

« *Game of Spies...* »

Sébastien Combéfis et Fabien Duchêne

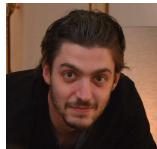
Bienvenue à l'Université ! Durant cette première semaine de cours, différentes activités sont organisées. Vous aurez notamment vos premiers cours de mathématiques, de sociologie et d'économie politique. Pour l'informatique, une activité spéciale vous est proposée : l'APP0. Il s'agit d'un mini-projet informatique nommé « *Game of Spies* » qui vous permettra de faire connaissance avec les autres étudiants, mais ce sera aussi l'occasion de poser une série de bases en informatique et en gestion d'un projet de programmation en équipe. Pour la durée du projet, vous serez amenés à travailler dans trois groupes différents de cinq étudiants. Chaque groupe sera supervisé par un tuteur junior qui sera votre guide et des assistants supervisant le bon déroulement du projet.



Cette année, l'APP0 vous emmène dans un monde où l'hiver arrive, et où les loups et les forêts infranchissables menacent la vie de ceux qui osent s'aventurer en ces terres inhospitalières...

1 Informations générales

Assistant Coordinateur : Fabien Duchêne



Professeurs :

Pr Charles Pecheur



Pr Olivier Bonaventure



Assistant(e)s :

Quentin De Coninck



Benjamin Hesmans



Vincent Branders



Quentin Cappart



Tous les documents relatifs à cet APP0 se retrouvent sur le site du cours **APP0 SINF** (Moodle) :

<http://moodleucl.uclouvain.be/course/view.php?name=APP0SINF>

Source de l'image de couverture : <http://kerim-b.deviantart.com/art/Game-of-Thrones-Dire-Wolf-386023294>.

2 Calendrier

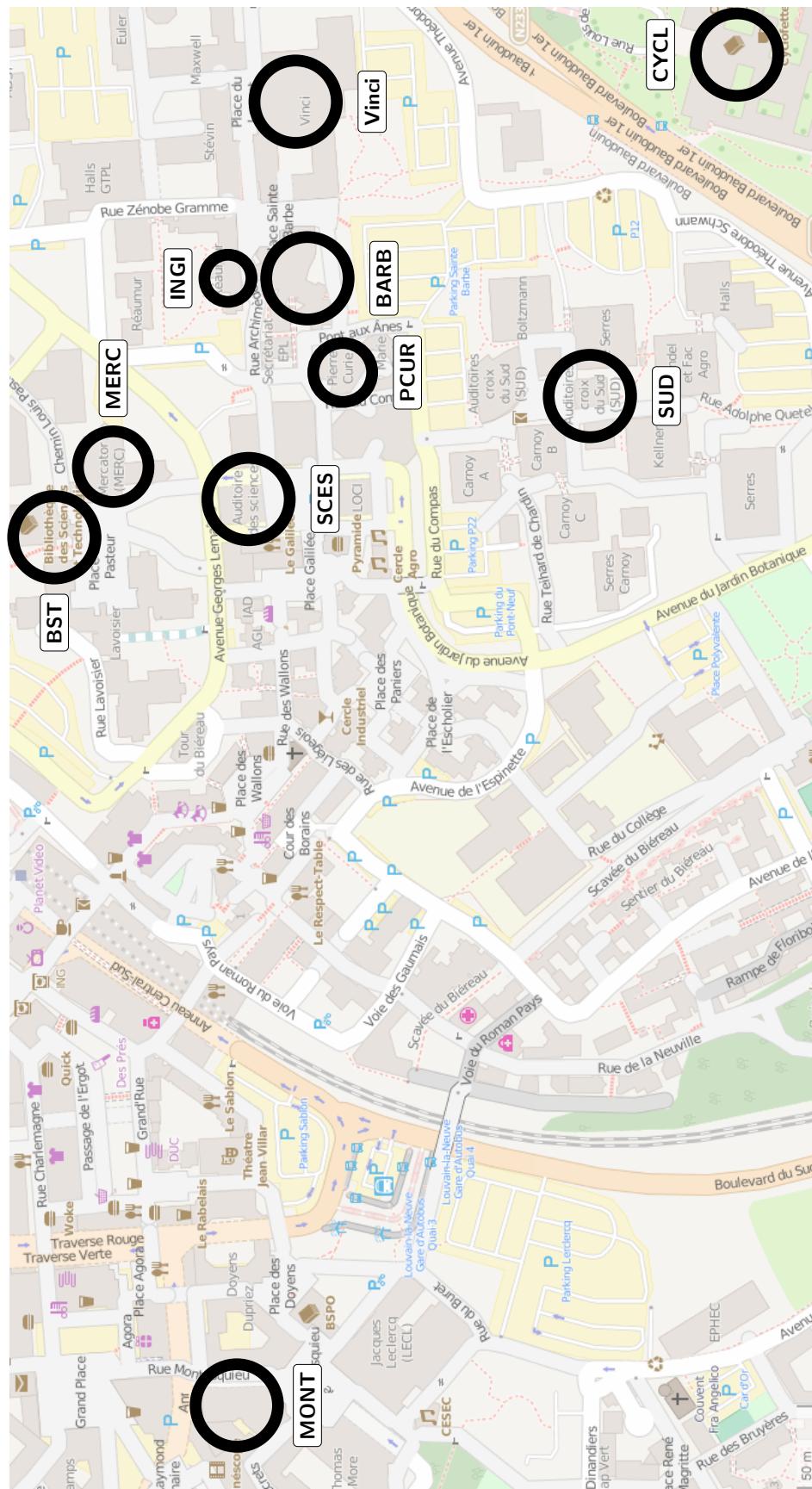
Voici le calendrier complet de votre première semaine de cours. Notez les barres horizontales plus épaisses, elles correspondent à des échéances où vous devrez remettre un travail. La suite de cette section détaille ce calendrier. **Veuillez être à l'heure et au bon endroit tout au long de la semaine.**

	Lu. 19/09	Ma. 20/09	Me. 21/09	Je. 22/09	Ve. 23/09
8h30	Accueil BARB 94	Intro BARB 94 Phase I PCUR 01,02,03,04	Cours I BARB 93	Cours II BARB 91 Passeport pour le BAC BARB 91	LSINF1111 CYCL 02,03,05,06
10h30					
10h45	Kickoff BARB 20,21,23	Travail perso I PCUR 01,02,03,04	Travail perso II CYCL 03,05,06,09b	LSINF1111 SCES 02	Machine I Salles informatiques
12h45					
14h00					
16h00					
16h15		LESPO1113D MONT 10	LEGCGE1115D MONT 11	Travail Perso III BARB 20,21,22,23	Machine II Salles informatiques
18h15		Bilan I PCUR 01,02,03,04	Bilan II CYCL 02,03,05,09b	Bilan III BARB 20,21,22,23	Concours Salles informatiques
				Test BARB 20,21,22,23	

- Durant la période *Intro*, le problème qui nous occupera dans le cadre de l'activité APP0 vous sera introduit en grand auditoire.
- Les périodes marquées comme *Travail Perso* sont réservées pour votre travail individuel ou en groupe, mais non encadrées. Des salles spécifiques sont mises à votre disposition. Si ces périodes sont explicitement mises à l'horaire, c'est sans doute qu'il y a une bonne raison...
- La période *Phase I* ainsi que les périodes marquées *Bilan* sont des périodes durant lesquelles vous serez encadrés par votre tuteur. Ces périodes sont rares, tâchez donc de les préparer à l'avance afin de les exploiter au mieux.
- Les périodes *Cours* correspondent à un cours magistral de restructuration et à un cours de feedback. Au terme de chacun de ces cours, vous serez affectés à de nouveaux groupes.
- Durant la période *Test*, vous passerez une évaluation du travail accompli au cours de la semaine.
- Les périodes *Machine* seront utilisées pour vous familiariser à votre environnement informatique, et vous permettront également de coder la solution de votre groupe pour le problème à résoudre.
- Enfin, la semaine se clôturera par un concours durant lequel les solutions des différents groupes seront exécutées et comparées en direct.

Votre présence est requise aux différentes activités de l'APP0.

Pour les auditoires, BARB = Sainte-Barbe, CYCL = Cyclotron, MERC = Mercator, SCES = Sciences, SUD = Sud et MONT = Montesquieu. Les salles informatiques sont les salles Candix, DAO et IAO se trouvant dans le bâtiment Vinci.



3 Mission

Dans la ville de Westeros DC, le roi Robert Baratheon gouverne d'une main de fer le royaume de la Couronne Intemporelle Acquise (CIA). Suite au décès de Chris Amador, son bras droit, il nomme son vieil ami *Eddard "Ned" Stark* au poste de Main du Roi. Si cette nomination est unanimement saluée dans le royaume, elle est un affront direct pour la reine : Cersei Lannister. En effet, depuis des temps immémoriaux, les familles Stark et Lannister se livrent une véritable guerre froide. Bien décidée à faire tomber la tête de Ned Stark, la reine décide d'espionner celui-ci afin de découvrir les secrets stratégiques de sa famille. Pour ce faire, elle décide d'avoir recours à deux membres de l'ordre des "*Knights of the Great Baratheon*" (KGB), les services secrets du roi. Vivant incognito à Winterfell (domaine des Stark) depuis des années, c'est *Philip et Elizabeth Jennings* qui se sont vu confier la lourde tâche d'approcher *Ned Stark* afin de l'espionner. Pour mener à bien leur mission, ils devront traverser les étendues glacées du nord du royaume tout en évitant de se perdre dans les épaisse forêts du nord et en se cachant des loups protégeant *Ned Stark*.



Pour guider ses agents sur le terrain, le KGB possède une cellule spéciale nommée *le Centre*. Cette cellule est composée d'agents ayant accès à toute une série d'outils leur permettant d'avoir accès aux informations du terrain pour guider au mieux les agents. La réussite d'une mission dépend donc en grande partie de la capacité des agents du *Centre* à prendre les bonnes décisions au bon moment. Afin de garantir la réussite d'un maximum de missions, la cellule a mis en place un processus très pointu afin de sélectionner ses agents. Durant une semaine, les aspirants agent seront confrontés à des situations proches de la réalité du terrain afin d'évaluer leurs capacités à guider les agents sur le terrain. Pour simuler la réalité du terrain, le KGB utilise un simulateur reproduisant le système utilisé pour guider les agents dans la réalité. À la fin de la semaine de sélection, les résultats des équipes seront comparés, et l'équipe la plus performante sera sélectionnée pour intégrer *le Centre*.

En tant qu'aspirant agent, votre mission de cette semaine sera d'apprendre à utiliser le simulateur pour aider *Philip et Elizabeth* à approcher *Ned Stark* en leur trouvant un chemin sûr et discret tout en évitant les bois et les loups.

Source image : <http://www.techtimes.com/articles/58294/20150605/game-dreaming-imagining-the-perfect-game-of-thrones-video-game.htm>

3.1 Description du simulateur

Pour vous permettre de passer les sélections, un simulateur est mis à votre disposition, un exemple de celui-ci est illustré par la figure 1.

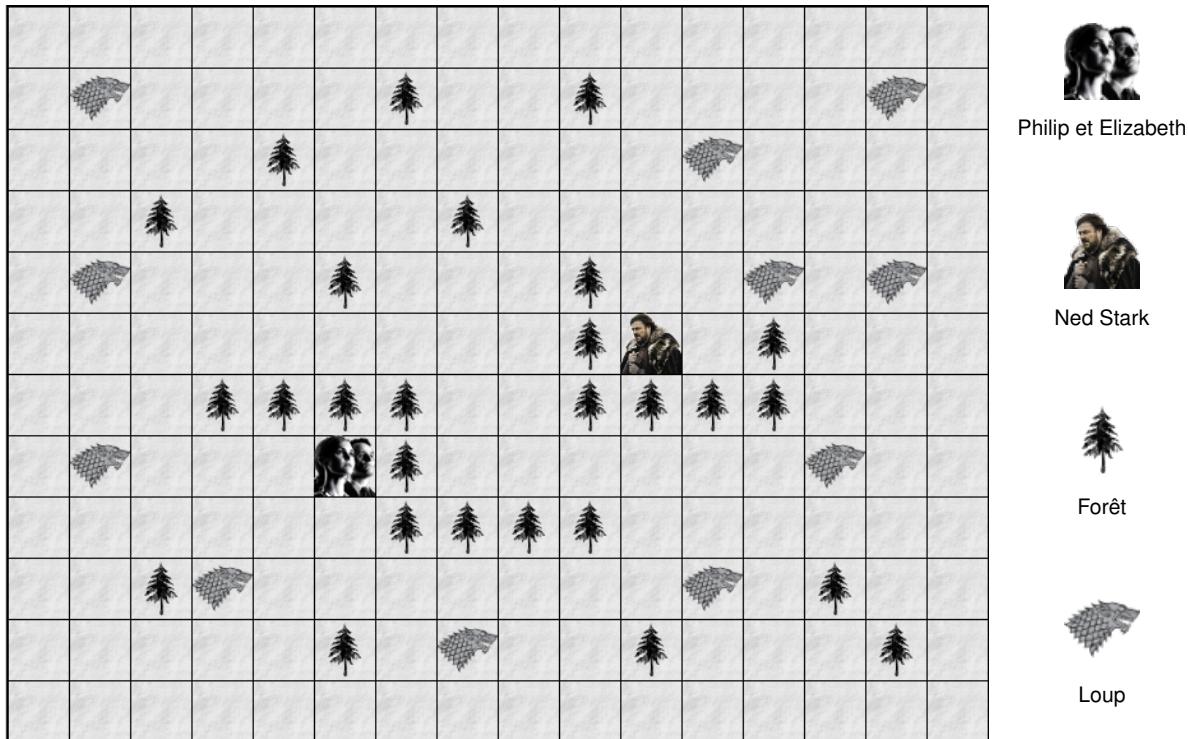


FIGURE 1. Exemple de monde avec des obstacles (loups et forêts).

Philip et Elizabeth ne peuvent effectuer que des opérations de base : avancer d'une case, tourner de 90 degrés sur eux-mêmes et regarder s'il y a une forêt ou un loup devant eux. À partir des primitives de base qu'ils sont capables d'effectuer, vous allez devoir établir une procédure systématique leur permettant d'atteindre Ned Stark. L'hiver venant, Philip et Elizabeth doivent veiller à effectuer le moins de déplacements possible, et ce quelles que soient les difficultés rencontrées.

Le problème étant complexe, il convient de procéder en plusieurs phases. Durant celles-ci vous apprêhenderez une partie du problème et trouverez des solutions en vous aidant du groupe. Chaque phase vous permettra de découvrir un concept clé en programmation.

3.2 Informations sur le monde

L'orientation sur le terrain se fait grâce à un système de coordonnées, celui-ci est différent du modèle habituellement utilisé en mathématiques. En effet, comme le montre la figure 2, l'axe des y est orienté vers le bas et l'origine du système se situe dans le coin supérieur gauche, de coordonnées $(0, 0)$. La figure 2 illustre les quatre

Ce système informatique commence à compter à partir de zéro.

points cardinaux. La direction dans laquelle Philip et Elizabeth se dirigent est nommée *direction courante* et peut être NORTH, SOUTH, WEST ou EAST.

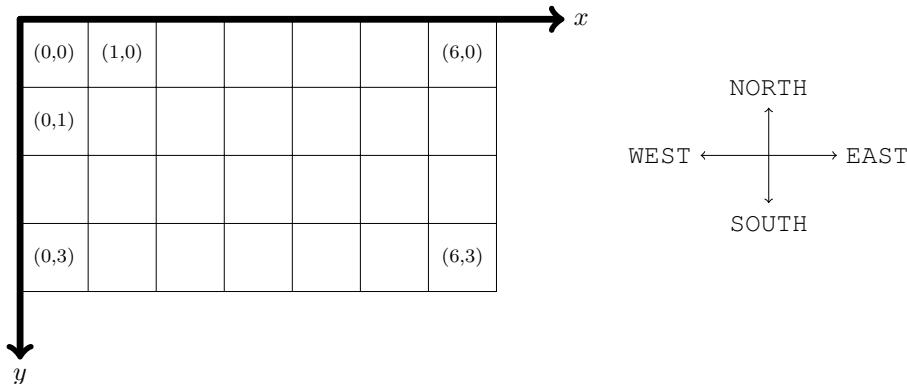


FIGURE 2. Système de coordonnées et points cardinaux.

En plus de Philip et Elizabeth que vous contrôlez et qui peuvent bouger, le monde est composé de forêts infranchissables, de loups qui peuvent les manger et de Ned Stark qui est l'objectif à atteindre. Afin d'espionner Ned Stark, vos espions doivent se déplacer le long d'un chemin reliant leur position initiale à sa position en évitant les forêts et les loups. Pour éviter l'hiver venant, vos agents doivent rejoindre Ned Stark par le chemin le plus court possible.

3.3 Transmission des ordres

Pour transmettre leurs ordres sur le terrain, le Centre utilise un système de messages chiffrés très ancien, mais très fiable. Afin de garder les messages les plus courts possible, le système ne permet de transmettre que des ordres très simples nommés *actions de base* (opérations primitives). Les agents de terrain ont une confiance absolue dans les ordres donnés par le centre, ce qui signifie que dans la mesure du possible ils obéiront toujours aux ordres donnés. Dans le cadre du simulateur, une action résultera en une modification de l'*état* du « *monde* », c'est-à-dire soit la position de vos agents dans le monde ou alors leur direction courante

Les ordres auxquels obéissent les agents sont :

- move fait avancer Philip et Elizabeth d'une case dans leur direction courante ;
- turnLeft fait tourner (sur place) Philip et Elizabeth de 90 degrés vers leur gauche ;
- turnRight fait tourner (sur place) Philip et Elizabeth de 90 degrés vers leur droite.

L'opération `move` ne peut être appliquée que si la case suivante, selon la direction courante, est libre. C'est-à-dire qu'il s'agit d'une case se situant dans les limites du monde et ne contenant pas de forêt. Si vous effectuez un `move` contre un forêt ou contre le bord du monde, l'instinct de survie de Philip et Elizabeth fera que l'opération n'aura aucun effet. Si vous effectuez un `move` et que la case suivante contient un loup, le loup étant trop rapide, il mangera malheureusement vos agents.

Les opérations `turnLeft` et `turnRight` permettent de faire tourner Philip et Elizabeth sur place vers la gauche ou vers la droite. La première opération correspond donc à une rotation de 90 degrés dans le sens anti-horlogique tandis que la seconde correspond à une rotation de 90 degrés dans le sens horlogique.

Les trois opérations suivantes permettent d'effectuer des tests. Celles-ci produisent un résultat booléen (*vrai* ou *faux*) et pourront être utilisées dans les tests et les boucles.

- `canMove` teste si Philip et Elizabeth peuvent avancer d'une case dans leur direction courante ;
- `isInFrontWolf` permet de vérifier si Philip et Elizabeth se trouvent face à un loup dans leur direction courante ;
- `isOnTarget` teste si Philip et Elizabeth ont trouvé (sont sur la même case que) Ned Stark.

L'opération `canMove` teste donc s'il y a une forêt devant vos agents ou si ceux-ci font face au bord du terrain. L'opération `isOnTarget` permet de tester s'ils se trouvent sur la même case que Ned Stark. Une fois qu'ils ont trouvé Ned Stark, il peuvent l'espionner grâce à l'opération suivante :

- `spyOnTarget` fait en sorte que Philip et Elizabeth espionnent Ned Stark.

Si Philip et Elizabeth ne se trouvent pas sur la même case que Ned Stark lors de l'exécution de l'opération `spyOnTarget`, un message s'affichera à l'écran indiquant qu'ils ne tenteront pas une manœuvre d'espionnage tant qu'ils n'ont pas trouvé Ned Stark.

Enfin, il est également possible d'obtenir des informations quant aux positions de vos agents et de leur cible. Les opérations suivantes sont prévues à cet effet :

- `getDirection` renvoie la direction courante de Philip et Elizabeth (NORTH, SOUTH, WEST ou EAST) ;
- `getX` renvoie la coordonnée *x* de Philip et Elizabeth ;
- `getY` renvoie la coordonnée *y* de Philip et Elizabeth ;
- `getTargetX` renvoie la coordonnée *x* de Ned Stark ;
- `getTargetY` renvoie la coordonnée *y* de Ned Stark.

4 Phase Ia : Notion d'algorithme

Un *algorithme* est une description d'une procédure systématique décrivant des instructions à exécuter. L'encadré ci-dessous montre un exemple d'algorithme utilisé pour illustrer cette notion. Sur la gauche, vous trouverez une description de l'algorithme en *pseudo-code* et sur la droite en code Java. L'algorithme présenté traduit le comportement suivant : « *Tant que Philip et Elizabeth peuvent bouger, avancer d'une case dans leur direction courante* ».

==== ALGORITHME 1 ====

```
TANT QUE Philip et Elizabeth peuvent bouger FAIRE      while (canMove ())
|           {                                         move ();
|             avancer d'une case                      }
|           }
```

La figure 3 montre une situation initiale possible. Si on *exécute* l'algorithme 1 sur cette situation initiale, en supposant que Philip et Elizabeth sont orientés initialement vers l'est, il ne pourront avancer que de deux cases dans cette direction avant d'être bloqués par une forêt.

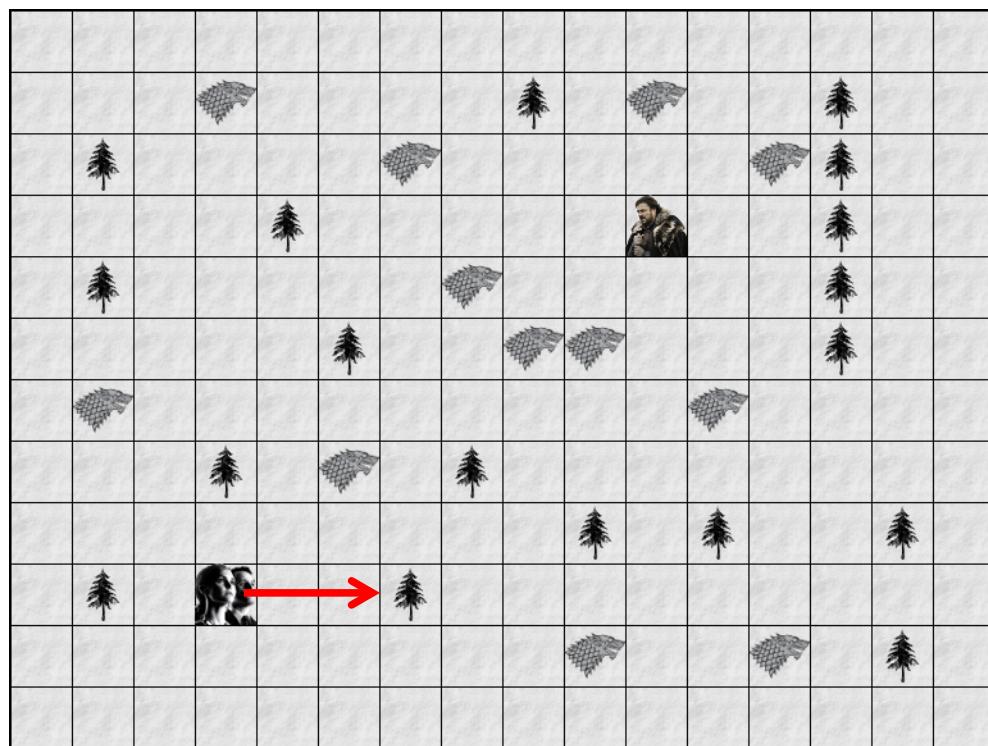


FIGURE 3. Un exemple de monde.

La construction illustrée ici est une *boucle*, celle-ci permet de répéter une séquence d'instructions tant qu'une certaine condition est satisfaite. Une autre construction de base utilisée pour construire des algorithmes est le *test* qui permet d'exécuter ou non une séquence d'instructions, une seule fois, en fonction d'une certaine condition.

L'encadré suivant montre un exemple d'algorithme plus complexe qui combine une boucle et un test. L'algorithme présenté traduit le comportement suivant : « *Tant que Philip et Elizabeth peuvent bouger, ils effectuent*

séquentiellement les deux actions suivantes. Ils commencent par avancer d'une case selon leur direction courante. Ensuite, deux situations peuvent se produire : soit ils peuvent bouger, ils avancent alors encore d'une case, soit ils sont coincés et tournent sur eux-mêmes de 90 degrés dans le sens anti-horlogique (vers leur gauche) ».

== ALGORITHME 2 ==

```
TANT QUE Philip et Elizabeth peuvent bouger FAIRE
|   |   avancer d'une case
|   |   SI Philip et Elizabeth peuvent bouger ALORS
|   |   |   avancer d'une case
|   |   SINON
|   |   |   tourner vers la gauche
|   |
|   }
while (canMove())
{
    move();
    if (canMove())
    {
        move();
    }
    else
    {
        turnLeft();
    }
}
```

La figure 4 montre une nouvelle situation initiale. Philip et Elizabeth étant libres d'avancer, ils vont d'abord avancer d'une case. Ensuite, étant toujours capables d'avancer, ils vont avancer d'une seconde case. Suite à la première *itération* de la boucle, ils auront donc avancé de deux cases en tout. Après cela, étant donné qu'ils peuvent toujours avancer, la boucle sera exécutée à nouveau. Ils avancent donc d'une case et, cette fois-ci, se retrouvent coincés par une forêt. Ils vont donc tourner sur eux-mêmes vers la gauche (en restant sur la même case). L'exécution continue alors jusqu'à ce qu'ils ne puissent plus avancer, comme indiqué sur la figure 4.

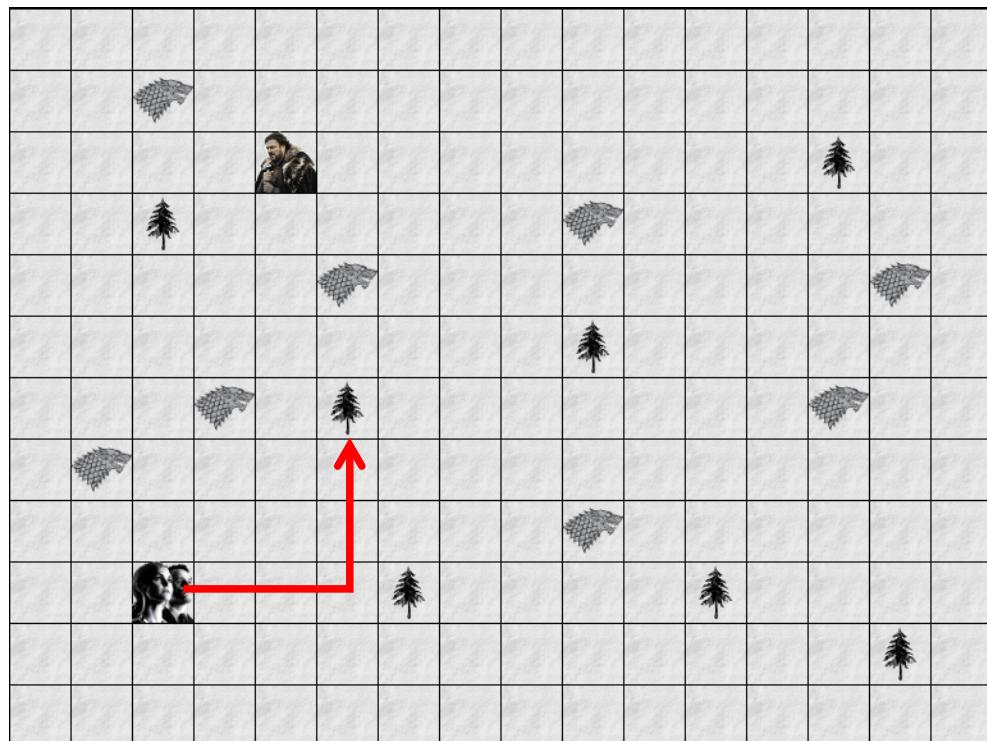
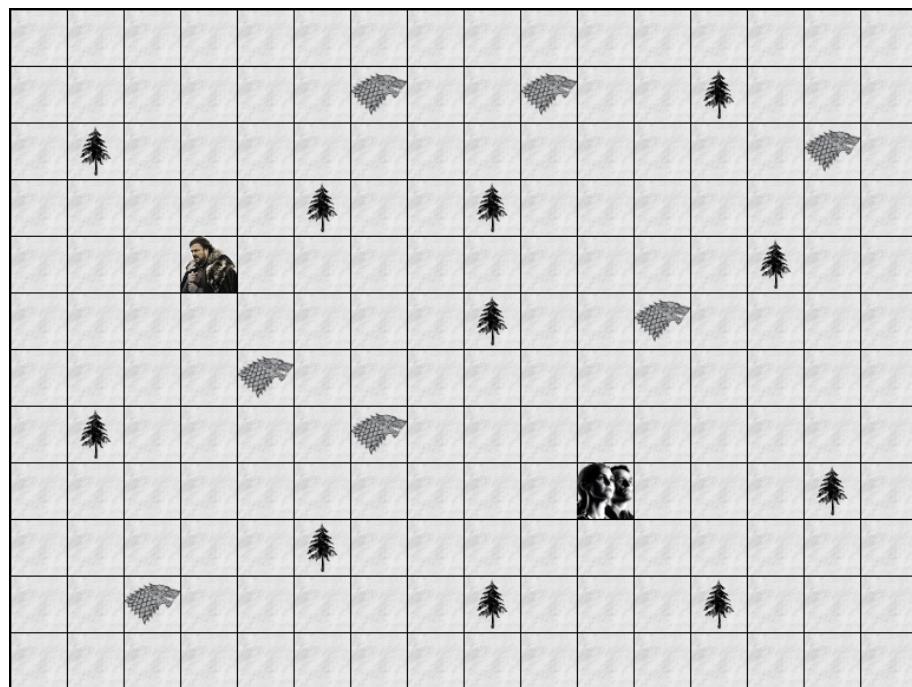


FIGURE 4. Un exemple de monde (2).

4.1 Exercices

- Pour la situation suivante, indiquez sur le dessin le résultat de l'exécution de l'algorithme 2, en supposant que Philip et Elizabeth sont initialement orientés vers l'est.



- Traduisez le comportement suivant en un algorithme :

- L'algorithme doit continuer à s'exécuter tant que Philip et Elizabeth ne se trouvent pas dans la dernière colonne tout à droite.
- Tant que Philip et Elizabeth peuvent se déplacer dans leur direction courante, il doivent avancer.
- Quand il ne savent plus avancer à cause d'une forêt ou parce qu'ils sont contre le bord, ils tournent sur eux-mêmes de 90 degrés vers la droite.
- Aucune attention particulière n'est portée aux loups.

Notes personnelles :

4.2 Rapport de mission

(à remettre au tuteur ce mardi à 16h15)

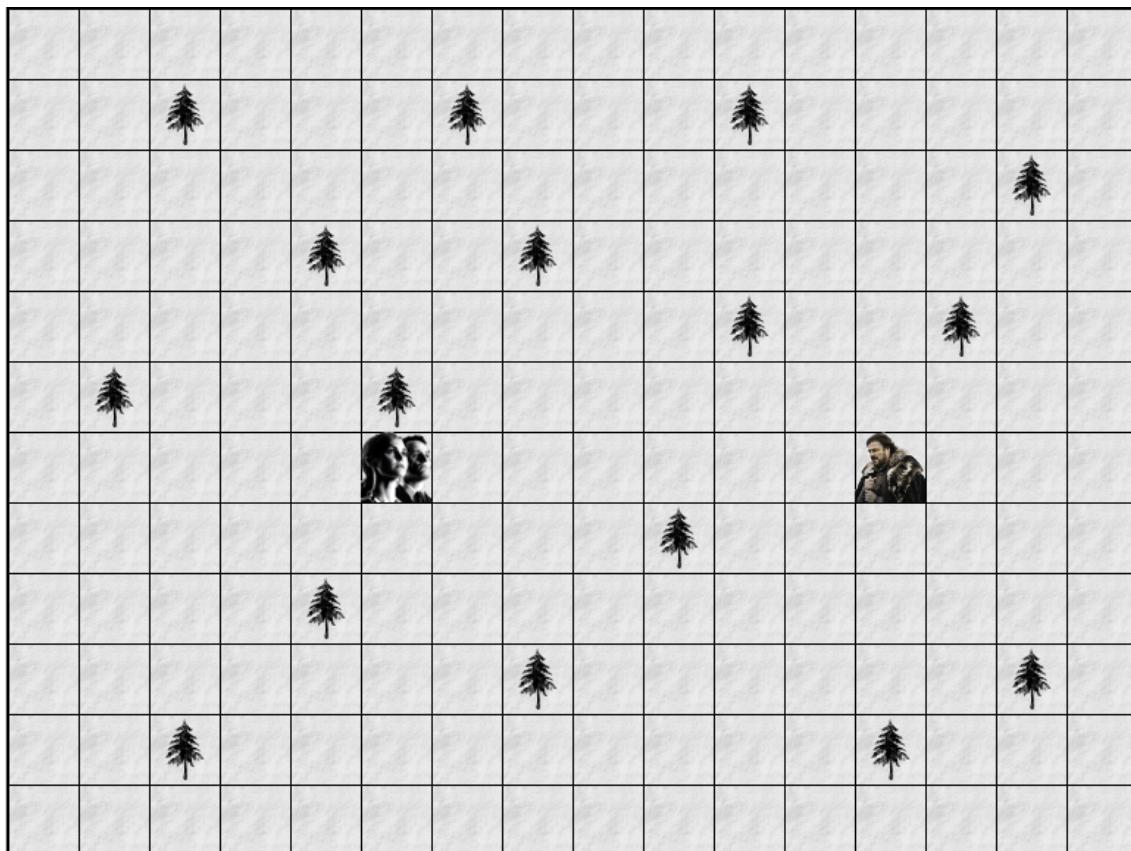
Prénom : _____ Nom : _____

- Soit l'algorithme 3 présenté dans l'encadré ci-dessous. Indiquez le résultat de l'exécution de l'algorithme 3 sur la situation suivante, en supposant que Philip et Elizabeth sont initialement orienté vers l'est.

== ALGORITHME 3 ==

```
TANT QUE Philip et Elizabeth N'ONT PAS trouvé Ned Stark FAIRE
|
|    avancer d'une case
|    avancer d'une case
|    SI Philip et Elizabeth peuvent bouger ALORS
|
|        tourner vers la gauche
|
|    SINON
|
|        tourner vers la droite
|
|
```

```
while (! isOnTarget ())
{
    move();
    move();
    if (canMove())
    {
        turnLeft();
    }
    else
    {
        turnRight();
    }
}
```



2. Définissez brièvement et avec vos propres mots une des notions suivantes, en utilisant éventuellement un exemple. (Les notions seront réparties entre les membres du groupe par le tuteur).

Algorithme / Opération primitive / Condition / Boucle / État (du « jeu ») / Pseudo-code

5 Phase Ib : Obstacles simples

Afin d'introduire les obstacles, une situation simplifiée vous est présentée. Dans celle-ci, le terrain ne contient aucun loup et n'est constitué que d'obstacles occupant exactement une case. De plus, les huit cases adjacentes à tout obstacle sont toujours libres ou contiennent Ned Stark. Il en est de même pour toutes les cases qui sont directement contre le bord du monde. En outre, Philip et Elizabeth se situent toujours tout à gauche du monde ($x = 0$) et Ned Stark se trouve sur la même horizontale, tout à droite du monde ($x = 15$). Enfin, Philip et Elizabeth sont initialement orientés vers l'est. La figure 5 illustre un exemple de cette situation simplifiée. Durant la première phase, vous travaillerez en équipes de cinq étudiants. L'objectif à atteindre à est d'aboutir à un *algorithme* permettant à Philip et Elizabeth de prendre le chemin le plus court pour rejoindre l'autre côté du terrain et espionner Ned Stark. Deux variantes de ce problème sont présentées plus bas dans l'énoncé.

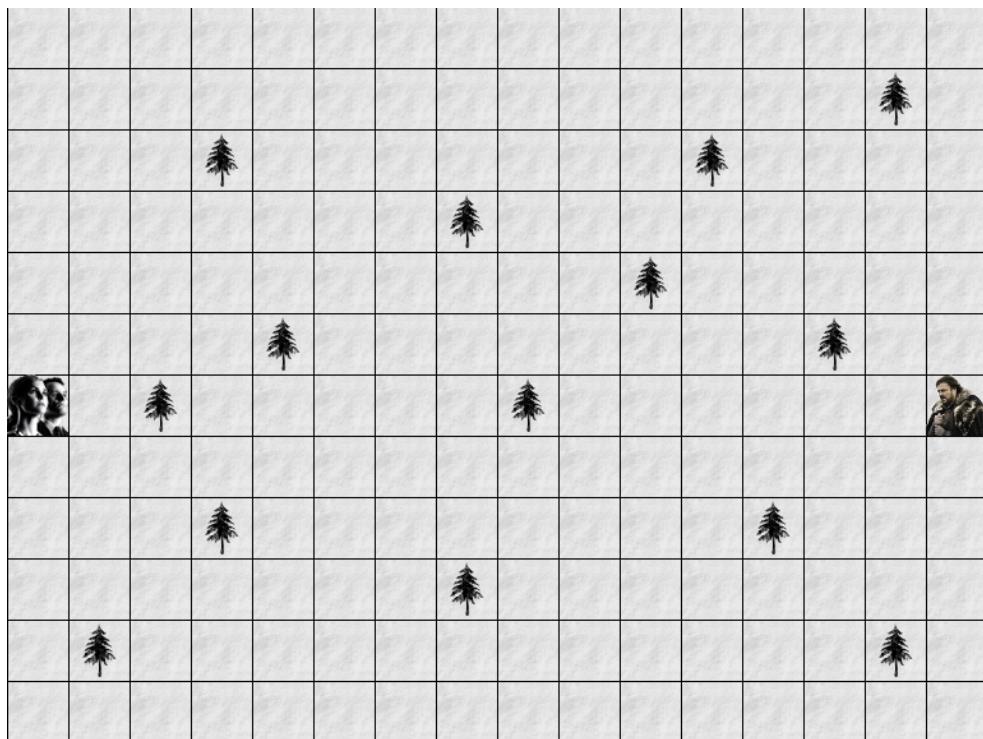


FIGURE 5. Exemple de terrain avec des obstacles simples.

5.1 Exemple d'algorithme

Lors des opérations sur le terrain, une des clés de la réussite est le travail en équipe. Une des clés du travail en équipe étant la communication, commencez pas discuter avec les membres de votre équipe afin d'être certain que tous les membres de l'équipe comprennent comment exécuter un algorithme. Comparez les réponses que vous avez trouvées pour les exercices de la phase Ia. En cas de réponses différentes, débattez au sein de l'équipe afin de trouver une bonne et unique réponse. Lorsque celle-ci a été validée par le tuteur, l'exercice peut continuer.

5.2 Résoudre le problème de base

Pour résoudre le problème de la phase Ib (illustré par la figure 5), l'algorithme doit permettre aux agents d'espionnée Ned Stark quelle que soit leur coordonnée y initiale et quelles que soient les positions des obstacles.

Commencez par proposer un algorithme simple, sans vous soucier du nombre de déplacements que vont faire Philip et Elizabeth (pour mémoire : le bord du terrain est exempt d'obstacles). Une fois la solution trouvée, améliorez-la en trouvant une stratégie plus efficace. Un algorithme est une description précise, complète et non-ambigüe d'une marche à suivre pour exécuter une certaine tâche. Il est donc important que celui-ci soit décrit correctement. Afin de tester ce point, voici comment va s'organiser cette phase :

1. Réfléchissez d'abord **individuellement** à un algorithme et rédigez-le ci-dessous. Dessinez la trajectoire que devraient suivre Philip et Elizabeth s'ils exécutaient l'algorithme que vous avez élaboré (utilisez la figure 8 en page 24) ;
2. Donnez ensuite votre algorithme à la personne se trouvant à votre gauche. Exécutez manuellement l'algorithme que vous avez reçu de la personne se trouvant à votre droite (utilisez la figure 9 en page 24) ;

Mon algorithme (Phase Ib)

Comparez ensuite les trajectoires obtenues par vous-même et par votre voisin. En cas de discordance, discutez avec votre voisin pour identifier la meilleure réponse.

5.2.1 Boucle et choix

Lors de la rédaction de votre algorithme, n'oubliez pas que vous avez deux *structures de contrôle* à votre disposition. Un comportement répétitif peut être obtenu grâce à une *boucle* :

TANT QUE condition FAIRE

Outre les boucles, il existe également les *choix* :

SI condition ALORS / SINON

La *condition* d'une boucle ou d'un choix doit être une opération de test. Pour rappel, à ce stade vous avez déjà trois test à votre disposition : `canMove`, `isInFrontOfWolf` et `isOnTarget`. Outre celles-ci, on peut également construire d'autres conditions à partir des opérations qui donnent de l'information sur la position de Philip et Elizabeth et de Ned Stark et de celle qui donne la direction courante de vos agents. Voici divers exemples :

==== EXEMPLES DE CONDITION ====

SI Ph. et El. sont orientés vers le sud ALORS	if (getDirection() == Direction.SOUTH)
SI Ph. et El. ne sont pas orienté vers l'est ALORS	if (! (getDirection() == Direction.EAST))
SI Ph. et El. se trouvent à gauche de Ned Stark ALORS	if (getX() < getTargetX())
SI Ph. et El. se trouvent plus haut que Ned Stark ALORS	if (getY() < getTargetY())

Pour mémoire : l'axe des *y* va du haut vers le bas, et donc, si vos agents se trouvent plus haut que Ned Stark, leur coordonnée *y* est inférieur à celle de Ned Stark. Notez que `!` permet de représenter une négation et que la négation d'une égalité peut également être notée avec `!=` qui signifie « *est différent de* ». Le deuxième exemple peut donc également s'écrire :

```
if (getDirection() != Direction.EAST).
```

5.2.2 Comparaison de solutions

Lorsque chaque membre de l'équipe a défini une solution possible, comparez celles-ci. Pour cela, choisissez au moins deux solutions différentes parmi celles élaborées au sein de votre équipe. Utilisez la grille de comparaison fournie à la page 18, en y ajoutant vos propres critères. Décidez ensuite quelle est, selon la comparaison, la meilleure solution. N'oubliez pas de prendre en compte le fait que la solution doit fonctionner pour toutes les configurations de monde satisfaisant les contraintes imposées, et pas seulement pour la seule configuration de la figure 5. Cette comparaison doit se faire **en travail de groupe**. Une fois l'algorithme de groupe décidé, recopiez-le dans le cadre de la page 19.

5.2.3 Grille de comparaison d'algorithmes

Critère	Sol 1	Sol 2	Sol 3	Sol 4
<p>Longueur – <i>L'algorithme est concis.</i></p> <p>Complexité – <i>L'algorithme reste simple, il n'y a pas trop de boucles ou tests imbriqués.</i></p> <p>Efficacité – <i>L'algorithme rejoint Ned Stark en faisant un minimum de mouvements, dans tous les cas.</i></p> <p>...</p>				

5.2.4 Solution du groupe

5.3 Première variante du problème initial

Le problème initial étant solutionné, il faut maintenant solutionner ses variantes. Pour solutionner les variantes, il n'est pas nécessaire de modifier la solution précédente. Pour ce faire, il est possible de définir un algorithme traitant la variation du problème pour après pouvoir utiliser la solution précédente. cette opération s'appelle une **décomposition en sous-problèmes**.

Les changements par rapport au problème initial sont les suivantes :

1. L'orientation initiale de Philip et Elizabeth est indéterminée ;
2. Les positions verticales initiales de Philip et Elizabeth et Ned Stark ne doivent plus coïncider. Philip et Elizabeth ne seront dès lors plus forcément sur la même horizontale que Ned Stark.

La figure 6 montre un exemple de configuration pour cette variante du problème. On peut y voir que Philip et Elizabeth sont initialement orientés vers le nord et qu'il se trouvent plus au sud que Ned Stark.

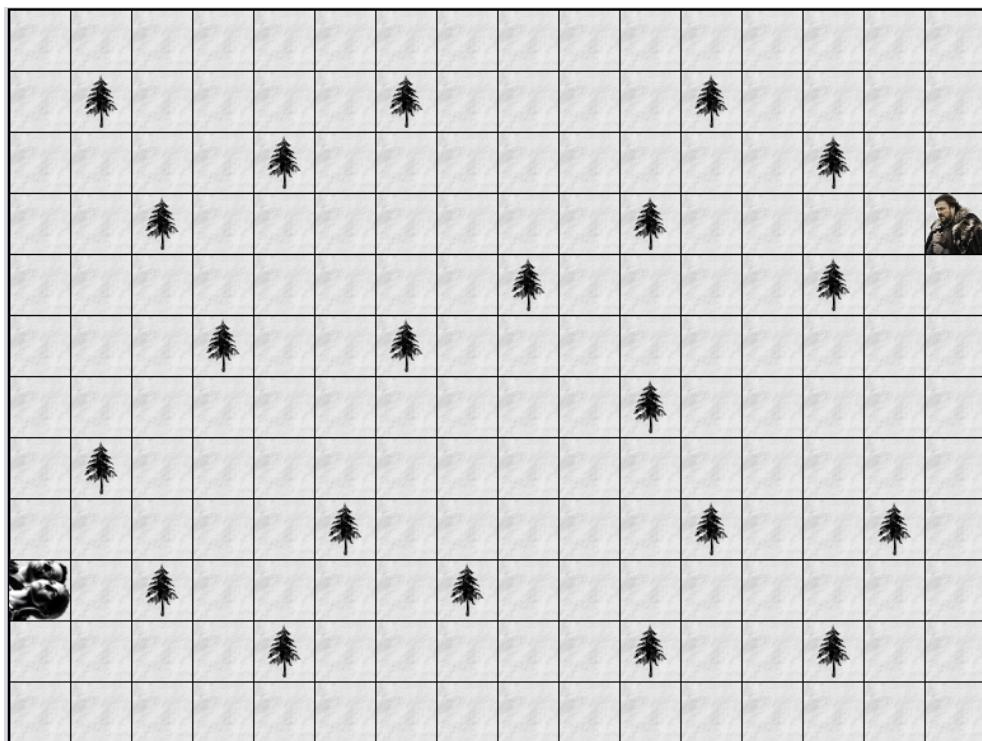


FIGURE 6. Exemple de monde avec des obstacles simples, pour la variante du problème initial.

Pour résoudre ce problème, il faut trouver deux algorithmes à exécuter avant et après celui développé pour le problème initial. La solution globale pour la variante du problème est donc décomposée en trois sous-problèmes illustrés par la figure 7.

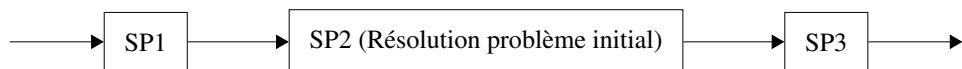


FIGURE 7. Décomposition en sous-problèmes, pour la variante du problème initial

Décrivez précisément les deux sous-problèmes SP1 et SP3 que vous allez définir. Ensuite, concevez un algorithme permettant de résoudre le problème. Attention à la distinction entre **problème** et algorithme.

SP1

SP3

Ceci fait, la solution ainsi obtenue est-elle la plus efficace ? Quels sont les avantages à avoir procédé comme cela pour trouver cette solution ? Pouvez-vous proposer une meilleure solution qui minimise le nombre de cases traversées par Philip et Elizabeth ? Le sous-problème SP2 peut être encore décomposé plus en détails. Comment pourrait-il être décomposé de manière plus fine ?

5.4 Seconde variante du problème initial

Dans cette seconde variante, Philip et Elizabeth ne seront initialement plus forcément situés tout à gauche du monde et Ned Stark ne sera plus forcément tout à droite. Par contre, Philip et Elizabeth seront toujours situés plus à l'ouest par rapport à Ned Stark. Que faut-il modifier dans votre solution de la variante précédente ? Idéalement, il ne faudrait modifier qu'un minimum de choses.

Le problème défini par cette nouvelle variante est un problème plus **général** que celui de la première variante. Autrement dit, le problème de la première variante est un cas particulier de celui de la deuxième variante. Ainsi, si vous trouvez une solution générale qui résout le deuxième problème, cette solution sera également directement valable pour la première variante.

Pour ce nouveau problème, il est possible de conserver le premier sous-problème SP1, vous devrez modifier légèrement SP2 et enfin vous devrez remplacer le sous-problème SP3 par un nouveau sous-problème SP4. Notez la très grande ressemblance entre SP2 et SP3. Comment pourriez-vous faire en sorte d'utiliser le même sous-problème à la place de SP2 et SP3 ?

5.5 Rapport de Mission

Pour la prochaine séance, rédigez **individuellement** un rapport à propos de la solution que vous proposez pour résoudre la seconde variante du problème initial. Ce rapport, qui fera **deux pages A4 au maximum**, doit contenir les éléments suivants :

- Une description de haut niveau de la solution (description en français de la solution et des sous-problèmes)
- Votre algorithme donné en pseudo-code ou en code Java (chaque sous-problème doit être défini séparément)

Indiquez vos nom et prénom sur le rapport et remettez-le à l'heure, aucun retard ne sera toléré. Vous pouvez utiliser le squelette de rapport présenté à la page suivante. Pour le travail de demain, **gardez une copie du rapport pour vous !**

Rapport à remettre pour le mercredi 21 septembre 2016, à 8h30 (début du cours).

5.6 Squelette de rapport

Prénom : _____ Nom : _____

Introduction

Le but de cette première phase du travail est de ...

Description générale

Ma proposition est de décomposer le problème principal en XXX sous-problèmes. Le premier sous-problème SP1 a pour but de ...

Je recopie bêtement le squelette de rapport qui m'est proposé ...

Algorithme

SP1 : TANT QUE Philip et Elizabeth ...

SP2 : SI Philip et Elizabeth ...

Conclusion

Ma proposition d'algorithme est la plus générale possible, mais il se trouve que si ... alors le chemin parcouru n'est pas le plus court possible puisqu'il est plus long. Le souci est que ...

5.7 Configuration de test

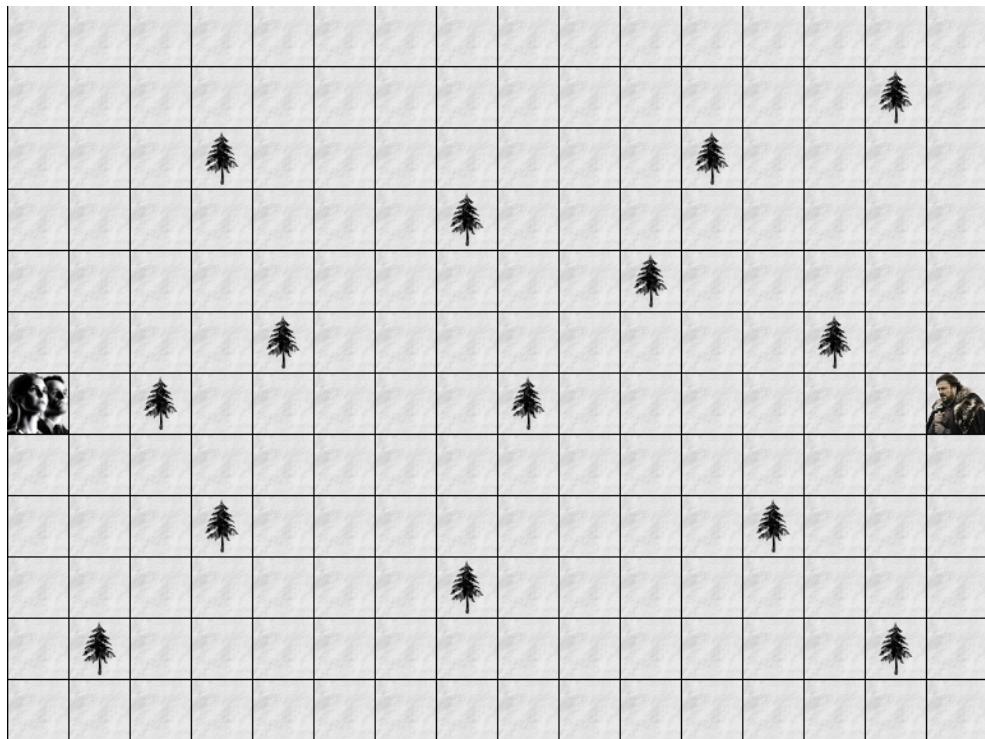


FIGURE 8. Résultat de l'exécution de mon algorithme.

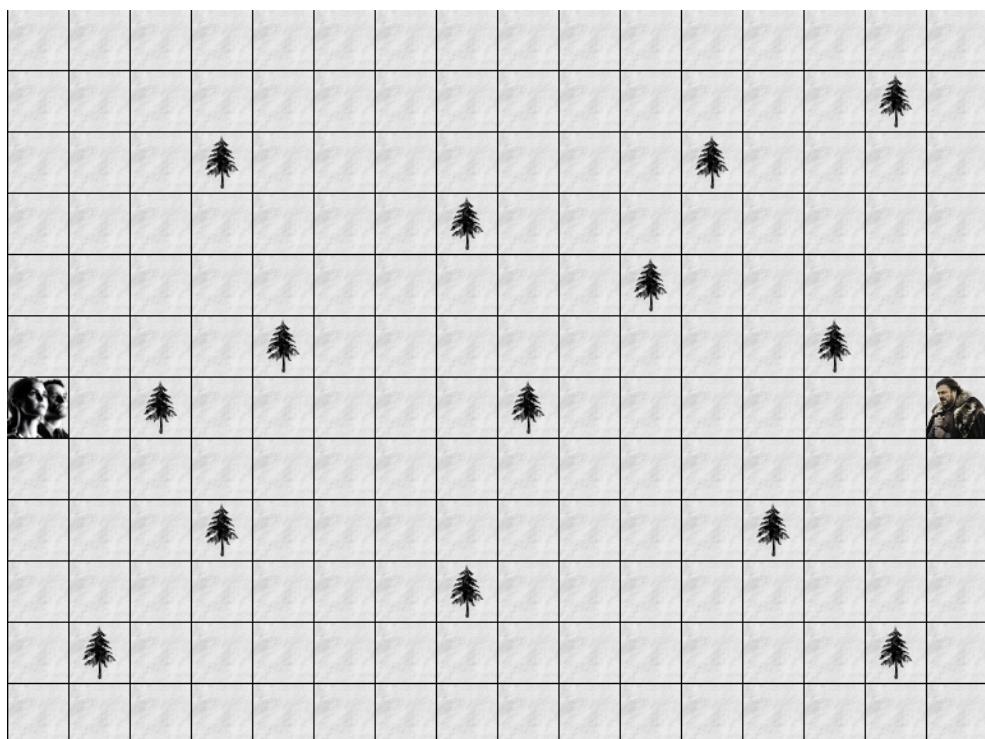


FIGURE 9. Résultat de l'exécution de l'algorithme de mon voisin de droite.

6 Phase II : Obstacles en ligne

Durant leur carrière au Centre, les agents changent régulièrement d'équipe. Afin d'évaluer la capacité d'adaptation des candidats, les équipes ont été modifiées. Afin que cette nouvelle équipe soit efficace, il est important de communiquer pour que l'équipe puisse travailler de manière cohérente. Pour ce faire, ses membres doivent discuter des différentes solutions apportées par ses différents membres. Pendant 20 minutes, lisez les rapports des membres de l'équipe et discutez-en afin d'identifier la meilleure solution parmi tout ce qui est proposé. Cette solution sera la base de votre travail d'aujourd'hui.

6.1 Obstacles en ligne

La mission du jour introduit deux nouveaux concepts : les **tests** et la **généralisation**. Le problème initial va être un peu enrichi. Les obstacles ne seront plus constitués de forêts isolées, mais ils prendront la forme de lignes de forêts. Un obstacle consiste donc en une suite de n forêts alignées selon l'horizontale ou la verticale. Pour le moment, ces lignes ne se touchent jamais et ne sont jamais contre les bords, c'est-à-dire qu'il est toujours possible de faire le tour complet des obstacles. La figure 10 montre une configuration possible du terrain.

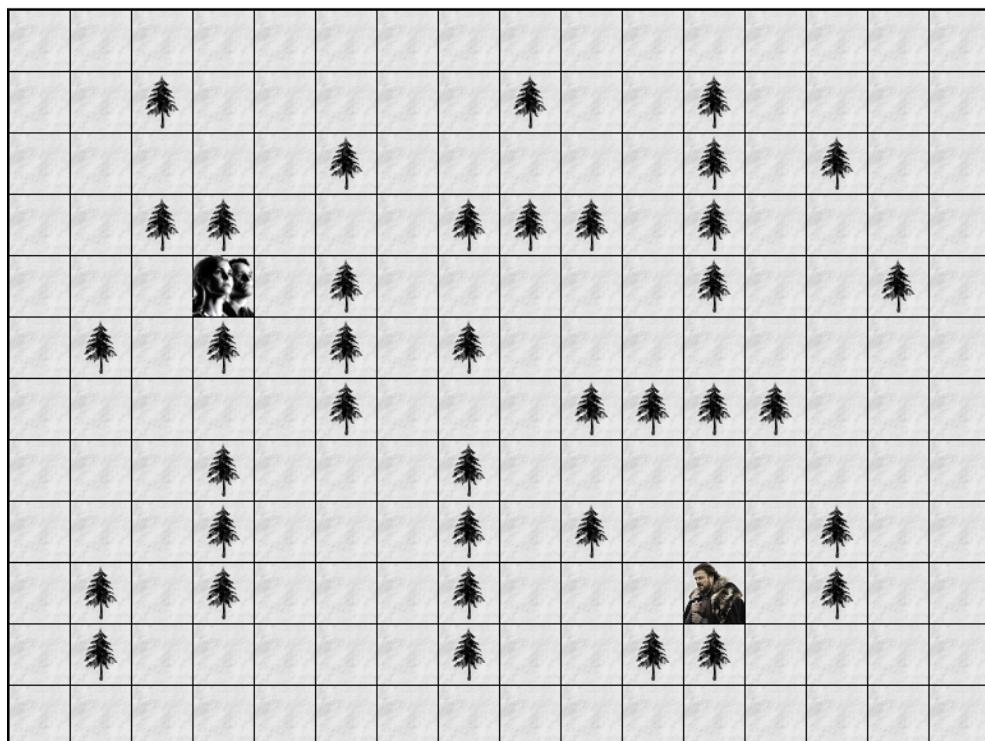


FIGURE 10. Exemple de monde avec des obstacles de type ligne.

L'objectif est de trouver une solution pour résoudre cette nouvelle variante du problème. Pour cela, il est obligatoire d'utiliser la décomposition en sous-problèmes présentée précédemment. Il est autorisé de ré-utiliser tous les sous-problèmes déjà été définis, la **réutilisabilité** étant un des avantages de la décomposition en sous-problèmes. Vous allez devoir définir un nouveau sous-problème qui permet de contourner un obstacle de forme « ligne de forêts ». Ce sous-problème sera ensuite réutilisé avec les autres sous-problèmes déjà résolus afin de

résoudre le problème général. Cette démarche permet en réalité de continuer à généraliser le problème initial. En effet, un obstacle constitué d'une seule forêt est en fait une ligne. La solution que vous trouverez ici devra donc pouvoir résoudre toutes les situations déjà étudiées jusqu'alors.

Divisez l'équipe en deux groupes, chacun de ces groupes se concentrera sur l'un des deux problèmes suivants : contourner une ligne horizontale ou une ligne verticale. Le contournement doit se faire jusqu'à atteindre une certaine position en x (resp. en y) de sorte que Philip et Elizabeth se retrouvent sur la même abscisse (resp. ordonnée) que Ned Stark.

Contourner ligne (horizontale ou verticale)

Une fois que chaque groupe a trouvé une solution, confrontez-les et analysez s'il n'est pas possible d'écrire un seul algorithme qui gère à la fois les lignes horizontales et verticales.

6.2 Définition des tests

Une fois l'algorithme défini, il est nécessaire de le tester. Pour cela, il faut imaginer un *jeu de tests*, chaque test correspondant à une configuration différente du monde. De bons tests doivent couvrir les différents cas d'exécutions possibles, en se basant sur la description du problème uniquement (vous vous assurerez d'avoir un test avec des trous isolés, en lignes, etc.). Un autre type de test doit couvrir les différentes parties de votre algorithme (par exemple, si vous avez une condition, il faudra un test pour lequel elle est vraie et un pour lequel elle est fausse).

Réfléchissez à des tests à exécuter sur votre algorithme en fin de semaine, lorsque vous coderez tout cela sur ordinateur. La figure 11 illustre quatre exemples de test à envisager. Chaque membre de l'équipe propose une situation de test en expliquant le cas particulier que son test vise à tester.

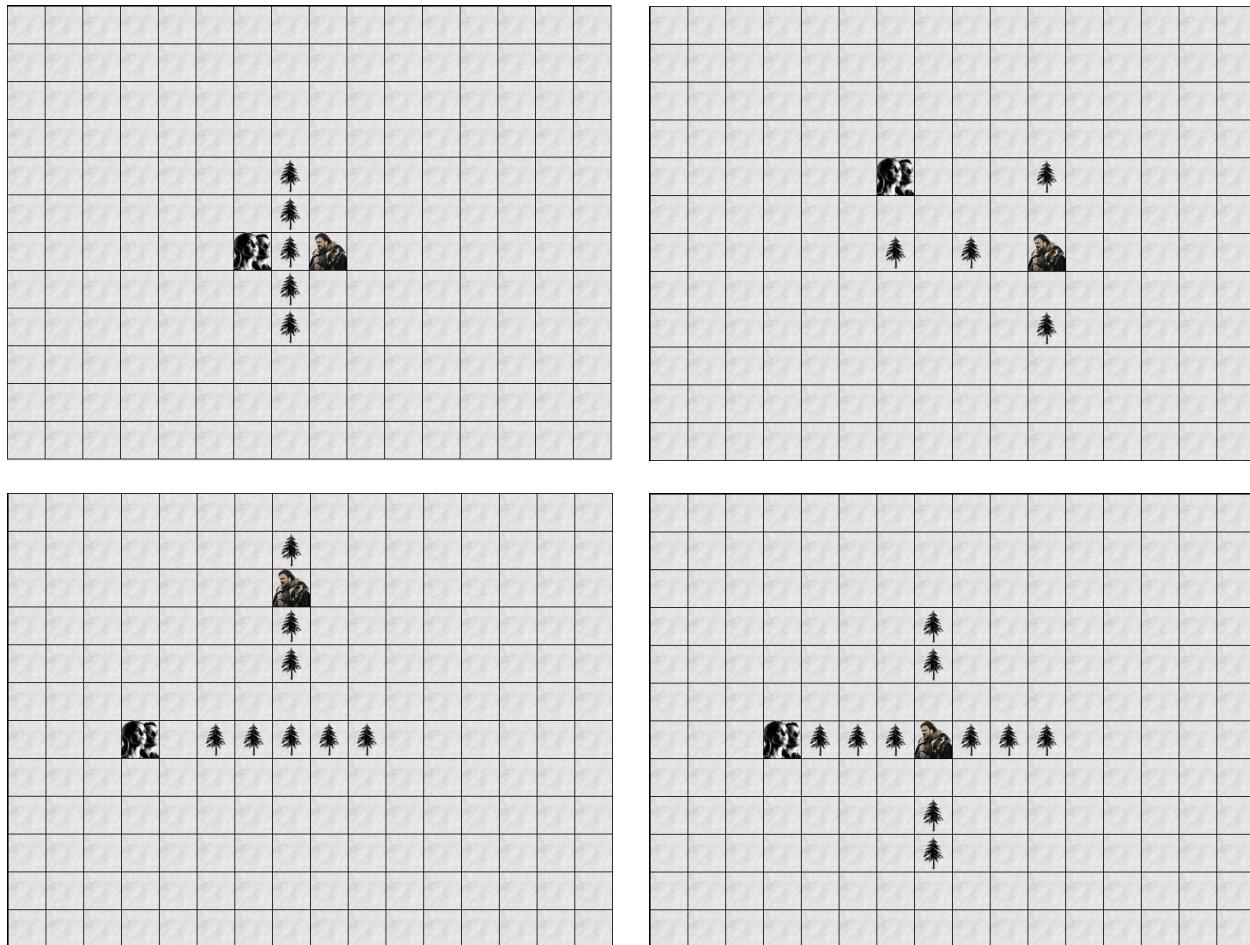


FIGURE 11. Quatre exemples de test pour le monde avec des obstacles de type ligne.

6.3 Avec des loups en plus...

Sentant le danger, les loups chargés de protéger Ned Stark se sont réveillés. Philip et Elizabeth peuvent vérifier si il n'y a pas de loup devant eux en utilisant l'opération `isInFrontOfWolf`. Si vos agents sont devant un pirate et que vous leur ordonnez de faire un `move`, confiant dans vos ordres, il avanceront sans repérer le loup et finiront mangés. L'exécution du programme sera donc terminée.

Dans un premier temps, les loups seront toujours isolés, c'est-à-dire qu'il n'y a pas d'autres obstacles (loup ou forêt) dans les huit cases adjacentes. Il vous faut trouver une solution élégante pour résoudre cette nouvelle variante du problème, dont un exemple de configuration se trouve sur la figure 12. Si vous trouvez une bonne solution, qui fait preuve de **généralisation**, tous les tests établis pour la variante précédente du problème devraient être suffisants pour cette nouvelle variante.

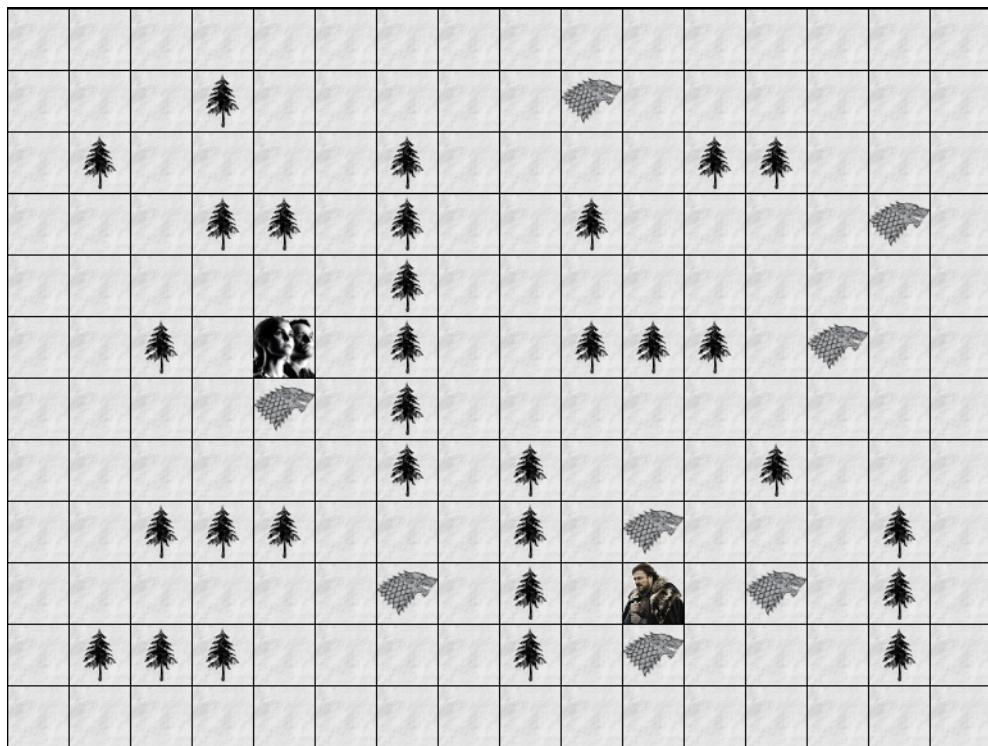


FIGURE 12. Exemple de monde avec des obstacles de type ligne et des loups.

6.4 Préparation pour le test

Revenons un moment sur quelques concepts clés qui ont été vus jusqu'à présent. Êtes-vous, chacun, capables d'expliquer avec vos propres mots les concepts suivants, et de les rattacher à un exemple traité ?

Opération primitive, algorithme, sous-problème, exécution, structure de contrôle, boucle, problème, condition, programme, décomposition en sous-problèmes, if/else, while

Que pouvez-vous dire à propos de l'algorithme suivant ? Qu'a-t-il de spécial ?

```
while (getDirection() != Direction.EAST)
{
    if (getDirection() != Direction.NORTH)
    {
        turnRight();
    }
    else
    {
        turnLeft();
    }
}
```

Comment peut-on réécrire les conditions suivantes de manière plus claire ?

```
! (getX() < getTargetX())
getX() - getTargetX() < 0
! (getY() - 2 * getTargetY() != - getTargetY())
```

6.5 Rapport de Mission

Pour la séance prochaine, vous devez rédiger **en groupe** un rapport à propos de la solution que vous proposez pour résoudre le problème de la phase II. Ce rapport, qui fera **quatre pages A4 au maximum**, doit contenir les éléments suivants :

- Une description de haut niveau de votre solution (description en français de la solution et des sous-problèmes)
- Votre algorithme donné en pseudo-code ou en code Java (chaque SPxx doit être défini séparément)
- Au moins deux tests accompagnés d'une description de ce que les tests permettent de vérifier

Indiquez les noms et prénoms des membres du groupe sur le rapport et remettez-le à l'heure. Aucun retard ne sera toléré. Vous pouvez utiliser le même squelette de rapport que celui du rapport précédent. Pour le travail de demain, **gardez une copie du rapport pour vous, une par étudiant !**

**Rapport à remettre pour le mercredi 21 septembre 2016,
à 18h30 au plus tard, au secrétariat INGI.**

Notes personnelles :

7 Phase III : Positions initiales quelconques et obstacles en L

Le problème du jour est plus compliqué, sa résolution implique l'utilisation des notions de **spécification**, **d'abstraction** et de **décomposition en sous-problèmes**.

Les positions initiales de vos agents et de Ned Stark ne sont maintenant plus du tout contraintes, ils peuvent donc se trouver n'importe où sur une case libre (sans forêt ou loup). La figure 13 montre une configuration possible d'un tel terrain.

Pour résoudre ce problème compliqué, il faut commencer par se concentrer sur le problème général en lui trouvant une solution basée sur des sous-problèmes clairement définis (sans nécessairement en donner l'algorithme complet).

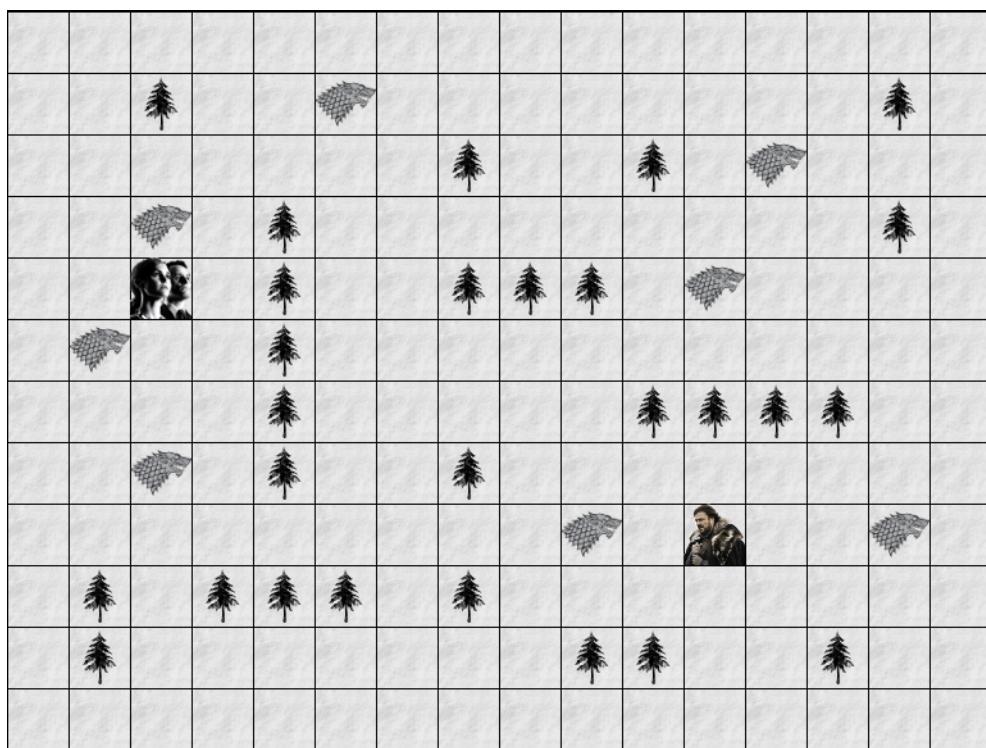


FIGURE 13. Exemple de monde avec les positions initiales indéterminées pour les agents et Ned Stark.

Ce problème doit donc être découpé en sous-problèmes. Dans un premier temps, ne cherchez pas à savoir comment résoudre les sous-problèmes. Commencez par définir ce qu'ils font et quelles sont les conditions initiales qui doivent être satisfaites afin de pouvoir les utiliser. Ceci est précisément décrit par les **spécifications** du sous-problème.

Deux sous-problèmes ont déjà été définis, et vous pouvez donc les utiliser pour résoudre le problème général. Ces sous-problèmes vont faire en sorte que Philip et Elizabeth effectuent un déplacement horizontal, pour se placer sur la même coordonnée x que celle de Ned Stark. Le premier sous-problème est applicable lorsque Philip et Elizabeth se situent à l'ouest de Ned Stark et le second lorsqu'ils sont à l'est de Ned Stark.

Voici la spécification (description précise) de l'un des deux sous-problèmes :

==== getToNedStarkXFromEast ====

```
<> Conditions à satisfaire :  
- Philip et Elizabeth se trouvent à l'est de Ned Stark.  
- Philip et Elizabeth sont orientés vers l'ouest.  
  
<> Résultat du sous-problème :  
- Philip et Elizabeth ont effectués un trajet sur le terrain, et ils se trouvent sur une case  
de coordonnées ( $x, y$ ) où  $x$  est la même abscisse que celle de Ned Stark  
- Philip et Elizabeth sont orientés vers l'ouest.
```

Le second sous-problème, `getToNedStarkXFromWest`, est similaire à la différence que Philip et Elizabeth doivent se trouver à l'ouest de Ned Stark et être orientés vers l'est.

Pouvez-vous indiquer où se situeront Philip et Elizabeth après exécution du premier sous-problème (`getToNedStarkXFromEast`), dans la situation dépeinte à la figure 13 ? Et où se situeraient Philip et Elizabeth si on exécutait plutôt le second sous-problème (`getToNedStarkXFromWest`) ?

7.1 Résoudre le problème

Vous devez maintenant proposer une solution pour résoudre le problème de la phase III. Pour ce faire, vous pouvez utiliser tous les sous-problèmes qui ont été définis lors des phases précédentes, ainsi que les deux sous-problèmes `getToNedStarkXFromEast` et `getToNedStarkXFromWest`. Vous pouvez également définir des nouveaux sous-problèmes, sans en définir les algorithmes, mais en donnant leurs spécifications précises.

Mon algorithme (Phase III)

7.2 Implémenter un sous-problème

Maintenant que le problème principal est implémenté, afin de pouvoir exécuter l'algorithme global, il faut implémenter tous les différents sous-problèmes. Pour cela, écrivez l'algorithme de l'un des deux sous-problèmes `getToNedStarkXFromEast` ou `getToNedStarkXFromWest`. Pour faire cela, vous devez faire complètement **abstraction** du problème principal, et ne prendre en compte que les spécifications du sous-problème choisi. Travaillez en deux groupes, un par sous-problème.

Mon algorithme (Phase III - Le sous-problème)

7.3 Minimiser les déplacements

Rappelez-vous que l'hiver venant, Philip et Elizabeth doivent retrouver Ned Stark, en se déplaçant le moins possible. La solution proposée pour résoudre le problème permet-elle de satisfaire cela ? Si oui, expliquez pourquoi. Si non, que pourriez-vous faire afin que ce soit le cas ?

7.4 Réflexion

Un sous-problème qui possède trop de conditions à satisfaire afin de pouvoir l'utiliser peut s'avérer assez contraignant. La tendance consiste donc à minimiser le nombre de conditions à satisfaire. Comment pourriez-vous adapter le sous-problème `getToNedStarkXFromEast` ou `getToNedStarkXFromWest` pour lequel vous venez d'écrire un algorithme afin d'éliminer la seconde condition à satisfaire (l'orientation imposée à Philip et Elizabeth) ?

Une autre question mérite votre attention. Deux sous-problèmes ont été définis, leur première condition à satisfaire ne diffère que par la position initiale de Philip et Elizabeth. Comment serait-il possible d'intégrer, proprement, un sous-problème `getToNedStarkX` dont les spécifications seraient :

```
== getToNedStarkX ==
<> Conditions à satisfaire :
    - Aucune

<> Résultat du sous-problème :
    - Philip et Elizabeth ont effectué un trajet dans le monde, et ils se trouvent sur une case
      de coordonnées (x, y) où x est la même abscisse que celle de Ned Stark
    - Philip et Elizabeth sont orientés vers l'Est.
```

7.5 Obstacles touchant le bord

À partir de maintenant, les obstacles pourront toucher le bord du monde. La figure 14 montre une configuration possible. Adaptez la solution que vous avez actuellement pour résoudre cette nouvelle variante. Essayez de garder votre solution la plus propre et décomposée en sous-problèmes possible. Tout nouveau sous-problème devra être accompagné d'une spécification.

7.6 Obstacles en L

Dans la dernière variante à solutionner, un nouveau type d'obstacle fait son apparition : les obstacles en L. Ces obstacles sont constitués de trois forêts et il existe évidemment quatre possibilités d'orientation possible pour ces L. La figure 15 montre un monde tout à fait général constitué, outre les obstacles isolés ou en lignes, de tous les types possibles d'obstacles en L. Définissez un algorithme permettant de contourner un obstacle en L. Pour ce faire, supposez que Philip et Elizabeth sont orientés vers l'est et que l'obstacle se trouve à leur droite. Spécifiez et définissez le sous-problème résolvant ce problème. Généralisez ensuite pour les autres situations possibles.

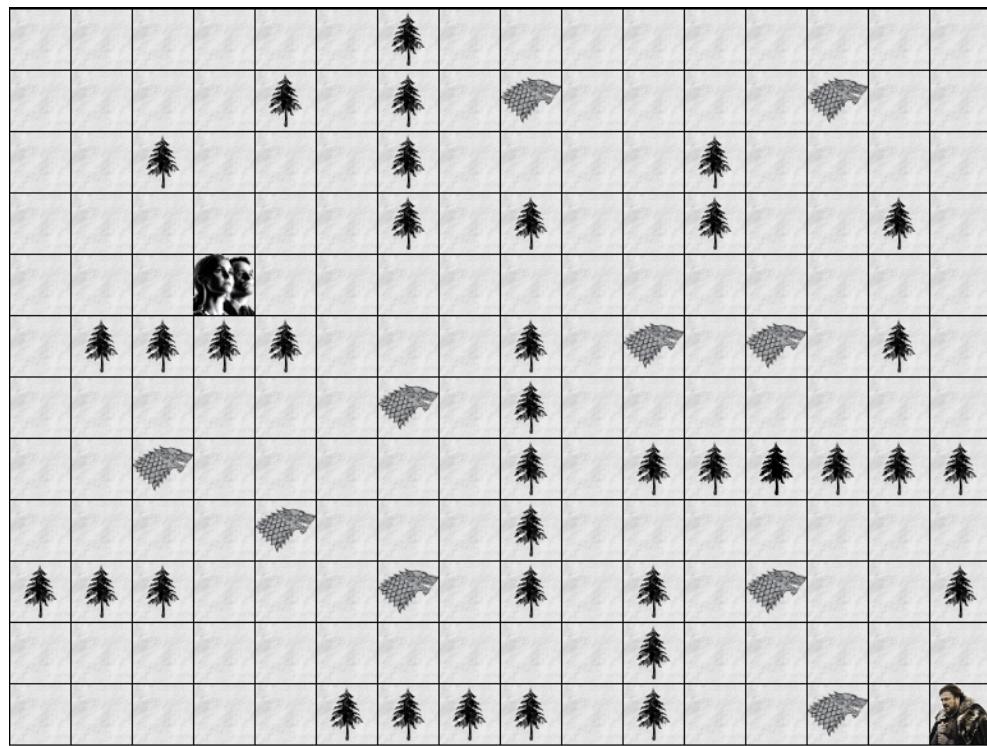


FIGURE 14. Exemple de monde avec des obstacles de type ligne pouvant toucher le bord et des loups isolés.

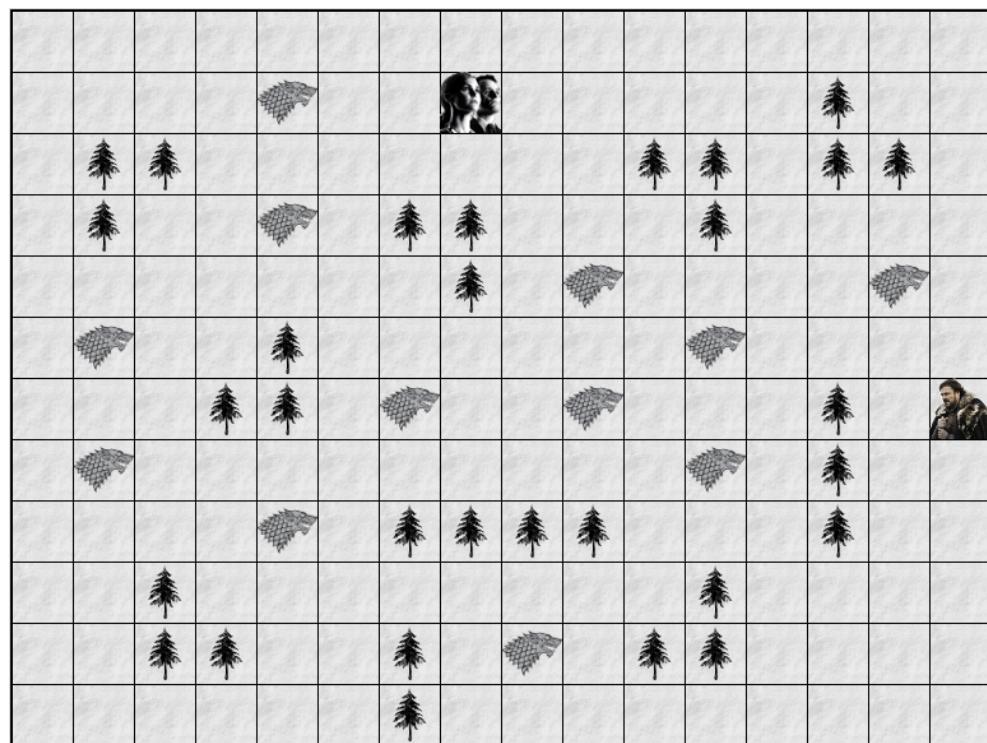


FIGURE 15. Exemple de monde avec des obstacles en L.

Notes personnelles :

7.7 Test

Prénom Nom :

Question 1

Soit l'algorithme suivant :

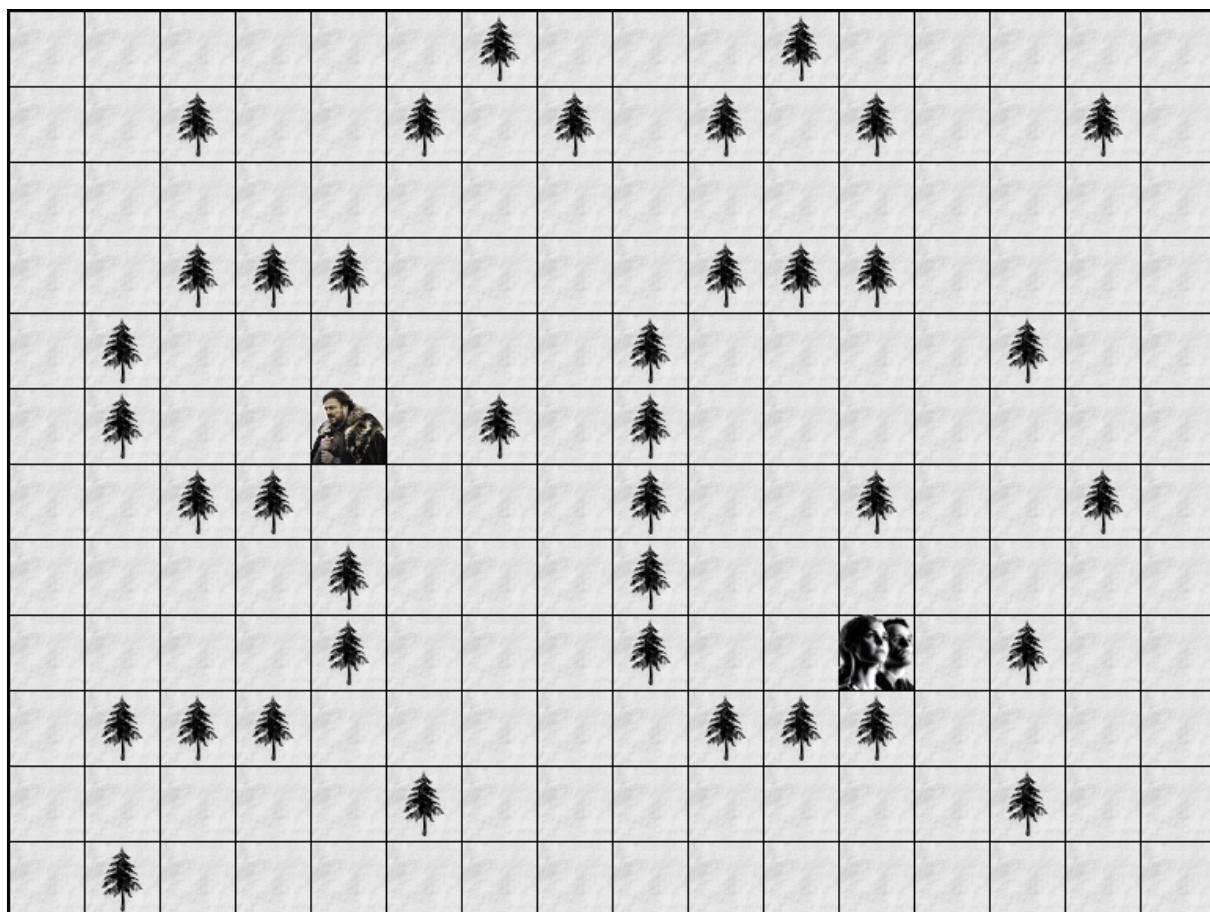
```

tourner vers la gauche
TANT QUE Philip et Elizabeth peuvent bouger FAIRE
|
|   avancer d'une case
|   avancer d'une case
|   tourner vers la droite
|   avancer d'une case
|   tourner vers la gauche
|
turnLeft();
while (canMove())
{
    move();
    move();
    turnRight();
    move();
    turnLeft();
}

```

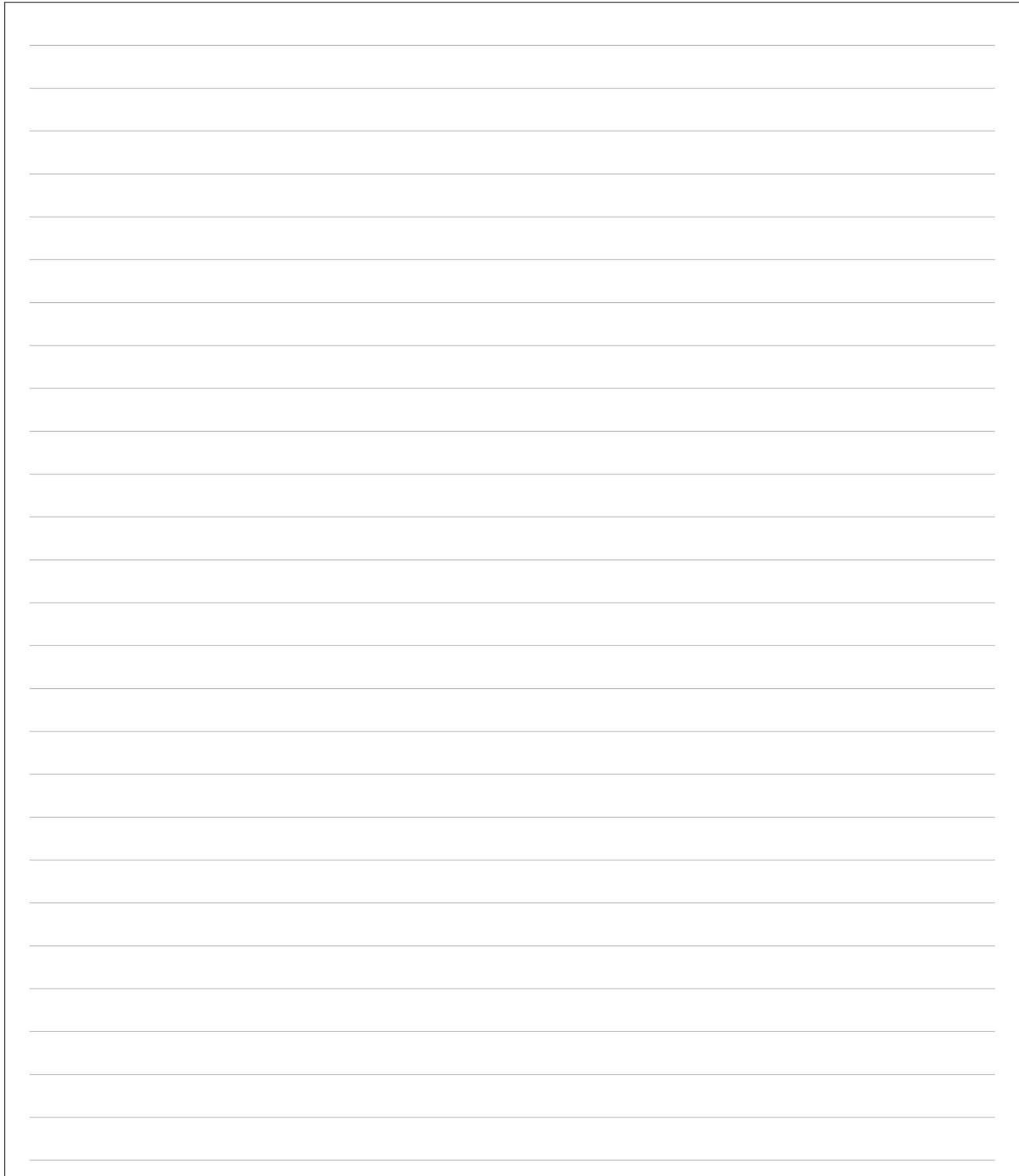
Tracez le chemin complet parcouru par Philip et Elizabeth, suite à l'exécution de l'algorithme, dans le monde suivant, en les supposant initialement orientés vers l'est.

Quelle est leur orientation après exécution de l'algorithme ? _____



Question 2

Soit la situation simplifiée suivante. Philip et Elizabeth se trouvent à gauche d'une ligne verticale de forêts ne touchant pas le bord du monde. Ned Stark se trouve à droite de ce mur de forêts. Écrivez un algorithme permettant à Philip et Elizabeth d'espionner Ned Stark, en faisant le moins de déplacements possible. Décomposez votre algorithme en sous-problèmes.



Question 3 (série A)

Expliquez brièvement, dans vos propres mots, le concept suivant. Citez également un moment où ce concept vous a été utile durant cette semaine, en commentant brièvement.

sous-problème

Question 4 (série A)

Simplifiez la condition suivante.

```
! (getX() - getNedStarkX() >= 0)
```

Question 3 (série B)

Expliquez brièvement, dans vos propres mots, le concept suivant. Citez également un moment où ce concept vous a été utile durant cette semaine, en commentant brièvement.

spécification

Question 4 (série B)

Simplifiez la condition suivante.

```
! (getY() - getNedStark() == 0)
```

Question 3 (série C)

Expliquez brièvement, dans vos propres mots, le concept suivant. Citez également un moment où ce concept vous a été utile durant cette semaine, en commentant brièvement.

généralisation

Question 4 (série C)

Simplifiez la condition suivante.

```
! (getX() - getX() == 0)
```

Question 3 (série D)

Expliquez brièvement, dans vos propres mots, le concept suivant. Citez également un moment où ce concept vous a été utile durant cette semaine, en commentant brièvement.

boucle

Question 4 (série D)

Simplifiez la condition suivante.

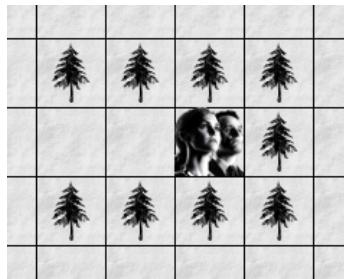
```
! (getTargetY() - getY() != 0)
```

Question 5

Quelles instructions faut-il donner à Philip et Elizabeth pour être certain qu'une fois exécutées, ils soient orientés vers le nord, peu importe la situation initiale ?

Question 6

Philip et Elizabeth sont coincés dans une cuvette en U horizontale (voir illustration ci-dessous). En supposant que vous ne connaissez pas leur orientation initiale, écrivez un algorithme qui leur permette de sortir de la cuvette. Philip et Elizabeth doivent se retrouver, après exécution de votre algorithme, à droite de la cuvette.



8 Phase IV : Implémentation

Vous allez maintenant traduire les différents algorithmes que vous avez définis en un programme informatique utilisable et exécutable sur ordinateur. Ceci va vous permettre de les tester et surtout de participer au concours qui aura lieu ensuite.

Vous allez travailler seul sur une machine. Vous pouvez bien entendu vous faire aider par d'autres. Reprenez l'algorithme que vous avez écrit lors de la séance précédente et codez-le. Vous pourrez tester vos programmes avec les exemples donnés tout au long de la semaine et également créer vos propres exemples. Vous avez **deux heures** pour réaliser cette phase, il n'est donc pas inutile de bien vous y préparer et de déjà réfléchir à la traduction de l'algorithme version papier en français vers un algorithme en Java entre jeudi et vendredi. Durant cette phase, des assistants circuleront dans les salles uniquement pour répondre à des questions techniques.

8.1 Lancement de Greenfoot

Le programme qu'on va utiliser pour cette phase est *Greenfoot* que vous trouverez sur les machines des salles informatiques qui sont à votre disposition. Vous pouvez également télécharger le programme pour l'installer sur votre machine personnelle depuis le site officiel : <http://www.greenfoot.org/>. La figure 16 montre l'écran de démarrage du programme.

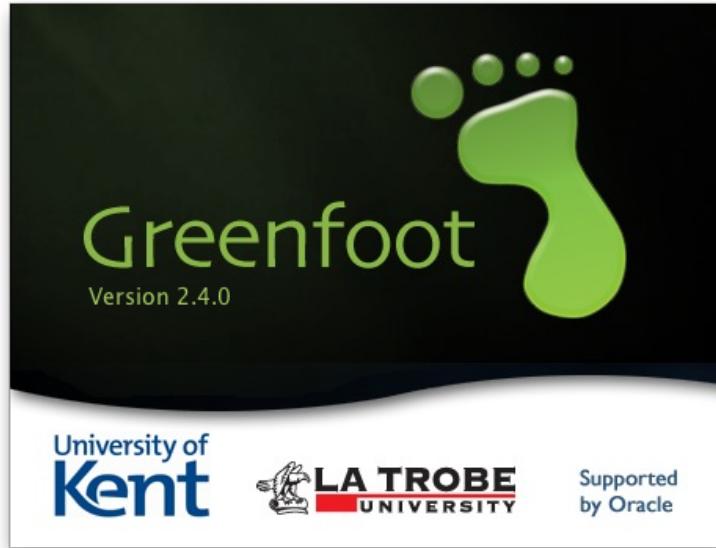


FIGURE 16. Lancement de Greenfoot.

Une fois le programme démarré, vous devez charger le projet. Rendez-vous sur le site du cours sur iCampus et téléchargez le fichier *GameOfSpies.zip* que vous pouvez trouver dans Documents et liens. Décompressez ensuite l'archive ZIP dans votre répertoire personnel. Dans le programme Greenfoot, sélectionnez le menu Scenario > Open... et indiquez le chemin vers le dossier que vous avez décompressé. Le projet se lance et vous vous retrouvez face à la fenêtre de la figure 17.

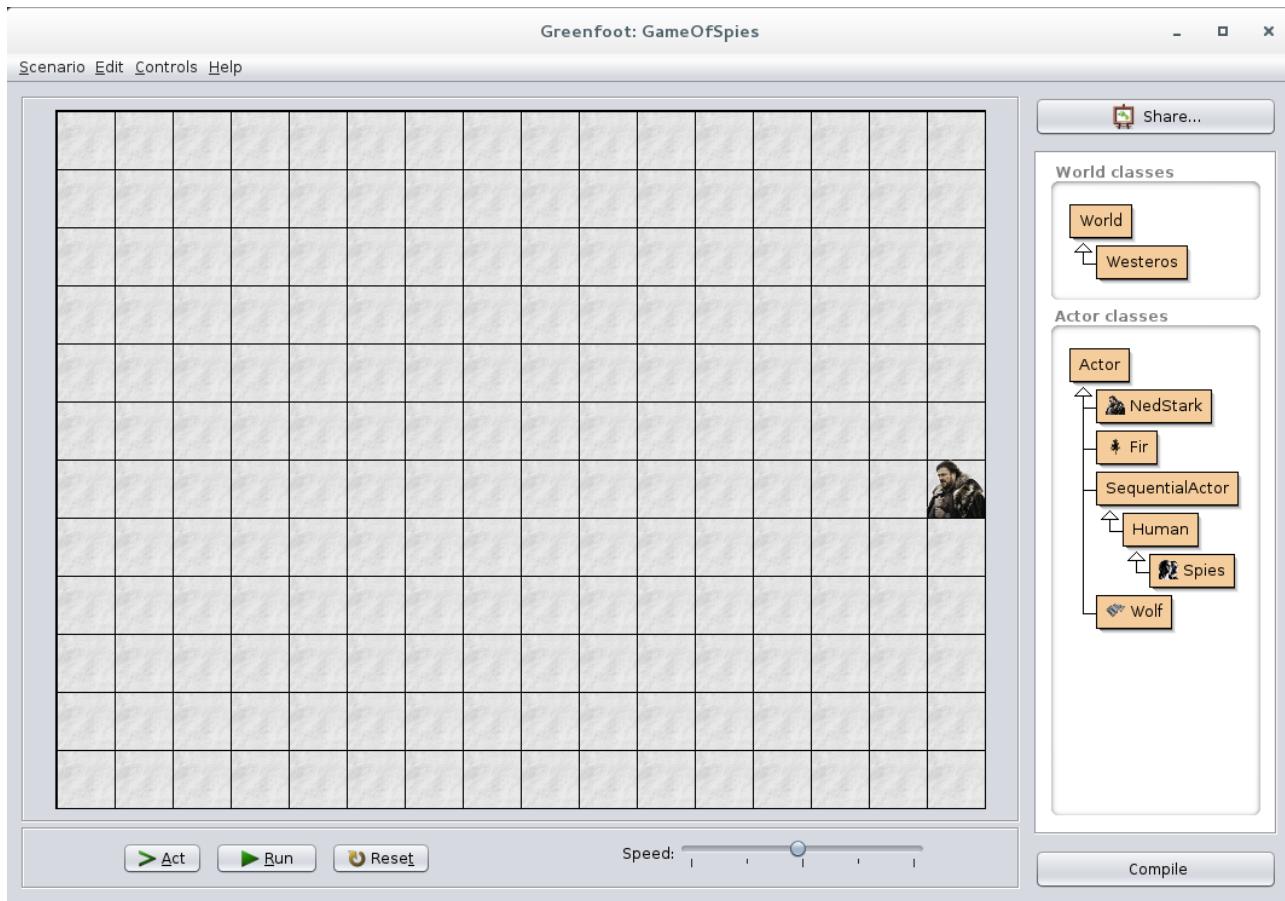


FIGURE 17. Le projet de la première semaine ouvert dans Greenfoot.

8.2 Utilisation de Greenfoot

Pour ajouter Philip et Elizabeth dans le monde, cliquez une fois sur Philip et Elizabeth à droite. Ensuite, tout en maintenant la touche **[Shift ↑]** enfonce, cliquez dans le monde à l'endroit où vous voulez placer Philip et Elizabeth. Vous pouvez procéder de la même manière pour placer des trous ou des pirates.

Vous pouvez ensuite exécuter le programme en une fois en cliquant sur le bouton Run en bas à gauche ou alors exécuter le programme pas à pas en utilisant le bouton Act. Si vous souhaitez obtenir des mondes d'exemple tout faits, il vous suffit de faire un clic droit sur le monde et sélectionner ensuite buildScenarioXXX().

8.3 Définir le comportement de Philip et Elizabeth

Pour définir le comportement de Philip et Elizabeth, double-cliquez sur Philip et Elizabeth à droite. Une fenêtre s'ouvre, il s'agit de l'éditeur de code Java. Vous pouvez y voir du code, en particulier :

```
protected void behave()
{
    // Placer le comportement de Philip et Elizabeth ici
}
```

C'est à cet endroit que vous devez écrire votre algorithme. À chaque fois que vous modifiez ce fichier, n'oubliez pas de l'enregistrer (menu Class > Save), et ensuite de le compiler (menu Tools > Compile). Vous pouvez ensuite exécuter le programme.

8.4 Définir un sous-problème

Pour définir un sous-problème que vous pourrez réutiliser, utilisez la syntaxe suivante où nom est le nom du sous-problème que vous pouvez choisir comme bon vous semble (n'utilisez que les lettres a-zA-Z, les chiffres 0-9 et le tiret de soulignement).

```
private void monSousProbleme()
{
}
```

Pour appeler un sous-problème, il vous suffit d'écrire monSousProbleme () ; .

8.5 Soumission du travail

Soumettez votre travail par Moodle, dans la section Travaux. Vous devez poster une solution pour votre groupe, pour 16h00. Vous n'avez pas beaucoup de temps, alors exploitez bien les deux heures de codage à votre disposition.

Attention, l'heure du serveur de Moodle n'est pas forcément exactement synchronisée avec celle de votre montre ou de votre ordinateur; prévoyez une marge de sécurité !

9 Concours

En fin de séance, un concours sera organisé entre les différents programmes qui auront été soumis. Nous allons exécuter vos algorithmes sur des nouveaux mondes inconnus. Le nombre de pas effectués par Philip et Elizabeth sera utilisé comme premier critère. En cas d'égalité, le nombre d'opérations move, turnLeft et turnRight sera comptabilisé.

10 Phase V : Trouver son chemin

Cette phase un peu particulière présente le premier projet de LSINF1102, durant cette phase vous devrez reprendre de zéro afin de trouver la meilleure solution possible à une problématique bien précise.

10.1 Enoncé du problème

Pour ce projet, vous devez repartir de zéro afin de permettre aux agents de trouver Ned Stark dans un labyrinthe.

Dans le cadre de cet énoncé, nous utiliserons des labyrinthes dits “parfaits”. Dans un labyrinthe dit parfait, chaque case est reliée à toute autre, et ce, de manière unique. Cela signifie que le labyrinthe ne contient pas d’îlots ou de possibilité de tourner en rond. Les loups protégeants Ned Stark sont à prendre en compte, mais il existera toujours en chemin possible entre la position initiale des agents et de Ned Stark. Dans cette configuration Ned Stark et les agents peuvent se trouver dans n’importe quelle position, mais Ned Stark se trouvera toujours dans une case jouxtée par au moins un mur.

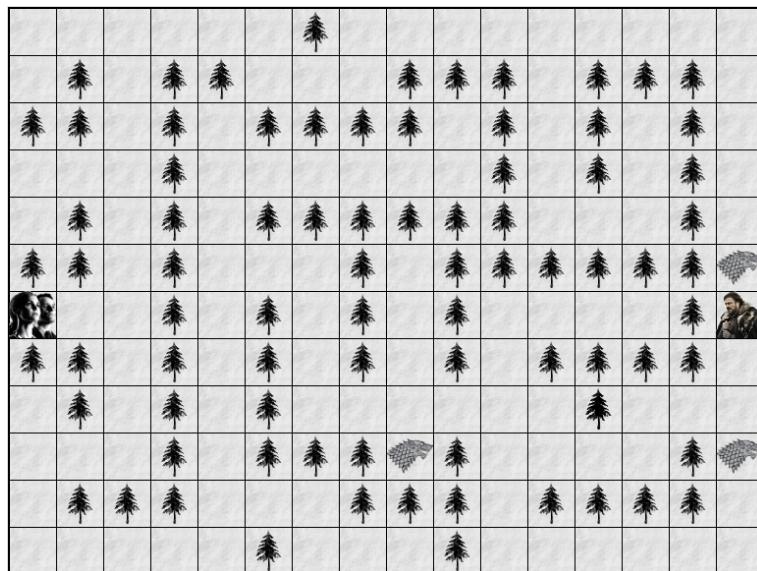


FIGURE 18. Un premier exemple de labyrinthe.

10.2 Calendrier

Sem.	Date	Contenu
S2	Mardi 27 septembre SUD ??	Séance plénière et activité “Quel étudiant, leader... suis-je ??”
	Mercredi 28 septembre	Debriefing de la première séance et lancement du projet labyrinthe
S3	Mardi 04 octobre	Discussion sur le travail de groupe
	Mercredi 5 octobre	Débriefing du “Passeport pour le BAC”
	Vendredi 7 octobre	Remise du projet

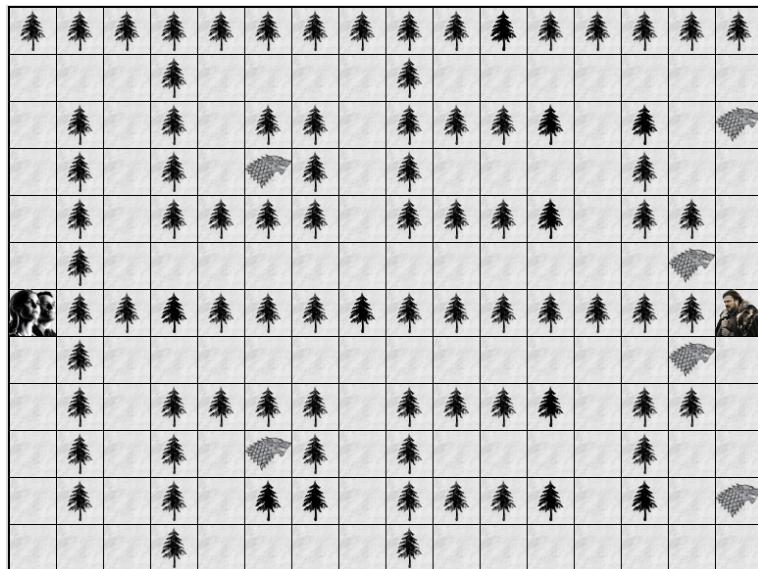


FIGURE 19. Un labyrinthe symétrique.

10.3 Délivrables

Chaque groupe d'étudiants doit rendre son programme (un seul fichier appelé P1_AAA_NN_Code.java) ainsi que son rapport (un seul fichier appelé P1_AAA_NN_Rapport.pdf de maximum 3 pages, où AAA est le nom de votre classe et NN le numéro de votre groupe) le **vendredi 7 octobre**, au plus tard à **18:00**, sur Moodle. Le fichier code doit contenir l'ensemble du fichier NedStark dans Greenfoot. Votre rapport doit contenir au minimum :

- La description et l'explication de votre algorithme
- La justification de vos choix
- La méthode et les tests utilisés pour tester votre solution
- Une analyse des points forts et faibles de votre solution ainsi que des possibilités d'amélioration

10.4 Évaluation

L'évaluation du projet P1 porte sur la qualité du programme (1 point), le rapport (1 point) et la participation à la vie de la classe (1 point).

Il est impossible de soumettre son travail sur Moodle après l'heure fixée. Si un groupe d'étudiants n'a pas téléchargé son travail sur Moodle avant l'heure fixée, chaque étudiant du groupe se verra attribuer la note 0 pour le programme et/ou le rapport.

11 Information pour les tuteurs

Le but premier de cette semaine d'APP0 est de permettre aux étudiants nouvellement arrivés de se rencontrer et cela en travaillant en groupe autour d'un projet informatique, tout au long de la semaine. Le second but est d'introduire gentiment les étudiants à toute une série de notions de base et de mot-clés qui seront utilisés tout au long de leurs études. L'important n'est pas du tout de leur donner des définitions ou des solutions, il s'agit d'une semaine purement formative. Votre rôle en tant que tuteur est d'animer les séances et de susciter les discussions au sein du groupe dont vous êtes responsable.

Vous trouverez toute une série d'indications, disséminées partout dans l'énoncé, qui devraient vous permettre de mener à bien votre tâche et que vous pourrez utiliser afin de diriger les conversations dans le bon sens et éviter que le groupe ne se perde en partant toujours dans toutes les directions. Rappelez-vous de votre formation tuteur, vous êtes là pour **C.Q.F.D. : Conduire, Questionner, Faciliter et Diagnostiquer**.

- **Conduire** : vous devez surveiller que le groupe s'organise et s'y prenne comme il faut, vous intervenez lorsqu'ils sont perdus ou lorsque plus rien ne se passe. Vous validez également les différentes étapes du processus de résolution de la situation-problème.
- **Questionner** : vous n'êtes pas là pour résoudre le problème à la place des étudiants, et vous ne devez donc *jamais* leur donner quelque réponse que ce soit. À toute question venant de leur part, vous leur répondez en leur posant une question qui vous permettra de situer leur niveau, de mettre en évidence des désaccords au sein du groupe, de relancer le débat et les discussions au sein du groupe, pour les pousser à aller plus loin et à utiliser leur esprit critique...
- **Faciliter** : vous facilitez le travail de groupe en aidant l'établissement d'un climat de travail favorable. Vous aidez une bonne circulation de la parole lorsque c'est nécessaire, vous relancez l'animateur lorsque celui-ci est en difficulté, vous donnez du feedback sur la situation du groupe, en particulier lorsque tout se passe bien.
- **Diagnostiquer** : vous devez à tout moment être capable d'observer la situation et de diagnostiquer ce qui se passe, afin de comprendre la dynamique de groupe qui s'installe, et pouvoir travailler sur les trois registres précédents.

Il y a une différence notable dans le dispositif mis en place durant cette semaine d'APP0, par rapport à un dispositif classique. Dans notre cas, les groupes ne seront pas figés durant la semaine. La raison essentielle est que de changer les groupes permet de mieux couvrir le premier objectif de cette semaine d'APP0, à savoir faire se connaître les étudiants. De plus, contrairement à ce qui se passe du côté des ingénieurs civils, les SINFs ne travailleront pas avec un même groupe tout au long du quadri, et il n'y a donc pas nécessité de, dès le début, travailler et mettre en place une dynamique de groupe. Le bémol de cela est que chaque jour, les étudiants se retrouveront avec un nouveau groupe et qu'une nouvelle dynamique devra à chaque fois s'installer. Chaque étudiant se retrouvera dans 3 groupes en tout. Les changements seront opérés mardi et mercredi soir. Pour jeudi et vendredi, les étudiants gardent le même groupe, étant donné qu'ils devront coder une solution de groupe au terme de la semaine.

La suite de cette section décrit dans le détail le déroulement précis de chacune des séances encadrées.

11.1 Intro

(mardi 8h30–9h00)

Durant 30 minutes, les étudiants recevront une introduction brève au déroulement de la semaine. La situation-problème qui les occupera sera très brièvement présentée, quelques informations logistiques et organisationnelles seront fournies et enfin ils recevront l'énoncé et on se dirigera vers les classes. À ce moment donné, les groupes seront formés aléatoirement par le coordinateur et ils seront dispatchés dans un local avec un tuteur.

<http://fr.wikipedia.org/wiki/Tutorat>

11.2 Phase I

(mardi 9h00–10h30)

Le but de cette première phase tutorée est de prendre en main la situation-problème qui va être utilisée tout au long de la semaine. Pour cela, diverses activités concrètes ont été prévues pour les étudiants et le groupe. Ce qui est important ici est de rappeler aux étudiants qu'après la séance, ils auront un moment de temps libre (durant lequel ils doivent travailler) qui sera suivi par une heure de bilan avec le tuteur. Pour le lendemain, 9h30, chaque étudiant aura un devoir individuel à remettre.

0’–5’	Commencez par vous présenter très rapidement et demandez à chaque étudiant d'écrire son prénom et son nom sur une feuille en papier et de la placer devant eux.
5’–15’	Laissez dix minutes aux étudiants pour parcourir rapidement la section 3 du document.
15’–30’	Pendant un quart d'heure, laissez les étudiants résoudre seul les exercices (section 4.1). Profitez de ce moment pour diagnostiquer et voir un peu comment les membres de votre groupe travaillent. N'hésitez pas à vous lever et faire le tour de la table.
30’–40’	Pendant 10 minutes, demandez aux étudiants de remettre en commun les différentes réponses obtenues. Le groupe doit se mettre d'accord sur les bonnes réponses.
40’–45’	Annoncez aux étudiants qu'ils auront un petit devoir individuel à faire et à rendre à 16h15 au début de la séance « Bilan I ». Il y a six termes différents à définir pour la question 2, attribuez un terme à chaque étudiant.
45’–60’	Les étudiants ont maintenant un quart d'heure pour réfléchir individuellement à une solution pour le problème de base. L'idée est d'écrire un algo clairement, et d'ensuite le faire exécuter par son voisin de gauche.
60’–80’	Les vingt prochaines minutes sont consacrées à l'établissement d'une comparaison d'algorithmes. Chaque étudiant aura vu son propre algorithme et celui de son voisin et maintenant, il faut se mettre d'accord sur une version de groupe. Pour ce faire, il faut trouver des critères de comparaison et utiliser la grille comparative. À la fin des vingt minutes, le groupe doit s'être mis d'accord sur un algorithme.
80’–90’	Clôturez la séance en demandant au groupe d'avoir une version finale de la comparaison et de l'algorithme de groupe. Récupérez la copie de ces deux pages, leur laissant l'occasion de le recopier. Dites leur directement au-revoir en leur annonçant qu'ils doivent lire les sections 5.3 et 5.4 et déjà commencer à travailler ça en groupe. Vous vous revoyez à 16h15 pour en discuter. Il faut donc qu'ils aient réfléchi à ça.

11.3 Bilan I

(mardi 16h15–17h15)

Cette phase de bilan fait suite à une phase de travail individuel. Votre rôle de tuteur lors de cette phase est donc essentiellement de la consultance et de la restructuration. La grosse nouveauté qu'ils auront travaillé entre le matin et maintenant, c'est la décomposition en sous-problèmes. En fin de séance, rappelez aux étudiants qu'ils ont un devoir individuel à remettre pour le lendemain. La suite de l'énoncé leur sera fourni demain lors du cours.

0'–5'	Commencez par récupérer les petits devoirs de chaque étudiant, de leur demander de placer la feuille avec leur prénom et nom devant eux et de leur demander si le travail non-encadré s'est bien passé.
5'–25'	Pendant 20 minutes, demandez à un porte parole du groupe de vous expliquer la solution à laquelle ils ont abouti. L'idée est que cette discussion soit interactive, c'est-à-dire que vous devriez interrompre la présentation en questionnant le groupe.
25'–35'	Pendant 10 minutes, discutez d'une manière plus fine de décomposer SP2. Les étudiants n'auront sans doute pas pensé à un sous-problème qui peut être appelé dans une boucle.
35'–50'	Le dernier quart d'heure devrait vous permettre de discuter de l'efficacité de leur algorithme. Vous devriez pouvoir aisément trouver des situations où leur algorithme n'est pas efficace. Vous devez susciter le débat au sein du groupe.
50'–60'	Clôturez la séance en rappelant au groupe qu'ils ont un devoir individuel à faire, et qu'ils devront le rendre au début du cours du lendemain, à savoir à 8h30. Prenez également le temps de parcourir avec eux le squelette de rapport, en insistant bien sur le fait que ce n'est qu'une suggestion.

11.4 Cours I

(mercredi 8h30–9h30)

Au début de ce cours, les étudiants remettent leur devoir individuel. Aucun retard ne sera toléré, les travaux remis en retard seront clairement identifiés.

Le but de ce cours est de restructurer les étudiants sur ce qui a déjà été vu jusqu'à présent. Les notions à revoir sont : algorithme, opération primitive, condition, boucle (while) et itération, test (if-else) et sous-problème. L'idée est que tous ces concepts devraient être expliqués et illustrés dans d'autres contextes que celui de la situation-problème et ce afin que les étudiants capturent bien les intuitions se trouvant derrière ces notions.

À la fin du cours, les étudiants reçoivent la suite de l'énoncé. Ils reçoivent également leur nouveau groupe, avec lequel ils doivent travailler durant la phase de travail non-encadré afin de préparer au mieux l'entrevue avec le tuteur à 16h15. Il faut rappeler aux étudiants qu'ils auront de nouveau un devoir à faire, en groupe, et à remettre pour le même jour, à 18h15.

11.5 Bilan II

(mercredi 16h15–17h15)

Cette nouvelle phase de bilan est à nouveau de type consultance et restructuration. La différence avec la journée d'hier est qu'ici, ils n'ont eu aucune phase tutorée avant cette phase de bilan, tout a été fait de manière non-encadrée, et dans un nouveau groupe.

0'–5'	Commencez par leur demander de placer la feuille avec leur prénom et nom devant eux et de leur demander si le travail non-encadré avec le nouveau groupe s'est bien passé.
5'–20'	Pendant 15 minutes, demandez à un porte parole du groupe de vous expliquer l'algorithme qu'ils ont défini pour contourner une ligne horizontale et à un autre porte parole de vous expliquer l'algorithme pour contourner une ligne verticale. N'hésitez pas à solliciter et questionner plusieurs membres du groupe pour vous assurer que tout le monde est bien d'accord.
20'–30'	Pendant 10 minutes, confrontez les solutions apportées pour les lignes verticales ou horizontales et demandez au groupe comment tout cela pourrait être vu comme un seul même problème.
30'–45'	Le prochain quart d'heure est consacré aux tests. Passez en revue avec eux les tests qu'ils vous proposent, tout en portant une attention particulière aux tests whitebox.
45'–55'	Enfin, terminez par dix minutes pour parcourir avec eux la préparation pour le test qui aura lieu le lendemain. Pour information, dites-leur bien qu'il s'agit d'un test à livre fermé.
55'–60'	Clôturez la séance en rappelant au groupe qu'ils ont un devoir de groupe à faire, et qu'ils devront le rendre en fin de journée, à 18h15 au plus tard, en main propre à la secrétaire INGI. Aucun retard ne sera toléré.

11.6 Cours II

(jeudi 10h45–11h45)

Le cours II a pour but de restructurer le travail qui a été fait dans le cadre du second devoir de groupe. Le cours reprend quelques algorithmes d'étudiants et discute de problèmes rencontrés. L'idée est de restructurer les notions d'abstraction, de sous-problème et de test.

À la fin du cours, les étudiants reçoit la suite de l'énoncé et les nouveaux groupes sont annoncés aux étudiants. Ces groupes ne changeront plus jusque la fin de la semaine. Des indications par rapport au test sont fournies (livre fermé, 1h, matière vue jusqu'à aujourd'hui).

11.7 Bilan III

(jeudi 16h15–17h15)

Une fois de plus, cette phase est une consultance et restructuration du travail qui a été accompli par un nouveau groupe. Cette fois-ci, le groupe ne changera plus jusque la fin de la semaine.

0'–5'	Commencez par leur demander de placer la feuille avec leur prénom et nom devant eux et de leur demander si le travail non-encadré avec le nouveau groupe s'est bien passé.
5'–20'	Pendant 15 minutes, demandez à un porte parole du groupe de vous présenter « Mon algorithme Phase III ». Le groupe doit bien comprendre la notion de spécification et vous veillerez à l'interroger afin de vous assurer que cette notion est bien assimilée.
20'–35'	Pendant 15 minutes, réfléchissez et discutez des réflexions décrites aux sections 7.3 et 7.4.
35'–50'	Terminez en discutant avec le groupe les solutions apportées pour les situations avec obstacles touchant le bord et avec obstacles en L.
50'–60'	Clôturez la séance en rappelant au groupe qu'ils devront demain coder leur algorithme et qu'ils n'auront que deux heures pour cela. Ils ont donc bien intérêt à finaliser leur solution de groupe et bien savoir comment ils vont, en pratique coder et tester leurs algorithmes, le lendemain. Enfin, profitez du temps qu'il reste pour leur demander s'ils sont prêts pour le test et s'ils n'ont pas l'une ou l'autre question. Vous ne répondez pas aux questions, mais utilisez le groupe.

11.8 Test

(jeudi 17h15–18h15)

Pendant 1h, les étudiants sont soumis à un test évaluatif. Ils recevront le feedback de ce test le lendemain.

11.9 Machine I

(vendredi 10h45–12h45)

À la fin de cette séance, les étudiants reçoivent l'énoncé de la dernière phase avec les explications sur Greenfoot. Ils sont invités à lire ce document avant la séance de 14h et également invités à se présenter à l'heure car ils n'auront que 2h de codage sur Greenfoot.

11.10 Machine II

(vendredi 14h–16h)

Les étudiants codent à deux sur une machine, et décident au terme des deux heures quelle est la version du programme qu'ils vont soumettre pour leur groupe. Une version par groupe doit être soumise pour 16h sans faute.

Votre rôle de tuteur sera de jouer les coach et consultant en programmation Java. Aidez-les à trouver les fautes de syntaxe Java, à utiliser Greenfoot et aidez-les à débugguer leurs algorithmes. Vous n'êtes pas obligé de rester avec votre groupe, vous pouvez tourner entre les différents groupes pour jouer votre rôle de consultant Java, et revenir vers votre groupe pour les soucis liés à leur algorithme.

11.11 Concours

(vendredi 16h15–17h15)

Les codes soumis par les étudiants seront testés sur des situations d'exemple de complexité croissante, afin de voir quel est le groupe gagnant.