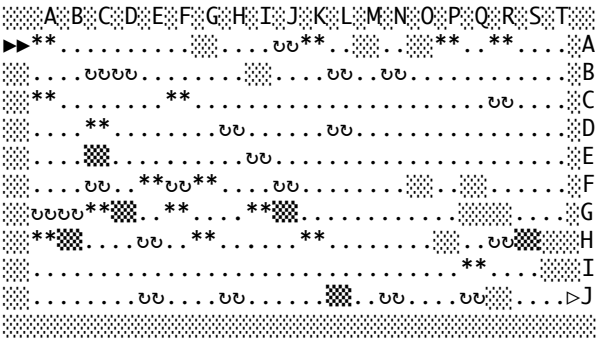
Université de Bourgogne
Année universitaire 2022-2023
Info 2A — programmation objet
Descriptif du projet

1. Description générale

L'objet de ce projet est de modéliser un jeu composé d'un plateau composé d'une grille de cases que le joueur doit parcourir d'une case de départ à une case d'arrivée situées de part et d'autre du plateau.
À chaque tour, le joueur peut se déplacer de la case où il est positionné à une case de son choix immédiatement voisine horizontalement ou verticalement.
Les cases peuvent contenir des éléments qui influent sur la progression du joueur ou ses capacités (voir les jeux d'essais de l'annexe). Par exemple :

- ✓ Si le joueur entre dans une case qui contient une mine, la mine explose et il perd.
- ✓ Une case de type passage est liée à une autre case du plateau appelée destination. Quand le joueur entre dans une telle case, il est téléporté à la case destination, ce qui peut le rapprocher ou l'éloigner de son objectif.
- ✓ Il ne peut pas entrer dans une case contenant un obstacle.
- ✓ Si le joueur entre dans une case qui stocke des pierres, il les récupère.



Dans l'exemple ci-dessus, l'entrée est symbolisée par ▶▶*, la sortie par* ▷▷*, une case contenant des mines (respectivement un passage, un obstacle et des pierres) par* ** *(respectivement* ʊʊ*,* ▒*,* ▓*). Vous pouvez adopter d'autres conventions de représentation.*

Au départ, le contenu des cases n'est pas visible. Le joueur avance donc a priori à l'aveuglette. Mais, pour ne pas prendre trop de risque, il dispose d'un lot de pierres qu'il peut lancer une par une dans une direction qu'il envisage de prendre. L'interaction d'une pierre avec le contenu des cases donne des indices sur leur contenu. Par exemple :

- ✓ Une pierre qui passe dans une case vide est propagée à la case voisine dans la même direction sans que la case soit démasquée. Aucune information n'est donnée.
- ✓ Si la pierre atteint une mine, celle-ci explose, la pierre est perdue mais la case devient visible et peut désormais être parcourue sans danger par le joueur et d'autres pierres (Elle se comporte comme une case vide).
- ✓ En entrant dans une case de type passage, la pierre est téléportée comme l'aurait été le joueur et continue son chemin dans sa direction d'origine. Elle peut alors atteindre le bord du plateau à un endroit non prévisible.
- ✓ Quand une pierre atteint un obstacle, elle est perdue et l'indication de l'événement (rencontre de l'obstacle) est donnée au joueur sans indiquer l'emplacement.
- ✓ Une pierre qui arrive dans une case qui stocke des pierres est perdue temporairement pour le joueur mais s'ajoute aux pierres de la case… qui pourront éventuellement être

récupérées par la suite. Ici aussi l'information de l'événement est donnée, mais sans indiquer d'emplacement précis.

- ✓ En sortant du plateau par un bord, la pierre est perdue mais l'emplacement de sortie est indiqué.

2. Précisions

Paramètres du jeu

Ce sont :

- ✓ Le crédit initial du joueur en nombre de pierres,
- ✓ Les nombres de lignes nbLig et de colonnes nbCol du plateau,
- ✓ Le pourcentage de cases de chaque catégorie (mine, passage, obstacle, pierres) par rapport au nombre de cases du plateau,
- ✓ Le nombre maximal de pierres qu'il peut y avoir dans les cases concernées.

Au début du jeu, lors du remplissage du plateau, la machine doit tirer au hasard l'emplacement des cases des différentes catégories, la destination de chaque passage, et le nombre de pierres dans chacune des cases concernées (entre 1 et le nombre maximal).

Tour du joueur

À son tour, le joueur effectue les actions suivantes :

1. Sélection d'une direction (gauche, droite, haut, bas…),
2. Choix de lancer une pierre ou d'avancer d'un pas dans la direction choisie.

La case sur laquelle arrive le joueur est rendue visible, ce qui permet de visualiser le chemin parcouru. Le joueur gagne quand il parvient à la sortie sans avoir explosé sur une mine.

3. Propositions et conseils de modélisation

Le programme doit au moins comporter une classe `Plateau` et une classe `Case`. Le plateau référence ses différentes cases via un tableau et peut contenir une méthode d'en-tête

```
Case get(int nLig,int nCol)
```

qui restitue la case située à la ligne de rang `nLig` et à la colonne de rang `nCol`. Chaque case du plateau a pour propriétés ses numéros de ligne et de colonne dans le plateau ainsi que le plateau lui-même (Elle contient une méthode `getPlateau()`). Une case peut ainsi accéder indirectement à ses cases voisines dans une direction donnée.
Pour la suite, vous n'êtes pas tenu-e-s de suivre les recommandations données. Néanmoins, vos programmes devront respecter le paradigme Objet, notamment en utilisant massivement le principe de délégation.

Gestion des bords du plateau

Pour éviter les erreurs de sortie du plateau, au lieu de tester chaque fois `nLig` et `nCol` avant d'utiliser la méthode `get` vue plus haut, il est possible d'intégrer les tests dans le code de cette méthode. Si `nLig` et `nCol` renvoient à une position externe au plateau, la méthode `get(nLig,nCol)` peut par exemple restituer `null`.

En fait, il est mieux de noter qu'un débordement du plateau n'est possible que quand une pierre ou un joueur se déplace case par case et atteint un bord (Voir point suivant). Le débordement ne se fait alors que d'une unité. Il est alors possible d'environner les cases internes du plateau par des cases de bordure dans lesquelles les déplacements sont impossibles (Pas de débordement possible quand une telle case a été atteinte). La case de bordure en

haut à gauche (respectivement en bas à droite) aurait alors dans le plateau les coordonnées (0,0) (respectivement (nbLig+1,nbCol+1)). Un autre avantage de ce principe est que les cases internes du plateau auront des coordonnées « naturelles » entre 1 et nbLig (respectivement entre 1 et nbCol).

Déplacement des pierres et des joueurs

Dans la logique de représentation objet, la progression d'une pierre peut être déléguée aux cases, ce qui signifie que les cases « se passent » la pierre à faire progresser tant qu'une case particulière ne l'arrête pas. Pour ce faire, chaque instance de case contient une méthode `propage` — ou équivalent — qui, à partir d'une direction donnée, effectue ce qui suit :

- ✓ La méthode `propage` d'une case vide avec l'information de direction donnée exécute la méthode `propage` de la case voisine dans cette direction (cela revient à transmettre la pierre à la case voisine sans changer sa direction).
- ✓ La méthode `propage` d'une case contenant une mine active (qui n'a pas explosé) fait exploser la mine (elle devient inactive et la case devient visible). Un affichage indique alors que la mine a explosé et que la pierre est perdue. Si la mine est inactive (elle a déjà explosé), la pierre est propagée à la case voisine de la même manière que pour les cases vides.
- ✓ La méthode `propage` d'une case de type passage exécute la méthode `propage` de la case de destination en conservant la même direction.
- ✓ La méthode `propage` d'une case qui stocke des pierres ajoute la pierre au stock et la fait perdre au joueur.
- ✓ La méthode `propage` d'un obstacle décrémente le nombre de pierres du joueur et signale la rencontre d'un obstacle.
- ✓ Si, comme vu plus haut, les bords du plateau ont été représentés par des cases, la méthode `propage` d'un bord effectue simplement l'affichage de ses coordonnées et décrémente le nombre de pierres du joueur.

À noter qu'une stratégie similaire peut être utilisée pour gérer le déplacement du joueur. Il est possible de créer une méthode `arrive` qui gère l'arrivée d'un joueur dans une case et dont le code se décline suivant les types de cases.

Structure des classes

Les cases effectuent des traitements différents suivant leur type et les informations qu'elles contiennent ne sont pas toutes les mêmes (la case de type « stock de pierres » contient le nombre de ses pierres, la mine est active ou pas, le passage contient la référence à une case de destination…).

Cela suggère fortement que les différents types de cases soient des sous-classes de la classe `Case` (classe `Mine`, `Pierres`, `Passage`, `Vide`…).
Chacune contient alors sa propre version de la méthode `propage` et de la méthode `arrive`.
La classe `Case` peut contenir l'action standard qui est effectuée pour l'essentiel des cases. Pour la méthode `propage`, cela peut être le passage de relais à la case voisine suivant la direction donnée, même si ce relais n'est pas effectué pour les mines, les passages et les bords. Les cases spécifiques qui auront besoin de cette action standard peuvent l'appeler par l'intermédiaire de `super.propage(…)`.
Pour ce qui concerne le déplacement du joueur, le comportement standard est la révélation du contenu des cases sur lesquelles passe le joueur.

À noter qu'il peut être utile de considérer les cases de départ et de sortie comme d'autres sous-classes de la classe `Case`.
La notion de direction est souvent utilisée, il peut être intéressant d'en faire une classe.

Affichage du plateau

À chaque type de case peut être associé un symbole conventionnel (Comme vu pour l'exemple) qui peut être utilisé pour l'affichage de la case via `toString`. Le `toString` du plateau est constitué des appels aux `toString` des cases.

Génération du plateau

C'est une partie complexe parce qu'il faut générer aléatoirement des cases de types différents en tenant compte du nombre de cases à générer par type. Une classe outil vous sera fournie pour vous aider à le faire.

Jeu

Le Jeu peut être représenté par une classe à une seule instance qui contient ou référence les paramètres globaux, le plateau, je joueur…
Il contient une méthode `joue` qui permet au joueur de choisir une direction, puis de choisir de se déplacer d'une case ou de lancer une pierre, cela jusqu'à ce qu'il ait gagné ou perdu.

4. Modalités du travail à effectuer

Le travail est prévu pour être effectué en binôme mais, avec l'accord de l'enseignant de TP, vous pourrez éventuellement le faire seul. Il doit être original, c'est-à-dire que vous pouvez échanger des idées entre monômes et/ou binômes, mais la réalisation finale présentée doit être spécifique et maîtrisée (Vous devrez pouvoir répondre à des questions sur le fonctionnement et le code). De même, le rapport de chaque binôme ou éventuellement monôme doit être original.

4.1. Réalisation

Vous devez réaliser un programme en java qui modélise les éléments décrits par l'énoncé. Créez le programme progressivement en le testant systématiquement à chaque étape. Vous pouvez par exemple commencer par un programme comportant uniquement des cases vides et, si vous en avez fait le choix, des cases de bordure. Une fois que vous aurez testé les méthodes `propage` et `arrive` dans ces cas simples.

Si TOUT ce qui est demandé fonctionne, vous pouvez ajouter d'autres fonctionnalités et des options ; par exemple, complexifier certaines cases (Une obstacle peut avoir une résistance qui décroit à force d'être frappé par des pierres), ajouter d'autres types de cases (Par exemple, une case gouffre, une case rebond…), ajouter d'autres joueurs avec éventuellement leurs propres entrée et sortie, proposer au joueur de choisir toutes les caractéristiques du jeu, de rejouer…

En bonus, vous pourrez étudier les cas de blocages, par exemple quand l'entrée ou la sortie sont bloquées par des obstacles ou des passages, les boucles infinies : un passage peut mener à lui-même ou à un passage qui, selon la direction choisie ramène au passage de départ.
L'inventaire de ces situations peut être fait dans le rapport et, éventuellement, donner lieu à une implémentation. Mais cette partie est optionnelle ; elle ne peut qu'ajouter des points.

Les classes devront comporter les constructeurs, accesseurs, modificateurs d'accès nécessaires ainsi, au besoin, que les méthodes `equals` et `toString` (le `toString` est utile pour les affichages du plateau, des cases, des joueurs…). Sauf cas particulier, les attributs ne devront être accessibles que par leurs accesseurs.

4.2. Démonstration

La démonstration se situera au cours de la semaine du 2 au 5 mai dans un créneau de 20 minutes, comportant votre présentation et les questions du/des enseignant/e/s.
Elle est destinée à présenter les principales fonctionnalités de votre réalisation.

Il est important de faire attention aux points suivants :

- ✓ Respectez le temps imparti.
- ✓ Testez au préalable ce que vous voulez présenter (n'improvisez pas) ; les problèmes sous-jacents apparaissent souvent pendant les démos.
- ✓ Structurez votre démonstration autour de jeux d'essais, c'est-à-dire d'exemples déroulés qui montrent bien ce que fait votre programme. Pour ce faire, vous pouvez avoir un mode particulier de votre application qui montre les cases (c'est très facile à faire).
- ✓ Ne vous perdez pas dans les détails techniques en parlant de votre programme. L'enseignant vous posera des questions techniques s'il le juge utile.
- ✓ Tous les participants au projet doivent prendre la parole de manière à peu près égale. Vous devez être le plus naturel possible lors des passages de parole et parler distinctement.
- ✓ Tous les participants doivent pouvoir répondre sur les fonctionnalités de tel ou tel élément du programme, même s'il n'ont pas réalisé cette partie.

En résumé, une démonstration doit être soigneusement préparée.

5. Échéances et conseils

Semaine du 2 au 5 mai (la deuxième semaine après les vacances de Pâques)
Démonstration du projet dans un des créneaux qui seront proposés (vous devrez vous inscrire). Vous devez fournir un fichier compressé (extension zip : format assez standard) à votre/vos nom/s contenant :

- ✓ le rapport au format pdf.
- ✓ Le code du projet,
- ✓ les autres documents que vous souhaitez communiquer (jeux d'essais).

Les modalités précises de la réservation des créneaux de passage et de la remise des documents vous seront précisées ultérieurement.

Il est conseillé de ne dépasser les fonctionnalités de base que si ces dernières sont bien réalisées et que vous êtes à jour dans les autres matières.
Rappel : il est (vivement) conseillé d'obtenir avant tout une version qui fonctionne, même avec des fonctionnalités plus restreintes que celles qui sont spécifiées par l'énoncé.

Annexe

Premier cas

```
Nom du joueur : Jo

▒▒▒▒A▒B▒C▒D▒E▒▒
▶♜          ▒▒A
▒▒          ▒▒B
▒▒          ▒▒C
▒▒          ▒▒D
▒▒          ▷▒E
▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒
```

→ Le joueur est symbolisé par ♜. Chaque position est composée de deux symboles identiques quand la position n'est pas occupée par le joueur, et par le symbole de la case suivi de celui du joueur dans le cas contraire. Vous pouvez adopter d'autres conventions.

```
Le joueur Jo a 4 pierres

Avancer ou lancer une pierre (A/P) : p
```

→ choix de lancer une pierre

```
Choisis une direction (g|d|h|b) : d
```

→ choix de la lancer à droite

```
Bord ligne A droit atteint. Pierre perdue
```

→ la pierre atteint la bordure de la première ligne, ce qui garantit qu'il n'y a rien de dangereux jusque là et semble même indiquer qu'il n'y a que des cases vides dans la première ligne (Ce n'est pas sûr, il pourrait y avoir des passages successifs ; voir plus loin).

```
▒▒▒▒A▒B▒C▒D▒E▒▒
▶♜          ▒▒A
▒▒          ▒▒B
▒▒          ▒▒C
▒▒          ▒▒D
▒▒          ▷▒E
▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒

Le joueur Jo a 3 pierres

Avancer ou lancer une pierre (A/P) : a
Choisis une direction (g|d|h|b) : d
```

→ Grâce au lancer de pierre précédent, on sait que le fait de partir vers la droite n'est pas dangereux ; c'est donc ce que fait le joueur.

```
▒▒▒▒A▒B▒C▒D▒E▒▒
▶▶.♜        ▒▒A
▒▒          ▒▒B
▒▒          ▒▒C
▒▒          ▒▒D
▒▒          ▷▒E
▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒

Le joueur Jo a 3 pierres

Avancer ou lancer une pierre (A/P) : a
Choisis une direction (g|d|h|b) : d

▒▒▒▒A▒B▒C▒D▒E▒▒
▶▶...♜      ▒▒A
▒▒          ▒▒B
▒▒          ▒▒C
▒▒          ▒▒D
▒▒          ▷▒E
▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒

Le joueur Jo a 3 pierres
Avancer ou lancer une pierre (A/P) : a
```

```
Choisis une direction (g|d|h|b) : d

▒▒▒▒A▒B▒C▒D▒E▒▒
▶▶.....♜     ▒A
▒▒           ▒B
▒▒           ▒C
▒▒           ▒D
▒▒           ▷E
▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒

Le joueur Jo a 3 pierres

Avancer ou lancer une pierre (A/P) : a
Choisis une direction (g|d|h|b) : d

▒▒▒▒A▒B▒C▒D▒E▒▒
▶▶.......♜   ▒A
▒▒           ▒B
▒▒           ▒C
▒▒           ▒D
▒▒           ▷E
▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒

Le joueur Jo a 3 pierres

Avancer ou lancer une pierre (A/P) : a
Choisis une direction (g|d|h|b) : d

▒▒▒▒A▒B▒C▒D▒E▒▒
▶▶.........♜▒A
▒▒           ▒B
▒▒           ▒C
▒▒           ▒D
▒▒           ▷E
▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒
```

→ L'hypothèse effectuée après le lancer de pierre était bonne : il n'y avait rien dans la première ligne.

```
Le joueur Jo a 3 pierres

Avancer ou lancer une pierre (A/P) : p
Choisis une direction (g|d|h|b) : b

Pierre arrêtée par un obstacle. Elle est perdue
```

→ Test de la colonne en dessous de la position du joueur. La pierre a été arrêtée par un obstacle. On ne sait pas où mais ce n'est pas dangereux ; il est donc possible d'aller dans cette direction.

```
▒▒▒▒A▒B▒C▒D▒E▒▒
▶▶.........♜▒A
▒▒           ▒B
▒▒           ▒C
▒▒           ▒D
▒▒           ▷E
▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒

Le joueur Jo a 2 pierres

Avancer ou lancer une pierre (A/P) : a
Choisis une direction (g|d|h|b) : b


▒▒▒▒A▒B▒C▒D▒E▒▒
▶▶..........▒A
▒▒         .♜▒B
▒▒           ▒C
▒▒           ▒D
▒▒           ▷E
▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒

Le joueur Jo a 2 pierres
```

```
Avancer ou lancer une pierre (A/P) : a
Choisis une direction (g|d|h|b) : b

Obstacle : accès à la case impossible !
```
→ L'obstacle a été atteint. Il faut le contourner.

```
▒▒▒A B C D E▒▒
▶▶.........▒A
▒▒        .♜▒B
▒▒        ▒▒▒C
▒▒          ▒D
▒▒          ▷E
▒▒▒▒▒▒▒▒▒▒▒▒▒▒
```

```
Le joueur Jo a 2 pierres

Avancer ou lancer une pierre (A/P) : p
Choisis une direction (g|d|h|b) : g
```
→ Lancement d'une pierre à gauche pour vérifier si la ligne est sûre.

```
Bord ligne B gauche atteint. Pierre perdue
```
→ Pas de danger effectivement. Dans la même logique que lors du premier lancer de pierre, la ligne semble vide.

```
▒▒▒A B C D E▒▒
▶▶.........▒A
▒▒        .♜▒B
▒▒        ▒▒▒C
▒▒          ▒D
▒▒          ▷E
▒▒▒▒▒▒▒▒▒▒▒▒▒▒

Le joueur Jo a 1 pierres

Avancer ou lancer une pierre (A/P) : a
Choisis une direction (g|d|h|b) : g
```

```
▒▒▒A B C D E▒▒
▶▶.........▒A
▒▒  .♜  ↻↻..▒B
▒▒    ↻↻  ▒▒▒C
▒▒          ▒D
▒▒          ▷E
▒▒▒▒▒▒▒▒▒▒▒▒▒▒
```

→ Contrairement à l'hypothèse faite ci-dessus, la ligne n'était pas vide. Elle contient un passage colonne B, qui conduit à un autre passage ligne C colonne C, lequel envoie finalement le joueur ligne B colonne B. Le joueur a donc été téléporté deux fois.

```
Le joueur Jo a 1 pierres

Avancer ou lancer une pierre (A/P) : p
Choisis une direction (g|d|h|b) : b

Bord bas colonne B atteint. Pierre perdue
```
→ De la même manière que précédemment, la colonne semble vide… mais ce n'est pas sûr.

```
▒▒▒A B C D E▒▒
▶▶.........▒A
▒▒  .♜  ↻↻..▒B
▒▒    ↻↻  ▒▒▒C
▒▒          ▒D
▒▒          ▷E
▒▒▒▒▒▒▒▒▒▒▒▒▒▒

Le joueur Jo a 0 pierres

Plus de pierres disponibles ; Jo avance.
```

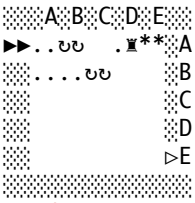
```
Choisis une direction (g|d|h|b) : b

▒▒▒▒A▒B▒C▒D▒E▒▒
▶▶.........▒▒A
▒▒  ..  ↻↻..▒▒B
▒▒  .♜↻↻  ▒▒▒▒C
▒▒          ▒▒D
▒▒          ▷E
▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒

Le joueur Jo a 0 pierres

Plus de pierres disponibles ; Jo avance.
Choisis une direction (g|d|h|b) : b

▒▒▒▒A▒B▒C▒D▒E▒▒
▶▶.........▒▒A
▒▒  ..  ↻↻..▒▒B
▒▒  ..↻↻  ▒▒▒▒C
▒▒  .♜      ▒▒D
▒▒          ▷E
▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒

Le joueur Jo a 0 pierres

Plus de pierres disponibles ; Jo avance.
Choisis une direction (g|d|h|b) : b

▒▒▒▒A▒B▒C▒D▒E▒▒
▶▶.........▒▒A
▒▒  ..  ↻↻..▒▒B
▒▒  ..↻↻  ▒▒▒▒C
▒▒  ..      ▒▒D
▒▒  .♜      ▷E
▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒

Le joueur Jo a 0 pierres

Plus de pierres disponibles ; Jo avance.
Choisis une direction (g|d|h|b) : d
```
→ Il n'est plus possible d'avancer sans risque. On tente d'aller dans la direction du but sans pouvoir vérifier au préalable le chemin grâce à un lancer de pierre.

```
▒▒▒▒A▒B▒C▒D▒E▒▒
▶▶.........▒▒A
▒▒  ..  ↻↻..▒▒B
▒▒  ..↻↻  ▒▒▒▒C
▒▒  ..      ▒▒D
▒▒  ...♜    ▷E
▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒

Le joueur Jo a 0 pierres

Plus de pierres disponibles ; Jo avance.
Choisis une direction (g|d|h|b) : d

Boum ! Jo perd !
Partie perdue !
▒▒▒▒A▒B▒C▒D▒E▒▒
▶▶.........▒▒A
▒▒......↻↻..▒▒B
▒▒....↻↻..▒▒▒▒C
▒▒**.......▒▒D
▒▒......*♜..▷E
▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒
```
→ Une mine a été rencontrée. Le jeu se termine sur la perte du joueur.

Deuxième cas

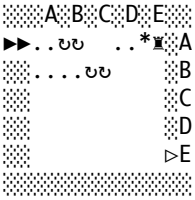
```
Nom du joueur : Jo

▒▒▒A B C D E▒▒
▶♜          ▒A
▒▒          ▒B
▒▒          ▒C
▒▒          ▒D
▒▒          ▷E
▒▒▒▒▒▒▒▒▒▒▒▒▒▒

Le joueur Jo a 4 pierres

Avancer ou lancer une pierre (A/P) : p
Choisis une direction (g|d|h|b) : d

Boum ! Pierre perdue
```
→ Une mine a été rencontrée sur le chemin mais elle a été désactivée par la pierre. Le parcours jusqu'à la case ligne A colonne E est désormais sûr.
```
▒▒▒A B C D E▒▒
▶♜        **A
▒▒          ▒B
▒▒          ▒C
▒▒          ▒D
▒▒          ▷E
▒▒▒▒▒▒▒▒▒▒▒▒▒▒

Le joueur Jo a 3 pierres

Avancer ou lancer une pierre (A/P) : a
Choisis une direction (g|d|h|b) : d

▒▒▒A B C D E▒▒
▶▶.♜       **A
▒▒          ▒B
▒▒          ▒C
▒▒          ▒D
▒▒          ▷E
▒▒▒▒▒▒▒▒▒▒▒▒▒▒

Le joueur Jo a 3 pierres

Avancer ou lancer une pierre (A/P) : a
Choisis une direction (g|d|h|b) : d

▒▒▒A B C D E▒▒
▶▶..ʊʊ     **A
▒▒.♜        ▒B
▒▒          ▒C
▒▒          ▒D
▒▒          ▷E
▒▒▒▒▒▒▒▒▒▒▒▒▒▒
```
→ Il y a ici aussi un passage « parasite » mais, comme la pierre, en continuant à aller sur la droite, avait atteint la bordure droite de la ligne A, il existe forcément au moins un autre passage qui remettra le joueur sur la ligne A. De plus, on sait que le trajet n'est pas dangereux.
```
Le joueur Jo a 3 pierres

Avancer ou lancer une pierre (A/P) : a
Choisis une direction (g|d|h|b) : d

▒▒▒A B C D E▒▒
▶▶..ʊʊ     **A
▒▒...♜      ▒B
▒▒          ▒C
▒▒          ▒D
▒▒          ▷E
▒▒▒▒▒▒▒▒▒▒▒▒▒▒
```

```
Le joueur Jo a 3 pierres

Avancer ou lancer une pierre (A/P) : a
Choisis une direction (g|d|h|b) : d
```



→ L'autre passage attendu est à la colonne C. Il téléporte le joueur à la colonne D de la ligne A.

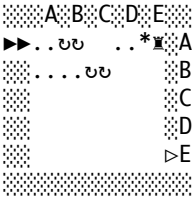
```
Le joueur Jo a 3 pierres

Avancer ou lancer une pierre (A/P) : a
Choisis une direction (g|d|h|b) : d
```



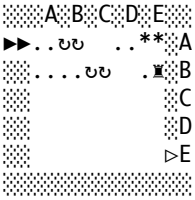
→ Le joueur a accédé à la case contenant la mine, mais elle avait été désactivée par la pierre et, donc, n'est plus dangereuse.

```
Le joueur Jo a 3 pierres

Avancer ou lancer une pierre (A/P) : p
Choisis une direction (g|d|h|b) : b

Bord bas colonne E atteint. Pierre perdue
```

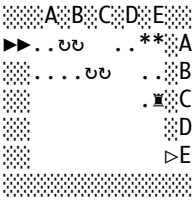
→ On est sûr de pouvoir gagner dès lors que la case ligne E colonne E, qui jouxte la sortie, peut être atteinte sans risque.




```
Le joueur Jo a 2 pierres

Avancer ou lancer une pierre (A/P) : a
Choisis une direction (g|d|h|b) : b
```




```
Le joueur Jo a 2 pierres

Avancer ou lancer une pierre (A/P) : a
Choisis une direction (g|d|h|b) : b
```



```
Le joueur Jo a 2 pierres

Avancer ou lancer une pierre (A/P) : a
Choisis une direction (g|d|h|b) : b

▒▒▒▒A▒B▒C▒D▒E▒▒
▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒
▶▶..ʚʚ  ..**▒A
▒▒....ʚʚ  ..▒B
▒▒▒▒▒▒▒▒..▒C
▒▒▒▒▒▒▒▒.♜▒D
▒▒▒▒▒▒▒▒▒▷E
▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒

Le joueur Jo a 2 pierres

Avancer ou lancer une pierre (A/P) : a
Choisis une direction (g|d|h|b) : b

▒▒▒▒A▒B▒C▒D▒E▒▒
▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒
▶▶..ʚʚ  ..**▒A
▒▒....ʚʚ  ..▒B
▒▒▒▒▒▒▒▒..▒C
▒▒▒▒▒▒▒▒..▒D
▒▒▒▒▒▒▒▒.♜▷E
▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒

Le joueur Jo a 2 pierres

Avancer ou lancer une pierre (A/P) : a
Choisis une direction (g|d|h|b) : d

Jo est arrivé à bon port.
▒▒▒▒A▒B▒C▒D▒E▒▒
▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒
▶▶..ʚʚ**..**▒A
▒▒....ʚʚ....▒B
▒▒▒▒........▒C
▒▒..........▒D
▒▒..........▷E
▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒
```

→ Le joueur a gagné.