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

Institut National Polytechnique

Génie des Logiciels et des Systèmes

Rapport de projet

BENDAGHA Youssef, BONNET Olivier, STEUX Yoann,
XAMBILI Robin

SOMMAIRE

1 Introduction

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.

2 Choix de conception du métamodèle

En ce qui concerne la conception du métamodèle associé à un jeu de défense, nous avons suivi certains points énoncés dans le sujet :

- Un jeu de défense est caractérisé par des éléments ;
- Chaque élément est caractérisé par un nom qui permet de le désigner ;
- Les éléments de jeu peuvent être des mobiles, des obstacles, des projectiles et des natures de terrain ;
- Une partie est caractérisée par un terrain de jeu, par une liste de niveaux de jeu et par la description de vagues de mobiles ;
- Un terrain de jeu est représenté par une liste de cases indexées par un numéro de ligne (entre 0 et nbLignes - 1) et de colonne (entre 0 et nbColonnes - 1) ;
- 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 ;
- Chaque case d'un terrain de jeu est caractérisée par une nature de terrain ;
- La nature d'un terrain peut être soit une entrée, soit une sortie, soit un chemin, soit un campement, soit une décoration ;

- Une case de chemin est caractérisée par une quantité d'énergie que doit consommer un mobile pour entrer dans la case ;
- 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 ;
- Un mobile est caractérisé par son volume ;
- Les mobiles et les obstacles sont caractérisés par une quantité d'énergie maximale ;
- 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) ;
- La force/faiblesse des mobiles/obstacles est déterminée par la quantité d'énergie disponible pour cet élément ;
- Un projectile est caractérisé par sa portée, par sa masse, par sa vitesse et par sa quantité d'énergie ;
- 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 déployée sans défaite du joueur et par les obstacles prépositionnés en début de vague ;
- 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 ;
- Chaque obstacle prépositionné est caractérisé par la case campement sur laquelle il est positionné.

Ainsi, nous pouvons en obtenir le métamodèle de la Figure 1.

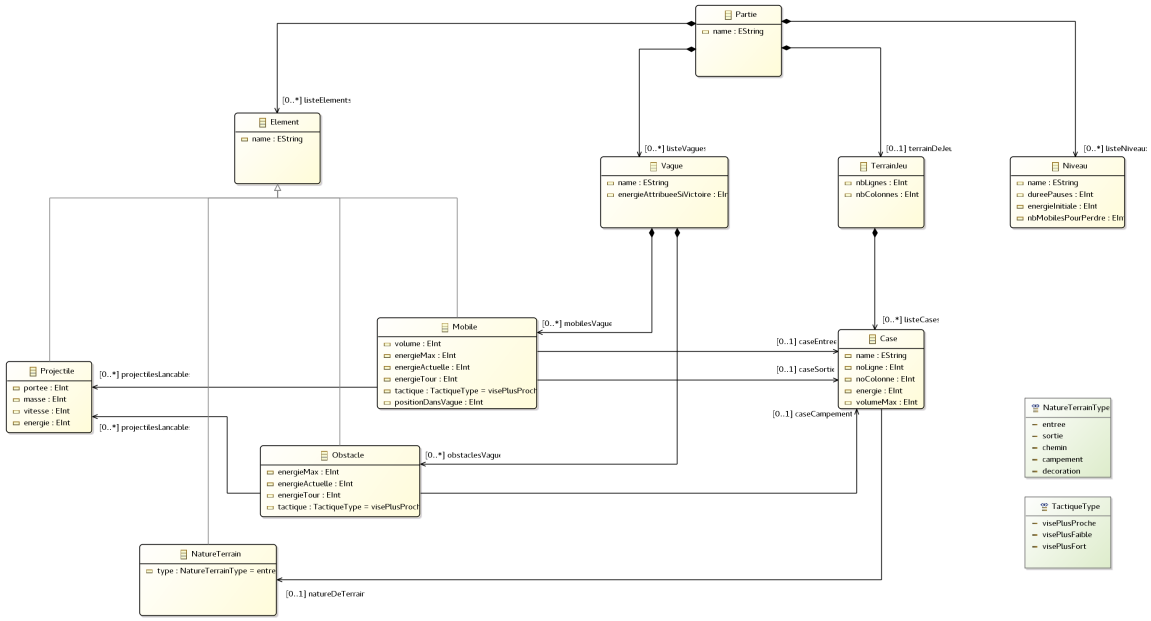

Figure 1: Métamodèle d'un jeu de défense.

3 Contraintes OCL du métamodèle

Concernant les contraintes OCL, nous nous sommes référés à certains points énoncés dans le sujet :

- Chaque élément est caractérisé par un nom qui permet de le désigner. Cela a été traduit en deux contraintes OCL : une pour s'assurer que deux éléments différents ont des noms différents, et une autre pour imposer à tout élément d'avoir effectivement un nom.

- Un terrain de jeu est représenté par un damier composé de cases indexées par un numéro de ligne (entre 0 et nbLignes - 1) et de colonne (entre 0 et nbColonnes - 1). Pour cela il suffit de vérifier que chacune des cases de ce terrain respectent cette indexation, et empêcher également que deux cases différentes aient les mêmes coordonnées.

- 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. Cela ce traduit en trois contraintes OCL qui fonctionnent de la même manière: en empêchant que le nombre de cases - dans ce terrain

de jeu - d'une certaine nature de terrain (entrée, sortie ou campement) soit inférieur à un.

- 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. Cela revient à définir une fonction qui évalue la contiguité de deux cases, et une autre qui renvoie le chemin relié à une certaine case (dans ce cas ce sera une case de nature de terrain entrée). Ces deux fonctions serviront à définir la contrainte précedemment citée. Cependant, la façon dont cette contrainte à été décrite en OCL ne permet pas de la définir correctement.

- 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. Pour cela nous vérifions que la somme des énergies de tout les mobiles de cette vague est inférieur à la somme des énergies de tout les projectiles transportés par tout les obstacles prépositionnés.

  - l'énergie attribuée au joueur pour une vague est suffisante dans le meilleur des cas pour éviter une défaite. Nous n'avons pas pu réaliser cette contrainte du fait de sa complexité et du temps qu'il faut pour bien la traduire en contrainte OCL.

  - l'énergie attribuée au joueur pour une vague est insuffisante dans le pire des cas pour éviter une défaite. Nous n'avons pas pu réaliser cette contrainte du fait de sa complexité et du temps qu'il faut pour bien la traduire en contrainte OCL.

- Les cases de chemin, d'entrée ou de sortie sont uniquement accessibles aux mobiles et aux projectiles lancés. Un projectile n'étant pas défini - entre autres - par la case (ie sa position sur le terrain) et de toute façon ayant accés à tout type de case (ie quelque soit sa nature de terrain), la contrainte OCL se résume donc à vérifier que bien aucun obstacle n'est positionné sur une case de type chemin, entrée ou sortie.

- Une case de campement est uniquement accessible aux obstacles et aux projectiles lancés. En suivant le même raisonnement que précedemment, il suffit dans ce cas de vérifier qu'aucun mobile n'est positionné sur une case de nature de terrain campement.

- Une case de décoration est uniquement accessible aux projectiles lancés. Pareil que précedemment, on doit vérifier qu'aucun obstacle ni mobile n'est positionné sur une case de nature de terrain décoration.

Remarque : Toutes les contraintes réalisée l'ont été à l'aide de CompleteOCL et non OCLinEcore.

4 Syntaxe textuelle avec Xtext

La syntaxe textuelle associée à notre métamodèle d'un jeu de défensé est basée sur la disposition de celui-ci.

Il y a tout de même quelques éléments notables :

- Nous avons ajouté des attributs name pour Partie, Vague, Case et Niveau pour faciliter la transformation modèle à texte (c.f. Section 6) ;
- Dans Obstacle, Mobile, Vague, Case, nous avons permis des références croisées (avec des crochets autour du type d'un attribut) afin de faciliter la création d'un jeu de défense.

On obtient alors la syntaxe textuelle, écrite avec Xtext, décrite sur la Figure 2.

5 Éditeur graphique avec Sirius

Pour l'éditeur graphique, nous nous appuyons sur une représentation graphique que nous avons choisie, que l'on peut voir sur la Figure 3.

À propos de cet éditeur graphique :

- Nous avons choisi de représenter TerrainJeu (respectivement Vague) par un *container* comprenant les *nodes* Case (respectivement Mobile et Obstacle) ;
- Pour chaque *node*, nous avons choisi de mettre dans son *label* toutes ses informations disponibles, pour avoir une vue d'ensemble ;
- Nous avons choisi de créer des *relation based edges* pour faire correspondre une nature de terrain à une case, un projectile lançable à un mobile / obstacle, une case de campement à un obstacle, etc. ;
- Nous avons essayé de choisir des couleurs cohérentes pour assurer la compréhension d'une partie de défense ;

```
grammar n7.Towdef with org.eclipse.xtext.common.Terminals

generate towdef "http://www.Towdef.n7"

////////////////////////////////////////////////////////////////////////////////

Partie :
	{Partie}
	'partie'
	name = ID
	'{'
		'listeElements' '(' listeElements += Element (',' listeElements += Element)* ')'
		'terrainDeJeu' '(' terrainDeJeu = TerrainJeu ')'
		'listeNiveaux' '(' listeNiveaux += Niveau (',' listeNiveaux += Niveau)* ')'
		'listeVagues' '(' listeVagues += Vague (',' listeVagues += Vague)* ')'
	'}'
;

Element :
	Element_Impl | Obstacle | Mobile | Projectile | NatureTerrain
;

Element_Impl returns Element :
	{Element}
;

Obstacle :
	{Obstacle}
	'obstacle'
	name = ID
	'{'
		'energieMax' '(' energieMax = INT ')'
		'energieActuelle' '(' energieActuelle = INT ')'
		'energieTour' '(' energieTour = INT ')'
		'tactique' '(' tactique = TactiqueType ')'
		'caseCampement' '(' caseCampement = [Case] ')'
		'projectilesLancables' '(' projectilesLancables += [Projectile] (',' projectilesLancables += [Projectile])* ')'
	'}'
;

Mobile :
	{Mobile}
	'mobile'
	name = ID
	'{'
		'volume' '(' volume = INT ')'
		'energieMax' '(' energieMax = INT ')'
		'energieActuelle' '(' energieActuelle = INT ')'
		'energieTour' '(' energieTour = INT ')'
		'tactique' '(' tactique = TactiqueType ')'
		'positionDansVague' '(' positionDansVague = INT ')'
		'caseEntree' '(' caseEntree = [Case] ')'
		'caseSortie' '(' caseSortie = [Case] ')'
		'projectilesLancables' '(' projectilesLancables += [Projectile] (',' projectilesLancables += [Projectile])* ')'
	'}'
;

Projectile :
	{Projectile}
	'projectile'
	name = ID
	'{'
		'portee' '(' portee = INT ')'
		'masse' '(' masse = INT ')'
		'vitesse' '(' vitesse = INT ')'
		'energie' '(' energie = INT ')'
	'}'
;

NatureTerrain :
	{NatureTerrain}
	'natureTerrain'
	name = ID
	'{'
		'type' '(' type = NatureTerrainType ')'
	'}'
;

Vague :
	{Vague}
	'vague'
	name = ID
	'{'
		'energieAttribueeSiVictoire' '(' energieAttribueeSiVictoire = INT ')'
		'mobilesVague' '(' mobilesVague += Mobile (',' mobilesVague += Mobile)* ')'
		'obstaclesVague' '(' obstaclesVague += Obstacle (',' obstaclesVague += Obstacle)* ')'
	'}'
;

TerrainJeu :
	{TerrainJeu}
	'terrainJeu'
	'{'
		'nbLignes' '(' nbLignes = INT ')'
		'nbColonnes' '(' nbColonnes = INT ')'
		'listeCases' '(' listeCases += Case (',' listeCases += Case)* ')'
	'}'
;

Case :
	{Case}
	'case'
	name = ID
	'{'
		'noLigne' '(' noLigne = INT ')'
		'noColonne' '(' noColonne = INT ')'
		'energie' '(' energie = INT ')'
		'volumeMax' '(' volumeMax = INT ')'
		'natureDeTerrain' '(' natureDeTerrain = [NatureTerrain] ')'
	'}'
;

Niveau :
	{Niveau}
	'niveau'
	name = ID
	'{'
		'dureePauses' '(' dureePauses = INT ')'
		'energieInitiale' '(' energieInitiale = INT ')'
		'nbMobilesPourPerdre' '(' nbMobilesPourPerdre = INT ')'
	'}'
;

enum NatureTerrainType :
	entree = 'entree' | sortie = 'sortie' | chemin = 'chemin' | campement = 'campement' | decoration = 'decoration'
;

enum TactiqueType :
	visePlusProche = 'visePlusProche' | visePlusFaible = 'visePlusFaible' | visePlusFort = 'visePlusFort'
;
```


Figure 2: Syntaxe textuelle d'un jeu de défense.

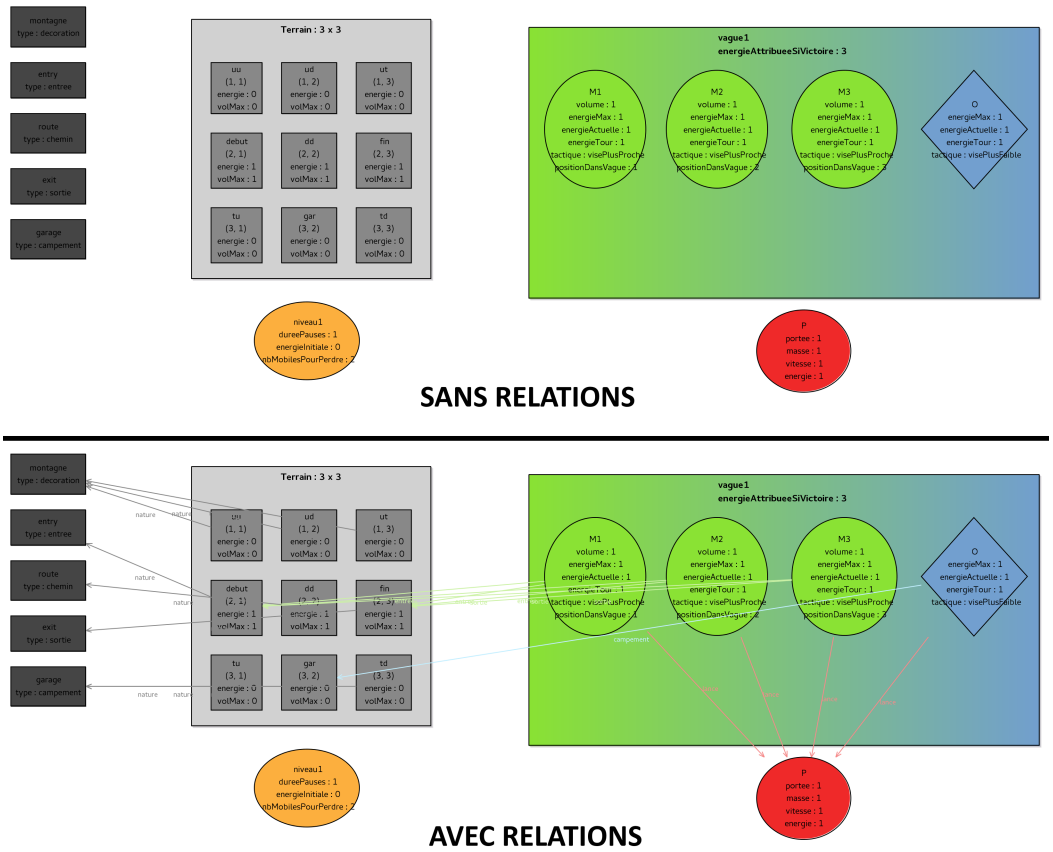

Figure 3: Vue de l'éditeur graphique (sans et avec les relations).

6 Transformation modèle à texte avec Acceleo

Cette partie correspond à l'automatisation de la création d'un fichier qui permettra l'exécution du jeu. Nous avons choisi le langage **Java**, car c'est celui que nous utilisons le plus souvent. Nous sommes donc le plus à l'aise avec celui-ci, et son orientation objet facilite la création de ce fichier.

Pour créer l'outil Acceleo qui permet la transformation modèle à texte, nous nous sommes appuyés sur le fichier que l'on aurait obtenu si l'on avait appliqué cet outil à l'exemple du sujet : Figure 4.

Nous avons donc écrit l'outil Acceleo permettant cette transformation : c.f. Figure 5.
Nous pouvons effectuer quelques remarques à propos du code Acceleo :

- Il s'agit globalement d'instancier tous les objets nécessaires à la création d'un objet Partie (permettant l'exécution) ;
- Le code étant plutôt long, nous avons décidé de créer plusieurs *sous-templates* effectuant la génération de code spécifique pour un élément (Mobile, Obstacle, Vague, etc.) ;
- Nous avons également utilisé les *query* ≪ typeToString ≫ qui permettent de renvoyer la bonne nature de terrain ou la bonne tactique, ce qui facilite l'écriture du code.

```java
import java.util.ArrayList;
import n7.towerDefense.game.*;
import org.newdawn.slick.AppGameContainer;
import org.newdawn.slick.SlickException;

public class enigme_sujet {
	public static void main(String[] args) throws SlickException{
		NatureTerrain montagne = new NatureTerrain("montagne", NatureTerrainType.Decoration);
		NatureTerrain entry = new NatureTerrain("entry", NatureTerrainType.Entree);
		NatureTerrain route = new NatureTerrain("route", NatureTerrainType.Chemin);
		NatureTerrain exit = new NatureTerrain("exit", NatureTerrainType.Sortie);
		NatureTerrain garage = new NatureTerrain("garage", NatureTerrainType.Campement);

		// Terrain de jeu
		Case[][] cases = new Case[3][5];
		cases[0][0] = new Case("uu", 0, 0, 0, 0, montagne);
		cases[0][1] = new Case("ud", 0, 1, 0, 0, montagne);
		cases[0][2] = new Case("ut", 0, 2, 0, 0, montagne);
		cases[0][3] = new Case("uq", 0, 3, 0, 0, montagne);
		cases[0][4] = new Case("uc", 0, 4, 0, 0, montagne);
		cases[1][0] = new Case("debut", 1, 0, 1, 1, entry);
		cases[1][1] = new Case("dd", 1, 1, 1, 1, route);
		cases[1][2] = new Case("dt", 1, 2, 1, 1, route);
		cases[1][3] = new Case("dq", 1, 3, 1, 1, route);
		cases[1][4] = new Case("fin", 1, 4, 1, 1, exit);
		cases[2][0] = new Case("tu", 2, 0, 0, 0, garage);
		cases[2][1] = new Case("td", 2, 1, 0, 0, garage);
		cases[2][2] = new Case("tt", 2, 2, 0, 0, garage);
		cases[2][3] = new Case("tq", 2, 3, 0, 0, garage);
		cases[2][4] = new Case("tc", 2, 4, 0, 0, garage);
		TerrainJeu terrain = new TerrainJeu(cases, 3, 5);

		// Liste des elements
		Projectile P = new Projectile("P", 1, 1, 1, 1);
		ArrayList<Projectile> projectilesLancables_M1 = new ArrayList<Projectile>();
		projectilesLancables_M1.add(P);
		Mobile M1 = new Mobile("M1", 1, new TactiquePlusProche(), projectilesLancables_M1, 1, 1, cases[1][0], cases[1][4]);
		ArrayList<Projectile> projectilesLancables_M2 = new ArrayList<Projectile>();
		projectilesLancables_M2.add(P);
		Mobile M2 = new Mobile("M2", 1, new TactiquePlusProche(), projectilesLancables_M2, 1, 2, cases[1][0], cases[1][4]);
		ArrayList<Projectile> projectilesLancables_M3 = new ArrayList<Projectile>();
		projectilesLancables_M3.add(P);
		Mobile M3 = new Mobile("M3", 1, new TactiquePlusProche(), projectilesLancables_M3, 1, 3, cases[1][0], cases[1][4]);
		ArrayList<Projectile> projectilesLancables_O = new ArrayList<Projectile>();
		projectilesLancables_O.add(P);
		Obstacle O = new Obstacle("O", 1, new TactiquePlusFaible(), projectilesLancables_O, cases[2][2]);

		ArrayList<Element> listeElements = new ArrayList<Element>();
		listeElements.add(P);
		listeElements.add(M1);
		listeElements.add(M2);
		listeElements.add(M3);
		listeElements.add(O);
		listeElements.add(montagne);
		listeElements.add(entry);
		listeElements.add(route);
		listeElements.add(exit);
		listeElements.add(garage);

		// Liste des vagues
		ArrayList<Mobile> mobilesVague = new ArrayList<Mobile>();
		mobilesVague.add(M1);
		mobilesVague.add(M2);
		mobilesVague.add(M3);
		ArrayList<Obstacle> obstaclesVague = new ArrayList<Obstacle>();
		obstaclesVague.add(O);
		Vague vague1 = new Vague(3, mobilesVague, obstaclesVague);
		ArrayList<Vague> listeVagues = new ArrayList<Vague>();
		listeVagues.add(vague1);

		// Liste des niveaux
		Niveau niveau1 = new Niveau(1, 0, 2);
		ArrayList<Niveau> listeNiveaux = new ArrayList<Niveau>();
		listeNiveaux.add(niveau1);

		// Partie finale
		Partie enigme_sujet = new Partie(listeElements, listeVagues, terrain, listeNiveaux);
		AppGameContainer app = new AppGameContainer(enigme_sujet);
		int width;
		int height;
		float ratio = terrain.getNbLignes()/terrain.getNbColonnes();
		if (ratio > 1) {
			width = app.getScreenWidth();
			height = (int) (width*ratio);
			if (height > app.getScreenHeight()) {
				height = app.getScreenHeight();
			}
		} else {
			height = app.getScreenHeight();
			width = (int) (height/ratio);
			if (width > app.getScreenWidth()) {
				width = app.getScreenWidth();
			}
		}
		app.setDisplayMode(width, height, false);
		app.start();
	}
}
```


Figure 4: Fichier Java que l'on aurait obtenu après l'utilisation de l'outil Acceleo.

```
[comment encoding = UTF-8 /]
[module toPrototype('http://www.Towdef.n7')]


[template public toPrototype(partie : Partie)]
[comment @main/]
[file (partie.name + '.java', false, 'UTF-8')]
import java.util.ArrayList;
import n7.towerDefense.game.*;
import org.newdawn.slick.AppGameContainer;
import org.newdawn.slick.SlickException;

public class [partie.name/] {
	public static void main(String['['/']'/] args) throws SlickException{
[for (natureTerrain : NatureTerrain | partie.listeElements)]
[genererJavaNatureTerrain(natureTerrain)/]
[/for]

		// Terrain de jeu
		Case['['/']'/']['['/']'/] cases = new Case['['/][partie.terrainDeJeu.nbLignes/]']'/]['['/][partie.terrainDeJeu.nbColonnes/]']'/];
[for (case : Case | partie.terrainDeJeu.listeCases)]
[genererJavaCase(case)/]
[/for]
		TerrainJeu terrain = new TerrainJeu(cases, [partie.terrainDeJeu.nbLignes/], [partie.terrainDeJeu.nbColonnes/]);

		// Liste des elements
[for (projectile : Projectile | partie.listeElements)]
[genererJavaProjectile(projectile)/]
[/for]
[for (mobile : Mobile | partie.listeVagues.mobilesVague)]
[genererJavaMobile(mobile)/]
[/for]
[for (obstacle : Obstacle | partie.listeVagues.obstaclesVague)]
[genererJavaObstacle(obstacle)/]
[/for]

		ArrayList<Element> listeElements = new ArrayList<Element>();
[for (projectile : Projectile | partie.listeElements)]
		listeElements.add([projectile.name/]);
[/for]
[for (mobile : Mobile | partie.listeVagues.mobilesVague)]
		listeElements.add([mobile.name/]);
[/for]
[for (obstacle : Obstacle | partie.listeVagues.obstaclesVague)]
		listeElements.add([obstacle.name/]);
[/for]
[for (natureTerrain : NatureTerrain | partie.listeElements)]
		listeElements.add([natureTerrain.name/]);
[/for]

		// Liste des vagues
[for (vague : Vague | partie.listeVagues)]
[genererJavaVague(vague)/]
[/for]
		ArrayList<Vague> listeVagues = new ArrayList<Vague>();
[for (vague : Vague | partie.listeVagues)]
		listeVagues.add([vague.name/]);
[/for]

		// Liste des niveaux
[for (niveau : Niveau | partie.listeNiveaux)]
[genererJavaNiveau(niveau)/]
[/for]
		ArrayList<Niveau> listeNiveaux = new ArrayList<Niveau>();
[for (niveau : Niveau | partie.listeNiveaux)]
		listeNiveaux.add([niveau.name/]);
[/for]

		// Partie finale
		Partie [partie.name/] = new Partie(listeElements, listeVagues, terrain, listeNiveaux);
		AppGameContainer app = new AppGameContainer([partie.name/]);
		int width;
		int height;
		float ratio = terrain.getNbLignes()/terrain.getNbColonnes();
		if (ratio > 1) {
			width = app.getScreenWidth();
			height = (int) (width*ratio);
			if (height > app.getScreenHeight()) {
				height = app.getScreenHeight();
			}
		} else {
			height = app.getScreenHeight();
			width = (int) (height/ratio);
			if (width > app.getScreenWidth()) {
				width = app.getScreenWidth();
			}
		}
		app.setDisplayMode(width, height, false);
		app.start();
	}
}
[/file]
[/template]

[template private genererJavaNatureTerrain(natureTerrain : NatureTerrain)]
		NatureTerrain [natureTerrain.name/] = new NatureTerrain("[natureTerrain.name/]", NatureTerrainType.[natureTerrain.type.typeToString()/]);
[/template]

[template private genererJavaCase(case : Case)]
		cases['['/][case.noLigne - 1/]']'/]['['/][case.noColonne - 1/]']'/] = new Case("[case.name/]", [case.noLigne - 1/], [case.noColonne - 1/], [case.energie/],
[case.volumeMax/], [case.natureDeTerrain.name/]);
[/template]

[template private genererJavaProjectile(projectile : Projectile)]
		Projectile [projectile.name/] = new Projectile("[projectile.name/]", [projectile.portee/], [projectile.masse/], [projectile.vitesse/], [projectile.energie/]);
[/template]

[template private genererJavaMobile(mobile : Mobile)]
		ArrayList<Projectile> projectilesLancables_[mobile.name/] = new ArrayList<Projectile>();
		[for (projectile : Projectile | mobile.projectilesLancables)]
		projectilesLancables_[mobile.name/].add([projectile.name/]);
		[/for]
		Mobile [mobile.name/] = new Mobile("[mobile.name/]", [mobile.energieMax/], new Tactique[mobile.tactique.typeToString()/](), projectilesLancables_[mobile.name/],
[mobile.volume/], [mobile.positionDansVague/], cases['['/][mobile.caseEntree.noLigne - 1/]']'/]['['/][mobile.caseEntree.noColonne - 1/]']'/], cases['['/][mobile.caseSortie.noLigne -
1/]']'/]['['/][mobile.caseSortie.noColonne - 1/]']'/]);
[/template]

[template private genererJavaObstacle(obstacle : Obstacle)]
		ArrayList<Projectile> projectilesLancables_[obstacle.name/] = new ArrayList<Projectile>();
		[for (projectile : Projectile | obstacle.projectilesLancables)]
		projectilesLancables_[obstacle.name/].add([projectile.name/]);
		[/for]
		Obstacle [obstacle.name/] = new Obstacle("[obstacle.name/]", [obstacle.energieMax/], new Tactique[obstacle.tactique.typeToString()/](), projectilesLancables_[obstacle.name/],
cases['['/][obstacle.caseCampement.noLigne - 1/]']'/]['['/][obstacle.caseCampement.noColonne - 1/]']'/]);
[/template]

[template private genererJavaVague(vague : Vague)]
		ArrayList<Mobile> mobilesVague = new ArrayList<Mobile>();
		[for (mobile : Mobile | vague.mobilesVague)]
		mobilesVague.add([mobile.name/]);
		[/for]
		ArrayList<Obstacle> obstaclesVague = new ArrayList<Obstacle>();
		[for (obstacle : Obstacle | vague.obstaclesVague)]
		obstaclesVague.add([obstacle.name/]);
		[/for]
		Vague [vague.name/] = new Vague([vague.energieAttribueeSiVictoire/], mobilesVague, obstaclesVague);
[/template]

[template private genererJavaNiveau(niveau : Niveau)]
		Niveau [niveau.name/] = new Niveau([niveau.dureePauses/], [niveau.energieInitiale/], [niveau.nbMobilesPourPerdre/]);
[/template]

[query private typeToString(type : NatureTerrainType) : String =
	(if (type = NatureTerrainType::entree) then 'Entree'
	else if (type = NatureTerrainType::sortie) then 'Sortie'
	else if (type = NatureTerrainType::chemin) then 'Chemin'
	else if (type = NatureTerrainType::campement) then 'Campement'
	else if (type = NatureTerrainType::decoration) then 'Decoration' else 'void' endif endif endif endif endif)
/]

[query private typeToString(type : TactiqueType) : String =
	(if (type = TactiqueType::visePlusProche) then 'PlusProche'
	else if (type = TactiqueType::visePlusFaible) then 'PlusFaible'
	else if (type = TactiqueType::visePlusFort) then 'PlusFort' else 'void' endif endif endif)
/]
```


Figure 5: Code du fichier Acceleo permettant la transformation modèle à texte.

7 Exécution d'un jeu

L'exécution du jeu utilise la librairie Slick2D permettant de développer des jeux en java en 2D. Cette librairie est un ensemble d'outils se reposant sur LWJGL OpenGL.

En particulier, la classe Partie du prototype hérite de BasicGame qui possède de 3 méthodes abstraite qu'il faut donc implémenter :

- init(GameContainer container) qui initialise le jeu notamment tout les ressources (images, sons, etc)
- update(GameContainer container, int delta) qui gère la logique du jeu
- render(GameContainer container, Graphics g) qui affiche les éléments du jeu

Dans le fichier java obtenu avec l'outil Acceleo, on crée une instance app de la classe AppGameContainer avec en argument l'instance de Partie créée grâce au modèle puis on appelle la méthode app.start() qui lance la boucle de jeu. L'appel à start() se décompose basiquement comme suit :

```
partie.init();
while (true) {
    partie.update(c,delta);
    partie.render(c,g);
}
```

De plus, le prototype reprend la structure du meta-modèle en ajoutant les éléments d'affichage et d'exécution (déplacement, etc). On ajoute notamment une méthode render et update dans la classe abstraite Element ainsi que dans Vague. Les méthodes render et update de Partie font ainsi appel aux méthodes render et update de Vague.

L'exécution du tower defense se déroule à chaque tour de la façon suivante :

- Initialisation de l'énergie des mobiles et des obstacles
- Les mobiles se déplacent le plus possible
- Les mobiles tirent
- Les obstacles tirent

Si une vague a été gagnée les obstacles vivants sont propagés à la vague suivante. Entre chaque vague le joueur dispose d'une pause pour réparer, déplacer, supprimer ou dupliquer un obstacle. Les commandes sont les suivantes :

- Double-clique gauche pour sélectionner un obstacle,
- Touche Suppr pour supprimer un obstacle sélectionné,
- Touche Espace pour réparer un obstacle sélectionné,
- Clique gauche et glisser pour déplacer un obstacle,
- Clique droit et glisser pour dupliquer un obstacle.

Toutes les ressources (images, sons, animations) proviennent du site opengameart.org et nous tenons remercier les contributeurs du site et en particulier l'auteur "Kenney" qui a réalisé les visuels des mobiles, obstacles et cases.

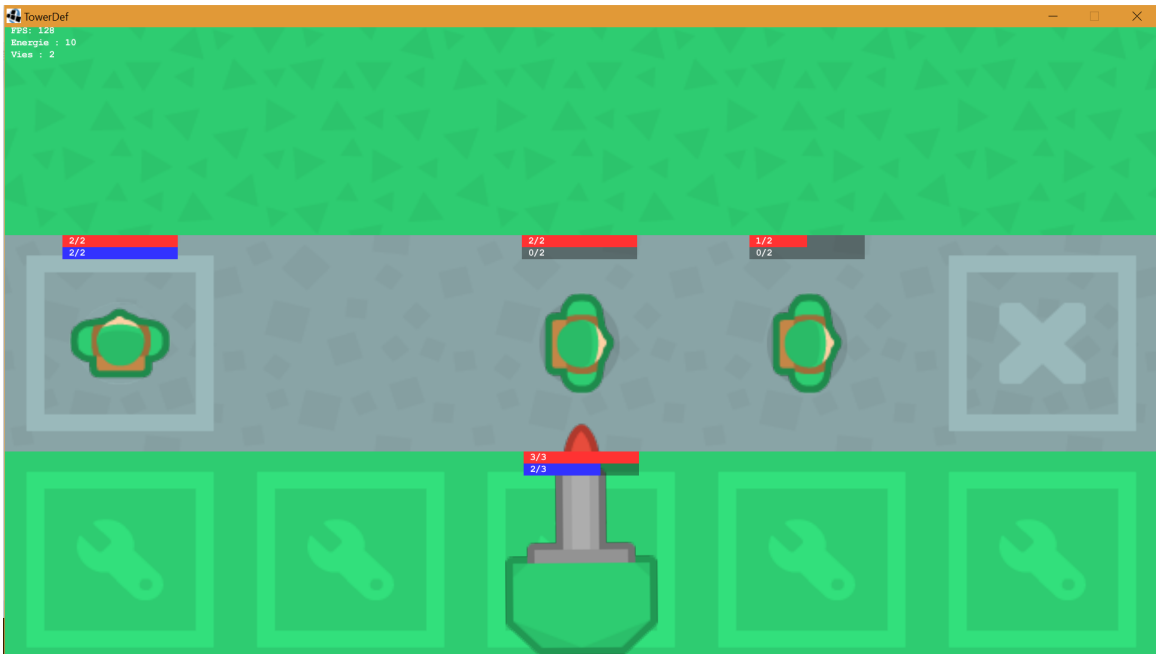

Figure 6: Une vue lors de l'exécution de jeu de défense proposé dans le sujet.

Figure 7: Une autre vue lors de l'exécution de jeu de défense proposé dans le sujet.