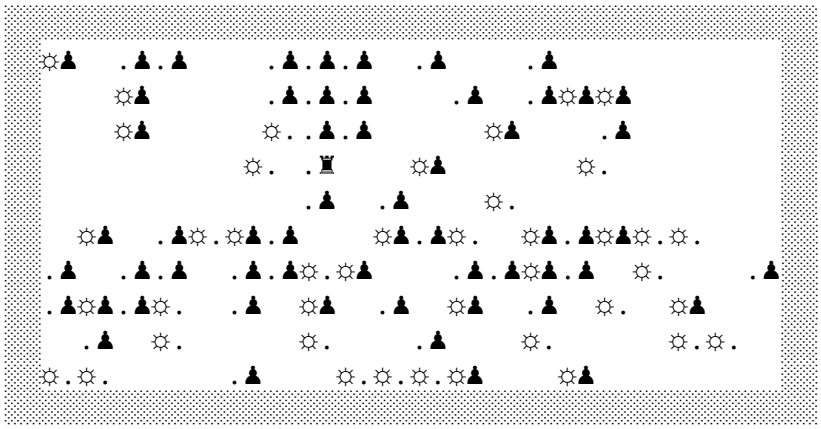
Licence STS première année — Année universitaire 2024-2025
Info 2a — programmation objet
Descriptif du projet

1. Description générale



```
Nombre d'adversaires neutralisés : 5
Collecteur : 29
```

♟ : adversaire,
♜ : le joueur,
☼ : bidon d'énergie.

Cadre du jeu

Ce projet a pour cadre un jeu qui se déroulera sur un plateau constitué d'une grille rectangulaire de salles — non délimitées dans le dessin — dans laquelle peuvent se déplacer des personnages.

Un personnage a une inertie qui lui est propre (entre 0 et 9) ainsi qu'une énergie positive ou nulle qui est initialisée à un maximum énergétique (Par exemple 10, mais cela peut constituer un paramètre du jeu) et qui diminue d'une unité à chaque déplacement élémentaire.

Un personnage a aussi une force qui est le produit de son inertie et de son énergie.

Un personnage qui a 0 comme énergie est neutralisé ; il ne peut plus bouger.

Le personnage principal est l'avatar du (seul) joueur et est contrôlé par lui. On l'appellera ci-dessous Le joueur. Au début du jeu, il est placé au centre du plateau. Son inertie est de 5. Il peut se déplacer dans une direction choisie.

Les autres personnages sont ses adversaires et sont placés aléatoirement sur le plateau autour du joueur avec une inertie de départ aléatoire entre 2 et 9 (Le minimum de 2 est destiné à ce qu'aucun adversaire ne soit neutralisé dès le début du jeu).

Le joueur aura à éviter ou à combattre des adversaires et à stocker le maximum d'énergie dans un collecteur sans être neutralisé. Il voit où sont ses adversaires mais n'a pas connaissance de leur inertie ni de leur énergie. Il voit aussi où sont des bidons d'énergie dans lesquels il pourra puiser de l'énergie.

Le jeu est terminé quand le joueur est neutralisé (Dans ce cas, il a perdu) ou quand tous ses adversaires sont neutralisés et que le joueur décide de la fin du jeu. Dans ce dernier cas, le score du joueur est la quantité d'énergie qu'il y a dans le collecteur.

Déplacement des personnages

À chaque tour de jeu, un personnage non neutralisé peut effectuer un déplacement élémentaire dans une direction donnée en passant de la salle où il est à une salle immédiatement voisine (Qui jouxte sa salle d'origine). S'il ne peut pas effectuer le déplacement dans la direction voulue, il « passe son tour ». Dans le cas contraire, il perd une unité d'énergie.

Le joueur est piloté par le joueur humain et se déplace dans la direction spécifiée par ce dernier. Suivant un principe proche de celui du jeu le Pacman, les adversaires tendent à se rapprocher du joueur quand ils sont plus fort que lui et à s'en éloigner s'ils sont plus faibles. Plus précisément, il peut exister plusieurs catégories d'adversaires ; par exemple, les suivants :

- ✓ les adversaires velléitaires se déplacent avec un tirage aléatoire biaisé : ils ont plus de chance d'aller vers/de fuir le joueur, mais ils peuvent aller aussi ponctuellement dans la direction opposée. Ils peuvent même ne pas bouger ponctuellement.
- ✓ les adversaires déterminés avancent systématiquement dans la direction du joueur ou dans la direction inverse, sauf s'ils sont bloqués par des accumulations d'autres personnages.
- ✓ les adversaires intelligents tentent d'avancer le plus efficacement possible en direction de (respectivement dans la direction inverse du) joueur, en tenant compte des zones de blocages.

À chaque tour, le joueur effectue deux déplacements, puis chaque adversaire en effectue un — si possible (Cela revient à donner une vitesse double au joueur, ce qui lui permet de rattraper les adversaires qui fuient). C'est le changement de salle d'un adversaire ou du joueur qui peut provoquer la rencontre entre eux.

Réservoirs d'énergie

Le jeu comprend des réservoirs d'énergie, qui contiennent une certaine quantité d'énergie. En particulier :

- ✓ des bidons disséminés dans certaines salles du plateau (un maximum par salle), dans lesquels on peut puiser de l'énergie mais pas en stocker. Ces bidons contiennent à leur création le maximum énergétique et leur énergie diminue en fonction des prélèvements par les personnages.
- ✓ un unique collecteur, accessible seulement au joueur, dans lequel on peut stocker de l'énergie de manière illimitée mais pas en prendre. Initialement, il ne contient rien.
- ✓ À noter que la réserve d'énergie propre qui permet à chaque personnage de se mouvoir peut être vue comme un troisième type de réservoir dans laquelle on peut à la fois puiser de l'énergie et en stocker, avec toujours le maximum énergétique comme quantité maximale d'énergie.

La quantité d'énergie dans un réservoir est entre 0 et le maximum énergétique, sauf pour le collecteur où il n'y a pas de limite supérieure.

Il peut y avoir un transfert d'énergie entre réservoirs d'énergie. Le transfert a une source (un bidon ou l'énergie d'un personnage) et un destinataire (un personnage ou le collecteur).
Le principe général du transfert d'énergie est que le montant du transfert ne doit pas excéder le niveau d'énergie disponible dans la source et ne doit pas permettre au niveau énergétique du destinataire de dépasser le maximum — sauf dans le cas du collecteur.

Dans le cadre de ce principe général, le joueur destinataire d'un transfert d'énergie peut prendre pour sa propre réserve ou stocker dans le collecteur la quantité d'énergie de son choix. Les autres personnages (Les adversaires) prennent systématiquement le maximum d'énergie possible pour leur propre réserve. Ils n'ont pas accès au collecteur.

Interaction entre éléments du jeu

Les personnages ont accès à un bidon d'énergie en rentrant dans la salle qui le contient. Ils y prennent de l'énergie suivant les modalités vues ci-dessus au paragraphe « Réservoirs d'énergie ».
Quand le joueur et un adversaire se rencontrent dans une salle, il y a un combat, qui est gagné par celui qui a la plus grande force. Le gagnant prend alors de l'énergie au perdant, toujours selon les modalités précédentes.
Un adversaire neutralisé disparaît du jeu.

Il ne peut jamais subsister plus d'un personnage dans une salle : à l'issue d'un combat, qu'il soit gagnant ou perdant, le personnage dernièrement entré dans la salle où il a rencontré son adversaire est rejeté dans la salle d'où il venait… sauf si le joueur arrivant dans la salle a neutralisé un adversaire qui s'y trouvait. Il l'y remplace alors. Un adversaire ne peut pas entrer dans la salle où un autre adversaire est déjà situé ; il doit aller dans une autre salle ou, par défaut, il est bloqué.

Paramètres du jeu

Le plateau de jeu est rectangulaire, avec un nombre de colonnes et un nombre de lignes impairs — pour avoir un centre.

Comme vu plus haut, les bidons et les personnages ont un maximum énergétique qui est leur énergie initiale.

Le jeu comprend un nombre d'adversaires et un nombre de réservoirs d'énergie disposés aléatoirement dans le plateau, qui doit être une proportion du nombre de salles du plateau.

La proportion entre le nombre d'adversaires et le nombre de réservoirs d'énergie doit aussi être bien ajustée.

Au début du jeu, lors du remplissage du plateau, la machine doit tirer au hasard l'emplacement des réservoirs d'énergie et des adversaires et placer le joueur dans la salle centrale. Au départ, il n'y a pas de bidon dans les salles où se situent des personnages. Ensuite, un personnage peut être dans la même salle qu'un bidon.

Tour de jeu

À chaque tour de jeu, l'état du plateau est affiché ainsi que le contenu du collecteur et le nombre d'adversaires neutralisés. Une direction est alors demandée au joueur humain pour le déplacement du joueur virtuel. Si la salle où ce dernier arrive contient un bidon, l'énergie restante dans le bidon est affichée et la machine demande au joueur humain la quantité d'énergie à prélever dans le bidon. Si la salle contient un adversaire, un combat a lieu. Le résultat du combat — qui peut mener à la fin du jeu si le joueur est neutralisé — est affiché, puis les adversaires non neutralisés se déplacent. Cela peut aussi mener à un combat dont le résultat est affiché. On passe ensuite au tour suivant.

2. 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.

2.1. 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 où tous les adversaires sont déterminés — ou même se déplacent complètement au hasard —, puis, quand le programme fonctionne, vous pouvez ajouter des types d'adversaires.

Si TOUT ce qui est demandé fonctionne, vous pouvez ajouter d'autres fonctionnalités et des options ; par exemple, créer d'autres types d'adversaires ajouter d'autres joueurs (avatars d'humains) qui peuvent se combattre, 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).

2.2. 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 informations 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.

2.3. Démonstration

La démonstration se situera au cours de la semaine du 5 au 9 mai dans un créneau de 20 minutes, où se déroulera votre présentation et le/les enseignant/e/s poseront des questions.
Elle est destinée à présenter les principales fonctionnalités de votre réalisation et la manière dont elles ont été réalisées.

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 mal préparées.
- ✓ 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.

3. Propositions et conseils de modélisation

Classes obligatoires

En plus de la classe principale d'amorce, le programme doit au moins comporter :

- ✓ une classe `Plateau` et une classe `Salle` : le plateau est constitué d'un tableau à deux dimensions de salles. 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 bidon et/ou un personnage, ce qui peut être représenté par des attributs d'instance de type bidon et personnage dans la classe salle, qui ont la valeur null si les éléments correspondants sont absents.
- ✓ une classe `Personnage` et des sous-classes correspondant aux différents types de personnages : il y a le joueur et différents types d'adversaires, qui avancent de manière différente et peuvent même avoir des caractéristiques différentes.
- ✓ une classe `Reservoir`, avec les classes dérivées `Bidon` et `Collecteur`. Il est conseillé aussi de créer une classe `ReserveEnergie` qui dérive aussi de `Reservoir` et qui contient l'énergie de « fonctionnement » de chaque joueur.
- ✓ La version suggérée comporte aussi la classe `Jeu` qui gère les paramètres globaux.

Le texte à partir d'ici ne contient plus d'obligations. Vous pouvez faire vos propres choix à condition d'utiliser massivement le principe de délégation.

Interaction des personnages et du contenu des salles

Une salle permet l'interaction entre un personnage qui veut y entrer et ce qu'elle contient. Pour respecter le principe de délégation, on peut par exemple exprimer les actions qui s'effectuent comme suit :

- ✓ L'avancée d'un personnage, par exemple par une de ses méthode d'instance `avance`, le fait passer d'une salle origine à une salle destination,
- ✓ Cette méthode déclenche une méthode `entre` dans la salle de destination, avec le personnage entrant pour paramètre. Cette méthode ne veut pas dire que le joueur entre dans la salle mais qu'il se présente à l'entrée de la salle. Elle permet de lancer l'interaction avec le contenu de la salle si celle-ci n'est pas vide, puis éventuellement, d'entériner l'entrée du joueur. La salle joue le rôle d'environnement commun à son contenu et au personnage arrivant.
- ✓ L'interaction entre le personnage arrivant et le contenu de la salle peut se faire en déclenchant pour l'arrivant sa méthode `interagit` en paramètre une entité de la salle (un bidon et/ou un personnage). Il y aurait alors deux exécutions de cette méthode si la salle comporte à la fois un bidon et un personnage.

Déplacement des personnages

Pour simplifier la gestion des déplacements au sein du plateau, vous pouvez aussi utiliser la classe `Direction` dont une instance est définie par un décalage de 1 négatif ou positif en ligne (`dLig`) et en colonne (`dCol`). Cette classe est disponible sur plubel à côté de cet énoncé. Elle permet d'entrer facilement une direction sous forme texte (Exemple : `haut gauche`) ou sous forme abrégée (`hg` ou `gh`). Elle permet aussi de créer une direction à partir d'un numéro (0 : droite, 1 : haut droite…), ce qui permet de générer une direction au hasard.
Il suffira, dans le plateau, d'ajouter au numéro de ligne/de colonne d'une salle les valeurs `dLig` et `dCol` d'une direction `d` pour obtenir les coordonnées dans la plateau de la salle voisine dans la direction `d`.

À noter que, comme c'est classique en informatique, le numéro de ligne le plus petit correspond à la ligne du haut. Autrement dit, remonter d'une ligne correspond à dLig = -1.

Comme dit plus haut, chaque déplacement élémentaire d'un personnage se fait de la salle où il se situe, vers une des salles voisines dans une direction spécifiée. Une des manières de faire est que chaque personnage référence la salle où il se situe, qui, elle-même, connaît ses coordonnées et peut accéder, via le plateau, à sa salle voisine dans une direction donnée.

Le déplacement du joueur se fait en demandant une direction à l'utilisateur (méthode `getDirectionQuelconque()` de la classe `Direction` si vous l'utilisez.). Il demande alors à la salle dans laquelle il se situe sa salle voisine dans la direction spécifiée, et lance la méthode `entre` de cette nouvelle salle.

Pour se déplacer, les adversaires, qui fuient ou coursent le joueur, doivent connaître les coordonnées ligne-colonne de la salle où il se situe.
Pour ce faire, chaque adversaire peut référencer le joueur et, donc, indirectement, connaître ses coordonnées en ligne-colonne : (lig_j,col_j). Il peut alors calculer les différentiels $dLig = lig_j - lig_{this}$ et $dCol = col_j - col_{this}$, où (lig_{this},col_{this}) sont ses propres coordonnées. Pour aller vers le joueur, la direction qu'il doit adopter est (signe(dLig), signe(dCol)) où signe(d) restitue 0 si d vaut 0, -1 s'il est négatif, et 1 s'il est positif. La direction inverse, correspondant à la fuite, peut être obtenue par la méthode d'instance `d.getInverse()` de la classe `Direction`, où d est la direction qui rapproche du joueur.

Pour obtenir le comportement d'un adversaire velléitaire, il faut lui donner une probabilité plus grande d'aller vers le joueur que dans une autre direction. Par exemple, si joueur est à droite de l'adversaire, la valeur dCol de la direction horizontale à adopter est 1 mais il peut aussi, de manière moins probable, avoir une valeur de dCol de 0 ou 1. Par exemple, il peut avoir 5 chances sur 10 d'avoir la valeur 1 (d'aller à droite), 3 chances sur 10 d'avoir la valeur 0 (ne pas bouger horizontalement), et 2 chances sur 10 d'avoir la valeur -1 (d'aller dans le sens inverse en allant à gauche).
Ce résultat peut être obtenu en créant le tableau suivant :

0	1	2	3	4	5	6	7	8	9
1	1	1	1	1	0	0	0	-1	-1

Il suffit alors de tirer au hasard un indice entre 0 et 9 pour obtenir une valeur dans la probabilité désirée.

Gestion des bords du plateau

La vérification que l'on reste sur le plateau peut être laborieuse. Il est intéressant de créer une méthode `isValide` dans le plateau, qui vérifie que des coordonnées en numéro de ligne et de colonne sont bien valides.

Une autre méthode consiste à créer plusieurs types de salles, dont des salles au bord du plateau, dans lesquelles on ne peut pas rentrer (la méthode `entre` ne fait rien ou affiche une erreur). Par exemple, pour un plateau ayant nbLig lignes et nbCol colonnes, les salles du bord gauche peuvent avoir comme numéro de colonne iCol, la valeur 0, pour celles du bord droit, on aura iCol = nbCol+1. Même principe avec les lignes pour les bords haut et bas. À noter qu'ici on considère que, pour les salles internes au plateau, iLig ∈ [1,nbLig] et iCol ∈ [1,nbCol]. Dans cette logique, il pourrait exister une classe `Salle` abstraite, définie uniquement par ses coordonnées `nLig` et `nCol` et une méthode `entre` abstraite (sans corps), ainsi que des classes dérivées `SalleBord` et `SalleDans`. Les salles de bord n'auraient pas de contenu, c'est-à-dire pas d'attributs `Bidon` et `Personnage` et une méthode `entre` qui ne fait qu'afficher éventuellement une notification d'erreur. La classe `SalleDans` a bien les attributs `Bidon` et `Personnage` et traiteraient l'entrée d'un personnage comme vu plus haut.

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 du plateau restitue une chaîne qui synthétise son contenu (`.♟` pour adversaire, `☼.` pour bidon, et `☼♟` pour les deux… ou des lettres). Il peut éventuellement y avoir des symboles différents pour un bidon vide ou un adversaire neutralisé. Une salle du bord peut par exemple restituer un (double) caractère plein "▒▒".

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

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.

4. Échéances

Semaine du 5 au 9 mai (la première semaine après les vacances de Pâques)
Démonstration du projet dans un des créneaux qui seront proposés (vous devrez vous inscrire auprès de votre responsable de TP). 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.