Projet de Génie du Logiciel et des Systèmes : Modélisation, Vérification et Génération de Jeux

Résumé

Ce document décrit le travail demandé aux étudiants du département IMA de l'ENSEEIHT inscrits pour la session 1 de l'année universitaire 2017-2018 de l'UE GLS.

Dans ce projet sont testées les facultés des étudiants à construire une infrastructure basée modèle autour d'une problématique concrète. Cette approche s'effectuera en deux axes : une **partie conception** sur papier et une **partie implémentation** en utilisant les outils du projet Eclipse Modelling pris en main au cours du module.

Deux rapports seront rendus au cours du projet : un dossier préliminaire de conception qui sera suivi d'un retour des enseignants et un rapport de projet qui devra contenir les détails d'implémentations et les difficultés par rapport au dossier de conception ainsi que toute autre information jugée pertinente par les étudiants.

Les travaux relatifs à la partie développement devront fonctionner sous Linux sur les machines des salles de travaux pratiques, qui sont pourvues des outils nécessaires à la bonne réalisation de ce travail, et qui serviront de lieu de test à la fin du projet.

Les dates de remise sont précisées à la fin de ce sujet. Il est impératif de rendre les travaux dans les délais. **Tout retard sera sanctionné**. Les étudiants sont encouragés à travailler de manière régulière, et ce dès la présentation du projet et la remise du présent sujet.

Table des matières

Attention : Des informations complémentaires, en particulier concernant les échéances, les documents à rendre, l'évaluation... peuvent être données sur la page du module.

Ce projet consiste à produire une chaîne de modélisation, de vérification et de génération de code pour des jeux de défense (« *Tower Defense* »).

La modélisation consiste à concevoir un langage dédié pour décrire un jeu de défense sous la forme d'un modèle et à implanter les outils d'édition et de visualisation associés.

La vérification permet de détecter certains modèles de jeux de défense qui ne possèderaient pas de solutions. Celle-ci s'appuiera sur des contraintes OCL et/ou des algorithmes implantés en Java/EMF.

La génération de code permet de construire un prototype avec une interface texte ou graphique simple permettant de tester le jeu de défense décrit par un modèle et de valider la jouabilité et l'intérêt du jeu avant de développer le contenu multimédia.

Vous utiliserez les différents outils exploités en TP et pendant le BE : Xtext, Sirius, Acceleo et EMF/Java.

Contrairement aux TPs/BE, le métamodèle sera généré à partir de la syntaxe concrète du langage définie en Xtext et les contraintes OCL ne seront pas intégrées dans le métamodèle avec OCLinEcore mais spécifiées dans un fichier séparé en utilisant CompleteOCL. Cet outil est présenté dans la documentation intégrée de l'outil OCL d'Eclipse.

1 Objectifs du projet

Ce projet consiste pour l'essentiel à définir un atelier de modélisation, vérification et génération de code pour des modèles de jeux de défense (leur description est donnée en section 1.1). Les principales étapes sont les suivantes :

1. Définition de la syntaxe concrète du langage de modélisation de jeu de défense en utilisant Xtext et génération du métamodèle Ecore. Cette solution permet de travailler plus facilement en équipe et de préserver, autant que possible, les exemples réalisés précédemment lors d'évolution du métamodèle (contrairement au format XMI).
2. Définition de la sémantique statique avec OCL dans un fichier séparé du métamodèle créé avec l'éditeur CompleteOCL. Vous n'utiliserez pas OCLinEcore comme en TP/BE car la combinaison avec la génération du métamodèle avec Xtext n'est pas opérationnelle avec la version actuelle d'Eclipse.
3. Utilisation de l'infrastructure fournie par EMF pour manipuler les modèles. Celle-ci est généré automatiquement par Xtext. Le code généré peut être modifié pour que la vérification des contraintes OCL soit intégrée à la validation dans l'éditeur texte produit par Xtext.
4. Définition de syntaxes concrètes graphiques avec Sirius pour certains points de vue.
5. Définition d'une transformation de modèle à texte (M2T) avec Acceleo, pour générer un prototype d'implantation du jeu à partir d'un modèle de ce jeu. Le code généré utilisera le langage et les bibliothèques choisis par chaque groupe.

1.1 Modélisation de jeux de défense

Nous vous proposons de concevoir un langage dédié (Domain Specific Modeling Language) pour modéliser les jeux de défense.

Une analyse des besoins/recueil des exigences a permis d'obtenir les informations suivantes :

E_1 L'objectif d'une partie d'un jeu de défense est d'empêcher les vagues de mobiles qui entrent dans le terrain de jeu de sortir de celui-ci. Pour cela, le joueur place des obstacles (appelés usuellement tours d'où le nom de « *Tower Defense Game* ») sur le terrain de jeu. Les obstacles, respectivement les mobiles, peuvent lancer des projectiles qui permettent d'éliminer les mobiles, respectivement les obstacles.

E_2 Un jeu de défense est caractérisé par les définitions partagées d'éléments de jeu qui pourront être utilisés dans différentes parties et de parties composées de ces éléments.

E_3 Chaque élément est caractérisé par un nom qui permet de le désigner.

E_4 Les éléments de jeu peuvent être des mobiles, des obstacles, des projectiles et des natures de terrain.

E_5 Une partie est caractérisée par un terrain de jeu, par une liste de niveaux de jeu et par la description de vagues de mobiles.

E_6 Un terrain de jeu est représenté par un damier composé de cases indexées par un numéro de ligne (entre 1 et L) et de colonne (entre 1 et C).

E_7 Un niveau de jeu est caractérisé par son nom, par la durée des pauses entre les vagues de mobiles, par l'énergie attribuée initialement au joueur et par le nombre de mobiles entrés dans le terrain qui doivent en sortir pour que le joueur perde la partie.

E_8 Chaque case d'un terrain de jeu est caractérisée par une nature de terrain.

E_9 La nature d'un terrain peut être soit une entrée, soit une sortie, soit un chemin, soit un campement, soit une décoration.

E_{10} Un terrain de jeu est bien formé si :
- il contient au moins une case d'entrée, une case de sortie et une case de campement.
- toute case d'entrée est réliée à au moins une case de sortie par une séquence de cases de chemin contiguës. Deux cases sont contiguës si elles possèdent un côté commun.

E_{11} Une vague de mobiles est bien formée si :
- les obstacles prépositionnés dans une vague ne suffisent pas à éliminer tous les mobiles de cette vague
- l'énergie attribuée au joueur pour une vague est suffisante dans le meilleur des cas pour éviter une défaite
- l'énergie attribuée au joueur pour une vague est insuffisante dans le pire des cas pour éviter une défaite

E_{12} Les cases de chemin, d'entrée ou de sortie sont uniquement accessibles aux mobiles et aux projectiles lancés.

E_{13} Une case de campement est uniquement accessible aux obstacles et aux projectiles lancés.

E_{14} Une case de décoration est uniquement accessible aux projectiles lancés.

E_{15} Une case de chemin est caractérisée par une quantité d'énergie que doit consommer un mobile pour entrer dans la case.

E_{16} Les projectiles ne consomment pas d'énergie pour entrer dans une case.

E_{17} Une case de chemin est caractérisée par un volume qui correspond au volume cumulé maximal de tous les mobiles qui peuvent se trouver simultanéement sur cette case.

E_{18} Les projectiles ont un volume nul.

E_{19} Un mobile est caractérisé par son volume. Un mobile ne pourra pas entrer dans une case chemin si le volume cumulé du mobile et des mobiles déjà présents dans la case chemin dépense le volume maximal de cette case.

E_{20} Les mobiles et les obstacles sont caractérisés par une quantité d'énergie maximale dont ils disposeront pour chaque tour de jeu et par les projectiles qu'ils transportent et qu'ils pourront lancer à chaque tour de jeu.

E_{21} Les mobiles, respectivement les obstacles, sont caractérisés par une tactique qui permet de déterminer sur quel obstacle, respectivement mobile, il lancera un projectile si plusieurs obstacles, respectivement mobiles, sont à sa portée :

— soit il vise le plus proche ;
— soit il vise le plus faible ;
— soit il vise le plus fort.

E_{22} La force/faiblesse des mobiles/obstacles est déterminée par la quantité d'énergie disponible pour cet élément.

E_{23} Un projectile est caractérisé par sa portée, par sa masse, par sa vitesse et par sa quantité d'énergie. La portée est le nombre de cases qu'il va parcourir avant de s'arrêter et de disparaitre. La masse est la quantité d'énergie que doit utiliser le mobile ou l'obstacle pour lancer le projectile. La vitesse est le nombre de cases que le projectile peut traverser dans un tour. La quantité d'énergie du projectile indique l'énergie que va absorber le projectile quand il touche un mobile ou un obstacle.

E_{24} Une vague de mobiles est caractérisée par les mobiles qui vont entrer sur le terrain pendant cette vague, par la quantité d'énergie attribuée au joueur si la vague est déployé sans défaite du joueur et par les obstacles prépositionnés en début de vague.

E_{25} Chaque mobile d'une vague est caractérisé par sa position dans la vague (i.e. l'ordre dans lequel il va entrer sur le terrain), par la case d'entrée par laquelle il arrive sur le terrain et par la case de sortie par laquelle il doit quitter le terrain.

E_{26} Chaque obstacle prépositionné est caractérisé par la case campement sur laquelle il est positionné.

Ces éléments permettent de modéliser de nombreux jeux différents.

Exemple 1 : Considérons les définitions *montagne* qui est une case de décoration, *route* qui est une case de chemin et *garage* qui est une case de campement.

La *route* a un coût de 1 et un volume de 1.

Définissons un jeu de défense composé d'une seule partie.

Le terrain de jeu est composé de 3 lignes de 5 cases. Les cases de la première ligne sont des *montagnes*. Les cases de la deuxième ligne sont des *routes* sauf la première qui est une entrée et la dernière qui est une sortie. Les cases de la troisième ligne sont des *garages*.

Un projectile P est défini avec une portée de 1, une masse de 1, une vitesse de 1 et une énergie de 1.

Un élément mobile M est défini avec une vitesse de 1 et une énergie de 1. Sa tactique est de viser le plus proche.

Un élément obstacle O est défini avec une énergie de 1 et un projectile P. Sa tactique est de viser le plus faible.

La partie possède un niveau unique avec une durée de pause de 1, une énergie initiale de 0 pour le joueur (il ne pourra pas créer d'obstacle pendant la partie), et une défaite si 2 mobiles sortent du terrain.

La partie possède une seule vague composée de 3 mobiles M en séquence qui doivent entrer par l'entrée unique et sortir par la sortie unique. Un seul obstacle O est prépositionné au milieu de la ligne de *garages*. L'énergie 3 sera attribuée si la vague échoue.

1.2 Vérification de l'existence de la solution dans un jeu

La complexité d'un modèle de jeu peut être suffisante pour que le concepteur ne puisse pas déterminer si le jeu possède une solution sans l'aide d'un outil informatique. En particulier il faut que le jeu soit bien formé et que les vagues de mobiles soient bien formées. Vous étudierez les conditions nécessaires pour que le jeu puisse avoir une solution. Ces conditions peuvent être plus fortes que nécessaires et rejeter des jeux corrects tant qu'elles n'acceptent pas des jeux incorrects.

1.3 Génération de prototypes

Le concepteur du jeu doit déterminer si celui-ci est intéressant avant de démarrer le développement de la partie multimédia du jeu (images, sons, musiques, vidéos, etc) qui est la plus coûteuse.

Nous proposons dans ce but de générer un prototype interactif en mode texte dans le langage de programmation de votre choix, en utilisant les bibliothèques de votre choix. L'objectif est la simplicité du code généré et de la transformation qui génère le code. Le prototype peut être rudimentaire.

L'analyse des besoins/recueil des exigences a permis d'obtenir les connaissances suivantes :

P_1 Le prototype doit permettre d'afficher le terrain de jeu avec la position de chaque obstacle, de chaque projectile lancé et de chaque mobile.

P_2 Le prototype doit permettre d'afficher les caractéristiques détaillées de chaque obstacle, projectile et mobile à la demande du joueur.

P_3 Cet affichage peut être textuel ou graphique. Chaque groupe dispose d'une totale liberté dans le choix des technologies, langages, bibliothèques, outils, ... pour réaliser cet affichage. Le but n'est pas de développer une nouvelle technologie d'affichage mais de s'appuyer sur des solutions existantes en maîtrisant les coûts de développement de cet affichage.

P_4 Une partie d'un jeu de défense consiste à déployer en séquence les vagues de mobiles. Le déploiement d'une vague se décompose en tours de jeux.

P_5 Un tour de jeux consiste à faire entrer les mobiles suivants de la vague tant que le volume maximum de la case n'est pas atteint, à activer chaque mobile, respectivement obstacle, respectivement projectile, présent sur le terrain.

P_6 L'activation consiste à initialiser l'énergie disponible du mobile, respectivement de l'obstacle, pour ce tour, avec la valeur actuelle de l'énergie maximale.

P_7 La valeur actuelle de l'énergie maximale est initialisée avec la valeur de l'énergie maximale qui caractérise le mobile, respectivement l'obstacle.

P_8 Lorsqu'un projectile entre dans la case contenant un mobile, respectivement un obstacle, si l'énergie du projectile est supérieure à la valeur actuelle de l'énergie maximale, alors le mobile, respectivement l'obstacle, est éliminé et l'énergie du projectile est diminuée de la valeur actuelle de l'énergie maximale puis le projectile poursuit son chemin ; sinon le projectile est éliminé et la valeur actuelle de l'énergie maximale est diminuée de l'énergie du projectile.

P_9 La valeur actuelle de l'énergie maximale peut être augmentée lors de la réparation d'un obstacle par le joueur.

P_{10} Une partie se termine par une défaite du joueur lorsqu'un nombre suffisant de mobiles des différentes vagues quittent le terrain par la sortie qui leur est assigné. Ce nombre dépend du niveau de difficulté de la partie.

P_{11} Une partie se termine par une victoire du joueur lorsque toutes les vagues de mobiles sont déployées sans conduire à une défaite.

P_{12} La fin du déploiement d'une vague de mobiles correspond soit à la défaite du joueur soit à la disparition de tous les mobiles qui composent la vague.

P_{13} Le déploiement d'une vague de mobile consiste à faire entrer par les entrées les mobiles composant la vague dans l'ordre prévu par le scénario défini pour la vague.

P_{14} Le joueur dispose d'une pause entre la fin du déploiement d'une vague de mobiles et le début du déploiement de la suivante. Sa durée dépend du niveau de difficulté de la partie.

P_{15} Pendant la pause, le joueur peut réparer, vendre, déplacer ou acheter et placer des obstacles.

P_{16} Chaque action effectuée par le joueur, en dehors de la vente, correspond à une perte d'énergie dépendant de l'obstacle et de son état.

P_{17} La vente correspond à un gain d'énergie dépendant de l'obstacle et de son état. Le gain ne peut pas dépenser la quantité maximale d'énergie d'une partie.

P_{18} Le joueur dispose d'une quantité d'énergie au début du jeu dépendant du niveau de difficulté de la partie.

P_{19} Le joueur reçoit un bonus en énergie à la fin de chaque vague. Ce bonus dépend du niveau de difficulté de la partie.

P_{20} Le joueur reçoit un bonus en énergie lors de l'élimination et le blocage de chaque mobile selon l'énergie maximale du mobile dépendant du niveau de difficulté de la partie.

P_{21} Pendant le déploiement d'une vague de mobile, le prototype doit :
- gérer l'entrée des mobiles sur le terrain par la case d'entrée qui est affectée à chaque mobile. Les entrées de mobiles doivent respecter l'ordre précisé dans la définition de la vague ;
- gérer le déplacement des mobiles sur le terrain ;
- gérer les actions des obstacles sur les mobiles ;
- gérer les actions des mobiles sur les obstacles ;
- détecter la disparition et le blocage des mobiles ;
- gérer l'énergie disponible globalement pour le joueur ;
- gérer l'énergie disponible pour chaque obstacle et chaque mobile.

P_{22} Les différentes activités précédentes dépendent du temps et doivent être coordonnées en fonction de la vitesse de chaque mobile et de chaque obstacle.

P_{23} Il n'est pas démandé une gestion concurrente des activités mais un entrelacement des activités élémentaires.

P_{24} Les technologies de simulation par événements discrets sont les plus appropriées pour cela. Il s'agit de remplir un agenda avec les activités élémentaires en prenant en compte la durée de chaque activité élémentaire puis de dérouler l'agenda jusqu'à sa fin.

2 Déroulement du projet

Ce projet est un travail **de groupe** qui devra être réalisé en quadrinôme constitué au sein d'un même groupe de TD. Pour prendre en compte la taille des groupes de TD qui n'est pas un multiple de

4, quelques trinômes seront autorisés. La composition des groupes est libre. Elle sera transmise par courrier électronique à `Marc.Pantel@enseeiht.fr` avant le vendredi 17 novembre 2017.

Les techniques mises en œuvre doivent être celles présentées dans le module de GLS : Ecore, EMF, OCL, Xtext, Sirius et Acceleoainsi qu'un langage de programmation cible de la génération de code qui sera choisi librement par chaque groupe.

Pour chaque partie, voici les tâches à réaliser, les documents à rendre par chaque groupe et les dates de remise.

2.1 Tâches à réaliser

T_1 Définir une syntaxe concrète textuelle pour les modèles de jeu avec Xtext puis générer le métamodèle associé.

T_2 Modéliser l'exemple donné en fin de section 1.1 avec la syntaxe textuelle proposée pour votre langage.

T_3 Réaliser une conception préliminaire du générateur de code à partir de modèles de jeux. Il s'agit de prendre l'exemple précédent et de programmer le code correspondant attendu. L'exemple sera exprimé avec la syntaxe textuelle précédente. Le code sera exprimé dans le langage de programmation que le groupe aura choisi.

T_4 Valider la pertinence, la complétude et l'usabilité de ce langage en rédigeant des tests plus simples pour chaque élement du langage ainsi qu'un test réaliste combinant toutes les capacités du langage.

Les tests seront exprimés avec la syntaxe concrète textuelle. Il n'est pas utile de faire des tests au format XMI avec l'éditeur arborescent. Les fichiers XMI seront générés à partir des modèles textuels avec l'outil générique de traduction de modèles textes gérés par Xtext vers du XMI inclus dans la distribution Eclipse GLS (dernière entrée du menu en faisant un clic droit sur un fichier texte valide géré par Xtext).

T_5 Définir les contraintes OCL pour capturer les contraintes sur les modèles de jeu qui n'ont pu l'être par les métamodèles.

T_6 Valider la pertinence et la complétude des contraintes en rédigeant des tests élémentaires ne respectant pas les contraintes et en montrant que les tests rédigés précédemment respectent les contraintes.

T_7 Développer un éditeur graphique avec l'outil Sirius offrant des vues graphiques pertinentes pour la saisie d'un jeu. Ces vues seront complétées par des contrôleurs pour pouvoir saisir graphiquement des éléments dans les modèles de jeux visualisés.

T_8 Implanter le générateur de code pour produire les prototypes dans le langage de votre choix en utilisant Acceleo.

T_9 Valider cette transformation en utilisant les tests rédigés précédemment.

2.2 Documents à rendre

Les consignes pour rendre les documents suivants seront données sur la page du module (les documents seront rendus via SVN).

D_1 Le modèle Xtext décrivant la syntaxe concrète textuelle des modèles de jeux (`game.xtext`).

D_2 Une vue graphique mise en page du métamodèle généré par Xtext (`game.png`).

D_3 Le modèle de jeux de l'exemple 1 exprimé dans la syntaxe concrète textuelle définie avec Xtext (`enigme.SUFFIXE`, utilisez le suffixe choisi lors de la création du projet Xtext).

D_4 Un programme écrit dans le langage de votre choix qui exécute le jeu de l'exemple 1 (`enigme.SUFFIXE`, utilisez le suffixe correspondant au langage de votre choix `java`, `py`, `ml`, `ada`, ...).

D_5 D'autres exemples de modèles de jeux (en expliquant l'intérêt de chaque exemple). Ces modèles serviront à illustrer les différents éléments réalisés dans le cadre du projet.

D_6 Les fichiers de contraintes OCL au format CompleteOCL associés à ce métamodèle, avec des exemples (et contre-exemples) qui montrent la pertinence de ces contraintes (`game.ocl`).

D_7 Le code Acceleo de la transformation modèle à texte vers le prototype dans le langage de votre choix (`game2prototype.mtl`).

D_8 Les modèles Sirius décrivant l'éditeur graphique pour les deux points de vue sur les modèles de jeux (le point de vue Territoire et le point de vue Interaction).

D_9 Un document concis (rapport) qui explique le travail réalisé. Attention, c'est ce document qui servira de point d'entrée pour lire les éléments rendus.

2.3 Dates importantes

Lundi 13 novembre 2017 : Présentation du Sujet en cours

Vendredi 17 novembre 2017 : Remise de la composition des groupes de projet

Mercredi 29 novembre 2017 : Remise du Sujet sur le site

Mercredi 6 décembre 2017 : Remise du métamodèle et de la syntaxe textuelle (format Xtext et image, délivrable D_1 et D_2)

Semaine du 27 novembre 2017 : Séance de TP avancement projet

Mercredi 13 décembre 2017 : Remise du dossier de conception préliminaire (exemple de la fin de la section 1.1 écrit dans la syntaxe textuelle (délivrable D_3), traduction manuelle dans le langage de programmation choisi (délivrable D_4)

Semaine du 11 décembre 2017 : Séance de TP avancement projet

Mercredi 20 décembre 2017 : Rendu des travaux d'implémentation (tous les délivrables sauf D_9)

Mercredi 20 décembre 2017 : Oral et test du projet

Vendredi 22 décembre 2017 : Rendu du rapport (délivrable D_9)