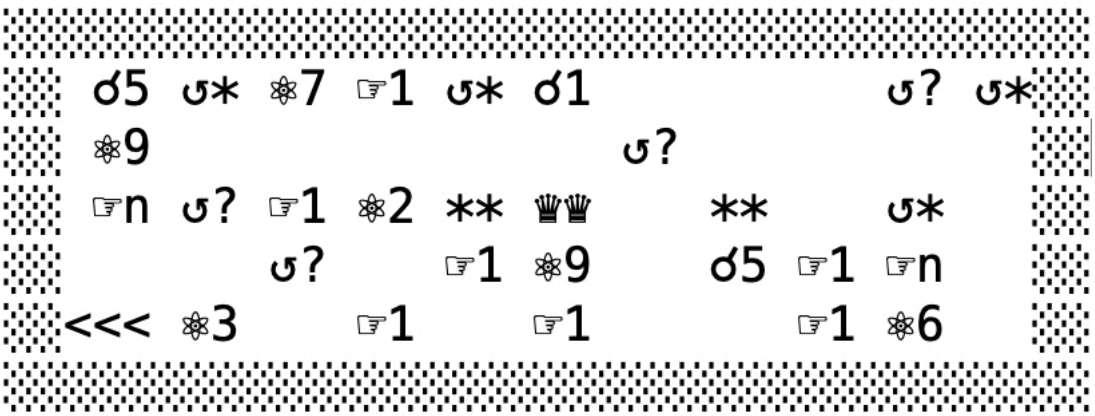
Université de Bourgogne
Année universitaire 2023-2024
Info 2A — programmation objet
Descriptif du projet

1. Description générale

L'objet de ce projet est de modéliser un jeu constitué d'un plateau, lui-même composé d'une grille de salles qui peuvent contenir un objet qui intéresse le joueur. Les objets peuvent être :

- ✓ Une mine… dangereuse pour le joueur,
- ✓ Une caisse de grenades pouvant contenir aléatoirement de 1 à 10 grenades,
- ✓ Une réserve d'énergie de 1 à 20 unités d'énergie,
- ✓ Un outil (voir plus loin).



Exemple de plateau au début du jeu dont *les salles ont été rendues visibles. Normalement, seule la salle centrale — qui contient le joueur — est visible ; on ne voit après que les salles dont un des murs a été détruit. Le joueur est symbolisé par* ♛♛*, la sortie par* <<<*, une caisse de grenades par* ძn *(n étant le nombre de grenades), une réserve d'énergie par* ⚛n*, une mine par* **, *un détecteur de mines par* ↺*… *(Ce n'est qu'un exemple de représentation possible).*

Le joueur doit parcourir les salles du plateau jusqu'à atteindre la sortie située dans un des angles — mais son emplacement exact n'est pas connu au départ. Initialement, les salles sont masquées et le joueur est emmuré dans la salle centrale. Il se déplace de salle en salle en perforant au préalable les murs[1] avec des grenades (Une par mur)[2]. Pour ce faire, le joueur dispose d'une réserve de grenades. À noter que l'utilisation d'une grenade sur un mur d'une salle qui contient une mine la fait exploser et tue le joueur ; le jeu est alors perdu. Le joueur possède aussi une batterie : une réserve d'énergie qui lui permet d'utiliser des outils s'il en a en sa possession. À chaque utilisation, un outil nécessite une quantité d'énergie qui lui est propre, et qui est prise sur la batterie. On distingue les outils suivants (Vous n'êtes pas obligés de tous les réaliser mais, si vous programmez bien dans la logique objet, l'ajout d'un nouveau type d'outil doit être simple) :

- ✓ un *scanner omnidirectionnel de courte portée* : il permet de détecter au travers des murs le nombre de salles contiguës contenant un objet. Son utilisation nécessite 1 unité d'énergie,

[1] Au départ, toutes les salles sont aveugles, fermées par des murs dans les directions haut, bas, gauche et droite. La perforation du mur de séparation entre deux salles permet définitivement d'accéder de l'une à l'autre.
[2] Chaque nouvelle salle dont un mur a été ouvert devient visible.

- ✓ un *scanner unidirectionnel de longue portée*, qui détecte à travers les murs la distance — en nombre de salles vides traversée — à laquelle se situe le premier objet — ou le mur du plateau — dans une direction donnée. L'estimation de la distance est approximative : elle est à 20% de la distance près[3]. L'utilisation de cet outil nécessite 2 unités d'énergie.
- ✓ un *détecteur de mines*, qui permet de connaître le nombre total de mines qui se situent dans les salles contiguës. Chaque utilisation de ce détecteur consomme 3 unités d'énergie.
- ✓ un *détecteur massique unidirectionnel* qui permet de connaître approximativement (à 10% près) le nombre d'objets — tous types confondus — situés dans une direction donnée. Son utilisation nécessite 2 unités d'énergie.
- ✓ une *excavatrice* qui permet de creuser un mur sans faire exploser la mine qu'elle pourrait contenir. Mais la mine n'est pas désactivée et le joueur ne peut pas entrer dans la salle qui la contient sans la faire exploser. Son utilisation nécessite 8 unités d'énergie.
- ✓ une *unité de déminage*, qui désactive moyennant 2 unités d'énergie une mine située dans une salle contiguë accessible (le mur entre les salles doit déjà avoir été détruit).

Au départ, la réserve de grenades et la batterie du joueur contiennent respectivement 10 grenades et 20 unités d'énergie. Il dispose aussi d'un scanner unidirectionnel de longue portée et d'un détecteur de mines.

À chaque tour, le joueur peut effectuer les actions suivantes :

- ✓ lancer une grenade sur un mur dans les directions horizontales ou verticales. S'il n'y a pas de mur, un affichage indique que la grenade est perdue. Dans le cas contraire, on considérera que la destruction du mur par l'explosion de la grenade provoque une dépression qui aspire le joueur dans la nouvelle salle ouverte… au risque de rencontrer une mine[4].
- ✓ se déplacer dans une salle accessible (Cela revient à se déplacer dans la direction de cette salle). Si la nouvelle salle contient une caisse de grenades, le joueur récupère la partie de son contenu qu'il souhaite (Il peut en laisser). Le principe est similaire s'il est confronté à une réserve d'énergie ; il charge sa batterie avec tout ou partie de l'énergie de la réserve. Si la salle atteinte contient un objet, celui-ci est ajouté à la liste d'outils du joueur — s'il n'en dispose pas déjà.
- ✓ utiliser un outil parmi ceux dont il dispose si sa réserve d'énergie est suffisante. L'énergie du joueur est alors diminuée de la quantité d'énergie nécessitée par l'utilisation de l'outil.

2. Précisions

Paramètres des outils et du joueur

À noter tout d'abord que les coûts d'utilisation des outils et leur précision — scanner unidirectionnel et détecteur massique — qui figurent dans l'énoncé ne sont qu'indicatifs ; vous pouvez faire d'autres choix et même éventuellement faire choisir ces paramètres par l'utilisateur. Il en est de même pour le nombre de grenades et la batterie du joueur ainsi que le nombre de grenades maximal dans chaque caisse et l'énergie dans chaque réserve.

Paramètres du jeu

Ce sont :

- ✓ Les tailles horizontale et verticale de la grille — qui doivent être impaires pour qu'il y ait une salle centrale (salle de départ du joueur),

---

[3] Autrement dit, si le nombre réel de salles traversées par le joueur est 10, le scanner peut aussi bien indiquer 8, 9, 10, 11, 12 (Entre 10-2 et 10+2). Si le nombre réel n est de 5, 6 ou 7, l'erreur arrondie est approximée à 1. L'indication est donc dans l'intervalle [n-1,n+1].

[4] Je sais, c'est très douteux selon les lois physiques mais c'est beaucoup plus facile à programmer ainsi<

- ✓ Le nombre de mines,
- ✓ Le nombre de caisses de grenades,
- ✓ Le nombre d'outils supplémentaires de chaque catégorie qui doivent être disposés dans les salles du plateau.

Au début du jeu, lors du remplissage du plateau, la machine doit tirer au hasard l'emplacement des mines, des outils de chaque catégorie et la salle permettant la sortie du jeu (Une des salles de coin). Elle doit aussi placer les caisses de grenades (Nombre de grenades par caisse constant ou tiré au hasard) et les réserves d'énergie (Idem pour la quantité d'unités d'énergie par réserve, constant ou tiré au hasard). Le nombre des objets de chaque catégorie peut être spécifié au départ en proportion du nombre de cases du plateau (3% de mines, 5% de caisses de grenades…).

Tour du joueur

À chaque tour, le joueur effectue une action parmi celles possibles en fonction des outils dont il dispose et de l'énergie qui lui reste. Cela peut être géré par un menu remis à jour en fonction des outils, de l'énergie et des grenades disponibles.

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

3. 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.

3.1. Réalisation

Vous devez réaliser un programme en java qui modélise les éléments décrits par l'énoncé (pas forcément tous). Créez le programme progressivement en le testant systématiquement à chaque étape. Vous pouvez par exemple commencer par un programme comportant uniquement des salles vides, ne pas prendre en compte l'énergie, ajouter un à un des outils en vérifiant leur fonctionnement avant de passer à l'outil suivant.

Si TOUT ce qui est demandé fonctionne, vous pouvez ajouter d'autres fonctionnalités et des options ; par exemple, créer d'autres outils ajouter d'autres joueurs, proposer au joueur de choisir toutes les caractéristiques du jeu, de rejouer…

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 salles, des joueurs…). Sauf cas particulier, les attributs ne devront être accessibles que par leurs accesseurs.

Les instances des classes doivent contenir des méthodes et pas seulement des attributs (Elles ne doivent pas seulement être des réservoirs d'information).

3.2. Rapport

Vous devez réaliser un rapport au format pdf (format aisément lisible sur tous les systèmes). Il doit rendre compte de votre analyse du problème. Le rapport doit contenir les informa-

tions suivantes :

- ✓ **Description des fonctionnalités de votre réalisation**. Il s'agit de présenter ce qui a été réellement fait par rapport à ce qui est demandé dans la première partie de cet énoncé. Vous pouvez avoir fait moins que ce qui a été demandé, ou, au contraire, avoir ajouté des fonctionnalités supplémentaires.
- ✓ **Diagramme de classes**. Description de la structure des classes à l'aide d'un diagramme, qui montre les relations d'utilisation et d'héritage ainsi que les attributs principaux.
- ✓ **Descriptif des classes**. Information sur le contenu des classes principales : rôles des attributs et des méthodes. Expliquez comment vous avez réalisé les méthodes les plus complexes.
- ✓ **Jeux d'essais commentés**. Un jeu d'essai illustre le fonctionnement du programme, soit dans sa globalité, soit seulement sur une partie de ses fonctionnalités.

3.3. Démonstration

La démonstration se situera au cours de la semaine du 29 avril au 3 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 salles (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.

4. Propositions et conseils de modélisation

En plus de la classe principale d'amorce, le programme doit au moins comporter une classe `Plateau`, une classe `Salle` et/ou une classe `Objet`, et une classe `Joueur`. La version suggérée comporte en plus une classe `Jeu` qui gère les paramètres globaux, une classe `Outil` (les outils sont des sortes d'objets que peut utiliser le joueur), ainsi qu'une classe pour chaque type d'outil, une classe `LesOutils` pour gérer les outils du joueur. Le plateau sert d'« environnement commun » aux salles, c'est-à-dire qu'une salle peut accéder à sa salle contiguë dans une direction donnée via le plateau. Pour ce faire, chaque salle référence le `Plateau`. Une salle peut contenir un objet (Mine, caisse de grenades, réserve d'énergie, outil…), ce qui peut être représenté par un attribut d'instance de type `Objet` dans la classe `Salle` ayant la valeur `null` si la salle est vide.

À noter qu'il n'y a qu'un plateau par jeu et qu'un objet par salle. Il est alors possible de regrouper `Plateau` et `Jeu` d'une part, et `Salle` et `Objet` d'autre part… mais cela surchargera les classes restantes.

Pour simplifier la gestion des positions et des directions, vous pouvez aussi utiliser la classe `Position` et la classe `Direction` dont une instance peut se combiner avec une position pour permettre d'obtenir la position contiguë dans une direction donnée. Ces classes sont disponibles sur plubel à côté de cet énoncé.
Dans cette hypothèse, une salle et le joueur ont une position donnée. À partir de la position du joueur, il est facile de connaître la salle dans laquelle il se situe et, donc, la salle contiguë 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

La classe `Position` a été dotée d'une méthode `isValide()` qui vérifie que la position est bien dans le plateau courant. De ce fait, pour vérifier que le déplacement d'un joueur ou l'exploration via un outil ne déborde pas du plateau, il suffit de voir si les positions successives atteintes sont toujours valides.

Si vous n'utilisez pas la classe `Position`, il est préférable de doter le plateau d'une méthode qui vérifie qu'une ligne et une colonne données sont bien situées dans les limites du plateau et de déléguer par la suite à cette méthode toutes les vérifications.

Destruction d'un mur par une grenade

Détruire un mur entre deux salles revient à créer un canal de communication entre ces deux salles. Une possibilité de représenter cela consiste à doter chaque salle de la liste des accès possibles à d'autres salles. Un accès peut être lui-même représenté par une instance de la classe `Direction`.

En d'autres termes, chaque salle comporte un attribut `acces` qui réfère à une liste de directions au départ vide (La salle est initialement complètement entourée de murs).
Quand le joueur lance une grenade dans une direction donnée à partir d'une salle origine, soit la direction fait déjà partie de la liste, et rien ne se passe — à part la perte de la grenade —, soit la direction est ajoutée aux accès de cette salle, et la direction opposée est ajoutée à la salle de destination qui est rendue visible. Dès lors, le joueur a accès à chaque salle à partir de l'autre salle.

Gestion globale des objets

Le joueur peut posséder une méthode `setPosition` ou `setSalle` qui le place virtuellement dans une nouvelle salle[5]. Cette méthode peut alors déclencher la méthode `entree(Joueur j)` — vue plus haut — de la salle d'arrivée, qui, si cette dernière contient un objet, peut elle-même déclencher une méthode `interaction(Joueur j)` associée à l'objet. Les objets se comportent différemment suivant leur type. Cela suggère fortement qu'il y ait une classe par type d'objet (C'est même imposé dans ce projet) avec une implantation spécifique de la méthode `interaction` (Polymorphisme hiérarchique).

Par contre, il ne doit pas y avoir de méthode dans `Objet` qui, avec un `switch` effectuerait l'interaction spécifique à chaque type objet. En effet, autant que possible, la classe `Objet` n'a pas à « connaître » les classes qui dérivent d'elle. Le programme est d'autant plus modulaire que les classes sont indépendantes les unes des autres.

---

[5] Suivant vos besoins, le positionnement du joueur dans une salle peut être matérialisé par un attribut d'instance de type position dans la description du joueur ou dans celle du plateau. Il est aussi possible d'avoir un attribut de type `Joueur` dans la description d'une salle qui référence le joueur quand il est dans la salle et qui vaut `null` sinon.

On peut distinguer les interactions suivantes quand le joueur entre dans la salle qui contient un objet :

- ✓ La mine explose et tue le joueur : le joueur peut avoir un attribut `perdant` initialisé à `false` et que l'interaction fait passer à `true`. C'est cette information qui permettra d'arrêter le jeu.
- ✓ Réserve d'énergie et caisse de grenades : dans les deux cas, l'interaction ajoute la ressource (quantité d'énergie ou nombre de grenades) à ce que possède déjà le joueur, qui est représenté par un attribut spécifique.
- ✓ Outil : sauf s'il le possède déjà, l'outil est « ramassé » par le joueur, c'est-à-dire ajouté à la liste des outils qu'il possède.

Les outils ramassés par le joueur sont retirés de la salle où ils étaient — sauf dans le cas des outils déjà possédés par le joueur qui sont alors laissés sur place.

Gestion spécifique aux outils

Le joueur peut utiliser un outil qu'il possède… si sa réserve d'énergie est suffisante, mais l'action à effectuer est spécifique à l'outil concerné.
De même que précédemment, il est alors conseillé d'avoir une méthode `utilisation(Joueur j)` spécifique à chaque type d'outil. Une méthode de même nom doit exister dans la classe `Outil`. Elle peut être abstraite ou prendre en charge ce qui est commun à tous les outils : la perte d'énergie que leur utilisation induit. Dans ce dernier cas, cette méthode sera appelée à partir du code des méthodes spécifiques en utilisant la syntaxe `super.utilisation(j)` (Même principe que pour le `toString` des objets géométriques).

Affichage du plateau

À chaque type d'objet peut être associé un symbole spécifique qui sera retourné par son `toString` (Voir l'illustration du début). Le `toString` d'une salle restitue un symbole différent suivant que la salle est ou non visible. Elle peut par exemple restituer un caractère plein "▤" si la salle n'est pas visible. Dans le cas où une salle est visible, elle restitue aussi un symbole quand elle est vide (par exemple un espace) et, sinon, elle restitue le symbole de l'objet qu'elle contient (Dans l'exemple, pour pouvoir donner assez d'information, trois caractères successifs sont associés à l'affichage d'une salle).

L'affichage du plateau peut aussi passer par son `toString()` qui est obtenu en combinant les `toString()` des salles qui le composent.

Génération du plateau

C'est une partie complexe parce qu'il faut générer aléatoirement des salles de types différents en tenant compte du nombre de salles à générer par type.

Si vous voulez le faire de manière variable, vous pouvez utiliser la classe `Catégorie` dont les instances crées des objets de types différents.

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 si possible au joueur de se déplacer d'une salle, de lancer une grenade, d'utiliser un outil ou d'abandonner, cela jusqu'à ce qu'il ait gagné ou perdu… ou abandonné.

5. Échéances et conseils

Semaine du 30 avril au 3 mai (la première 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.