

Tutoriel

JavaScript · Phaser · Jeu 2D

Labyrinthe — Construire un jeu avec Phaser

Construire un labyrinthe jouable dans le navigateur avec Phaser, des sprites pixel-art CC0, des murs solides et deux niveaux dont une carte complexe.

Auteur **Youness Boutyour**

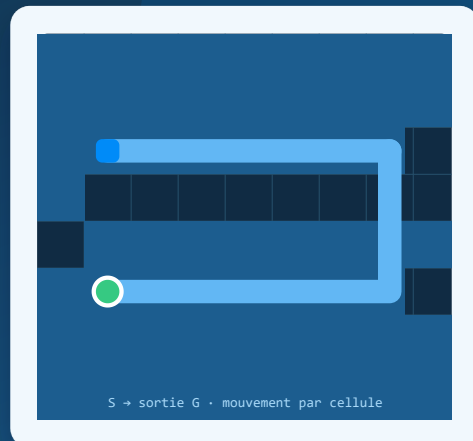
JEU 2D · JAVASCRIPT · PHASER 3

LABYRINTHE

Construire une démo interactive

Carte en caractères · scène Phaser · déplacements contrôlés

démo jouable dans le navigateur



DURÉE

60 minutes

NIVEAU

Débutant

FORMAT

Tutoriel pratique

JavaScript · Phaser · Jeu 2D · publié le 2026-10-23 · local

Objectifs et livrables

Un labyrinthe est un bon premier terrain pour apprendre Phaser : le monde est une grille, les règles tiennent en quelques fonctions et le résultat est visible immédiatement. Nous allons partir d'une carte écrite avec des caractères, construire un environnement de jeu avec un sol texturé, des murs en sprites et un personnage, puis déplacer ce personnage sans autoriser le passage dans les obstacles.

Le projet reste volontairement autonome. Phaser est chargé dans une page HTML, la scène est créée côté navigateur et aucune donnée n'est envoyée à un serveur applicatif. Le même dossier peut être lancé localement avec `python3 -m http.server`, sans installation de dépendances JavaScript ni service applicatif.

Nous ajouterons aussi une commande de recommencement, des contrôles tactiles et un état de victoire. Le tutoriel ne cherche pas à construire un moteur de jeu généraliste : il montre les frontières entre représentation, affichage et règle de déplacement. Un modèle génératif pourrait ensuite proposer des cartes ou des thèmes, mais le jeu de base reste déterministe afin que chaque règle soit vérifiable.

À la fin, vous aurez un `index.html`, un `src/main.js`, une planche de sprites locale sous licence CC0, deux cartes et un jeu autonome exécutable dans un navigateur. L'archive finale reprend exactement le code et les ressources nécessaires pour lancer le projet sur votre machine.

CONSTRUCTION PAS À PAS

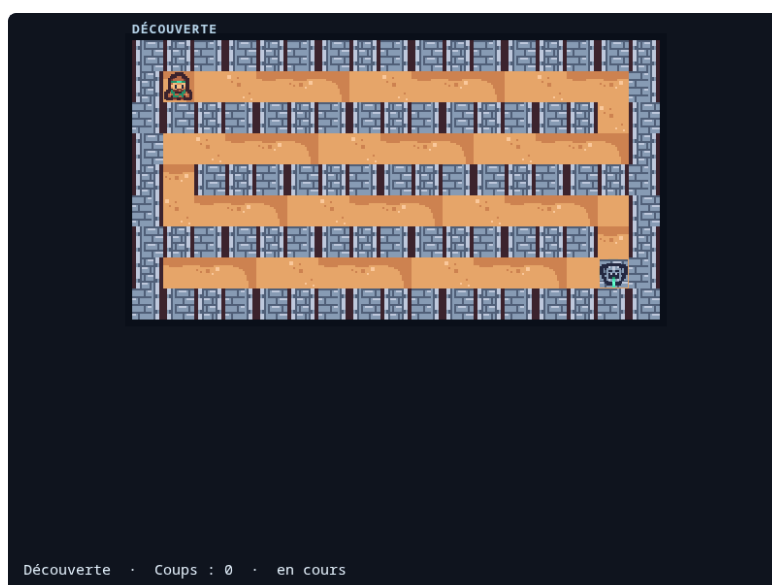
Du tableau de caractères à la partie jouable

Chaque couche garde une responsabilité simple : représenter, dessiner, déplacer, puis conclure.

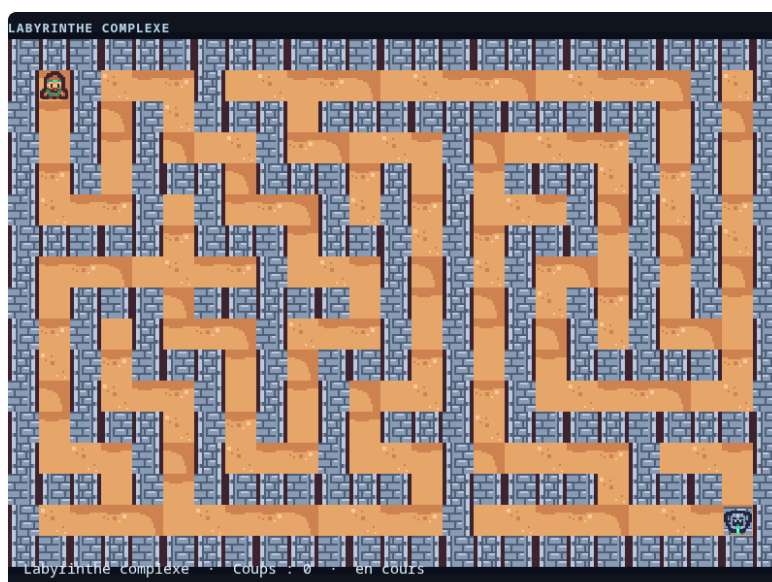


Le navigateur ne reçoit qu'une page statique : Phaser orchestre la scène, pas un serveur métier.

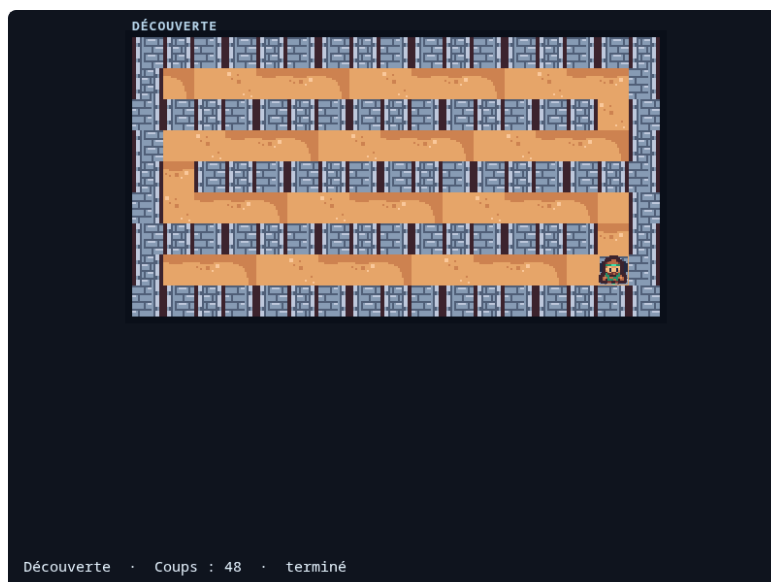
Le jeu avance par couches : représenter la carte, dessiner la scène, accepter une direction et conclure sur la sortie.



Le canvas de jeu réel affiche la carte, les obstacles, le personnage et la sortie avant la résolution du parcours.



Le niveau complexe utilise la même scène mais une carte de 25×17 cellules, avec des couloirs plus nombreux et plusieurs impasses.



Le canvas de jeu réel montre le personnage sur la sortie après le parcours minimal de 48 coups.

Comprendre la structure avant de coder

Phaser est un framework 2D exécuté dans le navigateur. Une instance de `Phaser.Game` démarre une ou plusieurs scènes ; la scène de ce tutoriel possède les objets du plateau, le joueur, le texte d'état et les entrées clavier. La documentation Phaser décrit cette organisation autour des scènes, chacune pouvant gérer son affichage, son entrée et son cycle de vie.

La géométrie du labyrinthe n'est pas importée comme une image : chaque ligne reste une chaîne où `#` désigne un mur, `.` une cellule libre, `S` le départ et `G` la sortie. La planche Kenney fournit seulement l'apparence du sol, des murs, du personnage et de la sortie. Cette séparation permet de lire la carte dans le terminal, de la versionner avec Git et de la transformer directement en coordonnées de jeu.

Le déplacement reste lisible pour un débutant : un appui demande une cellule voisine, la carte vérifie la cible, puis Arcade Physics déplace le sprite vers le centre de cette cellule. Les murs sont également enregistrés dans un groupe statique Phaser avec un collider. La carte de découverte permet de comprendre le cycle ; une seconde carte de 25×17 cellules ajoute des couloirs, des retours et des impasses pour obtenir une vraie expérience de labyrinthe.

1. Préparer la page HTML et le serveur local

Créez un dossier minimal. Un serveur HTTP local est préférable à l'ouverture directe du fichier avec `file://` : le navigateur traite alors la page comme une vraie ressource web et les chemins relatifs du jeu se comportent correctement. La planche graphique est téléchargée une fois puis servie localement ; le jeu ne dépend donc pas d'une image distante.

Code de l'étape

Terminal · Bash

```
mkdir -p labyrinthe-phaser/src labyrinthe-phaser/assets
cd labyrinthe-phaser
touch index.html src/main.js
```

Téléchargez ensuite le pack **Tiny Dungeon** de Kenney depuis sa page [OpenGameArt](https://opengameart.org/sites/default/files/kenney_tinydungeon.zip), vérifiez sa licence CC0 et extrayez la planche utilisée par le jeu :

Terminal · Bash

```
curl -L https://opengameart.org/sites/default/files/kenney_tinydungeon.zip -o /tmp/kenney_tinydungeon.zip
unzip -j /tmp/kenney_tinydungeon.zip 'Tilemap/tilemap_packed.png' -d assets
mv assets/tilemap_packed.png assets/kenney-tiny-dungeon.png
python3 -m http.server 8080
```

La planche contient des tiles 16×16 . Phaser les découpera en frames et les affichera en 32×32 avec le mode `pixelArt`, ce qui conserve les contours nets du pixel-art.

Dans `index.html`, placez un conteneur pour le canvas et chargez Phaser avant votre script. La version est fixée dans l'URL afin que le jeu ne change pas silencieusement entre deux exécutions.

HTML

```
<main>
  <h1>Labyrinthe</h1>
  <div id="game"></div>
</main>
<script src="https://cdn.jsdelivr.net/npm/phaser@3.90.0/dist/phaser.min.js" defer></script>
<script src="./src/main.js" defer></script>
```

Le fichier `src/main.js` charge ensuite la planche locale :

JavaScript

```
this.load.spritesheet('dungeon', './assets/kenney-tiny-dungeon.png', {
  frameWidth: 16,
  frameHeight: 16,
});
```

Sortie obtenue

Sortie / exemple

```
Serving HTTP on 0.0.0.0 port 8080 (http://0.0.0.0:8080/) ...
```

Ouvrez <http://localhost:8080/> . La page peut encore sembler vide : le canvas n'existe qu'après l'exécution de `main.js` et le premier démarrage de Phaser.

Commande de vérification

Terminal · Bash

```
curl -I http://localhost:8080/index.html
curl -I http://localhost:8080/src/main.js
```

Les deux réponses doivent commencer par `HTTP/1.0 200` ou `HTTP/1.1 200` . Si `curl` ne répond pas, gardez le terminal du serveur ouvert et contrôlez le dossier courant.

2. Décrire le labyrinthe avec une grille

Ajoutez la carte dans `src/main.js` . Toutes les lignes doivent avoir la même longueur : cette invariance garantit qu'un couple (`colonne`, `ligne`) correspond toujours à une cellule précise.

Code de l'étape

JavaScript

```
const TILE = 32;
const LEVELS = {
  simple: {
    label: 'Découverte',
    map: [
      '#####',
      '#S.....#',
      '#####.#',
      '#.....#',
      '#.#####',
      '#.....#',
      '#####.#',
      '#.....G#',
      '#####',
    ],
  },
  complex: {
```

```

label: 'Labyrinthe complexe',
map: [
  '#####',
  '#S#...#.....#.#',
  '#.#.###.#####.##',
  '#.#.#...#.....#.#',
  '#.#.###.###.###.###.###',
  '#...#.#...#.#...#.#.#',
  '#####.###.###.###.###.###',
  '#.....#...#.#.#...#.#',
  '#.###.#####.###.###.###.###',
  '#.#.#...#.#.#.#.#.#...#',
  '#.#.###.###.###.###.###.###',
  '#.#...#.#.#...#.#.....#',
  '#.###.###.###.#####',
  '#...#.#...#...#.....#...#',
  '###.#####.#####.###.###',
  '#.....#.....G#',
  '#####',
],
},
};

const MAP = LEVELS.simple.map;

function mapDimensions(map) {
  return { width: map[0].length, height: map.length };
}

const { width, height } = mapDimensions(MAP);
console.log(`map_size=${width}x${height}`);

function findCell(map, symbol) {
  for (let row = 0; row < map.length; row += 1) {
    const column = map[row].indexOf(symbol);
    if (column !== -1) return { column, row };
  }
  throw new Error(`Symbole absent de la carte : ${symbol}`);
}

function isWalkable(map, column, row) {
  return Boolean(map[row] && map[row][column] && map[row][column] !== '#');
}

```

`LEVELS` sépare les données de niveau du moteur. Le niveau complexe est une grille de 25 × 17 cellules générée de manière déterministe : chaque couloir reste connecté au suivant, mais plusieurs choix conduisent à des impasses. `findCell` transforme le symbole de départ ou de sortie en coordonnées. `isWalkable` refuse aussi les coordonnées hors de la carte : appuyer plusieurs fois vers le bord ne peut donc pas provoquer une exception JavaScript.

Sortie obtenue

Sortie / exemple

```
map_size=17x9
start=column 1, row 1
goal=column 15, row 7
wall_cells=91

complex_map_size=25x17
complex_goal=column 23, row 15
```

Les coordonnées sont comptées à partir de zéro dans le code. Le départ est dans la deuxième colonne et la deuxième ligne visibles, car la première ligne et la première colonne constituent la bordure de murs.

Commande de vérification

Terminal · Bash

```
node --check src/main.js
```

La commande doit se terminer sans message. À ce stade, elle vérifie uniquement la syntaxe du fichier ; elle ne remplace pas l'ouverture du jeu dans un navigateur.

3. Créer la scène Phaser et dessiner les cellules

Une scène Phaser regroupe l'initialisation et les objets à afficher. Dans `preload`, nous chargeons la planche de sprites ; dans `create`, nous trouvons le départ et la sortie, dessinons le sol, créons un groupe de murs statiques et plaçons le personnage. La position d'une cellule est convertie en pixels par son centre pour que les sprites restent alignés.

Code de l'étape

JavaScript

```
const BOARD_WIDTH = MAP[0].length * TILE;
const BOARD_HEIGHT = MAP.length * TILE;

class MazeScene extends Phaser.Scene {
  constructor() {
    super('MazeScene');
  }

  preload() {
```

```

    this.load.spritesheet('dungeon', './assets/kenney-tiny-dungeon.png', {
      frameWidth: 16,
      frameHeight: 16,
    });
  }

  create() {
    this.start = findCell(MAP, 'S');
    this.goal = findCell(MAP, 'G');
    this.playerCell = { ...this.start };
    this.walls = this.physics.add.staticGroup();
    this.drawMaze();
    this.player = this.physics.add.sprite(
      ...this.pixelPosition(this.playerCell),
      'dungeon',
      112,
    ).setDisplaySize(TILE * 0.92, TILE * 0.92);
    this.player.body.setSize(11, 13, true);
    this.physics.add.collider(this.player, this.walls);
  }

  pixelPosition(cell) {
    return [cell.column * TILE + TILE / 2, cell.row * TILE + TILE / 2];
  }
}

```

Le frame `112` correspond au personnage de la planche Kenney. Il est affiché à 32×32 pixels alors que la source fait 16×16 : le mode `pixelArt` évite un flou lors de cet agrandissement.

● ● ● JavaScript

```

drawMaze() {
  MAP.forEach((line, row) => [...line].forEach((symbol, column) => {
    const x = column * TILE + TILE / 2;
    const y = row * TILE + TILE / 2;
    this.add.image(x, y, 'dungeon', 50)
      .setDisplaySize(TILE, TILE);
    if (symbol === '#') {
      const wall = this.walls.create(x, y, 'dungeon', 58)
        .setDisplaySize(TILE, TILE);
      wall.refreshBody();
    }
    if (symbol === 'G') {
      this.add.image(x, y, 'dungeon', 20)
        .setDisplaySize(TILE * 0.9, TILE * 0.9);
    }
  }));
}

```

Sortie obtenue

Sortie / exemple

```
phaser_loaded=true
scene=MazeScene
canvas=800x600
cells=153
```

Une fenêtre ou un canvas apparaît avec le sol, les briques, le personnage et la sortie. Le niveau complexe utilise la même scène mais occupe 25×17 cellules ; la surface de jeu reste 800×600 et Phaser adapte le rendu avec `Scale.FIT`.

Commande de vérification

Terminal · Bash

```
node --check src/main.js
curl -s http://localhost:8080/src/main.js | grep -q 'class MazeScene'
```

Le second contrôle confirme que le serveur fournit bien le script contenant la scène, et non une ancienne copie ou une page HTML renvoyée par erreur.

4. Déplacer le joueur sans traverser les murs

Associez chaque touche à un vecteur (colonne, ligne). La méthode `move` calcule la cible et vérifie `is-Walkable` avant de demander à Arcade Physics de faire avancer le sprite vers le centre de la cellule. Le verrou `this.moving` ignore un nouvel appui pendant le déplacement ; `update` détecte l'arrivée et remplace le corps exactement sur la cellule atteinte.

Code de l'étape

JavaScript

```
bindKeyboard() {
  const moves = {
    UP: [0, -1], ARROWUP: [0, -1], W: [0, -1],
    DOWN: [0, 1], ARROWDOWN: [0, 1], S: [0, 1],
    LEFT: [-1, 0], ARROWLEFT: [-1, 0], A: [-1, 0],
    RIGHT: [1, 0], ARROWRIGHT: [1, 0], D: [1, 0],
  };
  Object.entries(moves).forEach(([key, vector]) => {
    this.input.keyboard.on(`keydown-${key}`, () => this.move(...vector));
  });
}
```

```
}

move(deltaColumn, deltaRow) {
  if (this.won || this.moving) return;
  const next = {
    column: this.playerCell.column + deltaColumn,
    row: this.playerCell.row + deltaRow,
  };
  if (!isWalkable(this.map, next.column, next.row)) {
    this.setExternalStatus('Mur : mouvement refusé.', '');
    return;
  }
  this.moving = true;
  const [x, y] = this.pixelPosition(next);
  this.moveTarget = { cell: next, x, y, startedAt: this.time.now };
  this.physics.moveTo(this.player, x, y, 220);
}

update() {
  if (!this.moveTarget) return;
  const distance = Phaser.Math.Distance.Between(
    this.player.x, this.player.y,
    this.moveTarget.x, this.moveTarget.y,
  );
  if (distance ≤ 2) this.finishMove();
}

finishMove() {
  const target = this.moveTarget;
  this.player.setVelocity(0, 0);
  this.player.body.reset(target.x, target.y);
  this.playerCell = target.cell;
  this.moveTarget = null;
  this.moving = false;
  this.moves += 1;
}
```

La vérification de grille donne un comportement déterministe et pédagogique : une destination `#` est refusée avant le déplacement. Le groupe `this.walls` et `this.physics.add.collider(this.player, this.walls)` ajoutent en plus une protection physique au niveau Phaser. Si la carte est modifiée plus tard, le joueur ne traverse donc pas un mur par accident.

Sortie obtenue

■ Sortie / exemple

```
position=(1,1)
input=ArrowRight
target=(2,1)
move=accepted
```

```
input=ArrowDown
target=(2,2)
move=blocked: wall
physics=static_wall_collision_enabled
```

Après le premier appui, le personnage avance d'une cellule. Le second appui est ignoré parce que la cellule demandée contient `#` ; le joueur reste à sa position précédente et le statut indique le blocage.

Commande de vérification

Terminal · Bash

```
node --check src/main.js
```

Dans le navigateur, appuyez rapidement vers un mur puis essayez une direction libre. Le compteur ne doit augmenter que pour les déplacements acceptés.

5. Ajouter la sortie, le recommencement et les contrôles tactiles

La victoire est un état explicite. Lorsque la nouvelle cellule correspond à `6`, la scène arrête les mouvements et met à jour un message accessible avec `role="status"`. Les boutons directionnels réutilisent la même méthode `move` que le clavier : il n'existe donc pas deux règles de collision à maintenir.

Code de l'étape

JavaScript

```
if (this.playerCell.column === this.goal.column
    && this.playerCell.row === this.goal.row) {
    this.won = true;
    this.setExternalStatus(`Sortie atteinte en ${this.moves} coups !`, 'won');
}

document.querySelectorAll('[data-move]').forEach((button) => {
    button.onclick = () => {
        const vectors = {
            up: [0, -1], down: [0, 1],
            left: [-1, 0], right: [1, 0],
        };
        this.move(...vectors[button.dataset.move]);
    };
});

document.querySelector('[data-reset]').onclick = () => this.scene.restart();
```

Le bouton de recommencement relance la scène complète. Il remet à zéro la position, le compteur, l'état de victoire et les objets graphiques sans recharger la page.

Sortie obtenue

■ Sortie / exemple

```
move=accepted count=41  
move=accepted count=48  
goal=reached  
status=Sortie atteinte en 48 coups !
```

Une fois la sortie atteinte, les nouveaux appuis n'ont plus d'effet. Le bouton « Recommencer » ramène le joueur au départ et réinitialise le statut.

Commande de vérification

Terminal · Bash

```
curl -s http://localhost:8080/ | grep -q 'data-move="up"'  
curl -s http://localhost:8080/ | grep -q 'data-demo-status'
```

Ces contrôles vérifient que la page expose bien les contrôles utilisés par `bindControls`. Pour l'accessibilité, testez aussi la navigation avec la touche `Tab` et observez le message de statut après une victoire.

6. Lancer et tester le jeu autonome

Depuis le dossier qui contient `index.html`, démarrez un serveur HTTP local :

Commande de lancement

Terminal · Bash

```
python3 -m http.server 8080
```

Ouvrez ensuite <http://localhost:8080/> dans un navigateur. Le fichier ne doit pas être ouvert directement avec `file://`, car Phaser et le navigateur doivent pouvoir charger les scripts et les sprites avec leurs chemins relatifs.

Sortie obtenue

```
Sortie / exemple

url=http://localhost:8080/
controls=keyboard,touch
network=Phaser CDN only
assets=Kenney Tiny Dungeon local
status=Jeu prêt
```

Le jeu est contenu dans son propre dossier : `index.html` charge `./src/main.js`, `main.js` ne dépend d'aucune route externe et les sprites sont lus depuis `./assets/kenney-tiny-dungeon.png`. Seule la bibliothèque Phaser vient du CDN ; le niveau, les règles et les ressources graphiques sont fournis avec l'archive.

Commande de vérification

```
Terminal · Bash

curl -I http://localhost:8080/
curl -I http://localhost:8080/src/main.js
curl -I http://localhost:8080/assets/kenney-tiny-dungeon.png
node --check src/main.js
```

Les trois ressources doivent répondre avec un statut `200`, et la vérification Node doit se terminer sans erreur. Dans le navigateur, contrôlez ensuite le parcours de découverte, passez au niveau complexe et terminez une partie.

Dépannage et diagnostics

Le canvas reste vide — ouvrez la console du navigateur et vérifiez que `phaser.min.js` est accessible. Le jeu utilise un CDN ; un bloqueur réseau ou une politique de sécurité trop stricte peut empêcher ce chargement. Pour une exécution sans dépendance distante, téléchargez Phaser dans un dossier local et remplacez le `src` du script.

Les touches ne répondent pas — cliquez une fois dans la page puis utilisez les flèches ou `WASD`. Sur mobile, utilisez les quatre boutons. Vérifiez aussi que le script `src/main.js` est servi avec un statut `200` et que le navigateur a bien terminé le chargement de Phaser.

Le joueur traverse un mur — contrôlez que toutes les lignes de `LEVELS.simple.map` ou `LEVELS.complex.map` ont la même longueur et que le test compare bien le caractère cible à `#`. Vérifiez aussi que chaque sprite créé dans `this.walls` appelle `refreshBody()` après son redimensionnement. Une modification de carte doit être testée à la fois depuis le départ et depuis la sortie.

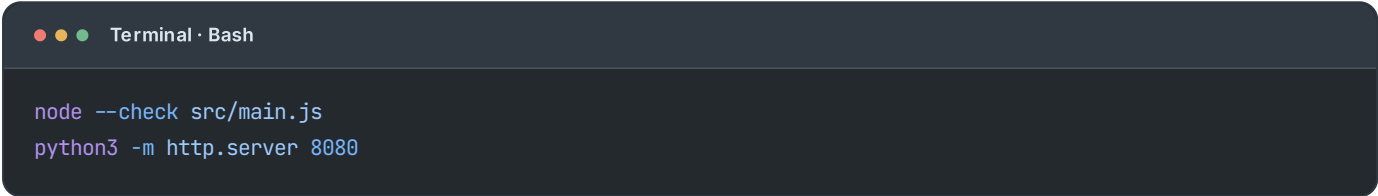
La sortie ne se déclenche pas — vérifiez la présence d'un seul `6` et comparez les coordonnées de `this.playerCell` à celles de `this.goal`. Le compteur peut s'incrémenter sans victoire si la cellule verte n'est pas atteignable dans la carte.

Le décor reste vide — contrôlez que `assets/kenney-tiny-dungeon.png` est servi avec un statut `200` et que le chemin du `spritesheet` est relatif à `index.html`. Si Phaser se charge mais si l'image échoue, le jeu ne peut pas découper les frames 16 × 16.

La page fonctionne mais pas les contrôles — vérifiez que vous utilisez bien l'adresse `http://localhost:8080/`, puis rechargez la page après le démarrage du serveur. Une ouverture directe avec `file://` peut empêcher le chargement de Phaser ou de la planche de sprites.

Vérification finale

Depuis le répertoire du jeu, exécutez :



```
Terminal · Bash  
  
node --check src/main.js  
python3 -m http.server 8080
```

Dans le navigateur, contrôlez le parcours suivant :

1. le plateau et la sortie sont visibles ;
2. les murs refusent un mouvement ;
3. les flèches et `WASD` déplacent le joueur ;
4. les boutons tactiles reproduisent les mêmes règles ;
5. la sortie affiche la victoire ;
6. « Recommencer » remet le jeu à son état initial.

L'archive du tutoriel contient le `README.md`, `ASSET-CREDITS.md`, la page HTML, `src/main.js` et la planche locale `assets/kenney-tiny-dungeon.png`. Elle ne contient ni `node_modules`, ni cache, ni résultat généré.

Résultats de référence

Avec la carte fournie et un parcours qui suit les couloirs sans appui inutile, le jeu affiche un message de victoire et un compteur de coups. Le nombre exact dépend des directions choisies : un appui contre un mur est ignoré et ne modifie pas le compteur.

■ Sortie / exemple

```
scene=MazeScene  
map=17x9 ou 25x17  
canvas=800x600  
goal=reachable  
status=Sortie atteinte en 48 coups !
```

Ces valeurs décrivent le scénario illustré par la figure de résultat sur le niveau de découverte. Le niveau complexe est plus long et nécessite davantage de décisions ; ces valeurs ne constituent pas une mesure de performance. Le jeu cherche ici à rendre le cycle d'une scène Phaser lisible et reproductible.

Limites et prolongements

La carte est écrite à la main et ne garantit pas automatiquement qu'un chemin existe. Une extension naturelle consiste à ajouter un parcours en largeur (BFS) qui valide la carte avant de démarrer la partie, puis à afficher la longueur du chemin minimal.

Le déplacement est par cellule et la caméra reste fixe. Pour une expérience plus riche, vous pouvez ajouter plusieurs niveaux, un chronomètre, des pièges ou une génération procédurale de labyrinthes. Dans un projet GenAI, un modèle pourrait proposer une carte ou une description de niveau, mais le navigateur devrait toujours valider la taille, les symboles, la présence d'un départ et l'existence d'un chemin avant de l'afficher.

Enfin, le jeu charge Phaser depuis un CDN. Ce choix simplifie le tutoriel ; une application distribuée avec une politique stricte de dépendances devra verrouiller, héberger et vérifier le fichier JavaScript localement.

Références

- [Phaser — documentation officielle des scènes](#)
- [Phaser — documentation officielle de la physique Arcade](#)
- [Phaser — guide officiel de démarrage](#)