École Nationale Supérieure
d'Électrotechnique, d'Électronique, d'Informatique,
d'Hydraulique et des Télécommunications

Institut National Polytechnique

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

Elouan Appere
Stanislas Deneuville
Ismail Moussaoui
Mohamed Smail

SOMMAIRE

1 Introduction

On s'intéresse dans ce Projet à produire une chaîne de modélisation, de vérification et de génération de code pour des jeux de défense,
Pour la modélisation de jeu de défense nous avons mis en place des éditeurs graphiques et textuels qui facilitent la création de ces modéles.
nous avons définit également des contraintes OCL ce qui permet de détecter certains modèles de jeux de défense qui ne possèderaient pas de solutions ou qui ne sont pas cohérents.
Enfin on a génère le code Java qui permet de construire un prototype avec une interface graphique simple permettant de tester le jeu de défense décrit.

2 Xtext

La première étape du projet consiste à concevoir un modèle représentant les jeux de defense. Et pour faire cela simplement, nous avons réalisé la grammaire du langage de description de ces modèles avec la syntaxe Xtext, cela sera ensuite converti en modèle Ecore par eclipse automatiquement.

Nous avons donc listé tous les composants d'un jeu de defense décris par les exigences du sujet, puis tous leurs attribus et les relations les reliants.

Puis nous avons réflechi sur comment adapter le modèle pour le rendre plus simple a écrire par l'utilisateur. En particulier les Projectiles, Mobile et Obstacle ont été chacun décomposé en 2 parties, les parties ModeleProjetiles, ModeleMobile et ModeleObstacle, qui sont décrits dans les élements communs et définissent des classe d'éléments réutilisables (définissant les caractéristiques générales, tels que l'energie maximal, la vitesse, etc.) et les instances de ces modèles qui sont décrites dans les vagues de mobiles et qui définissent les attribus propres à chaque instance (case de départ/arrivé, l'energie, etc.).
Il est ainsi possible de réutiliser autant de fois que souhaité ces modèles et limiter le travail en ne reécrivant pas tout les attribus pour chaque élements des vagues qui peuvent être très nombreux.

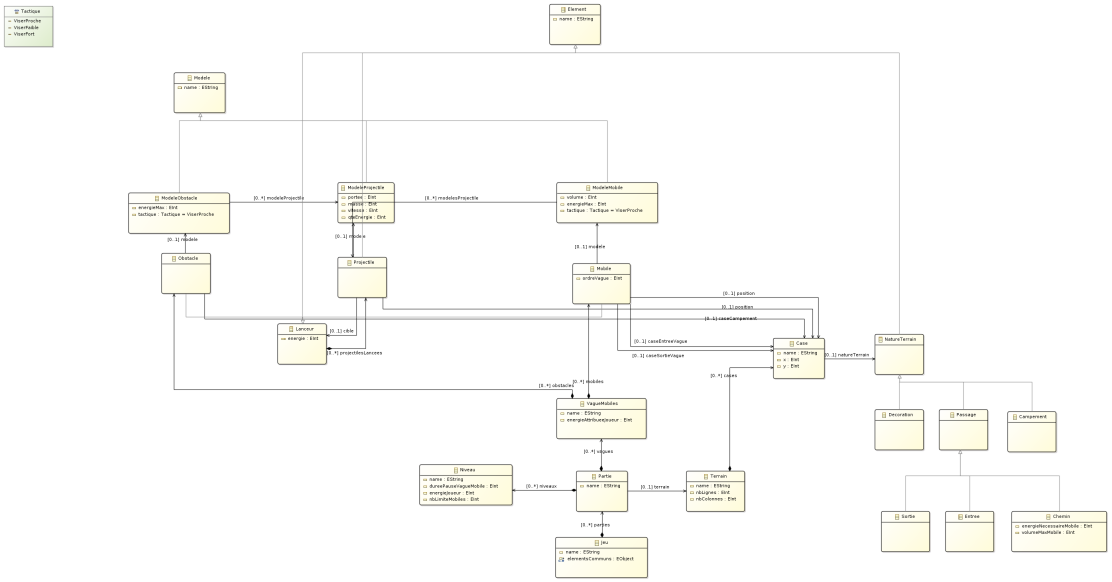
Nous avons aussi allégé le modèle en supprimant certains attributs uniquement nécessaire à l'execution comme l'energie courante durant un tour. Mais nous avons conservé les attributs de position, et d'énergie des Mobile, Projectiles et Obstacle, car bien qu'ils ne sont jamais écrits lors de la description d'un jeu les conserver peut permettre de sauvegarder une partie en cours de route en enregistrant l'état du modèle ce qui nous semblait une fonctionnalité interessante même si nous n'avons pas eu le temps de l'implanter.

A posteriori nous pensons qu'il aurait aussi été judicieux d'alléger la syntaxe décrivant les terrains qui est très lourde pour l'utilisateur sur de grande tailles car chaque case est définie independament des autres par sa propre position et sa nature. Même si cela nous aurait un peu compliqué la tâche il aurait par exemple été pratique de décrire toutes les cases d'un coup dans l'ordre en précisant seulement les natures de terrain sous forme de grille.

3 Métamodèle

Le modèle ecore correspondant a donc été intégralement généré par eclipse. Nous obtenons l'architecture suivante :

— Un jeu est composé par des éléments communs et des parties.
— Un élément commun peut être soit un modèle soit une nature de terrain soit un terrain.
— Une nature de terrain peut être soit un campement , soit une décoration soit un passage(qui peut être un chemin ,une entrée ou une sortie).
— Un modèle peut être soit un modèle de projectile soit un modèle de lanceur(qui peut être un modèle de mobile ou un modèle d'obstacle).
— Un terrain est composé de plusieurs cases, chaque case est liée à sa nature de terrain.
— Une partie est composé de vagues et de niveaux. Chaque vague est composé des mobiles qui vont entrer sur le terrain pendant cette vague, et les obstacles prépositionnés en début de vague.
— Chaque lanceur est composé des projectiles lancées par ce lanceur.
— Un lanceur peut être soit un mobile soit un obstacle.



Voici le diagramme du modèle, à trois exceptions près :

— il manque l'attribut "vitesse" de la classe ModeleMobile.
— il manque l'attribut "cout" de la classe ModeleObstacle
— l'attibut modelesProjectile de la classe ModeleMobile est de type Modele-Projectile et non Projectile,

Nous n'avons pas jugé essentiel de regénérer tout le diagramme et surtout de le réarranger pour ces changements mineurs.

4 contraintes OCL

Le métamodèle ne permet pas de capturer certains contraintes sur les modèles de jeux .
Pour cela on a définit certains contraintes OCL($game.ocl$) qui permetent de détecter certains modèles de jeux de défense qui ne possèderaient pas de solutions ou qui ne sont pas cohérents .
On a définit des contre-exemples qui montrent la pertinence de ces contraintes ($ko - modele - ocl$).
Alors Le modéle doit respecter plusieurs contraintes qu'on peut regrouper par type de contrainte :

Des contraintes de base des attribus

- positivité d'énergie des éléments du jeu.
- positivité de durée de pause entre les vagues.
- l'ordre d'un mobile dans la vague doit être supérieur à 1 etc ...

Des contraintes de nombre d'un composant(élément) dans son composite(constitué d'un ou de plusieurs composant)

- un terrain doit contenir au moins une case d'entrée et une case de sortie.
- un jeu doit contenir au moins une partie.
- une partie doit contenir au moins une vague.
- une partie doit contenir au moins un niveau etc ...

Des contraintes spécifique au jeu

- les cases d'un terrain sont validées :
  - si leurs Coordonnées sont valide. abscisse entre 0 et nombre des lignes moins 1et ordonnee entre 0 et nombre des colonnes moins 1).
  - nombre de case égale au nombre de lignes multiplie par nombre de colonnes.
  - les Coordonnée d'une case sont unique .
- les mobile peuvent aller sur entrée, sortie et chemin .
- la case entrée d'un mobile doit être de nature Entrée.
- la case sortie d'un mobile doit être de nature Sortie.
- l' obstacle peuvent être sur les case de nature. campement
- toute case d'entrée est réliée à au moins une case de sortie par une séquence de cases de chemin.
- les obstacles prépositionnés dans une vague ne suffisent pas à éliminer tous les mobiles de cette vague etc ...

5 L'éditeur graphique

L'éditeur graphiqe(*game.odesign*)permet à l'aide d'une palette d'outils de création de créer et d'organiser graphiquement un modèle. Les propriétés de chaque élement du diagramme sont modifiables via un menu adjacent.

Pour représenter graphiquement un modéle de ce jeu :

- On a représenté les modéles d'un mobile , les modéles d'un obstacle , les modéles d'un projectile par des noeuds .
- les terrains par des containers qui contient les cases , chaque case et lié par une relation a sa nature de terrain
- les parties par des containers qui contient les niveaux et les vagues.
- chaque vague est un container qui contient des mobiles et des obstacles.
- les mobiles et les obstacles sont des containers qui contients les projectiles lancés par ce mobile ou obstacle
- chaque mobile est lié par des relations a la case d'entre , la case de sortie , le modéle du mobile , la position actuelle
- chaque mobile est lié par des relations à sa position , à son modéle

Pour des raison de clarté, on a grouper dans des sections les outils de création et d'organisation de chaque groupe d'éléments

- une section pour la création de terrain et ses cases, de nature de terrain , et lier les case au nature de terrain
- une section pour la création de modéles de mobiles , de modéles de obstacles , des modéles de projectiles .
- une section pour la création d'une partie . ses niveaux ,et ses vagues
- une section pour la des mobiles et des obstacles d'une vague et les lier au différents éléments

Vous trouvez ci-joint l'image d'exemple du sujet représente par l'éditeur graphique (*exemple_sirius.jpg*)

6 Transformation modéle en java

L'objectif de cette partie est de pouvoir générer à partir d'un modèle valide un code Java complet réalisant le jeu.

Nous avons donc dans un premier temps programmé à la main le code Java d'un exemple simple.

Cela à principalement consisté à reproduire en java l'architecture de classe du modèle. Puis d'ajouter le code réalisant concrètement le comportement des éléments du jeu.

On a donc le fonctionnement suivant : Le main initialise un jeu et le lance, ce dernier lance à la suite ses différentes partie, chaque partie execute chacune ses vague dans l'ordre, chaque vague execute autant de tour que necessaire et chaque tour appel chaque Projectile, Obstacle et Mobile pour qu'ils réalisent leurs actions et interagissent ente eux.

Les classes ModeleMobile, ModeleObstacle et ModeleProjectiles sont abstraites, ainsi chaque modele de mobile, obsacle ou projectile decrit se concretise après conversion en java en une classe local de Jeu_* heritant d'une des classes abstraites et redéfinissant les paramètres communs. Un obstacle, mobile ou projectile est alors une instance de cette classe locale.

Finalement l'ensemble de l'architecture est assez générale et ne décri que les comportement et interactions des différents types d'éléments de tel sorte que pour générer le code du jeu il ne suffit de ne générer que le code de la classe Jeu_* qui va dans son constructeur instancier chaque élements du jeu, et le main qui appel cette classe Jeu_*. Le reste est copié à l'identique par rapport à l'exemple que nous avons réalisé à la main.

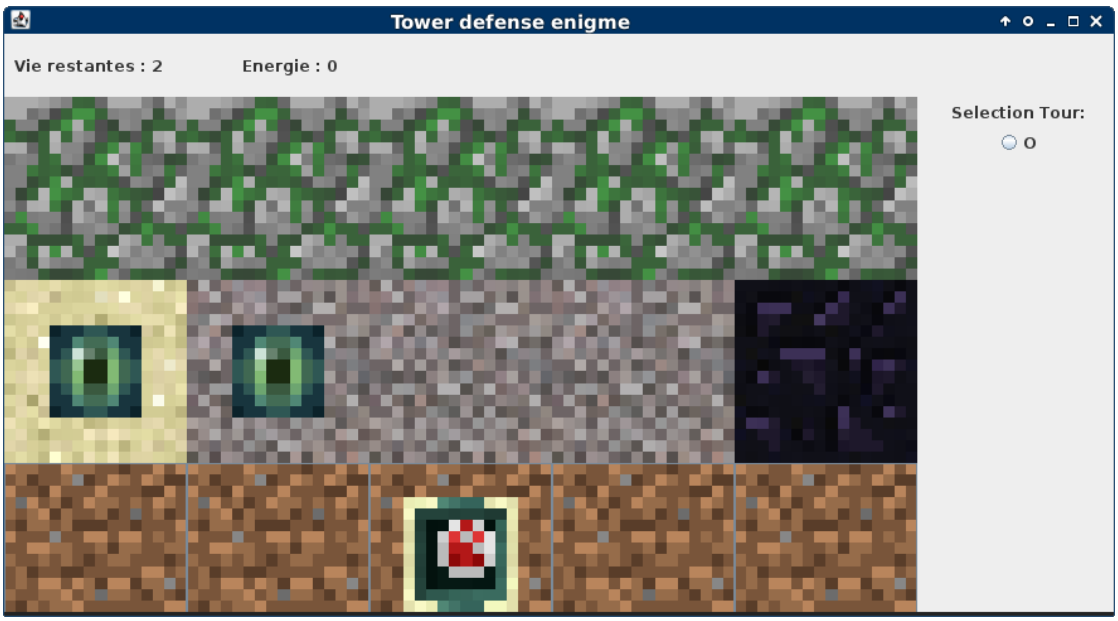
La génération du code de Jeu_* est faite à l'aide d'acceleo de manière très simple car les architetures du code java et du modèle sont identiques, il suffit donc de convertir chaque élément du modèle en son homologue java en l'initalisant dans Jeu_*.

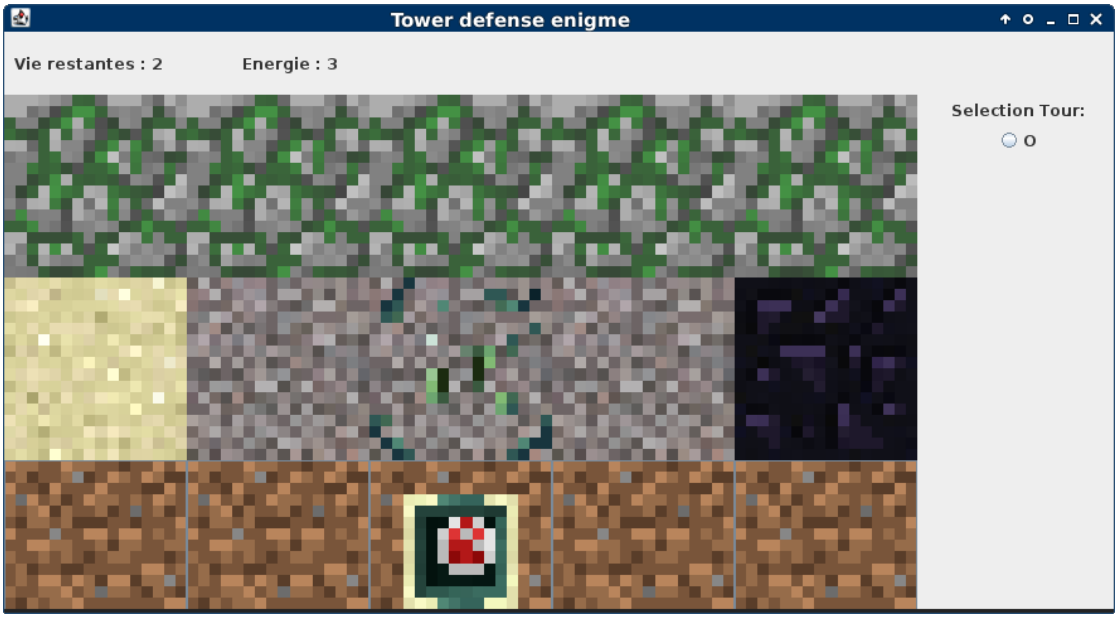
Le fichier .mtl de conversion est assez conséquent mais très simple.

La seule tâche qu'il reste après la génération est la création des images dans le dossier "ressources" si l'on souhaite utiliser l'interface graphique. A part cela le code est immédiatement compilable et executable.

7 Exemples

Voici ce que donne l'exemple décrit par le sujet avec l'interface graphique :





8 Conclusion

En premier lieu, on a définit la syntaxe concrète du langage de modélisation de jeu de défense en utilisant Xtext qui permet de générer le métamodèle Ecore . On a mis en place également des editeurs graphiques(avec sirius) qui facilitent la creation de ces modèles de jeu
Ensuit on a définit certains contraintes OCL qui permettent de détecter certains modèles de jeux de défense qui ne possèderaient pas de solutions ou qui ne sont pas cohérents.
Enfin on a réalisé 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.