

## CPS 2017-2018

### Examen Réparti 1 : *Dungeon Master*



Figure 1: Capture d'écran de *Dungeon Master* (Atari ST).

**Consignes.** L'objectif de l'examen réparti est d'écrire une spécification cohérente, complète (autant que possible) et non-redondante d'un programme (un jeu vidéo) décrit dans l'énoncé. Ainsi :

- Le sujet est trop long pour être traité entièrement en deux heures. Une bonne spécification couvrant une petite partie du cahier des charges sera plus valorisée qu'une spécification médiocre en couvrant une grande partie. Il n'est absolument pas nécessaire de traiter toutes les questions pour obtenir la note maximale.
- L'examen n'est pas linéaire mais il est conseillé de commencer par l'exercice 1. L'exercice 5 peut être traité directement après.
- Certaines questions sont indépendantes les unes des autres et les questions sont de difficulté et de longueur variables; il est conseillé de lire l'intégralité du sujet avant de commencer à rédiger une réponse.
- La description donnée dans l'énoncé est ambiguë, plusieurs spécifications correctes sont possibles, il faut faire des choix (et expliciter ces choix au maximum dans la copie).
- La spécification doit être semi-formelle. Sa syntaxe peut s'écarter de celle vue en cours et TD si elle est suffisamment claire (quantificateurs  $\forall \exists$ , ensemble d'éléments, appel d'opérateurs, ...)
- La définition d'un service **S** incluant (ou raffinant) un service existant **E** ne doit pas comporter la partie de la spécification présente dans **E**, seul ce qui est ajouté (observateur, opérateur, observation, préconditions, ...) doit être décrit.

**Description du Jeu.** Sorti en 1987, *Dungeon Master* (FTL Games) propose un jeu de rôle classique (paradigme porte-montre-trésor) en vue à la première personne. La structure du jeu est similaire à un roguelike (par exemple *Rogue* de Toy, Wichman, Arnold, 1980), mais sans génération aléatoire: on déplace un personnage (ou un groupe de personnages) sur une grille comportant des murs et des portes, formant un labyrinthe dans lequel évoluent des monstres.

L'innovation est la vue à la première personne. Le joueur voit à l'écran ce qu'il y a devant lui (jusqu'à une distance limitée). Il déplace son groupe en utilisant six actions de mouvement (avancer, reculer, faire un pas de côté ou se tourner). Le déplacement est discret, chaque action (à l'exception des actions de rotation, qui font rester au même endroit) font bouger d'exactly une case sur la grille. Divers monstres se déplacent aussi sur la grille, et se dirigent vers le joueur quand ils le repèrent.

Le jeu inclut un système de combat (on peut attaquer un monstre se trouvant strictement devant soi), un inventaire, un système faim/nourriture, des sortilèges magiques, des portes fermées à clef, des énigmes, ...

### Exercice 1: Grille

La grille est l'environnement dans lequel vont évoluer le joueur et les monstres. La case au Sud-Ouest de la grille a les coordonnées (0,0). Une case (*cell*) de la grille est soit l'entrée du niveau, soit la sortie (qui peuvent être placées

n'importe où), soit un emplacement vide (sur lequel on peut marcher), soit un mur (infranchissable), soit une porte. Une porte "Nord-Sud" est une porte qui autorise le passage de l'Est à l'Ouest et vice-versa. Une porte "Ouest-Est" est une porte qui autorise le passage du Nord au Sud et vice-versa. Les portes peuvent être soit ouvertes, soit fermées. On représente donc la nature d'une case par le type suivant:

**type** Cell {IN, OUT, EMP, WLL, DNO, DNC, DWO, DWC }

par exemple **DNC** sera utilisé pour décrire une porte "Nord-Sud" fermée (*Door, North-south, Closed*).

On décrit un service **Map** pour représenter la grille. Il possède deux observateurs **Height** et **Width** pour, respectivement, le nombre de lignes (Ouest-Est) et le nombre de colonnes (Nord-Sud) de la grille ainsi que **CellNature** qui observe la nature d'une case de la grille. En outre, une **Map** possède deux opérations, permettant d'ouvrir et de fermer les portes, attendant en paramètres les coordonnées de la porte en question.

**Q1.1** Compléter la spécification ci-dessous avec des préconditions, des invariants et des post-conditions (ne pas recopier les signatures).

```

Service:    Map
Types:    bool, int, Cell
Observers:  const Height: [Map] → int
               const Width: [Map] → int
               CellNature: [Mat] × int × int → Cell
Constructors:  init: int × int → [Map]
Operators:  OpenDoor: [Map] × int × int → [Map]
               CloseDoor: [Map] × int × int → [Map]

```

**Q1.2** Donner le code Java de la méthode **OpenDoor** de la classe **MapContract**, implémentation du contrat associé au service **Map**.

**Q1.3** Rédiger un jeu de test MBT partiel pour **Map** contenant deux tests de préconditions (un positif et un négatif), un test de transition et un test de paires de transitions. Calculer, dans ces trois cas, les taux de couverture.

## Exercice 2: Objets Mobiles

Les joueurs et les monstres (ainsi que d'autres éléments éventuels, comme des projectiles ou des pierres/caisses que l'on peut pousser) sont considérés comme des objets mobiles (*Mobile objects*). On les décrit au sein du service **Mob**. L'observateur **Env** renvoie la **Map** dans laquelle évolue le **Mob**. Les observateurs **Col** et **Row** renvoient les coordonnées, colonne (*column*) et ligne (*row*) du **Mob**. Enfin, **Face** donne la direction dans laquelle regarde le **Mob**, un point cardinal du type **Dir** {N, S, W, E }

Un **Mob** est initialisé avec des coordonnées, une **Map** et une direction initiale. Les **Mob** possèdent six opérations de déplacement: **Forward** (pour avancer dans la direction vers laquelle il fait face), **Backward** (pour reculer), **StrafeL** (pour faire un pas chassé vers la gauche), **StrafeR** (vers la droite), **TurnL** (pour faire un quart de tour vers la gauche), **TurnR** (vers la droite). Un **Mob** ne peut jamais occuper une case qui est un mur ou une porte fermée (mais il peut occuper une case qui est une porte ouverte).

On fera en sorte qu'on puisse toujours appeler chaque opération de déplacement. Par exemple, si le **Mob** fait face à un mur (ou une porte fermée, ou au bord de la grille), l'opération **Forward** est possible, mais n'a aucun effet.

**Q2.1.** Donner la spécification de **Mob**, avec l'intégralité des signatures et des invariants. Pour les préconditions et les postconditions, se restreindre uniquement à **Forward** et **TurnL**. Il n'est pas nécessaire d'écrire toutes les observations si certaines sont répétitives (l'indiquer dans la copie).

On veut qu'il y ait au plus un **Mob** par case de la grille. Cela implique que les déplacements des différents **Mob** ne sont pas indépendants (par exemple, un **Mob** ne peut pas avancer si la case juste devant lui est occupée par un autre **Mob**). On décide donc de décrire un sous-service de **Map** appelé **Environment** qui contient un observateur **CellContent**, qui prend en paramètres des coordonnées et renvoie un **Option[Mob]**, c'est à dire **No** si la case ne contient pas de mob, et **So(M)** si elle est occupée par le mob **M**.

**Q2.2.** Compléter la spécification de **Environment** ci-dessous, pour qu'on ne puisse pas fermer une porte ouverte sur laquelle se trouve un **Mob**.



**Q3.3** Décrire la spécification du service **Player**, qui étend **Entity** et qui possède des observateurs **Content**, **Nature** qui prennent en entrée des coordonnées relatives (dans  $[-1, 1] \times [-1, 3]$ ) et renvoient, respectivement, le **Mob**[Option] contenu ou non à cet endroit et la nature de la case à cet endroit. Par exemple, si le joueur **P** fait face vers l'Ouest, **Nature**(**P**, -1, 0) donne la nature de la case située directement à sa gauche (donc au Sud) et **Content**(**P**, 0, 2) le contenu de la case située deux cases en face de lui (donc deux cases à son Ouest). Faire attention aux nouvelles postcondition pour les opérations de déplacement (donner uniquement celles pour **Forward** et **TurnL**).

Une case est visible (*viewable*) d'un joueur si elle est située dans les 9 cases devant lui et si il n'y a pas entre elle et le joueur un mur ou une porte fermée.

**Q3.4** Ajouter au service **Player** un observateur **Viewable** qui prend en paramètre des coordonnées relatives et décide (bool) si oui ou non la case est visible du joueur.

## Exercice 4: Moteur

Le moteur (*engine*) du jeu gère les tours de jeu. C'est un service qui maintient un environnement et une structure de données d'entités. Les deux doivent rester synchronisés, c'est-à-dire que l'observation **CellContent** de l'environnement appelée en  $x, y$  renvoie quelque chose si et seulement si la structure de données contient une entité dont les coordonnées sont  $x, y$ . Le moteur possède une opération **step** qui ne peut être appelée que quand toutes les entités ont un nombre de point de vie strictement positif; cette opération déclenche l'opération **step** de toutes les entités.

**Q4.1** Donner un service **Engine** pour le moteur de jeu, avec les observations et opérations nécessaires à la gestion (ajout/suppression) d'une structure de données d'**Entity**.

**Q4.2** Expliquer pourquoi est-il difficile d'obtenir une bonne postcondition pour la méthode **step**.

## 1 Exercice 5: Editeur de Grille

Afin de permettre aux utilisateurs de définir et partager leurs propres grilles, on se propose de gérer les grilles modifiables. Le service **EditMap** raffine **Map** et ajoute un opérateur **SetNature** qui permet de mettre à jour une case de la grille (de changer sa nature).

**Q5.1** Donner un service **EditMap** qui raffine **Map** et ajoute l'opération **SetNature**.

On veut ajouter à **EditMap** un observateur **isReady** qui indique si oui ou non une grille est prête pour être utilisée par le jeu. On considère qu'une grille est prête quand les conditions suivantes sont réunies:

1. Chaque porte, ouverte ou fermée, est telle que les cases devant et derrière la porte sont vides, et les cases des deux côtés de la porte sont des murs. Par exemple si une case est une porte Nord-Sud ouverte, les cases à son Ouest et son Est doivent être vides et les cases à son Nord et à son Sud doivent être des murs.
2. La grille possède une unique entrée et une unique sortie,
3. La sortie doit être atteignable depuis l'entrée.

**Q5.2** Intégrer l'observateur **isReady** en considérant uniquement les 2 premières conditions.

**Q5.3** Ajouter un observateur **isReachable** au service, qui prend en paramètres les coordonnées de deux cases de la grille et décide si elles sont atteignables, c'est à dire s'il existe un chemin dans la grille de l'une à l'autre passant uniquement par des cases vides ou des portes (ouvertes ou fermées). Compléter les observations relatives à **isReady** à l'aide de cette observateur.