RAPPORT PROJET IDM

Modélisation, Vérification et Génération de Jeux

Justine BANNAY, Mylène BERCÉ, Arthur COUTURIER, Cédric VILLEMONT-JEAN

18/01/2022

IDM – L12-03

Tables des matières

INTRODUCTION 2

DOCUMENT 1 : Game.xtext 2

DOCUMENT 2 : game.png 2

DOCUMENT 3 : enigme.game 3

DOCUMENTS 4 : enigme.ltl & enigme.net 3

DOCUMENT 5 : Enigme.java 4

DOCUMENTS 6 : exemple*.game 4

DOCUMENTS 7 : game.ocl & exemple (ok_*.xmi) / contre-exemples (ko_*.xmi) 4

DOCUMENT 8 : game2petrinet.atl 5

DOCUMENTS 9 : game2ltl.mtl & game2ltl_invariants.mtl 6

DOCUMENT 10 : game2java.mtl 6

DOCUMENTS 11 : game.odesign & representations .aird 6

CONCLUSION 7

INTRODUCTION

Dans le cadre de l'UE Ingénierie Dirigée par les Modèles il nous a été demandé de définir une chaîne de modélisation, vérification et génération de code pour des modèles de jeux.

Nous allons présenter l'ensemble des documents produits en expliquant pour chacun les différents éléments le composant et les principales difficultés rencontrées.

DOCUMENT 1 : Game.xtext

Ce document décrit la syntaxe imaginée pour représenter des jeux d'exploration (exemple enigme document 3) et permet de générer un méta-modèle ecore.

Choix de conception :

- Les noms des concepts sont les identifiants des classes
- Exigences interprétables (conditions stockables au choix et descriptions obligatoires ou non)

Difficultés :

- Actions

- Conditions :
    - Disjonctives et Conjonctives
    - Give Conditionnel
    - Descriptions Conditionnelles
- ObjetMultiple / ItemMultiple

DOCUMENT 2 : game.png

La difficulté a été d'ordonner les classes et relations afin de faire un diagramme lisible car il y a beaucoup de classes. Nous avons envisagé de faire des sous-diagrammes représentant une partie des classes afin de gagner en lisibilité.

DOCUMENT 3 : enigme.game

Nous avons commencé par ce document pour nous aider. Nous avons rencontré quelques difficultés dans la compréhension du texte. Certaines exigences menaient à des interprétations différentes. Nous avons donc fait certains choix de conception afin de répondre au mieux aux exigences attendues.

Choix de conception :

- Description des choix de syntaxe :
    - Définition d'un concept avec son nom de classe en amont en tant que mot-clé (exemple : action Réussir …).
    - Dans les actions, les mots-clefs 'give' et 'consume' sont suivis d'une liste d'items placés entre crochets pour ressembler à un tableau.
    - Les conditions sont représentées entre parenthèses et éventuellement séparées par des opérateurs & (et) et | (ou).
    - Les caractéristiques des personnes, de l'explorateur et des interactions sont données entre accolades pour plus de lisibilité aux vues des nombreux attributs.
    - Les descriptions de concepts sont données entre guillemets (convention dans de nombreux langages).
    - Les lieux destination et source des chemins sont décrits à l'aide des mots-clés "from" et "to"
    - Les attributs conditionnels (isVisible, isObligatoire …) sont suivis du terme "on" puis de la condition associée pour indiquer qu'ils ne se réalisent que lorsque la condition est vérifiée.
    - Les lieux de départ et de fin du jeu sont indiqués grâce aux mots-clés "start" et "end".

DOCUMENTS 4 : enigme.ltl & enigme.net

Le document "enigme.ltl" décrit des propriétés LTL définies pour notre modèle. Dans notre cas, ce fichier exprime la propriété suivante "il existe une solution pour ce jeu" ou autrement dit, on arrivera forcément dans un état ÉCHEC ou SUCCÈS lors de l'exécution du jeu. Les quelques lignes du fichier sont décrites ci-dessous :

"<> dead" signifie que le blocage est inévitable

“<> ([] (Echec \/ Succès)) = (Echec \/ Succès)” sera définitivement vraie -> existence d’une solution

Le document “enigme.net” permet de représenter l’exemple de l’énigme à travers un réseau de Petri, visualisable grâce à l’outil Tina. Les contraintes LTL permettent de vérifier la terminaison et les invariants. Pour construire ce document, nous avons divisé le travail en sous-réseaux pour nous aider à construire le réseau final. Ce document nous servira de base ensuite pour la transformation modèle à modèle (Game vers Petrinet).

DOCUMENT 5 : Enigme.java

Nous avons décidé d’écrire le prototype à l’aide du langage Java puisque c’est celui avec lequel nous étions le plus à l’aise. Ce document nous servira de base pour la génération automatique du prototype à partir d’un modèle respectant le méta-modèle Game. Afin de faciliter l’écriture du fichier Game2Java.mtl, nous avons décidé d’utiliser une méthode générique qui associe à un choix l’exécution d’une méthode. Pour celà il nous a fallu une Map de Int (choix) / Method (La méthode à exécuter). Ainsi pour obtenir l’objet Method, nous avons utilisé la réflexivité que nous avons pu aborder dans la matière Méta Modélisation et Tests.

DOCUMENTS 6 : exemple*.game

L’ExempleRealiste.game a pour thème l’exploration d’une île avec des épreuves, et reprend chaque fonctionnalité et option du métamodèle, rassemblant tout dans un même exemple fonctionnel.

L’ExempleCombat.game a pour thème un combat avec un monstre, et illustre un exemple plus simple que le précédent qui permet de traiter quelques fonctionnalités dont les suivantes : Choix conditionné et final, Action conditionnée et chemin visible sous condition.

D’autres exemples fournis dans le rendu nous permettent plus facilement de couvrir une partie des fonctionnalités.

DOCUMENTS 7 : game.ocl & exemple (ok_*.xmi) / contre-exemples (ko_*.xmi)

Les principaux éléments mis en place dans les contraintes OCL sont les suivants :

- Chaque nom doit commencer par une lettre et peut être suivi d'une ou plusieurs lettres et d'un ou plusieurs chiffres. Chaque nom a au moins deux caractères.
- Chaque attribut représenté par un entier modélise une taille ou une quantité. Nous imposons donc que chaque entier soit positif.
- Nous avons aussi rempli les exigences 17 et 26.

Nous avons rencontré des difficultés à implémenter les contraintes induites des exigences 17 et 26 :

- Une seule personne obligatoire par lieu :
    - Nous avons mis la contrainte que le nombre de personnes obligatoire par lieu doit être inférieur ou égal à 1.
- Un seul chemin visible obligatoire et ouvert par lieu :
    - Nous vérifions qu'il y a au plus un chemin obligatoire et sans condition de visibilité et d'ouverture de la même manière que la contrainte précédente. Cependant ce test n'est pas exhaustif car plusieurs conditions peuvent être vraies en même temps et dans ce cas là plusieurs chemins peuvent être obligatoires, ouverts et visibles en même temps.

Le fichier d'exemple "ok_game.xmi" illustre le fonctionnement de toutes les contraintes. Dans cet exemple, chaque contrainte est respectée.

Les fichiers de contre-exemples (ko_cheminReflexif.xmi, …) illustrent chacun le non-respect d'une des contraintes.

DOCUMENT 8 : game2petrinet.atl

Concernant la transformation modèle à modèle (.game à .petrinet) faite à l'aide d'ATL, nous sommes partis du fichier enigme.net ainsi que des deux méta-modèles. Notre objectif était d'arriver (après une transformation modèle à texte à l'aide de la feature toTina à un modèle respectant le méta-modèle Petri Net), à un fichier similaire à enigme.net construit à la main d'après un réseau de pétri. Nous avons ainsi traduit les différents objets (personnes, explorateur, items etc…) en ATL avec pour chacun une règle précise. Puis, dans chaque règle, nous avons lié ces différents éléments, tout en suivant enigme.net.

Cependant, notre architecture a rendu la transposition des conditions complexe, c'est pourquoi cette partie n'est pas implémentée.

DOCUMENTS 9 : game2ltl.mtl & game2ltl_invariants.mtl

Nous avons repris la structure des fichiers mtl créés lors du mini-projet et nous l'avons adapté aux conditions ltl de ce projet qui sont :

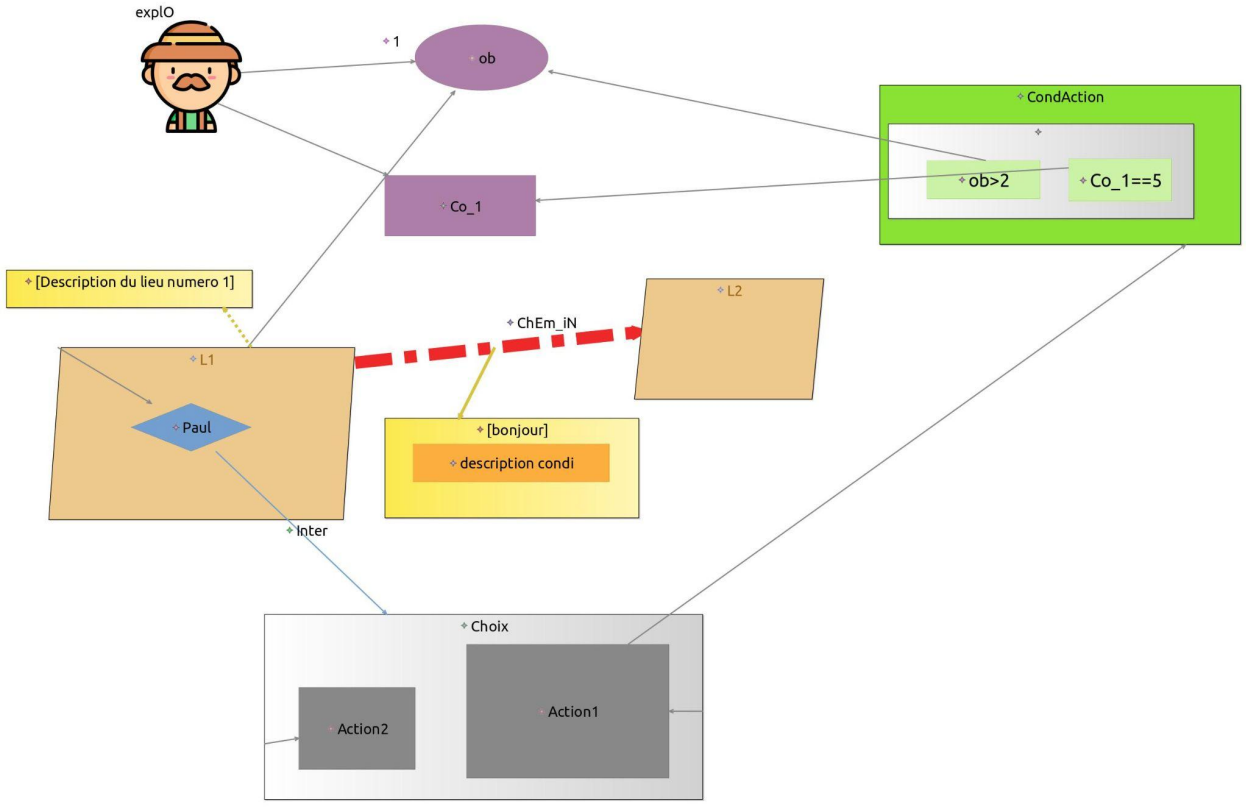
- Il existe une solution pour un modèle de jeu.
- Le jeu se termine lorsque l'explorateur est dans un lieu de fin.
- L'explorateur est dans un seul lieu à la fois.

DOCUMENT 10 : game2java.mtl

En reprenant le document 3 généré à la main (enigme.game), nous avons pu simplement générer ce fichier mtl qui nous permet de transformer un modèle respectant le métamodèle de Game en fichier .java compilable et exécutable afin de se donner une idée concrète de l'exécution du jeu en mode console.

DOCUMENTS 11 : game.odesign & representations .aird

Nous avons défini une syntaxe graphique pour représenter les modèles de Game. Chaque élément du méta-modèle est représenté par une forme graphique (rectangle, image, flèches, …). Nous avons développé tous les outils permettant de les créer via l'interface



graphique directement. Chaque élément est accompagné d'une description lors de sa création. De plus, une personne est créée directement avec son interaction et son choix vide. Une condition disjonctive est forcément composée d'une condition conjonctive, elle-même composée d'une condition simple.

CONCLUSION

Ce projet nous a permis de mettre en application les connaissances que nous avons pu acquérir durant l'intégralité du cours et des TPs d'IDM, et de rendre concret et utile l'utilisation de cette chaîne de transformation. De plus, il nous a été possible d'écrire simplement, grâce à notre syntaxe, le jeu que nous avions développé lors du projet long de TOB l'année dernière, l'exécuter en java grâce à l'auto génération toJAVA, le transformer en réseau de Pétri et le visualiser sur l'outil Tina.