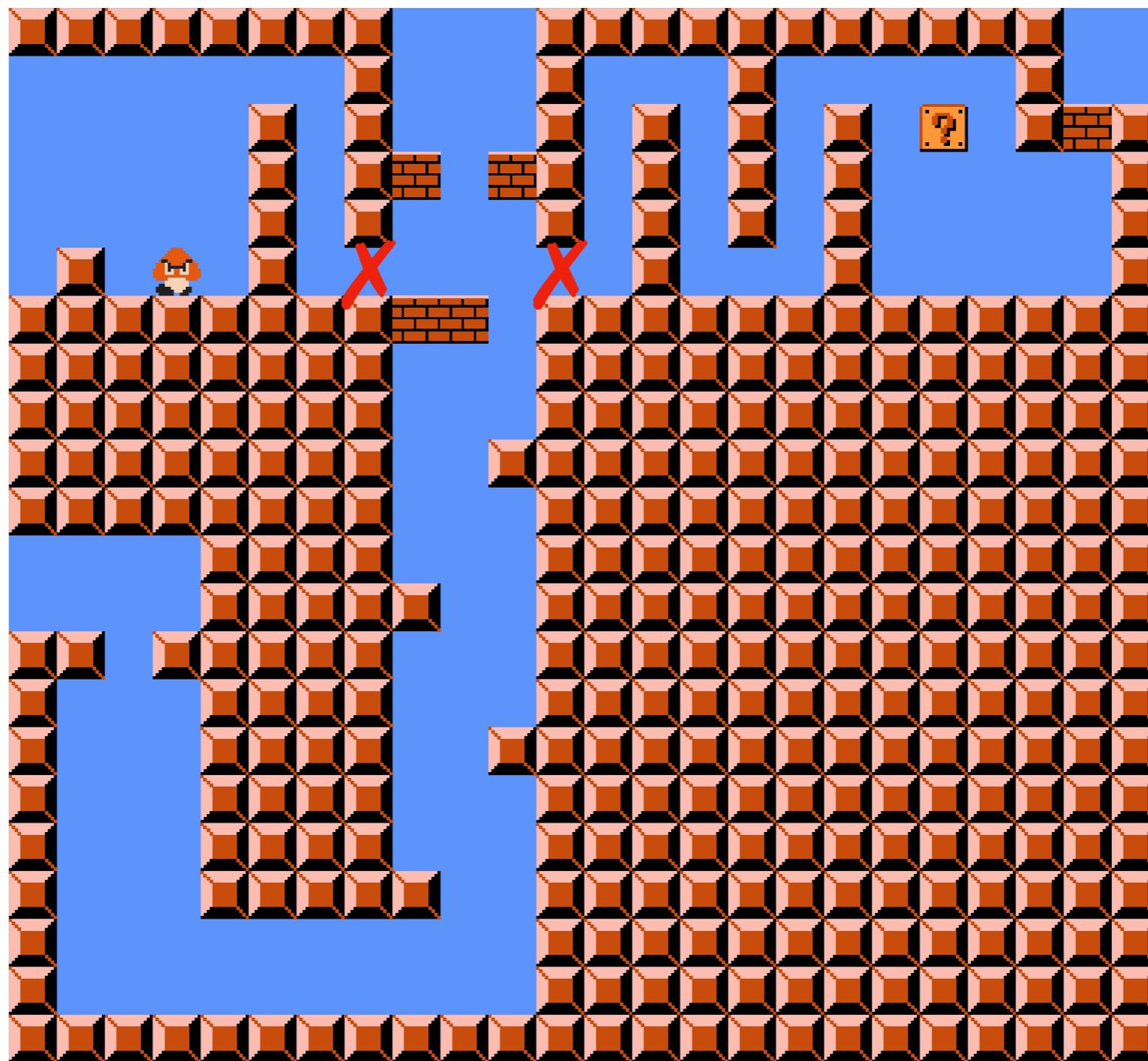
# Gadget pour les croisements



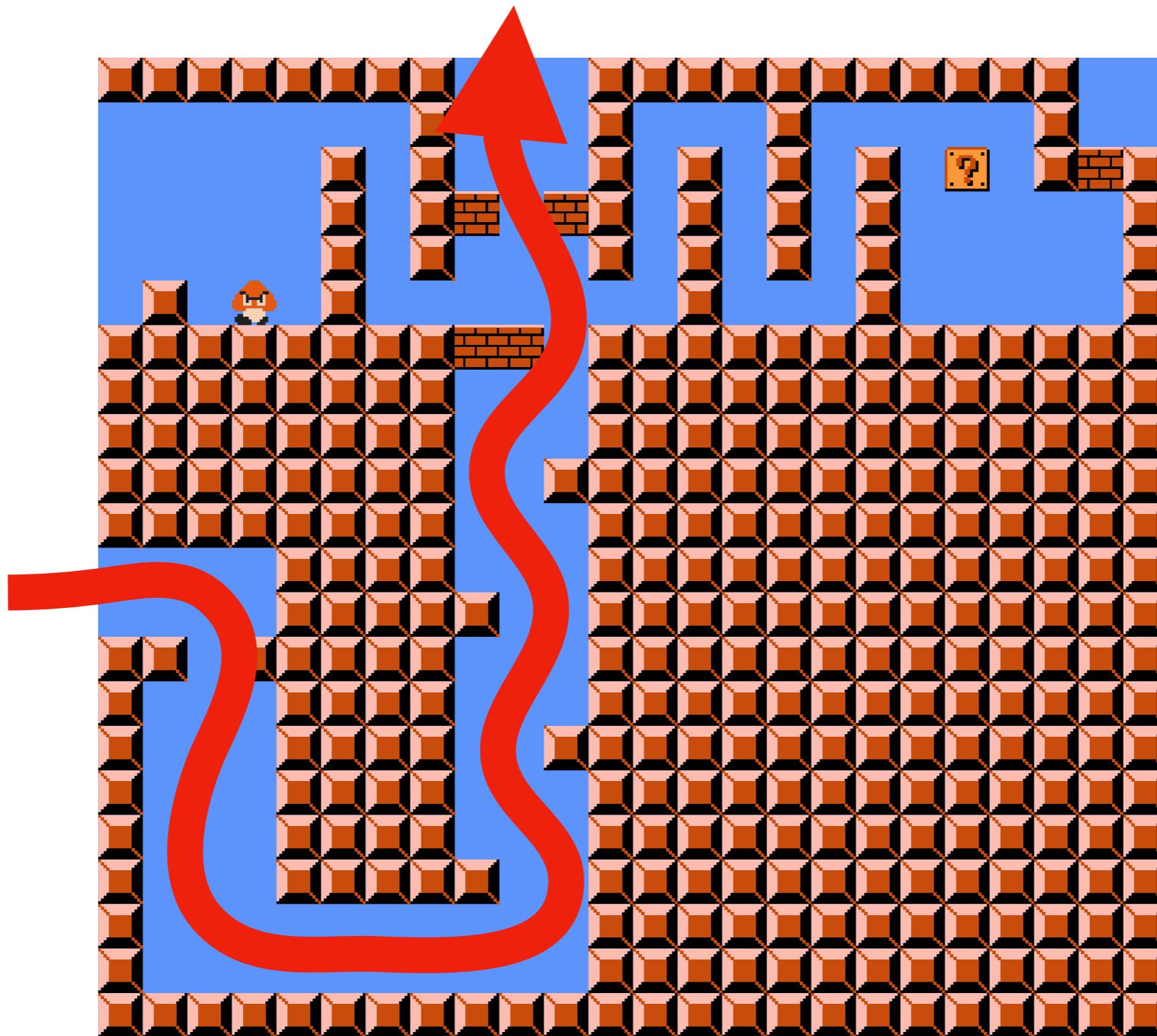
# Gadget pour les croisements



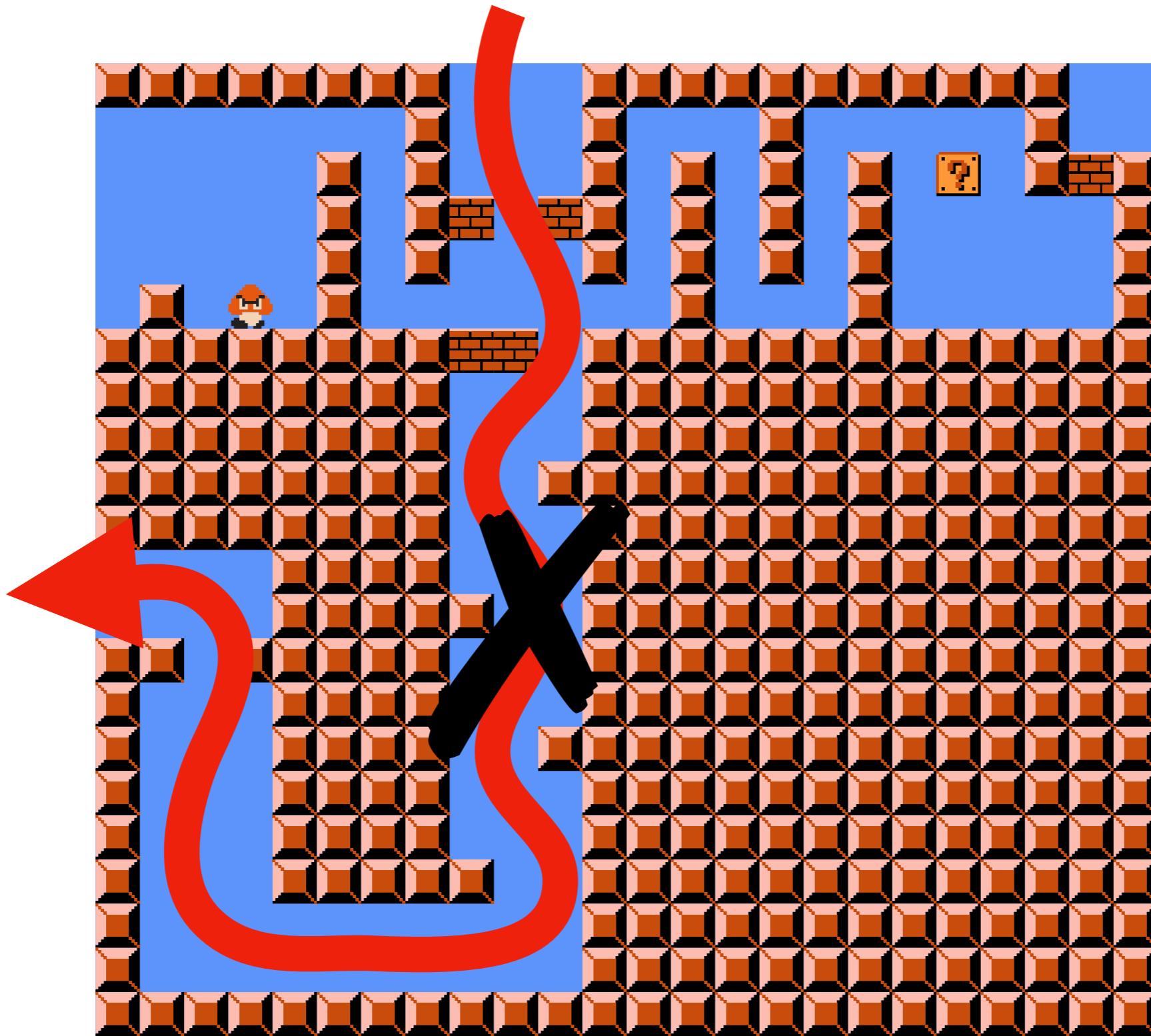
# Gadget pour les croisements



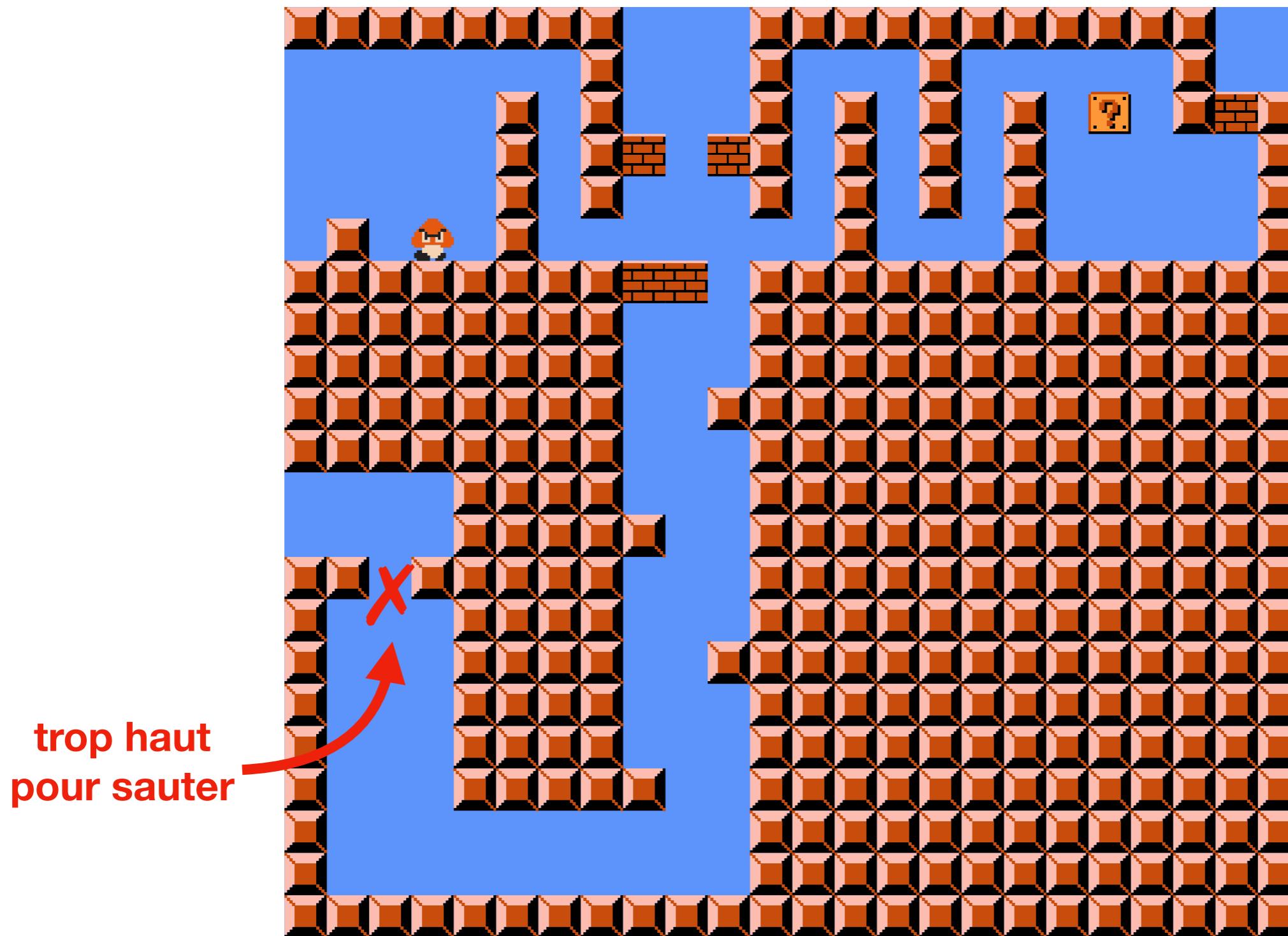
# Gadget pour les croisements



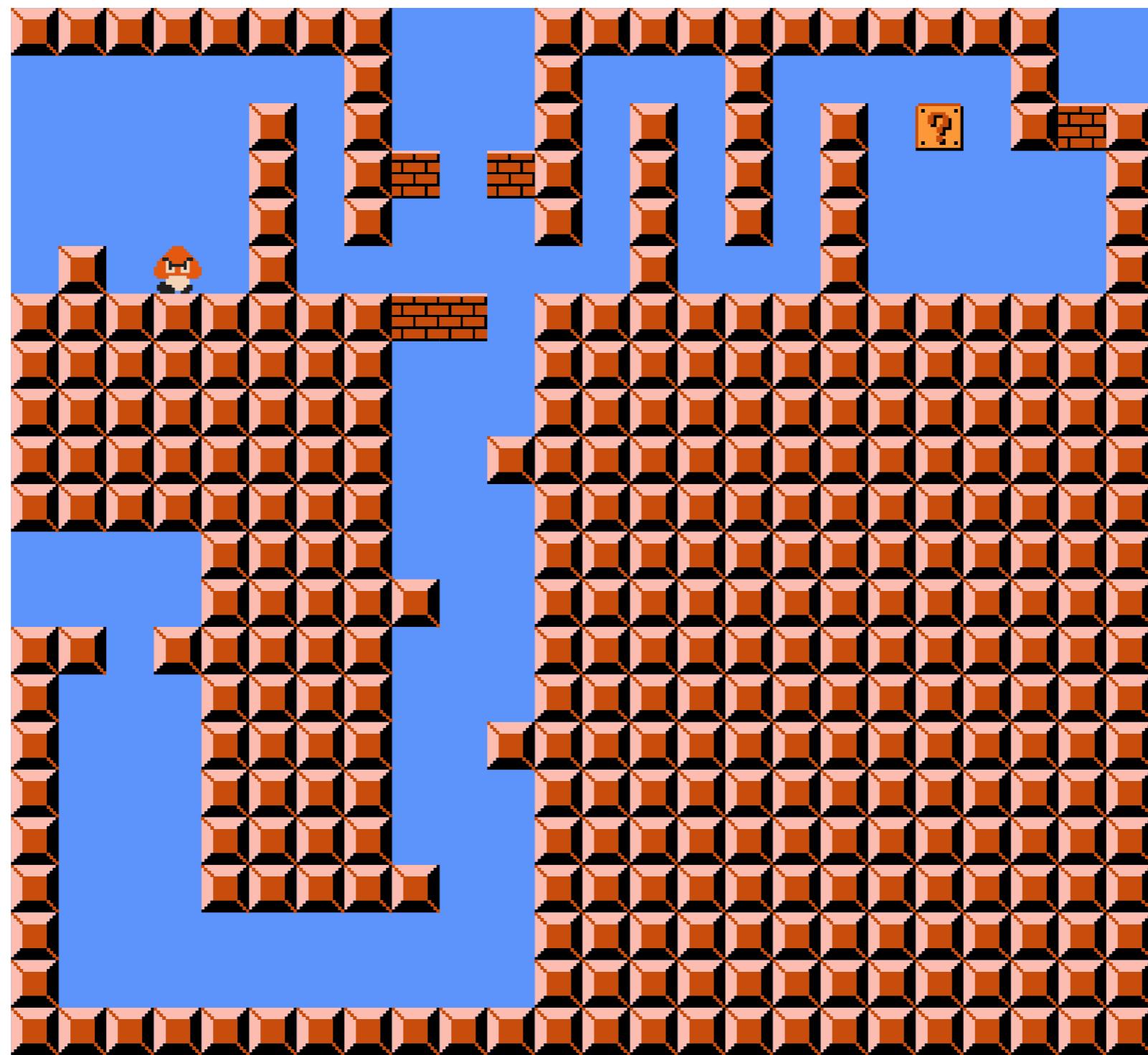
# Gadget pour les croisements



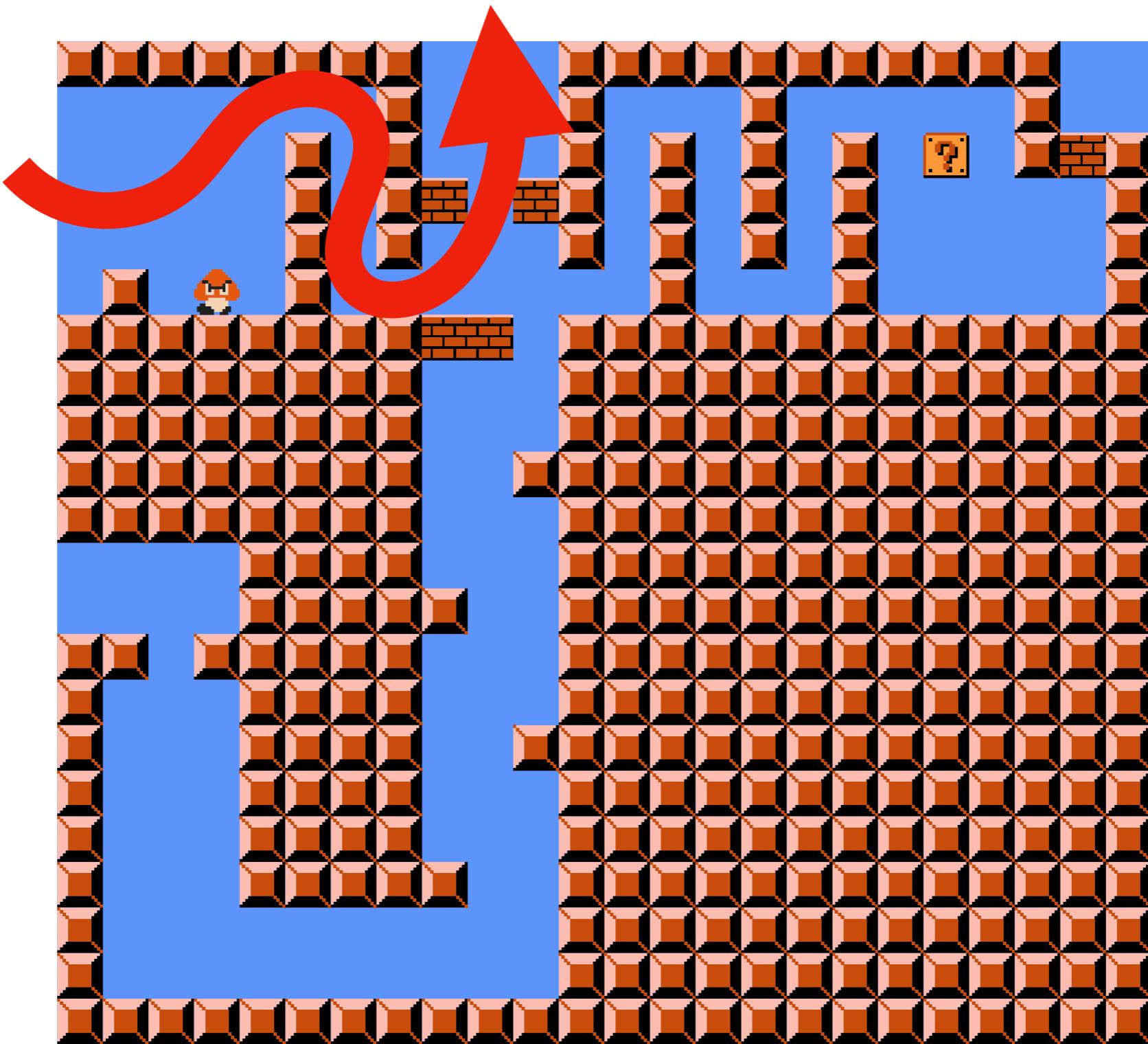
# Gadget pour les croisements



# ! Remarque !

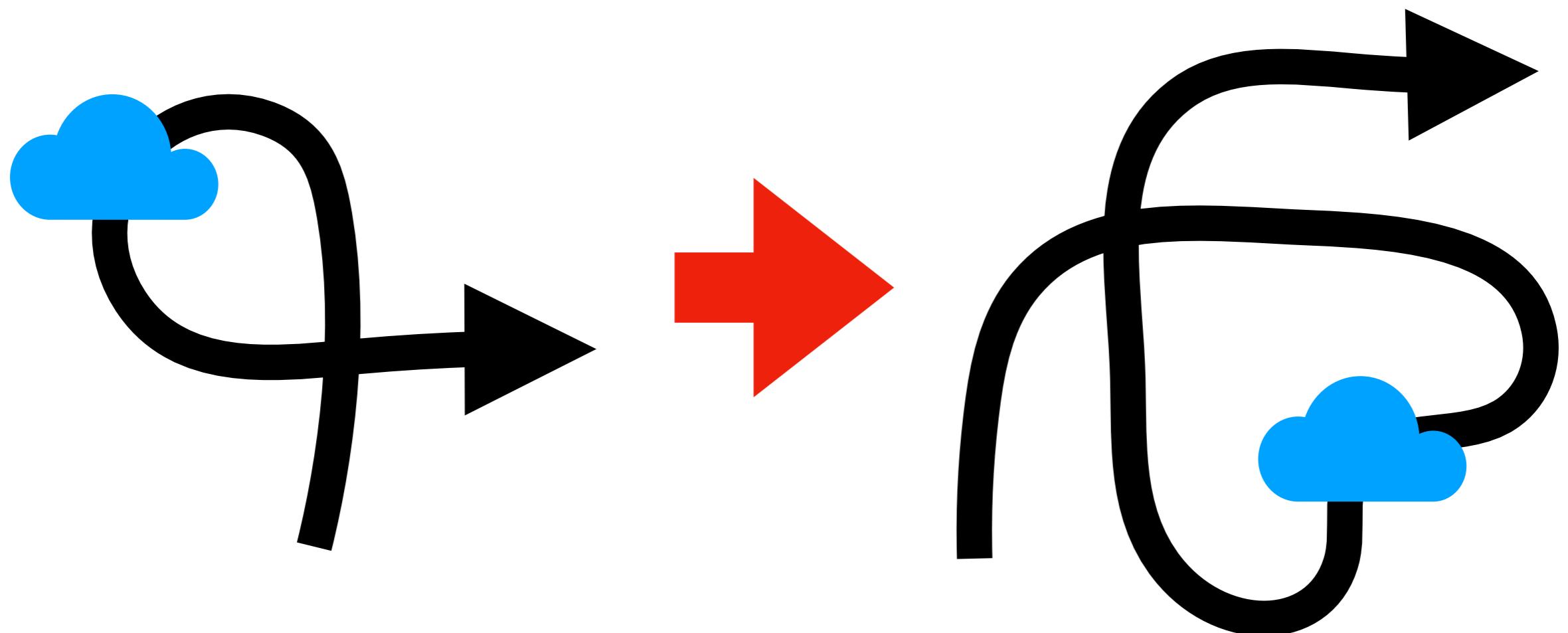


# ! Remarque !

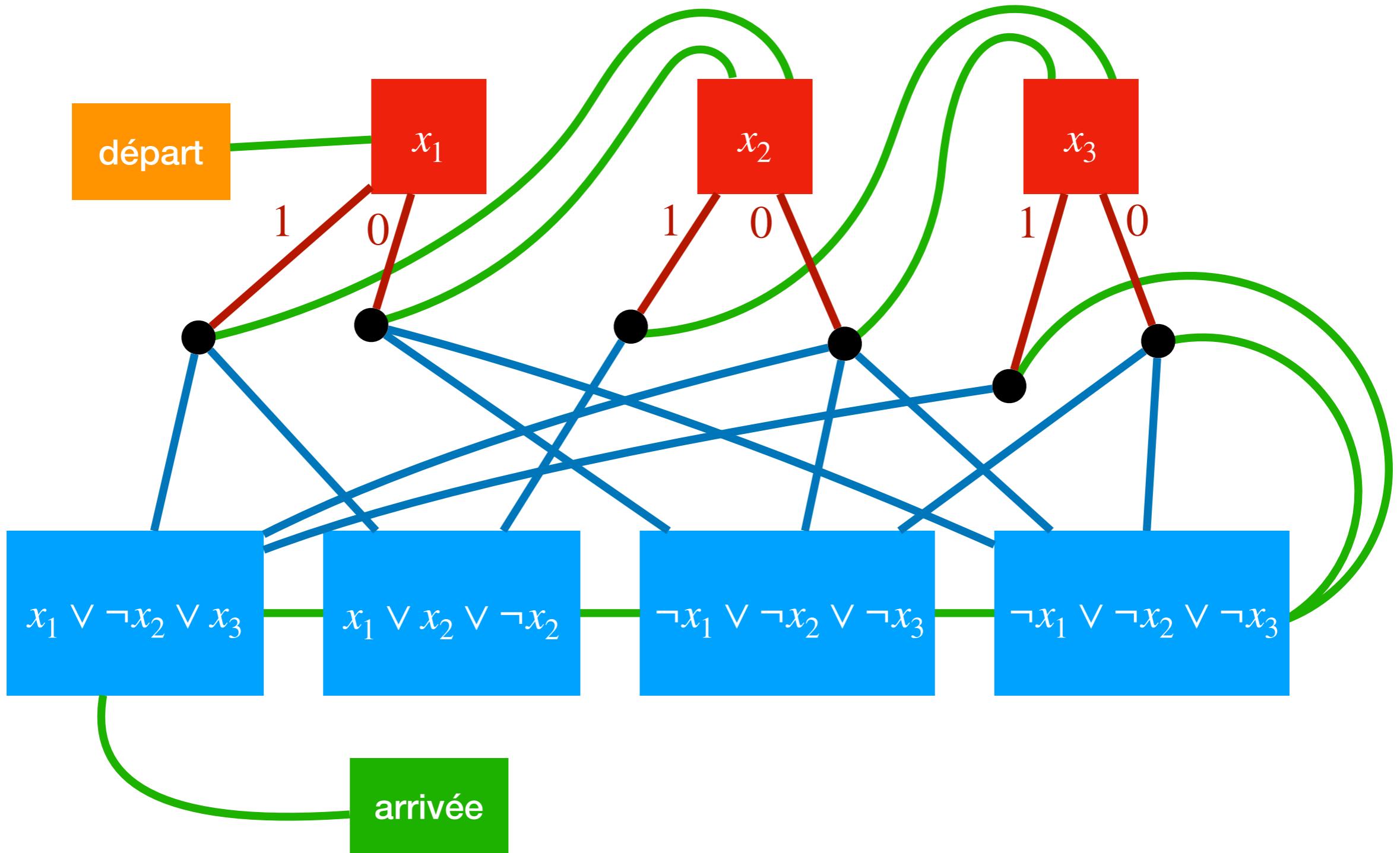


# ! Remarque !

Ce n'est pas un problème, on peut toujours forcer le plombier à passer d'abord en horizontal et seulement après en vertical :



# Reste des arcs = couloirs



# Conclusion

- Le plombier arrive à la fin ssi il arrive à traverser tous les gadgets pour les clauses
- Ce qui demande d'avoir au moins une étoile pour chaque clause
- Ce qui correspond à un littéral vrai pour chaque clause
- Ce qui donne une affectation qui satisfait la formule
- Donc Super Plombiers Italiens est **NP-difficile**



# Remarque

- On a montré que Super Plombiers Italiens est **NP-difficile**
- Mais en réalité\* il est même plus difficile que **NP** !
- Il est complet pour la classe **PSPACE**,  
la classe des problèmes solvables en espace  
(= quantité de ruban ou mémoire) polynomial

\* Sous certaines assumptions

# Problèmes d'optimisation

# Problèmes d'optimisation

- Pour chaque **instance  $x$**  (un mot dans  $\Sigma^*$ ) on a un ensemble  $F(x) \subseteq \Sigma^*$  de **solutions réalisables** et une **fonction de coût** (ou **fonction objectif**)  $c: F(x) \rightarrow \mathbb{R}^+$
- Le **coût optimal** est  $\text{OPT}(x) = \min\{c(s) : s \in F(x)\}$  pour les problèmes de **minimisation** , ou bien  $\text{OPT}(x) = \max\{c(s) : s \in F(x)\}$  pour les problèmes de **maximisation** 
- On veut trouver une **solution optimale** , c'est-à-dire un  $s \in F(x)$  tel que  $c(s) = \text{OPT}(x)$

# Problème du voyageur de commerce (TSP)

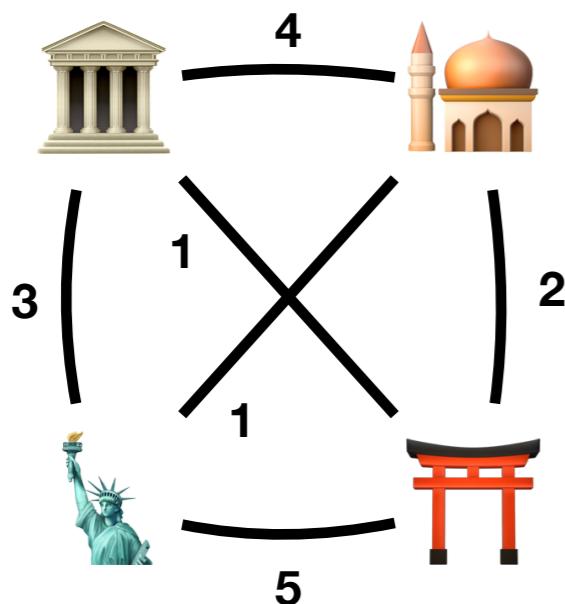


- Entrée : un **graphe non orienté complet pondéré**  $G = (V, E = V^2, w)$  où  $w: E \rightarrow \mathbb{R}^+$  est la distance entre chaque paire de points
- Sortie : un **cycle hamiltonien** (un cycle qui traverse chaque sommet une et une seule fois) **de poids minimal** A small icon of a blue map with a winding blue line representing a path or route.

# Problème du voyageur de commerce (TSP)



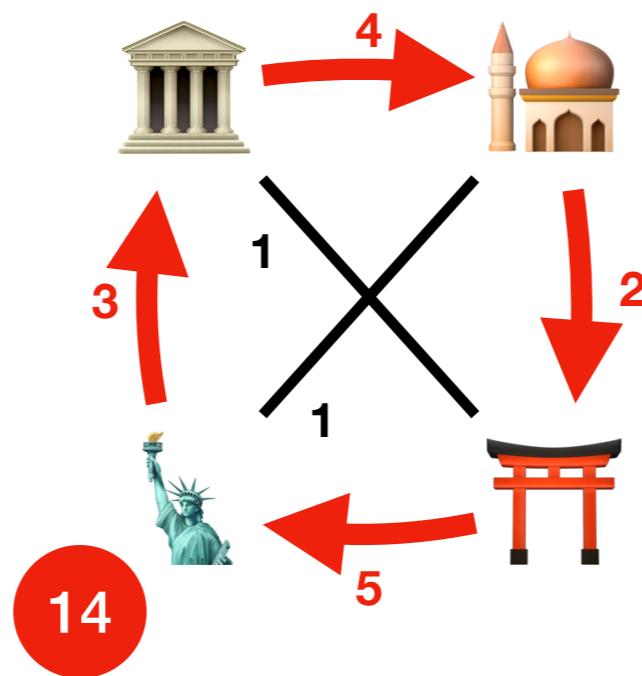
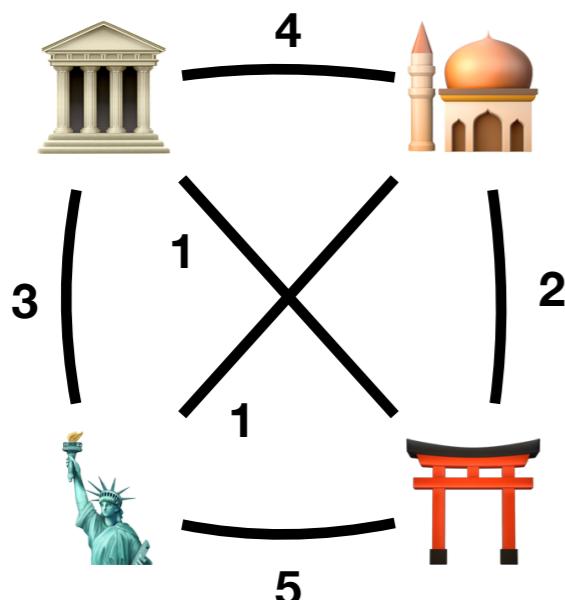
- Entrée : un **graphe non orienté complet pondéré**  
 $G = (V, E = V^2, w)$  où  $w: E \rightarrow \mathbb{R}^+$  est la distance entre chaque paire de points
- Sortie : un **cycle hamiltonien** (un cycle qui traverse chaque sommet une et une seule fois) **de poids minimal** 



# Problème du voyageur de commerce (TSP)



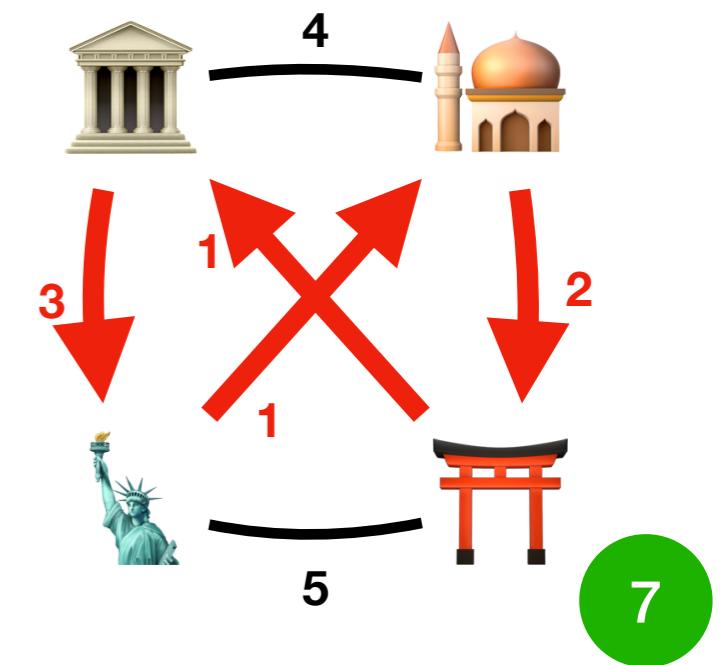
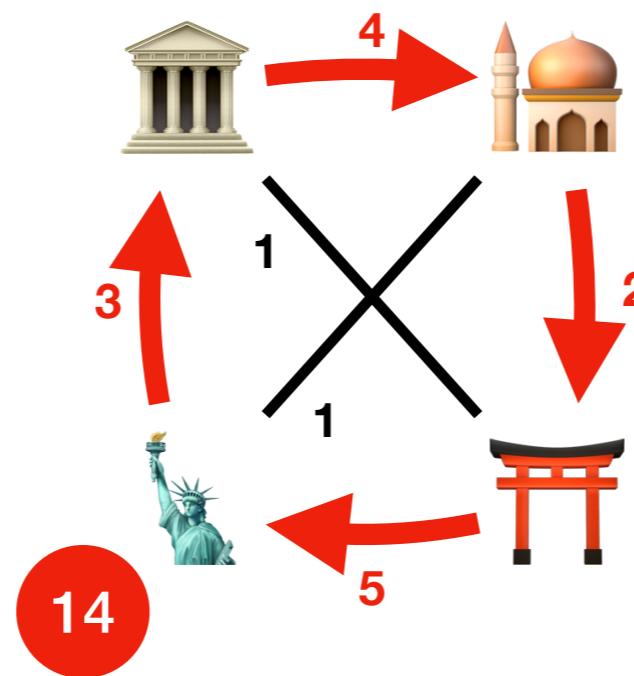
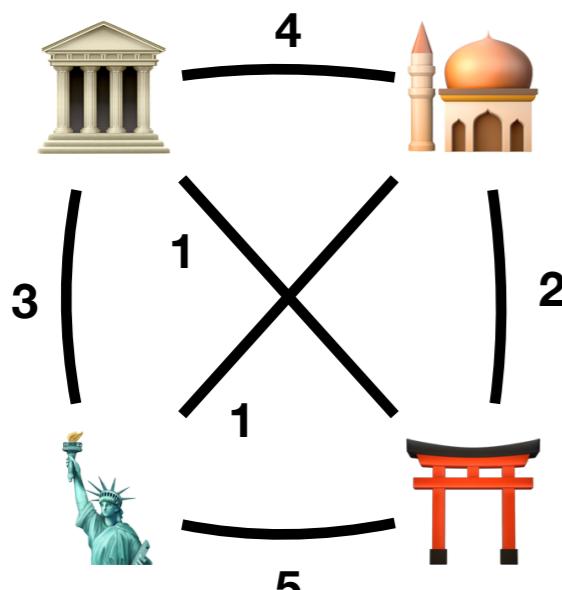
- Entrée : un **graphe non orienté complet pondéré**  
 $G = (V, E = V^2, w)$  où  $w: E \rightarrow \mathbb{R}^+$  est la distance entre chaque paire de points
- Sortie : un **cycle hamiltonien** (un cycle qui traverse chaque sommet une et une seule fois) **de poids minimal**



# Problème du voyageur de commerce (TSP)



- Entrée : un **graphe non orienté complet pondéré**  $G = (V, E = V^2, w)$  où  $w: E \rightarrow \mathbb{R}^+$  est la distance entre chaque paire de points
- Sortie : un **cycle hamiltonien** (un cycle qui traverse chaque sommet une et une seule fois) **de poids minimal**



# Problème du voyageur de commerce (TSP)



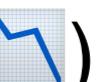
- **Instance**  $x$  = un graphe non orienté complet  $G = (V, E = V^2)$  et sa fonction de coût  $w: E \rightarrow \mathbb{R}^+$  codés sur un alphabet  $\Sigma$
- **Solutions réalisables**  $F(x) =$  tous les cycles hamiltoniens de  $G$ , c'est-à-dire, toutes les permutations de son ensemble de sommets  $V$
- **Fonction de coût**  $c(v_0, \dots, v_{n-1}) = \sum_{i=0}^{n-1} w(v_i, v_{(i+1)\text{mod}n})$
- **Coût optimal**  $\text{OPT}(x) =$  longueur de l'un des cycles hamiltoniens les plus courts
- **Solution optimale**  $s =$  un cycle hamiltonien tel que  $c(s) = \text{OPT}(x)$

# Problèmes de décision vs problèmes d'optimisation

Chaque problème d'optimisation de maximisation  (resp., de minimisation )  $A$  a un **problème de décision associé** DECISION- $A$  :

- Etant donné une entrée  $x$  du même type que  $A$  et un entier  $k$ , existe-t-il une solution réalisable  $s \in F(x)$  telle que  $c(s) \geq k$   (resp.,  $c(s) \leq k$  <img alt="cross icon" data-bbox="628 743 662 787</math>) ?

# Problèmes de décision vs problèmes d'optimisation

- Si on peut résoudre  $A$  avec un algorithme déterministe en temps polynomial, alors **DECISION- $A \in P$** 
  - Il suffit de résoudre  $A$  en temps polynomial et vérifier si le coût de la solution optimale obtenue est  $\geq k$   ( $\leq k$  pour les problèmes de minimisation )
- Inversement, si on ne peut pas résoudre en temps polynomial **DECISION- $A$** , par exemple s'il est **NP-complet** et que  $P \neq NP$ , alors on ne peut pas résoudre  $A$  en temps polynomial non plus

# **Algorithmes d'approximation**

# Algorithmes d'approximation

Un algorithme (MT déterministe)  $M$  est dit **algorithme de  $\varepsilon$ -approximation** (avec  $0 \leq \varepsilon \leq 1$ ) si pour toute entrée  $x$  il renvoie une solution réalisable  $M(x) \in F(x)$  telle que

$$\frac{|c(M(x)) - \text{OPT}(x)|}{\max\{\text{OPT}(x), c(M(x))\}} \leq \varepsilon$$

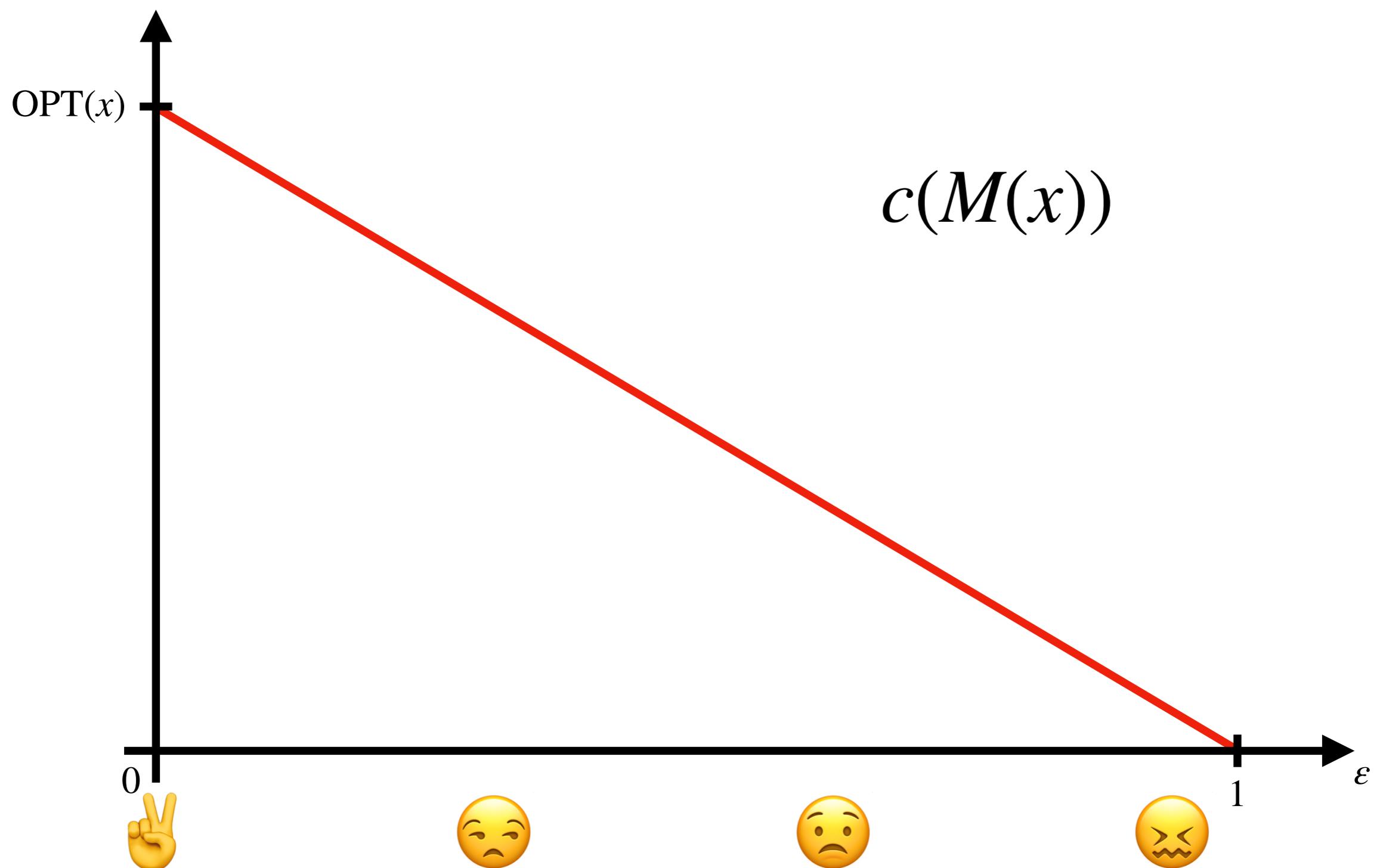
# Signification de $\frac{|c(M(x)) - \text{OPT}(x)|}{\max\{\text{OPT}(x), c(M(x))\}} \leq \varepsilon$

- Pour un problème de **maximisation**  ça veut dire que

$$\frac{\text{OPT}(x) - c(M(x))}{\text{OPT}(x)} \leq \varepsilon$$

- Donc  $c(M(x)) \geq (1 - \varepsilon)\text{OPT}(x)$
- Un algorithme de **0-approximation** renvoie donc toujours une solution **optimale** :  $c(M(x)) \geq \text{OPT}(x)$  donc  $c(M(x)) = \text{OPT}(x)$
- Par contre, un algorithme de **1-approximation** peut renvoyer n'importe quelle solution réalisable :  $c(M(x)) \geq 0$

**Max**  $c(M(x)) \geq (1 - \varepsilon)\text{OPT}(x)$



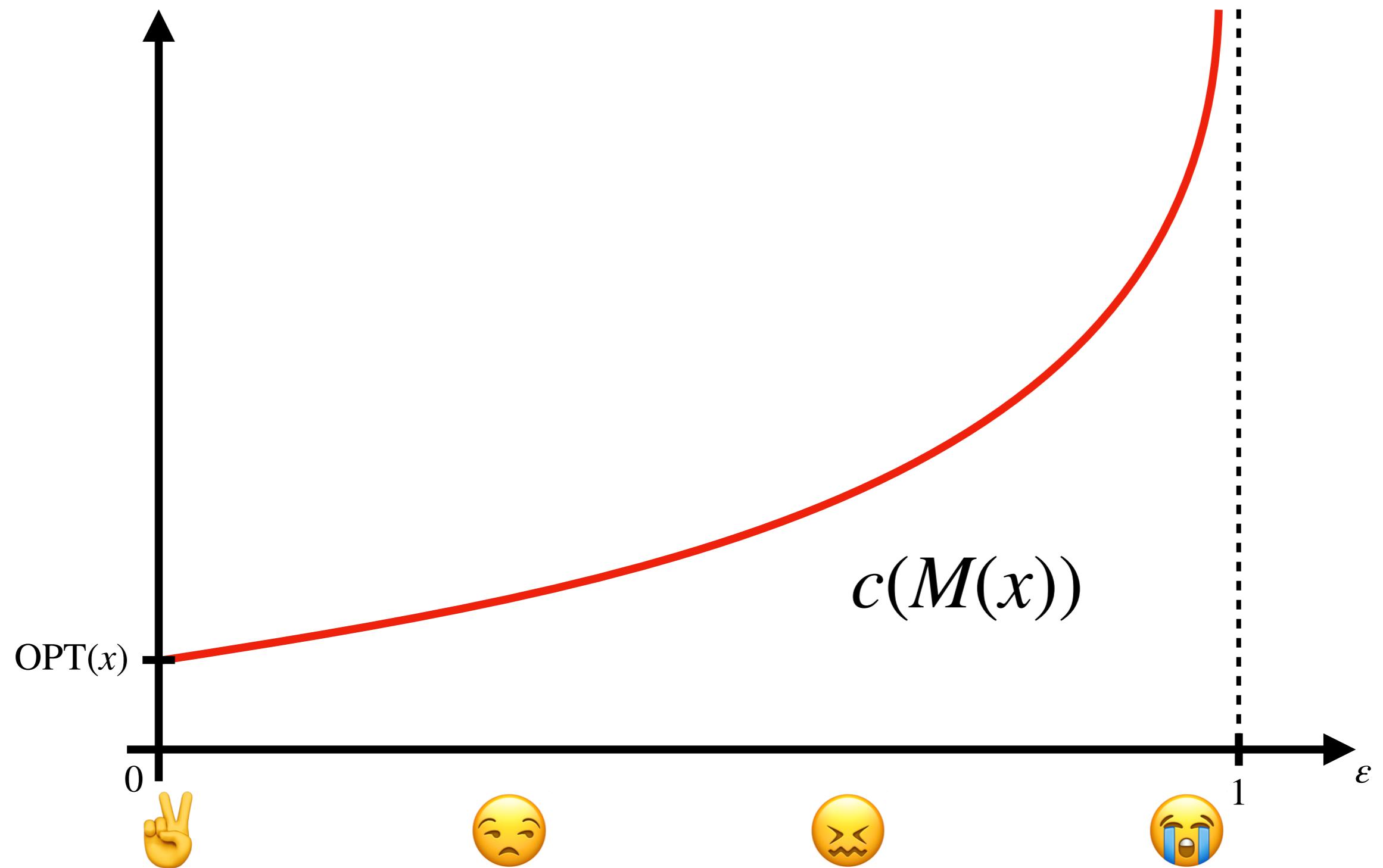
# Signification de $\frac{|c(M(x)) - \text{OPT}(x)|}{\max\{\text{OPT}(x), c(M(x))\}} \leq \varepsilon$

- Pour un problème de **minimisation** ↘ ça veut dire que

$$\frac{c(M(x)) - \text{OPT}(x)}{c(M(x))} \leq \varepsilon$$

- Donc  $c(M(x)) \leq \frac{1}{1 - \varepsilon} \text{OPT}(x)$
- Un algorithme de **0-approximation** renvoie donc toujours une solution **optimale** :  $c(M(x)) \leq \text{OPT}(x)$  donc  $c(M(x)) = \text{OPT}(x)$
- Par contre, un algorithme de **1-approximation** peut renvoyer n'importe quelle solution réalisable :  $c(M(x)) \leq \infty$  (👉 petit abus de notation ici)

$$\text{Min } c(M(x)) \leq \frac{1}{1 - \varepsilon} \text{OPT}(x)$$



# Problèmes approximables

- On est intéressé à trouver des algorithmes de  $\varepsilon$ -approximation (avec un petit  $\varepsilon$  !) qui fonctionnent en temps polynomial pour les problèmes dont la version de décision est **NP-complete**
- Où possible, on voudrait trouver **le plus petit  $\varepsilon$** , mais **parfois ça n'existe pas**, même si on peut trouver des  $\varepsilon$  **arbitrairement petits**
- La **seuil d'approximation** d'un problème d'optimisation  $A$  est **le plus petit  $\varepsilon$**  tel qu'il existe un algorithme de  $\varepsilon$ -approximation pour  $A$

**Un problème difficile  
mais approximable**



# Problème d'optimisation VERTEX-COVER

- Entrée : un graphe non orienté  $G = (V, E)$
- Sortie : le **plus petit** ensemble  $C \subseteq V$  tel que chaque arête dans  $E$  a au moins un sommet dans  $C$
- Il s'agit donc de **minimiser**  le coût  $c(C) = |C|$  parmi toutes les couvertures  $C \subseteq V$
- Rappel : DECISION-VERTEX-COVER (**existe-t-il une couverture de taille  $\leq k$  ?**) est **NP-complet**, donc il n'existe pas\* un algo exact polynomial pour le problème d'optimisation

\* sous l'hypothèse que **P**  $\neq$  **NP**

# Algo $M$ pour VERTEX-COVER

- $C := \emptyset$
- **tant que**  $E \neq \emptyset$  faire
  - choisir **n'importe quelle arête**  $\{u, v\} \in E$
  - ajouter  $u$  et  $v$  à  $C$
  - éliminer tout les arêtes avec  $u$  et  $v$  de  $E$
- **renvoyer**  $C$

# Algo $M$ pour VERTEX-COVER

- $C := \emptyset$
- **tant que**  $E \neq \emptyset$  faire
  - choisir **n'importe quelle arête**  $\{u, v\} \in E$
  - ajouter  $u$  et  $v$  à  $C$
  - éliminer tout les arêtes avec  $u$  et  $v$  de  $E$
- **renvoyer**  $C$



# $M$ est un algo de $\frac{1}{2}$ -approx 😱

- Ça veut dire qu'il renvoie toujours une solution de taille au pire  $\frac{1}{1 - \varepsilon} = \frac{1}{1 - 1/2} = 2$  fois la taille d'une solution optimale 🙌
- $C$  contient les extrémités de  $\frac{1}{2} |C|$  arêtes de  $G$  qui ne partagent jamais un sommet
- Chaque couverture, y compris une optimale, doit contenir au moins un sommet de chacune des arêtes de  $C$  (sinon il y aurait une arête qui n'est pas couverte), donc  $\text{OPT}(G) \geq \frac{1}{2} |C|$
- Et donc  $\frac{|c(M(G)) - \text{OPT}(G)|}{\max\{\text{OPT}(G), c(M(G))\}} = \frac{|C| - \text{OPT}(G)}{|C|} \leq \frac{|C| - \frac{1}{2} |C|}{|C|} \leq \frac{1}{2}$

# Considerations sur VERTEX-COVER

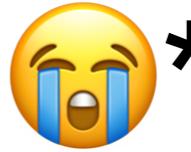
- Il est possible d'obtenir une solution de coût tout au plus double comparé à une solution optimale en temps polynomial
- Par contre, s'il existe un algo polynomial avec  $\varepsilon = 0$ , c'est-à-dire qui donne toujours une solution optimale, alors **P = NP**
- En effet, on pourrait résoudre DECISION-VERTEX-COVER en temps polynomial en calculant la solution optimale  $s$  et en vérifiant si  $c(s) \leq k$

# HAMILTONIAN-CYCLE est NP-complet

- Entrée : un **graphe orienté**  $G = (V, E)$
- Question : existe-t-il un **cycle** dans  $G$  qui visite **chaque sommet exactement une fois** (un cycle hamiltonien) ?



est difficile et même  
pas approximable



\*

\* sous l'hypothèse que  $P \neq NP$

# Théorème

- À moins que  $P = NP$ , le seuil d'approximation pour  est 1
- Ça veut dire que on n'a **aucune garantie du tout** sur le coût de la solution trouvé par  $M$  par rapport à une solution optimale
- (On sait seulement que  $c(M(x)) \leq \infty \dots \text{:(}$ )

# Démonstration

- Par contradiction, soit  $M$  un algo de  $\varepsilon$ -approximation pour  avec  $\varepsilon < 1$
- Avec ça, on va construire un algo polynomial pour HAMILTON-CYCLE ( c'est une sorte de réduction aussi !)
- Étant donnée une instance  $G = (V, E)$  de HAMILTON-CYCLE, on construit un graphe complet  $G' = (V, E' = V^2)$  pondéré :

$$w(u, v) = \begin{cases} 1 & \text{si } (u, v) \in E \\ \frac{|V|}{1 - \varepsilon} & \text{si } (u, v) \notin E \end{cases}$$

# Démonstration

- Par contradiction, soit  $M$  un algo de  $\varepsilon$ -approximation pour  avec  $\varepsilon < 1$
- Avec ça, on va construire un algo polynomial pour HAMILTON-CYCLE ( c'est une sorte de réduction aussi !)
- Étant donnée une instance  $G = (V, E)$  de HAMILTON-CYCLE, on construit un graphe complet  $G' = (V, E' = V^2)$  pondéré :

$$w(u, v) = \begin{cases} 1 & \text{si } (u, v) \in E \\ \frac{|V|}{1 - \varepsilon} & \text{si } (u, v) \notin E \end{cases}$$


# Démonstration

- Maintenant **on utilise notre algo** de  $\varepsilon$ -approximation hypothétique  $M$  sur le graphe  $G'$  pondéré par  $w$
- Si  $M(G', w)$  donne une solution de coût  $|V|$ , alors on ne parcourt que des arêtes originales de  $G$  qui, du coup, **aura un cycle hamiltonien**

# Démonstration

- Si, par contre,  $M(G', w)$  donne une solution avec au moins une des « nouvelles » arêtes de poids  $\frac{|V|}{1 - \varepsilon}$ , alors le poids de la solution sera  $c(M(G', w)) > \frac{|V|}{1 - \varepsilon}$
- Vu que  $M$  est un algo de  $\varepsilon$ -approximation, on a  $c(M(G', w)) \leq \frac{1}{1 - \varepsilon} \text{OPT}(G', w)$ , donc
- Donc une solution optimale a coût  $> |V|$ , ce qui implique que  $G$  n'a pas de cycle hamiltonien



# Conclusions

- VERTEX-COVER est difficile, mais au moins on peut l'approximer avec un facteur  $1/2$
- Par contre,  en général est difficile et on ne peut même pas l'approximer
- Donc soit on se débrouille avec les algos qu'on connaît, soit on trouve une sous-classe d'instances  qu'on peut bien approximer

*The End*