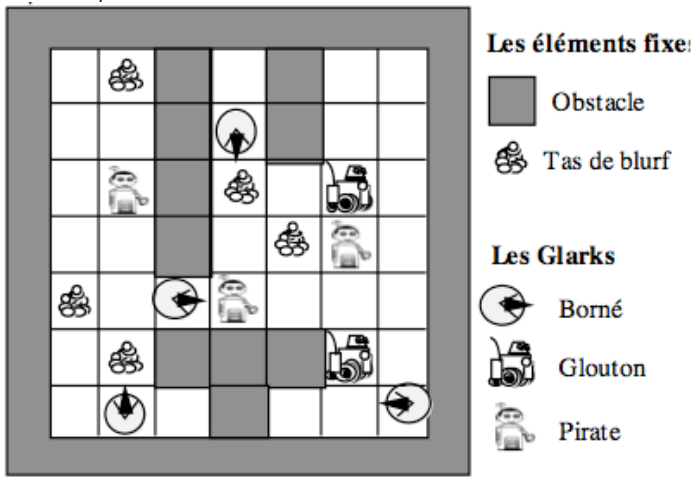
Modélisation puis réalisation en Java d'un programme de jeu vidéo imaginaire, le monde des Glarks

1. Enoncé

Nous allons programmer la base d'un jeu vidéo imaginaire: **le monde des Glarks**. On ne cherchera pas à programmer le joueur, mais simplement le comportement du jeu en l'absence de joueur. La plupart des jeux vidéos donnent l'impression que les êtres se déplacent en parallèle. Puisque les machines ne sont pas parallèles, on simule le parallélisme en activant le comportement caractéristique de chaque être au cours d'une boucle d'exécution. Un comportement est donc un ensemble d'actions qu'un être peut accomplir au cours d'un cycle de la boucle d'exécution.

Le jeu **le monde des Glarks** est un jeu vidéo dans lequel de petits êtres, les Glarks se déplacent, se nourrissent et donc vivent dans un espace à deux dimensions, qui est représenté sous la forme d'un ensemble de cases disposées en damier.



Il existe plusieurs catégories d'éléments dans le monde des Glarks. On distingue essentiellement les éléments fixes des éléments mobiles.

Les éléments fixes

Les éléments fixes sont les *obstacles* et les *tas de blurf*.

Les obstacles sont des cases particulières sur lesquelles il n'y a ni Glark ni tas de blurf.

Les tas de blurf sont fixes et sont disséminés dans tout l'espace. Ils servent de nourriture aux Glarks. Un tas de blurf contient 100 points d'énergie. Si un Glark dévore un tas de blurf (et pour cela il suffit au Glark de se placer sur la case où se trouve le tas de blurf) alors il récupère ces points d'énergie. Il ne peut y avoir qu'un tas de blurf par case.

Les éléments mobiles

Les Glarks constituent les éléments mobiles du jeu. Ils se divisent en trois catégories: les Bornés, les Gloutons et les Pirates. Les déplacements comme les perceptions des glarks s'effectuent toujours de manière orthogonale. Ils ne peuvent donc pas se déplacer ni percevoir en diagonale. Les bornés foncent tout droit jusqu'à rencontrer un obstacle et changent alors de direction. Les gloutons et les pirates peuvent percevoir leurs cases voisines (dans des directions orthogonales) et se diriger vers l'une d'elles en fonction de ce qu'ils perçoivent.

Les Bornés : à chaque cycle, un Borné avance d'une case dans une direction choisie initialement de manière aléatoire. S'il rencontre un obstacle, c'est-à-dire s'il perçoit un obstacle se trouvant devant lui, il change de direction, en allant vers la gauche ou la droite. Si les cases à gauche et à droite sont aussi des obstacles, il fait demi-tour et rebrousse chemin.

Lorsqu'il arrive sur une autre case, un borné dévore les tas de blurf et tous les Glarks qui s'y trouvent sauf les Bornés. Lorsqu'il dévore un Glark il gagne 10 points d'énergie (indépendamment de l'énergie de ce Glark). Un Glark dévoré est supprimé de l'espace de jeu.

Les Gloutons : comme les bornés, les gloutons avancent d'une case à chaque cycle. Mais à la différence des bornés, ils sont capables de percevoir les tas de blurf à une case de distance dans les quatre directions (haut, bas, gauche, droite). Ils peuvent ainsi se diriger vers eux. S'ils ne perçoivent rien, ils se déplacent en choisissant une direction de manière aléatoire.

Les Pirates: comme les bornés et les gloutons, ils avancent d'une case à chaque cycle. Ils sont capables de percevoir ce qui se passe à une case de leur position. Ils ne sont pas capables de manger les tas de blurf directement (lorsqu'ils vont sur une case où se trouve un tas de blurf il ne se passe rien), mais ils peuvent récupérer les points d'énergie des autres Glarks (et en particulier des autres pirates aussi) en les dévorant, ce qui élimine ces derniers du jeu. S'ils perçoivent un (ou plusieurs) glarks sur une case voisine, ils se dirigent donc vers cette case pour les dévorer.

Les glarks disposent au départ d'un capital d'énergie de 100 points. À chaque cycle ils perdent 5 points rien que pour se maintenir en vie. Un glark qui n'a plus d'énergie meurt. Un glark ne peut dévorer un autre glark qu'en arrivant sur la case où se trouve sa proie.

Remarques :

- plusieurs glarks peuvent se trouver en même temps sur une même case ;
- pour chaque glark, les déplacements se font toujours d'une case par cycle.

2. Conception : les objets de l'application

Les objets sont

- le **jeu** composé d'un damier et de Glarks
- le **damier** qui est un tableau à double entrée de cases
- les **cases** sont de deux sortes : soit elles contiennent des obstacles et sont inaccessibles, soient elles peuvent contenir du blurf et/ou des glarks
- les **glarks** sont de trois sortes : les Bornés, les Pirates et les Gloutons qui se différencient par leur régime alimentaire et leur stratégie de déplacement

2. Présentation des classes et de leur responsabilité

Les classes à considérer sont donc

- la classe **Jeu** qui est responsable de l'initialisation du damier, de l'initialisation des Glarks et du lancement de la boucle d'exécution
- la classe **Damier** mémorise l'ensemble des cases ; c'est un tableau à double entrée de cases dont le rôle est de faciliter l'accès aux cases
- On a deux sortes de cases : les cases obstacles et les classes libres. Une CaseObstacle ne fait rien de spécial si ce n'est prévenir le glark qui voudrait s'y rendre que c'est impossible. Une CaseLibre mémorise le blurf qu'elle contient éventuellement, les Glarks qu'elle contient éventuellement. Toutes les cases mémorisent la position de la case. On peut dire que par défaut une case est-un obstacle et une case libre est une case plus spécialisée. Nous choisissons (d'autres choix auraient été possibles) de définir une classe **Case** et une sous-classe de cette classe, la classe **CaseLibre**. La responsabilité de la classe Case est de gérer la position de la case sur le damier et de définir le comportement par défaut des cases.

Remarque : ici les cases sont passives. C'est la nature des Glarks qui décide de leur déplacement et non la nature des cases comme c'était le cas dans le jeu de l'oie.

- La classe des **Glarks** a trois sous-classes : les Bornés, les Pirates et les Gloutons. Tous les glarks gèrent leur énergie, leur nourriture et leur position sur le Damier. Chaque type de glark se déplace et se nourrit différemment des autres.
- La classe **Direction** sert à gérer les changements de direction des Bornés.

3. Description précise des classes

La classe **Jeu**

C'est une classe concrète qui a une instance unique.
• Elle possède deux champs
- *damier* où sont mémorisées les cases
- *listeDesGlarks* liste où sont mémorisés les Glarks (morts ou vifs)
• Elle répond aux demandes de services suivants
- *initialiseToi* qui crée le Damier, place les obstacles, crée les Glarks et les dispose sur le Damier
- *lanceToi* tant qu'il y a un glark en vie, chaque Glark en vie reçoit le message vasY

La classe **Damier**

C'est une classe concrète qui a une instance unique. C'est un tableau à double entrée de cases qui ne fait rien de plus que d'assurer les services d'un tableau à double entrée
• accès en consultation et en modification à chaque case en fonction de sa position (ligne, colonne)
donneTaCase : i et : j retourne la case de la ième ligne et j ième colonne du damier
modifieTaCase : i et : j avec : uneCase remplace par uneCase la case de la ième ligne et j ième colonne du damier.

La classe **Direction**

C'est une classe concrète dont la responsabilité est de retourner la voisine d'une case passée en paramètre.
Une direction possède un champ appelé *sens* qui peut posséder 4 valeurs appartenant à la rose (des vents bien sûr) nord, sud, est ou ouest.
Elle permet de retourner la case voisine d'une case dans une direction donnée (nord, sud, ouest, est) d'une case à gauche (respectivement à droite, à l'opposé) d'une direction donnée
exemple : si l'on considère la case c de coordonnées (2, 3) quand on envoie à la direction nord le message :
voisineDansDir: c , la direction nord retourne la case de coordonnées (1,3)
caseGauche: c, la direction nord répond la case de coordonnées (2, 2) (la gauche du Nord est l'ouest)
caseDroit: c, la direction nord répond la case de coordonnées (2, 4) (la droite du Nord est l'est)
caseOpposée: c la direction nord répond la case de coordonnées (3, 3) (l'opposé du Nord est le sud)

La classe **Case**

C'est une classe abstraite. Elle possède deux sous-classes concrètes CaseObstacle et CaseLibre dont le seul comportement commun est de savoir répondre au message envoyé par les Glarks leur demandant si elles sont un obstacle.
Chaque case connaît sa *position* (ligne, colonne) :
Elle répond aux demandes de service suivant
- *estObstacle* retourne faux par défaut (par défaut une case n'est pas un obstacle) ; ce comportement est redéfini par la classe CaseObstacle

La classe **CaseObstacle**

C'est une classe concrète sous-classe de Case. Elle ne dispose d'aucun champ supplémentaire et redéfinit le comportement hérité
- *estObstacle* retourne vrai.

La classe **CaseLibre**

C'est une classe concrète sous-classe de Case.
• Elle dispose de champs supplémentaires
- *damier* qui permet d'accéder au damier
- *blurf* qui est indique la présence d'un tas de blurf
- *listeDesGlarks* qui donne la liste des glarks présents sur la case
• Elle répond aux demandes de services suivants :
- des méthodes d'accès en consultation et en modification à ses contenus (il faut pouvoir lui demander de supprimer son tas de blurf quand il est mangé, d'effacer un glark qui la quitte ou meurs, d'ajouter un glark qui lui arrive) ,
- des méthodes d'accès à ses voisines (nord, est, sud, ouest) ; pour cela, la case doit connaître le damier ,
- une méthode *menu* qui offre au glark passé en paramètre de manger le blurf (s'il y en a), de manger les glarks présents (s'il y en a).
Remarque : c'est le Glark qui sait ce qu'il aime manger ; la case elle, propose démocratiquement le même menu à tout le monde.

La classe **Glark**

C'est une classe abstraite qui est responsable de ce qui est commun à tous les Glarks.
• Elle possède deux champs :
- énergie (un nombre),
- caseCourante (une caseLibre).
• Elle fournit les services suivants (communs et non redéfinissables)
- accès en consultation et modification (uniquement pour les sous-classes) à ses champs ;
attention quand la case courante d'un Glark est modifiée ne pas oublier de gérer la liste des Glark de la case
- *meurs* énergie à zéro, disparais de la listeDesGlarks de la case courante,
- *vasY* : message envoyé par le jeu qui provoque diminution de l'énergie nécessaire au cycle, si le Glark est épuisé, il meurt sinon il recherche sa destination; si la case retournée est différente de la classe courante modification de la case courante (qui ensuite offre son menu au nouveau venu),
• Elle fournit des **comportements par défaut**
- *estBorné* tous les Glarks savent s'ils sont bornés ou non (retourne faux par défaut) ; ce comportement est redéfini par la classe Borné .
- *devoreBluf* le glark s'attribue 100 points d'énergie supplémentaire et supprime le blurf de la case courante ; ce comportement est redéfini par la classe Pirate qui ne mange pas de blurf
- *devoreGlark:* unGlark ; ce message est envoyé par la case destination quand un glark se déplace ; le glark receveur est celui qui mange et le paramètre est le repas ; par défaut un glark n'est pas anthropophage mais certains type le sont (Bornés et Pirates) ; ils redéfinissent donc cette méthode
• Toutes les sous-classes doivent définir les méthodes suivantes (elles sont déclarées dans la classe abstraite, mais leur réalisation est de la **responsabilité des sous-classes**)
- *trouveDestination* retourne la case où va se diriger le Glark

La classe Borné

C'est une classe concrète sous-classe de Glark.
• Elle possède un champ supplémentaire :
- *direction* qui indique dans quel sens le borné se déplace si le champ est libre
• Elle redéfinit évidemment le comportement par défaut
- *estBorné* retourne vrai
• Elle définit ainsi les obligatoires des Glarks (de la responsabilité des sous-classes)
devoreGlark: unGlark ; ce message est envoyé par la case destination quand un glark se déplace ; le glark receveur est celui qui mange et le paramètre est le repas ;
Si le repas n'est pas un borné alors
augmente ton énergie de 10
et toi pauvre repas meurs !
- *trouveDestination*
si la voisine dans la direction est libre
retourner cette voisine ;
sinon
direction = direction à gauche de direction
si la voisine dans la direction est libre retourner cette voisine ;
sinon direction = direction opposée de direction si la voisine dans la direction est libre retourner cette voisine ;
sinon direction = direction droite de direction si la voisine dans la direction est libre retourner cette voisine ;
sinon retourner direction = direction opposée (ie. crever sur place, on est coincé)

La classe Glouton

C'est une classe concrète sous-classe de Glark.
• Elle ne possède aucun champ supplémentaire :
• Elle ne redéfinit aucun comportement par défaut
• Elle définit ainsi les comportements obligatoires des Glarks de la responsabilité des sous-classes
- *devoreGlark:* unGlark ; les gloutons ne mangent que du blurf ; la méthode héritée convient parfaitement
- *trouveDestination*
chercher la première case qui contient du blurf et s'il en existe une la retourner
sinon retourner la première case libre

La classe Pirate

C'est une classe concrète sous-classe de Glark.
• Elle ne possède aucun champ supplémentaire :
• Elle redéfinit un comportement par défaut
- *devoreBlurf* ne fait rien, les pirates ne touchent pas au Blurf
• Elle définit ainsi les comportements obligatoires des Glarks (de la responsabilité des sous-classes)
- *devoreGlarks:* repas
Si le repas n'est pas le receveur du message alors
augmente ton énergie de l'énergie du repas
repas tu es mort !
- trouveDestination
retourne la première case libre qui contient un Glark s'il en existe une sinon retourner la première case libre

Remarque malveillante : estBorne, estObstacle, ce ne serait pas par hasard quelque sournois test de type déguisé ? Alors mettez-vous des claques et programmez objet vous-même...Faites ce que je dis pas ce que je fais…
Ouais bon ça va…
En général tester le type d'un objet est une erreur de programmation. Mais ici comment faire autrement ? on est coincé :
- estBorné : on a en besoin car le régime alimentaire de ces anthropophages dépend à la fois de celui qui mange et de celui qui est mangé
- estObstacle : il est plus simple de demander à une case si on peut y aller ; plutôt que d'y aller et de se faire mettre à la porte.