Labo - Questions

Objectifs

Parties

- Décrire les *plateaux de jeu* et ses composantes, où il s'agit de modéliser les éléments constituant les leçons et niveaux.
- Définir un *langage de stratégie* permettant de traduire en instructions précises les étapes guidant le personnage vers la sortie.

Artefacts

Pour chacune des deux parties, trois artefacts :

1. Un **diagramme de classes** capturant les concepts présents dans chaque partie;
2. Un ensemble d'**expressions OCL** énonçant les contraintes et contrats portant sur les diagrammes précédents;
3. Un **diagramme d'objets** illustrant une utilisation des diagrammes et expressions sur chaque partie;

Description du jeu (DJ)

Dans cette section, on cherche à définir un **diagramme de classes** (UML) et ses contraintes permettant de capturer les éléments constituant un match, comme décrit dans la section "*Description d'un Match*".

Pour simplifier, on pourra considérer que dans la version réactive du jeu, l'ordinateur est un joueur comme un autre, avec un personnage propre évoluant sur le plateau : ceci permet de s'exercer contre un joueur fictif.

1. Établir une première version d'un diagramme de classes UML qui fixe les éléments principaux : le jeu rassemble des joueurs qui sauvegardent des personnages/avatars, qu'ils utilisent pour jouer des matchs, eux-mêmes constitués d'un certain nombres de rencontres ayant lieu dans un monde.

2. Enrichir cette première version avec les détails nécessaires concernant les joueurs et leurs personnages; les matchs, les rencontres et les mondes. Les joueurs et personnages sont décrits dans la Section 2.1; les rencontres et les mondes dans la Section 2.2.

   La modélisation doit tenir compte des éléments indispensables pour décrire le déroulement d'une partie, comme spécifié en Section 2.3.
   On prêtera une attention particulière aux éléments UML suivants :

   - la multiplicité des associations doit correspondre aux contraintes de l'énoncé;
   - le type des attributs choisis doit permettre l'expression des contraintes;
   - les associations portant des agrégations / compositions doivent être choisies avec soin;
   - si besoin, la nature des collections (bag, set, sequence, ordered set) doit être justifiée;
   - les éventuelles hiérarchies doivent être pleinement spécifiées (complétude et couverture).

   Puisqu'il s'agit d'une modélisation conceptuelle, les détails de visibilité de propriétés de classe (attributs / associations), ainsi que la présence d'opérations, sont à proscrire.

3. Spécifier les contraintes d'unicité suivantes :

   1. Les matchs sont identifiés de manière unique au sein du jeu;
   2. Les joueurs ont des pseudos différents au sein du jeu;
   3. Les personnages/avatars d'un joueur sont nommés différemment, et ont une représentation physique distincte pour pouvoir les reconnaître visuellement au premier coup d'oeil;
   4. Les rencontres d'un match portent un numéro d'ordre unique.

4. Les contraintes énumérées ci-après expriment des règles permettant d'obtenir des modèles bien formés.
   Spécifier ces contraintes à l'aide d'éléments structurels dans le **diagramme de classes**,

éventuellement complétés par des expressions OCL (dépendamment de la manière dont votre **diagramme de classes** est construit).

1. le potentiel de vie d'un joueur est toujours positif;
2. les ratios d'attaque et de défense sont des ratios, c'est-à-dire des valeurs strictement positives et inférieures à 1;
3. Le nombre de rencontres constituant un match est toujours impair;
4. Le numéro identifiant la rencontre au sein d'un match correspond à l'ordre dans lequel il sera joué (ou plus précisément, l'ordre dans la collection qui le stocke, à supposer que les rencontres sont jouées dans l'ordre de stockage).
5. L'inventaire d'un joueur contient trois types d'items : les items ramassables (nourriture, boisson ou munitions), les objets d'aide à l'orientation, et les objets d'aide (cotte de maille et cape).
6. Une seule des cases d'un monde contient un Graal;
7. Les coordonnées d'une case ne peuvent excéder la longueur d'un monde;
8. La disposition des cases correspond à leur coordonnées. Par exemple, l'abscisse d'une case à droite d'une case donnée est l'entier successeur de l'abscisse de la case de référence (et similairement pour les autres directions, en tenant compte des bords du plateau).
9. Un joueur se trouve toujours dans la case correspondant à sa position absolue;
10. La bordure d'un plateau contient toujours des éléments infranchissables (pour éviter aux personnages de « sortir » du monde).
11. Le nombre de cases visibles correspond à la valeur de visibilité liée au joueur. Par exemple, dans le plateau à droite dans la Figure 2, la visibilité de 3 rend « découverte » trois lignes au-dessus, en dessous, et trois colonnes à droite et à gauche.
12. Un personnage ne peut pas se trouver sur une case portant un obstacle infranchissable.
13. Deux personnages (y compris les zombies) ne peuvent pas se trouver sur la même case.

5. La définition des deux Dsls, pour spécifier les mondes et pour définir les stratégies, est séparée en deux morceaux. Cependant, pour que le jeu soit utilisable, ces deux éléments doivent être harmonieusement connectés. Comment, et où, apparaît la notion de stratégie dans votre **diagramme de classes** pour spécifier les mondes?

6. Définir un diagramme d'objets pour un monde de taille 4 (càd. 2x2) dont les bordures sont délimitées par des murs, et où les coins en bas à gauche et à droite sont occupés par les deux joueurs; et les coins en haut à gauche et à droite sont occupés par le Graal et un zombie. On donnera au joueur et au zombie les mêmes potentiel et

caractéristiques, et on supposera que le joueur possède un item de chaque sorte. Les autres valeurs nécessaires pour compléter le modèle peuvent être choisies arbitrairement

Description de Stratégies (DS)

Dans cette section, on cherche à définir un **diagramme de classes** (UML) et ses contraintes permettant de modéliser le langage de stratégies comme décrit dans la Section 3. La Question 5 précisait déjà que les deux Diagrammes sont en réalité liés, même si pour des besoins de notations et pour faciliter votre progression, leur modélisation est séparée. Il est donc normal de devoir faire référence à des classes du Diagramme de Classe précédent. Dans ce cas,

- Ne répéter que la classe référencée, sans reprendre les détails de celle-ci (attributs et/ou associations);
- Rester cohérent dans les noms choisis : le nom de classe doit correspondre à une classe effectivement présente dans le **diagramme de classes** précédent.

La cohérence entre les deux diagrammes est un critère prépondérant de notation.

7. Établir une première version d'un diagramme de classe UML qui fixe les éléments principaux : une stratégie comporte une série de déclarations de modules et de variables, et est composée d'une vision à court et long terme articulées autour d'objectifs, qui sont réalisés par des règles (dont une règle par défaut). Chaque règle comporte une priorité, un déclencheur et une réaction.

8. Modéliser le concept de Type comme décrit dans la Section 3.1, en s'assurant de pouvoir déclarer tous les exemples donnés pour les énumérations, les tableaux et les enregistrements, sur la base des types primitifs classiques (on utilisera une classe abstraite TypePrimitif / PrimitiveType que l'on n'instanciera pas).

9. Modéliser le concept de Declaration de variables et de modules; puis les relier avec leurs éléments constitutifs.

10. Modéliser explicitement le concept d'Objectif (Objective) réalisant les visions, sans oublier leur(s) paramètre(s), comme déjà modélisés dans le Diagramme décrivant les mondes.

11. Modéliser ensuite explicitement le concept d'Action, comme décrit en Section 2.3, en veillant à en respecter le vocabulaire.

12. Modéliser le concept d'Expression comme défini en Section 3.5, en s'assurant de pouvoir représenter tous les exemples de la Table 1.

13. Modéliser le concept d'Instruction (Statement) comme défini en Section 3.6, en s'assurant de pouvoir décrire tous les exemples fournis.

14. Spécifier les contraintes d'unicité suivantes :

    - Les modules ont un nom unique au sein d'une même stratégie;
    - Les variables globales ont un nom unique au sein d'une même stratégie;
    - Les variables locales au sein d'un module possèdent un nom unique;
    - Les noms des paramètres d'un modules sont tous différents;
    - Les noms des types déclarés sont globalement uniques (en particulier, on ne peut pas nommer une énumération et un tableau de manière identique);

15. Spécifier une contrainte OCL permettant de vérifier qu'une déclaration de type est bien formée :

    - La liste des litéraux sont uniques au sein d'une énumération;
    - La liste des champs d'un enregistrement est non-vide;
    - Les noms des champs sont uniques au sein d'un même enregistrement;
    - Un tableau comporte au moins une dimension;
    - Chaque dimension de tableau doit être strictement positive.

    Pour les questions qui suivent et qui demandent la spécification d'un contrat, il s'agit de définir des pre-/post conditions sur l'exécution de l'opération associée.

16. Supposons l'existence d'une opération Expression :: typepq : Type définie dans la classe Expression. Spécifier le contrat sur le résultat produit par cette opération pour vérifier que le Type retourné correspond à :

    - le type du littéral qui correspond au littéral (par exemple, le type de la valeur true doit être Boolean, celui de la chaîne "123"doit être String et celui de l'entier 122 Integer, etc.)
    - le type correspondant à l'opérateur unaire, à condition que sa sous-expression corresponde à ce type. Par exemple, les types de expressions unaires - 123 et not isVisible doivent respectivement être Integer (puisque 12 est entier) et Boolean (à condition que la variable isVisible soit déclarée comme une variable booléenne).
    - le type correspondant à l'opérateur binaire, à condition que les types des sous-expressions soient cohérentes. Par exemple, 12 + total utilise un opérateur entier + sur deux sous-expressions entières, mais 12 + isVisible est incohérent.
        - le type de la sous-expression dans une expression parenthésée;

        - le type de sa déclaration pour une expression d'accès à la valeur d'une variable (cf. exemples plus haut avec isVisible);
    - le type de l'énumeration pour une expression d'un accès à un littéral d'énumération. Par exemple, Direction.Up doit renvoyer le type énumération Direction.
    - le type de la déclaration du champ dans une expression d'accès à un champ. Par exemple, origine.x doit retourner Integer puisque le champ x est déclaré comme tel.
    - le type du contenu du tableau pour une expression d'accès à un élement de tableau. Par exemple, à partir des exemples déclarés en Section 3.1, les expressions visible[0,0] et area[0] doivent respecti- vement retourner Integer et VisibleLine.

17. Supposons l'existence d'une opération estTraversableGraceAuxItemsp…q : Boolean, qui rend vrai ssi pour un personnage donné, une case passée en paramètres est traversable grâce aux items que le personnage possède sans l'intervention du joueur, c'est-à-dire que la case est sur une zone de feu, d'eau ou de glace et que le personnage possède respectivement des bottes pare-feu, un kit de plongée et des bottes à crampon.

    - Quelle classe de votre Diagramme de Classe pourrait contenir cette opération?
    - Définir cette opération en précisant quel(s) serai(en)t ses paramètre(s), et quel serait le contrat OCL sur cette opération.

18. Supposons l'existence d'une opération ramasser(…) : Void dont l'effet est le suivant : si la case passée en paramètre contient un item ramassable, alors

    - si l'item est un bonus de défense ou d'attaque, le ratio correspondant du personnage se trouve modifié en multipliant l'ancienne valeur par la valeur du bonus;
    - Si l'item est un radar, il double la visibilité de la position de l'adversaire;
    - si l'item est de la nourriture, de l'eau, des munitions, des bottes, des palmes, une cape ou une cotte de maille, l'item est simplement rajouté dans l'inventaire.

    Préciser le contexte (¸ càd. sur quelle classe est défini l'opération) et la signature (¸ càd. les paramètres) de l'opération ramasser et définir le contrat sur son résultat.

19. Spécifier le contrat OCL sur une opération Instruction :: estValidepq : Boolean qui vérifie qu'une instruction est valide :

    - L'instruction Skip est toujours valide;
    - La garde d'une Conditionelle ou d'une Itération doit être de type booléen;
    - La partie gauche et droite d'une Affectation doivent être de même type.

20. Spécifier une contrainte OCL vérifiant la cohérence des règles d'une stratégie par rapport à son objectif :

1. Toutes les règles d'un même objectif (à court ou long terme) ont des priorités différentes pour assurer leur déclenchement déterministe.
2. Une réaction de règle ne contient qu'une seule action SeDéplacer.
3. Seules les métarègles peuvent contenir l'action changer.

21. À partir de la situation définie en Question 4.6, définir une stratégie constituée des visions suivantes :

    - à long terme, se rendre vers le Graal;
    - à court terme, ramasser un maximum de nourriture.

    On ne définira pas les règles correspondantes, mais on déclarera en plus dans la stratégie un module estCaseGraal(…) : Boolean qui renvoie vrai ssi la case contient le Graal. On supposera une variable réservée result dont le type correspond au type de retour du module, et qui sera affectée du résulat.
    Note : Il faut donc utiliser une instruction affectation (cf. Section 3.6) avec la bonne expression en partie droite!

Maturité & Modularité

Dans cette section, on teste la robustesse des Diagrammes de Classes définis au terme des Sections précédentes, en introduisant des variantes dans le jeu.

22. Dans l'état actuel du jeu, une case ne peut pas supperposer un item sur un obstacle. Pourtant, ce serait parfois utile de donner au joueur un gros bonus à condition qu'il puisse traverser une zone de feu.
    - Comment proposeriez-vous de modifier votre diagramme de définition du monde pour intégrer cette possibilité?
    - Quel impact cela a-t-il sur votre modélisation du langage de stratégie, et sur les contraintes OCL définies de part et d'autre?