

TP 2 - Node Runner

IFT1015 ~ Automne 2017

Le TP2 a pour but de vous faire pratiquer les concepts suivants : les opérations sur les chaînes de caractères, l'algorithmie, l'intelligence artificielle, les opérations sur les tableaux, la décomposition fonctionnelle, la programmation événementielle, l'affichage graphique dans une page web en HTML et les tests unitaires.

Vous avez à coder et tester un ensemble de fonctions en JavaScript, à l'extérieur de l'environnement de développement codeBoot.

Vous pouvez réutiliser le code qui a été montré dans le cours mais vous ne devez pas utiliser du code provenant d'ailleurs (du web).

1. Introduction

Ce travail pratique consiste à développer une intelligence artificielle pour un jeu inspiré de *Lode Runner*, un jeu de plateforme datant des années 80.

Pour citer Wikipédia :

Le joueur incarne un personnage évoluant dans un décor en deux dimensions constitué d'échelles, de barres de franchissement, de murs et de passerelles de briques et de pierre. Le but du joueur est de ramasser les lingots disséminés dans le décor (sur les passerelles, en haut des échelles ou suspendus dans le vide) tout en évitant des gardes qui essaient de l'attraper. Une fois que tous les lingots sont récupérés, il doit s'échapper en rejoignant le sommet du décor (éventuellement une échelle apparaît pour l'y aider lorsque tous les lingots sont récupérés) pour passer au tableau suivant.

Une version web du jeu original est disponible ici : <http://loderunnerwebgame.com/game/>

Notez que *Node Runner* est une version simplifiée du jeu original :

- Il n'y a pas de gardes
- La sortie est une case comme une autre, plutôt qu'une échelle vers le haut du niveau

Fichiers fournis

- **tp2-ia.js** : ce fichier contient votre code pour l'intelligence artificielle de votre runner
- **tp2-graphique.js** : ce fichier contient la logique pour afficher le jeu graphiquement
- *runner.js* : ce programme est un client de jeu qui permet de jouer avec les flèches du clavier (utile seulement pour tester le jeu)
- *index.html* : c'est la page web qui est utilisée pour l'affichage graphique du jeu
- *style.css* : définition du CSS pour afficher une belle page web
- *ia.js* : ce fichier contient la logique de base pour connecter votre intelligence artificielle (**tp2-ia.js**) avec le serveur
- *html-base.js* : ce fichier contient la logique de base pour connecter la page html avec le serveur
- *img/* : ce dossier contient les images à utiliser pour l'affichage graphique

Notez que vous n'aurez qu'à modifier les fichiers **tp2-ai.js** et **tp2-graphique.js**

Pour tester le jeu :

1. Lancez la commande `node runner.js {mot de passe secret}` depuis un terminal dans le dossier de fichiers fournis
2. Ouvrez *index.html* dans un navigateur, entrez votre `{mot de passe secret}` dans la boîte puis cliquez sur le crochet
3. Retournez à votre terminal qui exécute `node` et utilisez les flèches ou WASD pour déplacer votre runner et Z/C pour creuser. **Gardez votre navigateur visible, c'est là que se fait l'affichage du jeu**

Le `{mot de passe secret}` est ce qui identifie votre partie sur le serveur, assurez-vous de choisir quelque chose d'unique (et de difficile à deviner, si vous ne voulez pas que d'autres gens regardent vos parties !)

Notez que vous n'aurez pas à modifier `runner.js`, c'est un programme qui vous est donné pour que vous puissiez tester le jeu avant de commencer à coder.

2. Déroulement du jeu

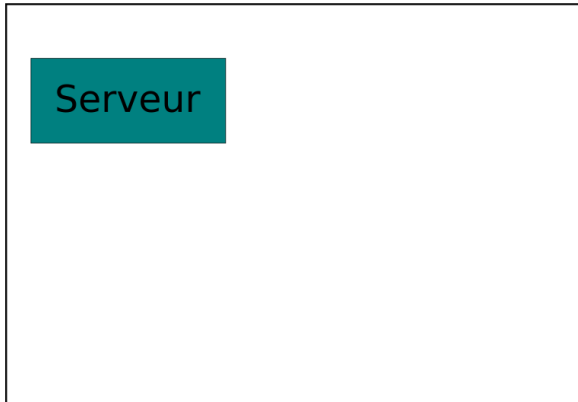
Le jeu est conçu avec une architecture client/serveur. Autrement dit, lors d'une partie, deux programmes interagissent :

- Un serveur, qui attend que des joueurs se connectent puis qui coordonne les parties
- Des clients qui se connectent au serveur
 - Un client de type "runner" (soit `runner.js`, soit `ia.js`) envoie périodiquement les déplacements que le joueur souhaite faire
 - Un client de type "affichage" (`index.html`) reçoit toute la grille à chaque tour et se charge de l'afficher

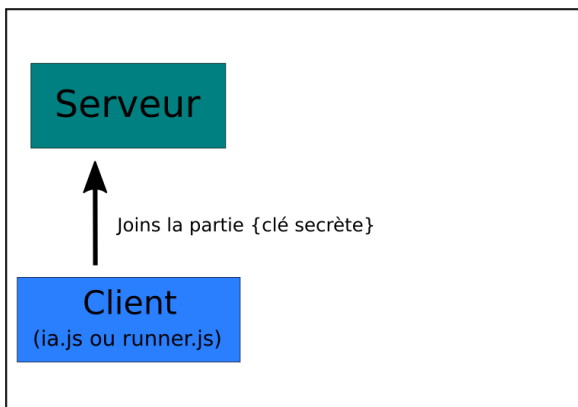
Le jeu se joue tour par tour, à chaque tour, le joueur envoie une commande au serveur : soit se déplacer dans une des quatre directions (haut/bas/gauche/droite), soit creuser dans une direction (gauche/droite).

Visuellement :

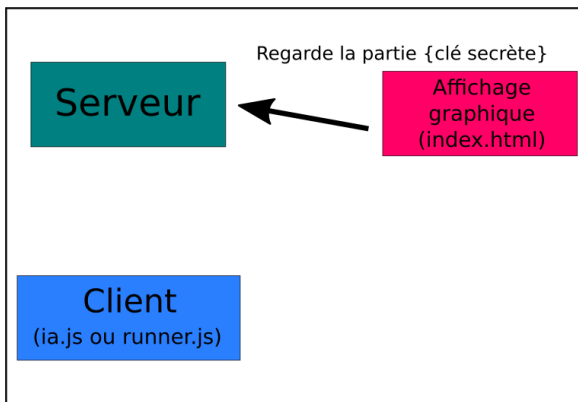
Le serveur se situe quelque part dans le cloud infonuagique, plus précisément à l'adresse *138.197.153.140* et attends des connexions :



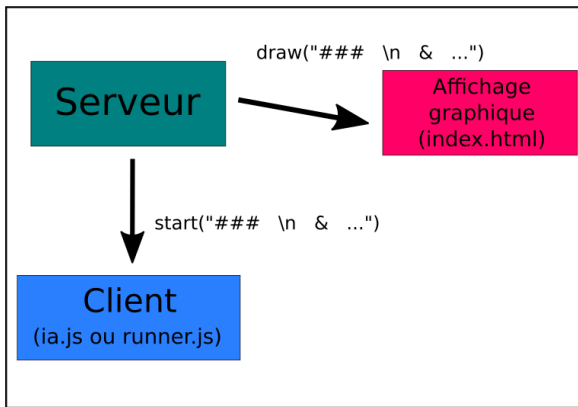
Lorsqu'un runner souhaite se connecter pour jouer, il envoie un message au serveur :



Pour afficher la partie, il faut également qu'un client de type "affichage" joigne le serveur :

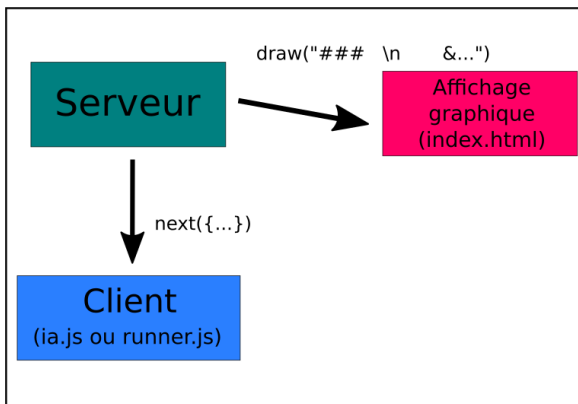


Lorsque le "runner" est prêt, le serveur lance la partie et la fonction `start()` dans `tp2-ia.js` est appelée avec une représentation textuelle de la grille de jeu initiale en paramètre :

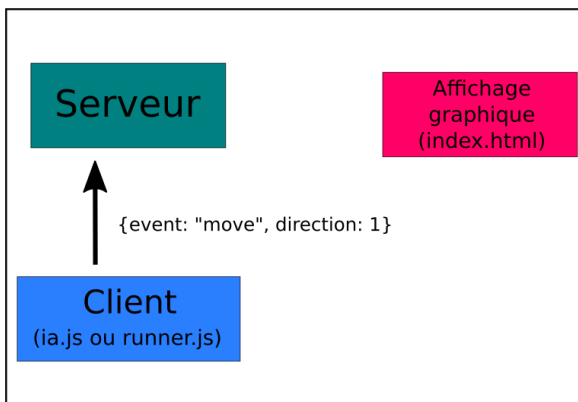


Le runner en profite pour noter l'état initial de la grille et attend le premier tour.

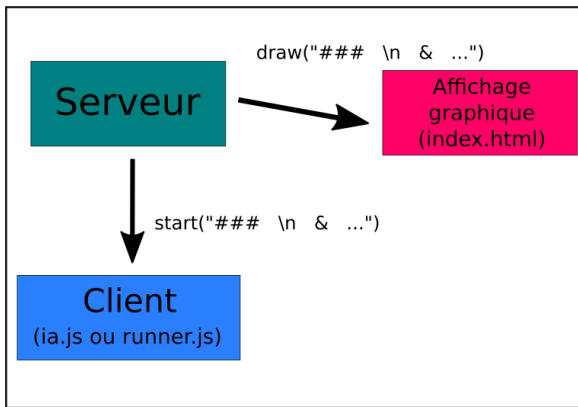
À chaque tour, la fonction `next()` est appelée dans `tp2-ia.js` et la fonction `draw()` est appelée du côté de `tp2-graphique.js`.



La page web doit alors s'actualiser pour afficher le nouvel état du jeu et le runner doit envoyer au serveur dans quelle direction il doit aller. C'est le résultat (le `return`) de la fonction "next" qui est envoyé au serveur :









Les deux dernières étapes se répètent jusqu'à ce que le runner atteigne la sortie. Une fois la sortie atteinte, le serveur envoie un nouveau level à résoudre au runner (toujours en appelant la fonction `start()`) :



Blocs

La grille peut être composée de différents blocs :

- **Case vide** (espace) : rien de spécial, on peut se déplacer dedans
-  **Bloc de brique** (#) : on peut marcher dessus et *creuser dedans pour créer un chemin
-  **Corde** (-) : on peut se déplacer latéralement dessus (gauche/droite), ou se laisser tomber (bas)
-  **Échelle** (H) : on peut monter, descendre, ou aller de côté pour se laisser tomber en bas de l'échelle. On peut également marcher sur la partie de l'échelle la plus haute.
-  **Lingot d'or** (\$) : on doit ramasser tous les lingots d'or avant de passer à la sortie
-  **Joueur** (&) : la case où se trouve le joueur
-  **Sortie** (S) : la case de sortie

*Notez qu'à moins de faire le bonus, vous n'aurez pas besoin de creuser dans des blocs. Vous pourrez vous concentrer sur les déplacements à faire dans la carte, qui constituent déjà un gros morceau.

3. Implantation

Vous devrez :

1. Programmer une intelligence artificielle pour le jeu (**tp2-ia.js**)
2. Programmer l'affichage du jeu avec des belles images (**tp2-graphique.js**)

Intelligence Artificielle (**tp2-ia.js**)

Vous devrez programmer un algorithme qui saura ramasser les lingots d'or puis se diriger vers la sortie.

Lorsque le jeu commence, la fonction **start()** est appelée automatiquement. C'est dans cette fonction que vous devrez stocker la grille de jeu initiale reçue en paramètre

À chaque tour, la fonction `next()` est appelée automatiquement. C’est cette fonction qui décide de la direction dans laquelle vous voulez vous déplacer. La fonction `next()` doit retourner un enregistrement de la forme :

```
{event: ..., direction: ...}
```

Où :

- **event** est une chaîne de caractères :
 - “move” pour se déplacer
 - “dig” pour creuser
- **direction** est un nombre compris entre 1 et 4 inclusivement :
 1. haut
 2. gauche
 3. bas
 4. droite

Notez encore que “dig” ne servira qu’aux équipes qui décident de faire le bonus. Vous pouvez donc commencer en retournant toujours des actions de type “move”.

Vous devez écrire votre code dans le fichier `tp2-ia.js` fourni. Lisez les commentaires déjà présents dans le code pour vous guider.

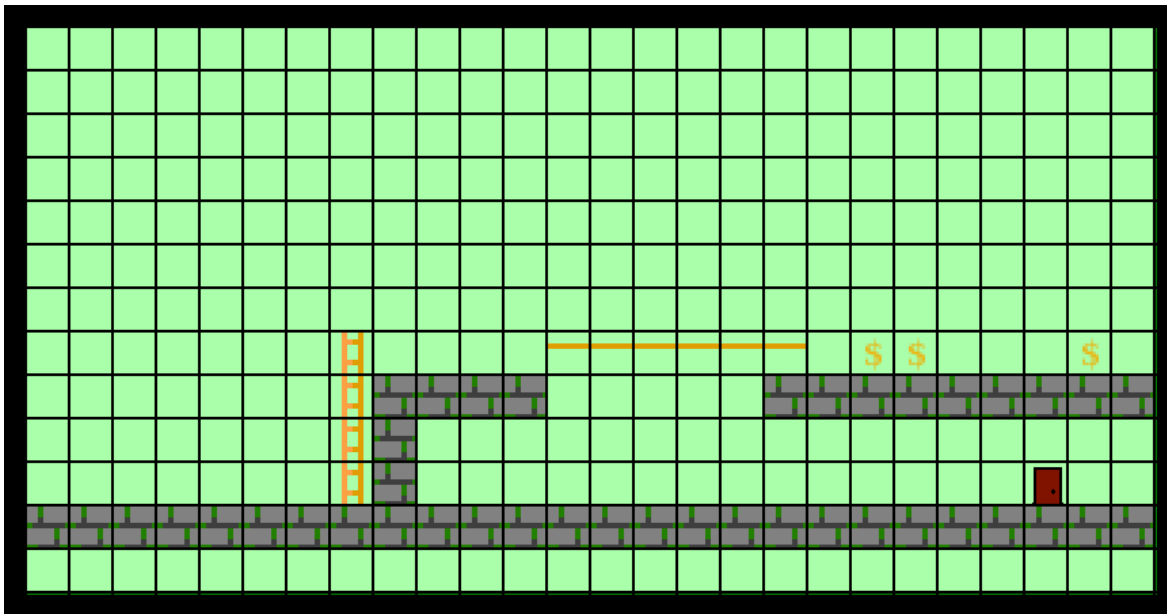
Pour tester votre IA, vous pouvez lancer la commande :

```
node ia.js
```

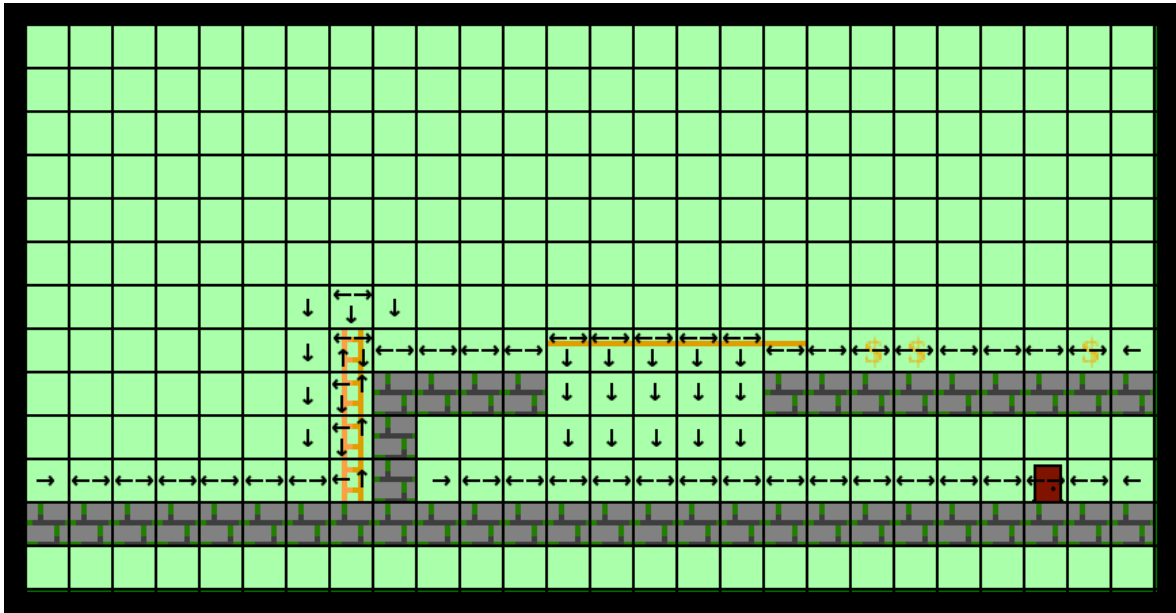
Qui se chargera de la connexion au serveur et qui appellera vos fonctions `start` et `next` aux bons moments.

Trouver son chemin

À partir de la grille du jeu :

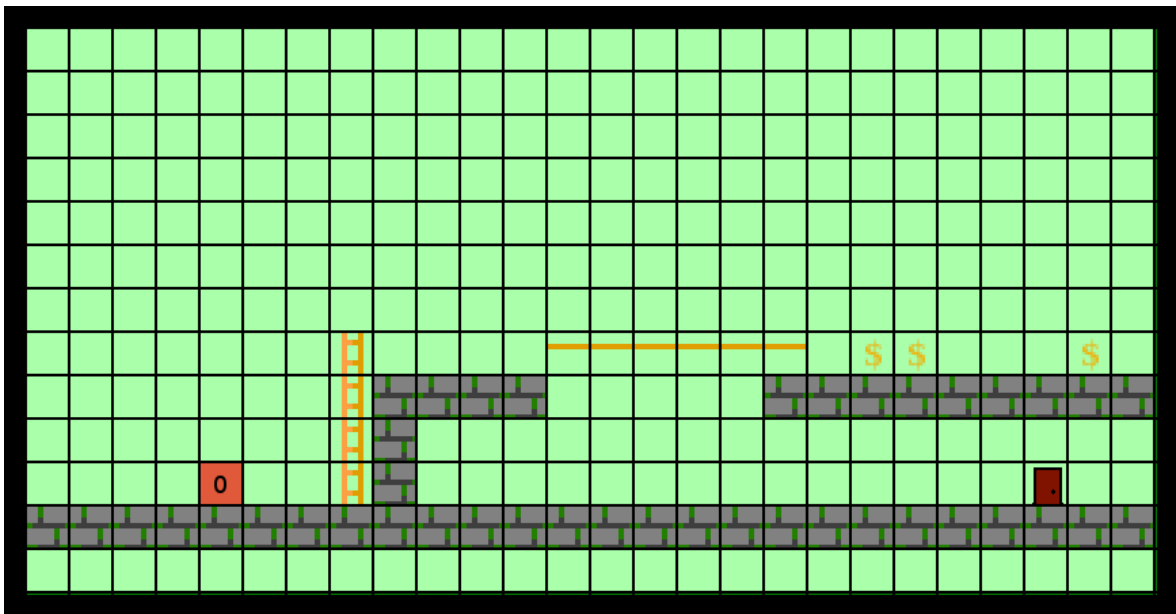


On peut déterminer quels déplacements sont valides pour chaque case, indépendamment de la position du joueur :

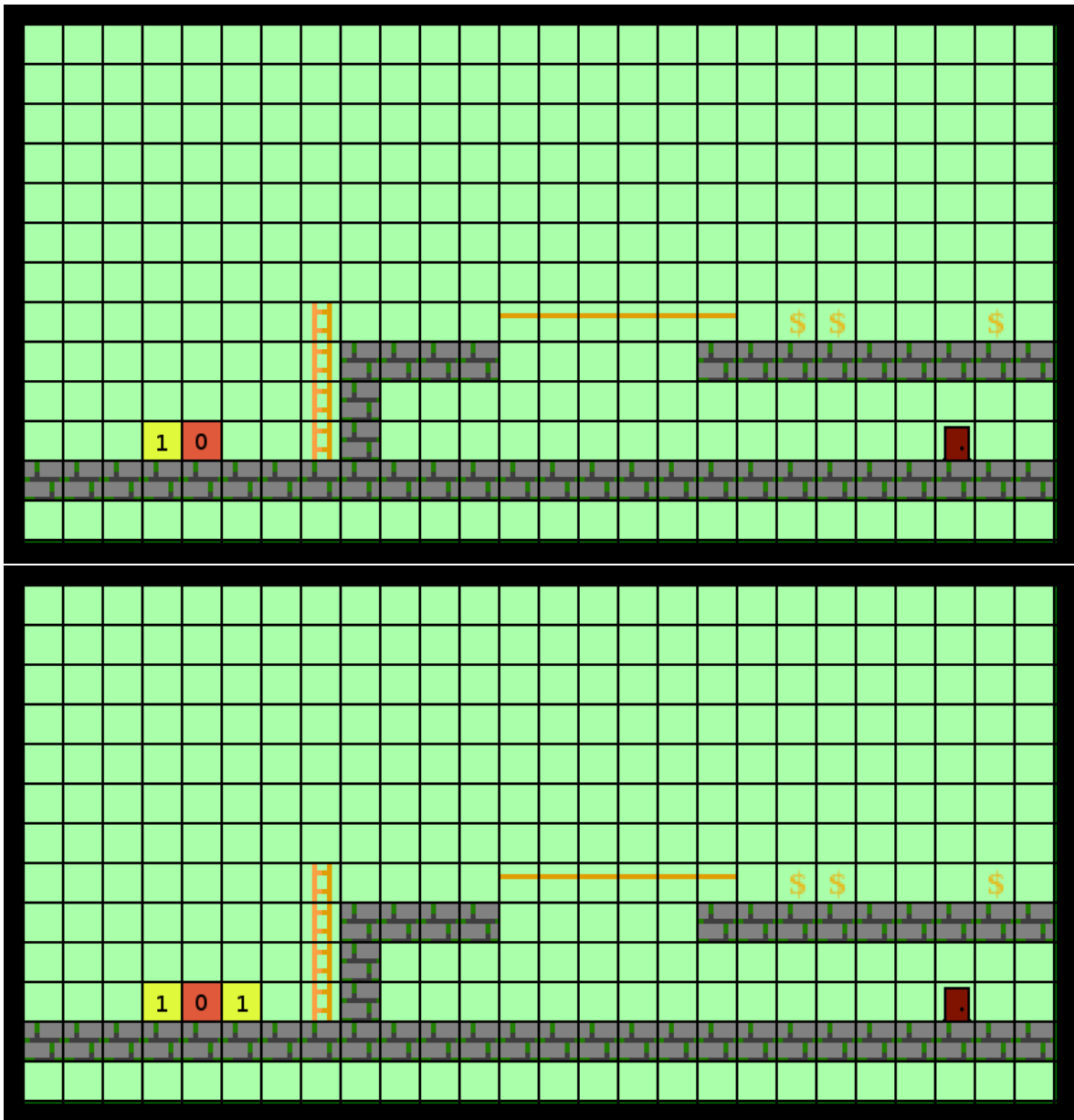


Une façon efficace de décider par où aller est d'utiliser une recherche *breadth first* (ou fouille en largeur) dans le graphe des déplacements possibles.

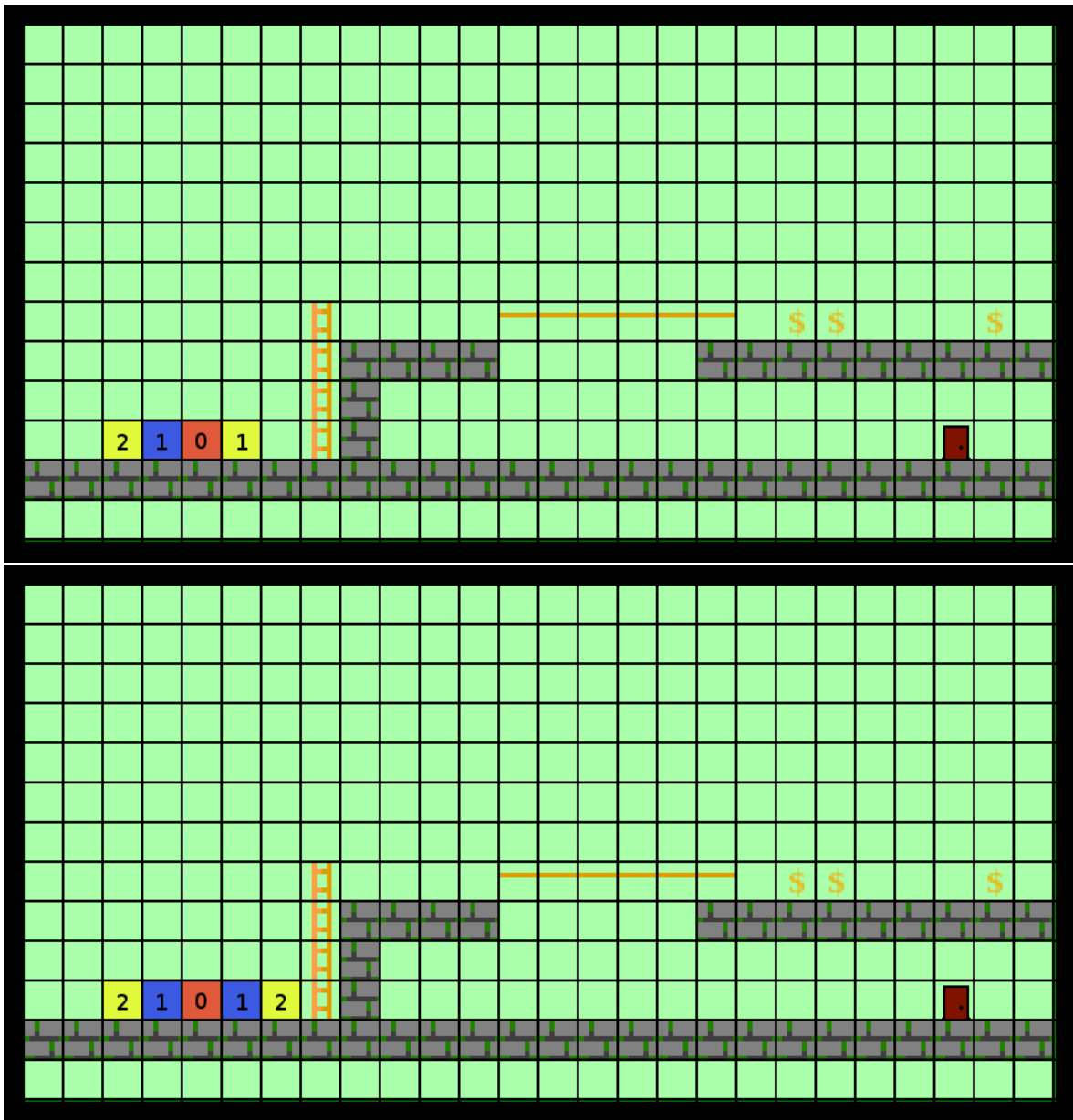
On part d'une position initiale, où on fixe la distance à 0 :



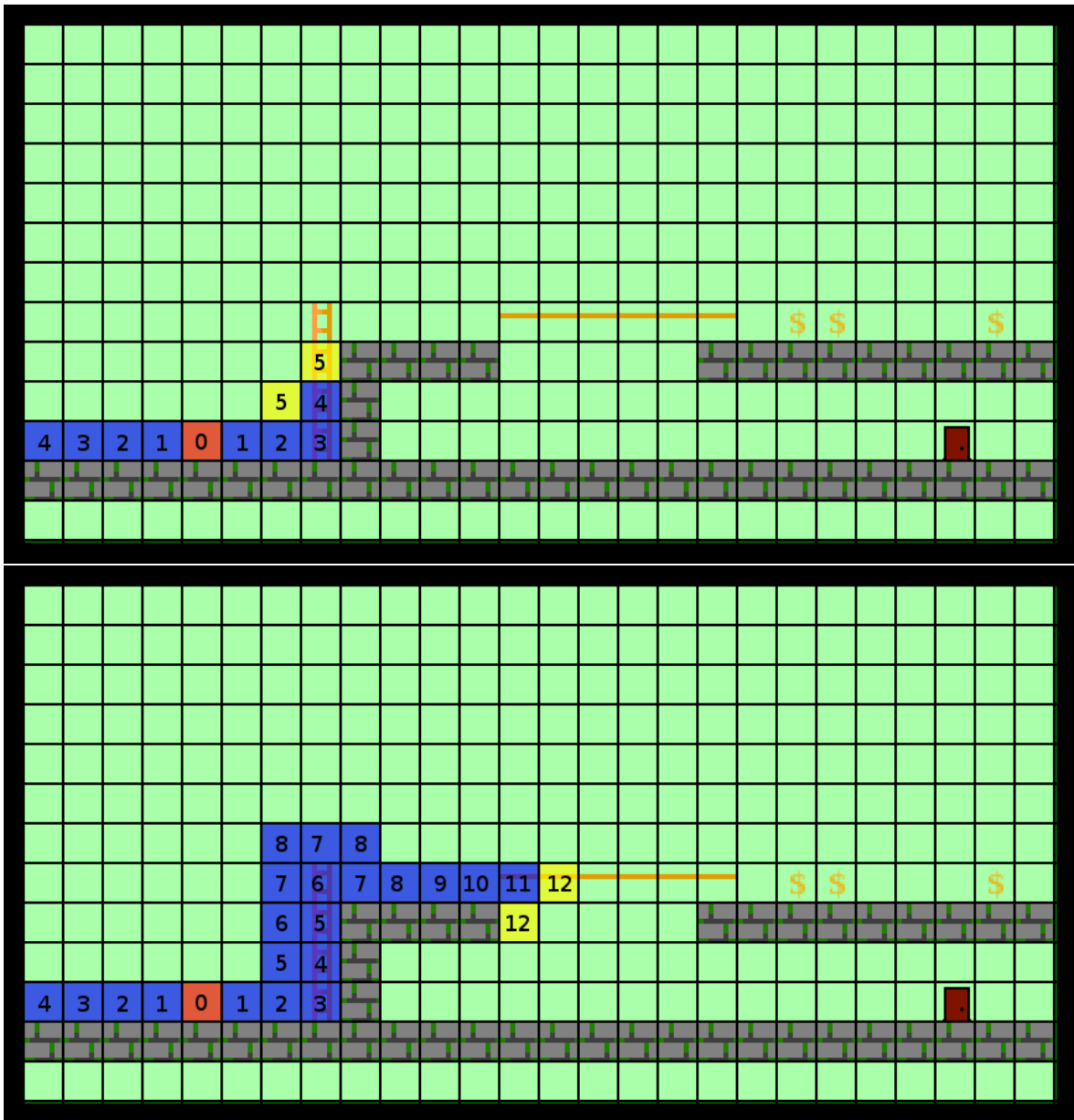
Et pour chaque voisin accessible depuis la position, on calcule leur distance à la position initiale (un de plus que la position considérée) et on ajoute ces voisins à une queue (*first-in-first-out*) de cases à visiter plus tard (cases marquées en jaune) :

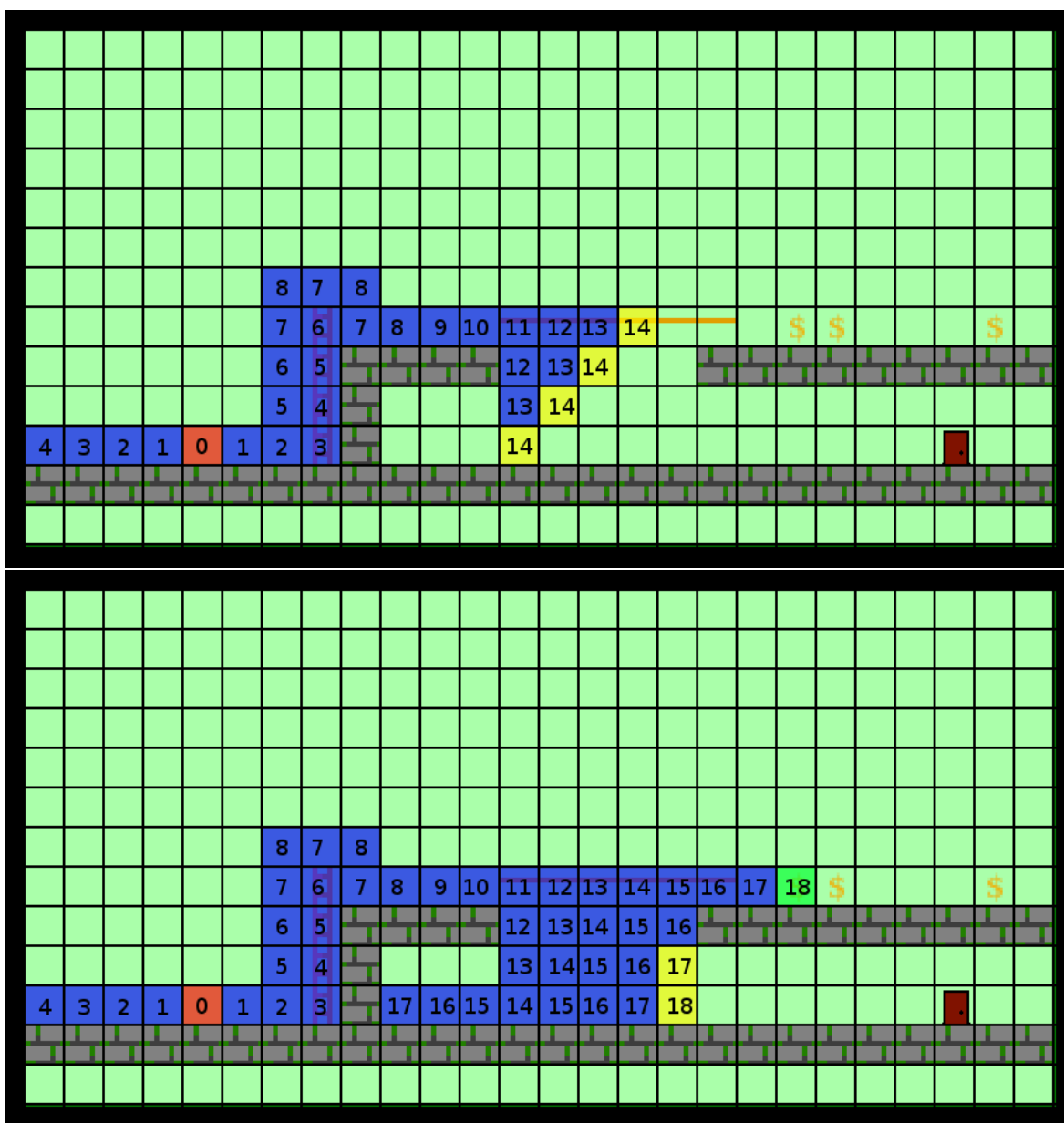


Lorsqu'on a visité tous les voisins de la case actuelle, on recommence à partir de la première case sur la file d'attente :

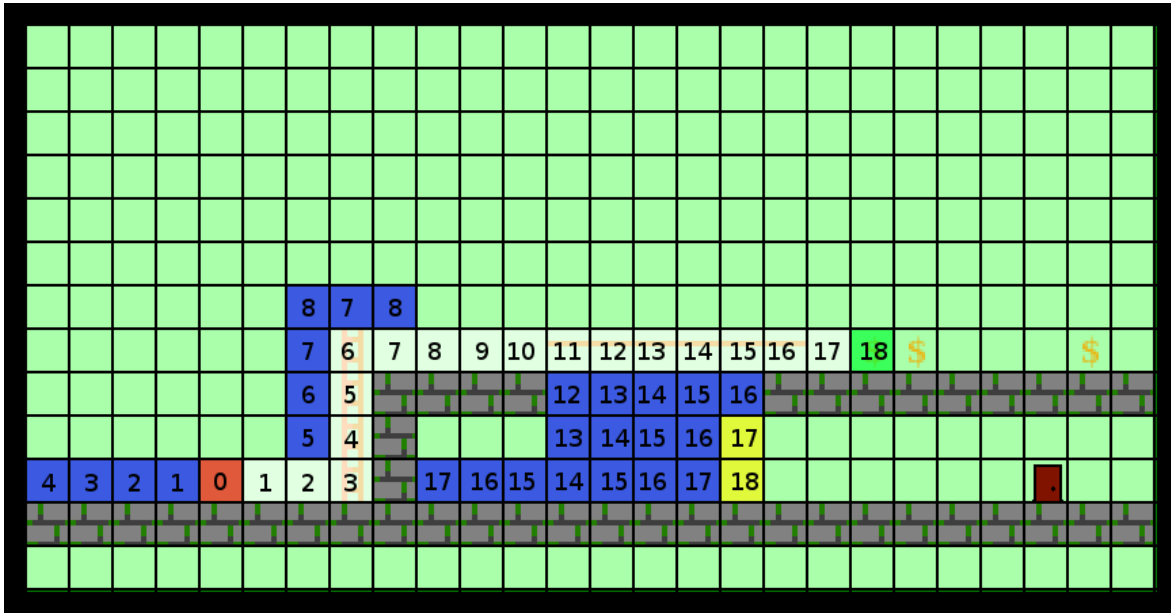


On continue jusqu'à arriver à une case intéressante (soit parce qu'elle contient un lingot d'or, soit parce qu'on a terminé de collecter l'or et que c'est la sortie)





Lorsqu'on trouve une case intéressante, on part de cette case et on suit les cases ayant la valeur de distance la plus petite jusqu'à retrouver la position de départ :



Cet algorithme va vous donner le chemin le plus court pour passer de votre position jusqu'à la case d'intérêt la plus proche.

Si jamais vous faites les niveaux bonus, vous remarquerez que ce n'est pas systématiquement la bonne chose à faire...

Affichage du jeu (tp2-graphique.js)

Un affichage primitif (ASCII) vous est déjà donné dans `tp2-graphique.js`. Pour le voir, vous devez lancer un client de joueur dans node et ouvrir la page `index.html` dans votre navigateur (voir la section 1 - *Introduction*).

Vous devrez modifier le code de la fonction `draw()` pour arriver à quelque chose de (discutablement) plus joli. La fonction est appelée automatiquement à chaque tour et reçoit en paramètre une représentation textuelle de la grille.

Pour l'affichage graphique, vous pouvez utiliser les images fournies dans `img/*.png`

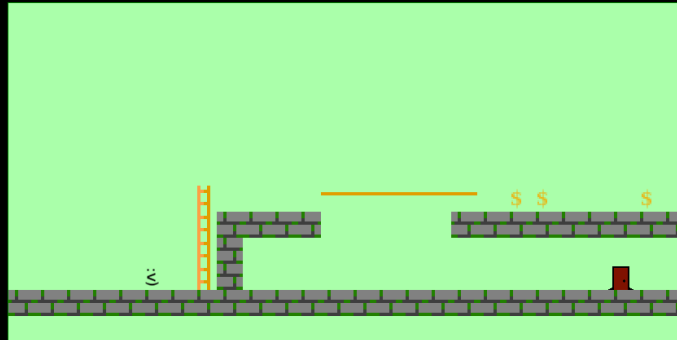
La façon la plus simple de dessiner la grille de jeu est en créant un élément HTML `<table>` avec autant de cellules `<td>` que de cases dans la grille et en changeant le `background-image` associé à chaque case pour refléter ce que la grille contient. Ce n'est pas très efficace, mais ça devrait faire l'affaire pour ce TP¹. Un exemple de génération de `<table>` est donné dans la fonction `draw()`, vous pouvez vous en inspirer pour faire votre fonction.

Le résultat final de votre page web devrait ressembler à quelque chose comme :

¹ Notez : Si jamais vous souhaitez aller un peu plus loin, vous pouvez fouiller sur l'utilisation des '`<canvas>`', qui sont beaucoup plus efficaces pour dessiner des images arbitraires sur l'écran.

Node Runner

room



4. Évaluation

- 30% sur l’affichage graphique en HTML
 - La grille de jeu est affichée sans erreur
 - 40% sur l’intelligence artificielle
 - Votre IA doit *réussir* les niveaux 1 à 4 inclusivement
 - **3% bonus** par niveau atteint à partir du 5ème niveau (donc 3% bonus si vous réussissez le niveau 5, et 12% bonus si vous réussissez tout jusqu’au niveau 8)
 - 30% sur la qualité du code (le style, les commentaires, le découpage fonctionnel, le choix des noms de fonctions et de variables, etc.)
 - *15% de pénalité & les points bonus ne sont pas comptés si le travail est fait individuellement*
-

5. Indications supplémentaires

- **Vous aurez besoin d’une connexion internet pour faire ce TP.** Si jamais c’est un problème, assurez-vous de trouver une solution avec un démonstrateur rapidement.
- La date de remise est le *18 décembre 2017 à 23h55*. Il y a une pénalité de 33% pour chaque jour de retard.
- Vous devez faire le travail par groupes de 2 personnes. Indiquez vos noms clairement dans les commentaires au début de votre code. Un travail fait seul engendrera une pénalité (qui ne peut pas être compensée par les points bonus). Les équipes de plus de deux seront refusées.
- Vous devez seulement remettre deux fichiers : `tp2-graphique.js` et `tp2-ia.js`
- Voici les critères d’évaluation du code :
 - l’exactitude (respect de la spécification)
 - l’élégance et la lisibilité du code
 - la présence de commentaires explicatifs lorsque nécessaire
 - le choix des identificateurs
 - la décomposition fonctionnelle et le choix de tests unitaires pertinents
- De plus :
 - La performance de votre code doit être raisonnable.
 - Chaque fonction devrait avoir un bref commentaire pour indiquer ce qu’elle fait.
 - Il devrait y avoir des lignes blanches pour que le code ne soit pas trop dense (utilisez votre bon sens pour arriver à un code facile à lire)
 - Les identificateurs doivent être bien choisis pour être compréhensibles (évitez les noms à une lettre, à l’exception de `i`, `j`, ... pour les variables d’itérations des boucles `for`).
 - Vous devez respecter le standard de code pour ce projet (soit, les noms de variables en camelCase).
 - Il ne devrait plus y avoir de code de debug (aka, `console.log(...)`) dans la version finale remise.