INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES RENNES

Projet Modélisation et Programmation Orientées Objet

4e année

BedBihan

Rapport de conception

Étudiants

Valentin ESMIEU

Corentin NICOLE

12 novembre 2014

Table des matières

Introduction

Ce document est le premier rapport du projet *Modélisation et Programmation Orientées Objet*, qui a lieu dans le cadre de la formation de quatrième année informatique de l'Insa de Rennes. L'objectif de ce projet est la réalisation d'un jeu vidéo librement inspirée de Small World, un jeu de plateau créé par Philippe Keyaerts en 2009. Le jeu ainsi réalisé portera le nom de *Bedbihan*, traduction bretonne du nom du jeu original.

La première partie de ce rapport présentera les règles précises du jeu, déterminées à partir de celles formulées dans le cahier des charges initial. La deuxième partie illustrera à l'aide de diagrammes les différentes phases de jeu, les actions possibles pour le joueur, les interactions entre les différents éléments ainsi que leur séquencage dans le temps. Enfin, la troisième partie présentera la modélisation de notre programme en précisant les différents patrons de conceptions utilisés.

Pour des raisons de cohérence avec le monde professionnel dans le développement de notre projet, *Bedbihan* sera intégralement en anglais. Par conséquent, bien que le texte de ce rapport soit en français, l'ensemble de ses illustrations, figures et diagrammes, sont en anglais.

1 Règles du jeu

1.1 But du jeu

BedBihan est un jeu de stratégie au tour-par-tour où chaque joueur dirige un peuple sur une carte. Le but est de marquer le plus de points possible en dirigeant les unités de son peuple pour qu'elles occupent au mieux les cases de la carte. Les unités d'un joueur peuvent attaquer les unités d'un autre joueur pour les détruire, limitant ainsi l'acquisition de points adverses. La partie se termine quand le nombre de tours imparti est atteint, ou qu'il ne reste qu'un seul joueur en jeu — les autres ayant perdu toutes leurs unités [1].

1.2 Carte du monde

La carte du monde, sur laquelle s'affrontent les joueurs, se compose de cases hexagonales. Chaque case possède un type de terrain : Plaine, Désert, Montagne ou Forêt. Chaque case peut être occupée par une ou plusieurs unités alliées, mais deux unités adverses ne peuvent se trouver sur la même case. Par défaut, une case occupée par une ou plusieurs unités alliées rapporte 1 point. Des bonus et malus sont possibles selon les caractéristiques de chaque peuple (voir Section 1.3).

À chaque nouvelle partie, la carte est générée aléatoirement avec le même nombre de case de type Désert, Forêt, Montagne et Plaine, afin de ne pas avantager un peuple. La carte peut avoir trois tailles différentes :
- Démo : 4x4 cases, 5 tours, 4 unités par peuple.
- Petite : 10x10 cases, 20 tours, 6 unités par peuple.
- Normale : 14x14 cases, 30 tours, 8 unités par peuple.

1.3 Peuples

BedBihan permet de jouer avec trois peuples différents : les Elfes, les Humains et les Korrigans. Chaque peuple présente des caractéristiques particulières [2].

Pour les Elfes :
- le coût de déplacement sur une case Forêt est divisé par deux.
- le coût de déplacement sur une case Désert est multiplié par deux.
- Une unité elfe a 50% de chance de survivre avec 1 point de vie lors d'un combat conduisant normalement à sa mort.

Pour les Humains :
- le coût de déplacement sur une case Plaine est divisé par deux.
- une unité humaine ne rapporte aucun point lorsqu'elle occupe une case Forêt.
- lorsqu'une unité humaine détruit une autre unité, elle marque 1 point. Les points ainsi gagnés sont liés à l'unité : si celle-ci meurt, les points sont perdus.

Pour les Korrigans :
- le coût de déplacement sur une case Plaine est divisé par deux.
- une unité korrigane ne rapporte aucun point lorsqu'elle occupe une case Plaine.

1. Pour ce projet, le nombre de joueurs sera limité à 2.
2. Ces caractéristiques se limitent pour l'instant à celles spécifiées dans les consignes du projet. Elles sont toutefois susceptibles d'être étoffées en cours de développement.

– lorsqu'une unité korrigane occupe une case Montagne, elle peut se déplacer sur n'importe quelle autre case Montage de la carte, à condition que celle-ci ne soit pas occupée par une unité adverse.

1.4 Unités

Chaque unité possède des points de vie, des points de déplacement, des points d'attaque et des points de défense. Les points de vie représentent la santé de l'unité : plus ceux-ci sont élevés, plus l'unité attaquera ou se défendra efficacement (voir Section 1.6). Lorsque les points de vie d'une unité atteignent 0, cette dernière est détruite. Ces points de vie ne se régénèrent pas d'un tour à l'autre.

Les points d'attaque représentent la force brute d'une unité lorsqu'elle en attaque une autre. Plus ceux-ci sont élevés, plus l'unité infligera de dégâts à l'adversaire. De la même façon, les points de défense représentent la capacité d'une unité à se défendre : plus ceux-ci sont élevés, et plus l'unité sera susceptible de résister aux attaques.

Les points de déplacement, quant à eux, représentent la capacité de mouvement d'une unité à travers la carte. Plus ces points sont élevés, plus l'unité pourra se déplacer loin lors d'un tour. Ces points de déplacement sont restaurés d'un tour à l'autre.

1.5 Déroulement d'un tour

Lorsque vient le tour d'un joueur, celui-ci peut bouger chacune de ses unités tant qu'elles ont suffisamment de points de déplacement. Par défaut, se déplacer d'une case coûte 1 point de déplacement. Une unité peut également passer son tour.

Lorsqu'une unité tente de se déplacer sur une case sur laquelle se trouve une ou plusieurs unités ennemies, un certain nombre de combat s'engagent (voir Section 1.6). Une fois les combats terminés, si toutes les unités adverses ont été détruites, l'unité se déplace effectivement sur la case, en consommant le nombre de points de déplacement nécessaire. Si au moins une unité ennemie est encore en vie, l'unité alliée reste sur sa case actuelle mais consomme malgré tout le même nombre de points de déplacement.

Ainsi, lors d'un tour, une unité peut attaquer et se déplacer autant de fois que le permet son nombre de points de déplacement. Lorsqu'un joueur a terminé de déplacer toutes ses unités, c'est au tour du joueur suivant, et ainsi de suite.

1.6 Combats

Un combat a lieu lorsqu'une unité tente de se déplacer sur une case occupée par l'ennemi. Une unité attaque seule et ne peut attaquer qu'une seule unité adverse à la fois : dans le cas où plusieurs unités ennemies se situent sur la même case, c'est l'unité adverse ayant la défense la plus élevée qui est affrontée.

Un combat se compose d'un certain nombre d'affrontements. Ce nombre est choisi aléatoirement au début du combat selon la formule suivante :

$$NombreAffrontements = random(3, 2 + max(PVattaquant, PVattaqué)).$$

À chaque affrontement, un calcul de probabilités de victoire de l'une ou l'autre unité est effectué. Si l'attaque de l'attaquant est supérieure à la défense de l'attaqué, la formule est la suivante :

$$probaVictoireAttaquant = 1 - (0.5 * defense/attaque)$$

Par exemple, une unité avec 2 en attaque faisant face à une unité ayant 1 en défense possède 75% de chances de gagner à chaque affrontement — et donc d'infliger des dégats à chaque fois. Inversement, l'unité attaquée n'a que 25% de chances de gagner l'affrontement, et donc de se défendre.

Si toutefois l'attaque de l'attaquant est inférieure à la défense de l'attaqué, la formule est alors celle-ci :

$$probaVictoireAttaquant = 0.5 * attaque/defense$$

Par exemple, une unité avec 3 en attaque faisant face à une unité ayant 4 en défense possède 37.5% de chances de gagner chaque affrontement. L'unité défensive possède donc 62.5% de chances de se défendre et d'infliger des dégats à l'attaquant.

L'unité remportant la victoire inflige des dégâts à l'unité adverse selon la formule suivante :

$$DégâtsInfligés = partieEntière((PVactuels/PVmax) * attaque)$$

Par exemple, une unité ayant 5 en attaque et 3 points de vie sur ses 6 points de vie maximum fera perdre 2 points de vie à une unité adverse lors d'un affrontement remporté.

Le combat se termine lorsque le nombre d'affrontements est atteint, ou que l'une des deux unités est détruite.

2 Conception

2.1 Déroulement du jeu

Le diagramme d'état-transition de la FIGURE 1 présente les différents états dans lequel *Bedbihan* peut se situer.

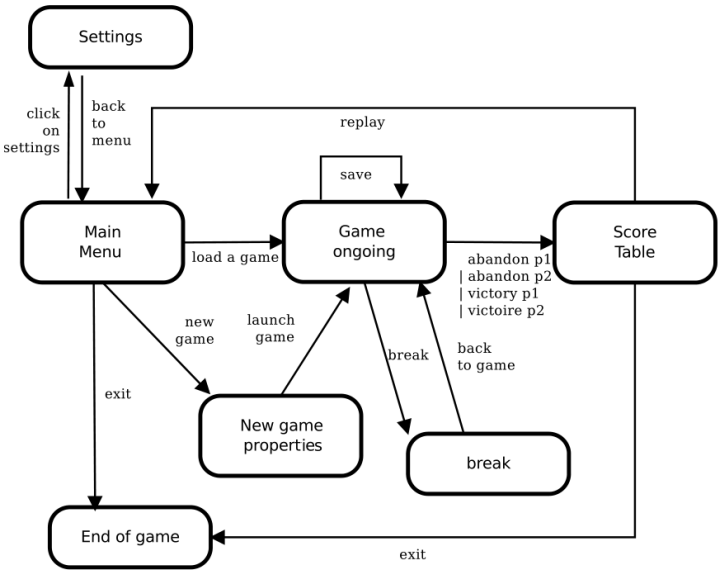


FIGURE 1 – Diagrame d'états-transitions du jeu.

2.2 Lancement du jeu

Au lancement de *Bedbihan*, l'utilisateur peut débuter une nouvelle partie ou reprendre une partie préalablement sauvegardée. Le lancement d'une nouvelle partie implique de choisir une taille de carte et de sélectionner un peuple pour chacun des joueurs. Ceci est illustré par le diagramme d'utilisation de la FIGURE 2.

2.3 Déroulement d'un tour pour un joueur

BedBihan se jouant à tour de rôle, chaque joueur dispose de différentes actions possible lorsque c'est à lui de joueur. Les différentes possibilités sont illustrées par le diagramme d'utilisation présenté en FIGURE 3.

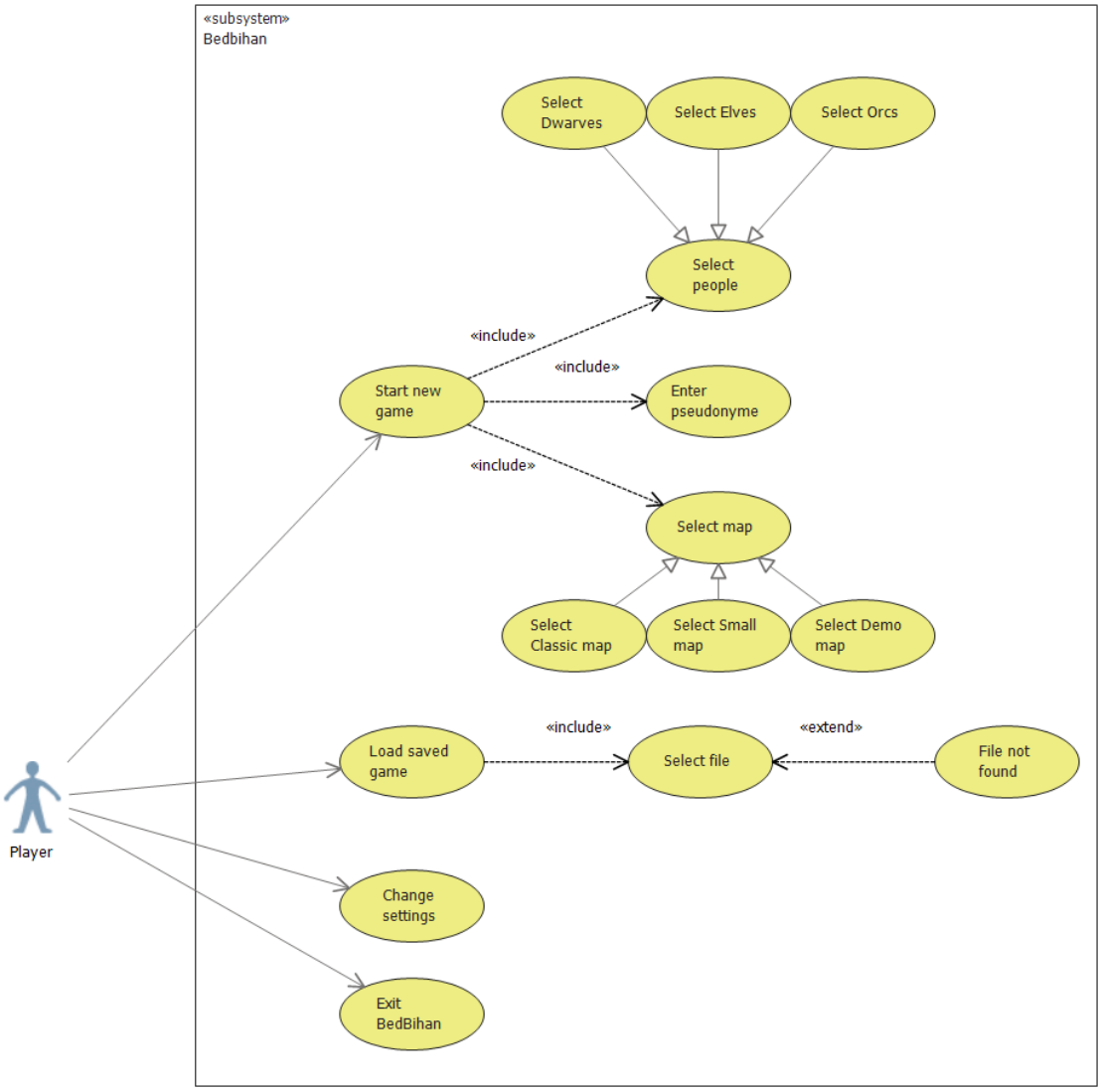


FIGURE 2 – Actions possibles lors du démarrage du jeu.

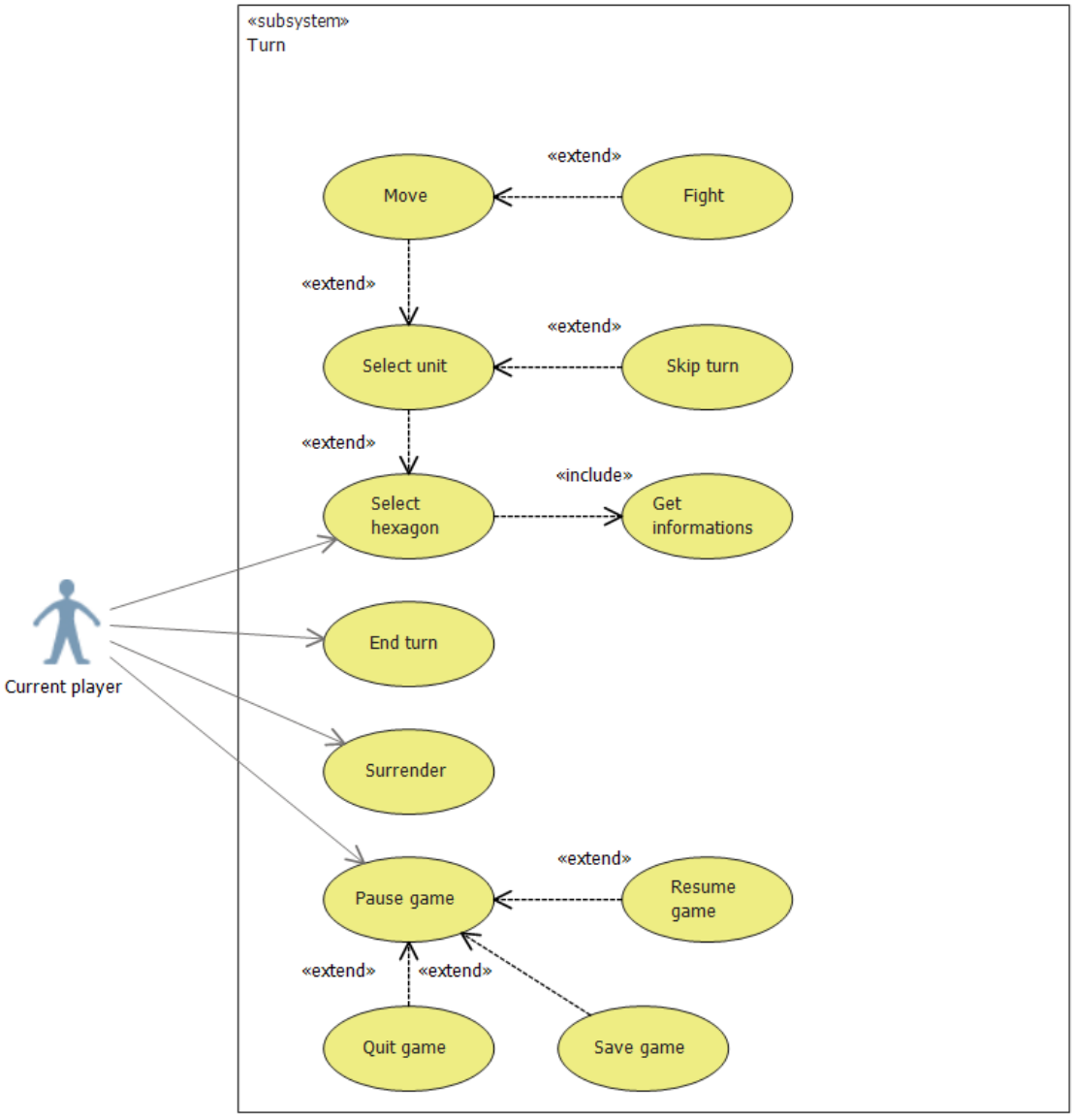


FIGURE 3 – Actions possibles pour un joueur lors d'un tour.

2.4 Cycle de vie d'une unité

Les unités d'un joueur ont un cycle de vie illustré par le diagramme d'activité en FIGURE 4.

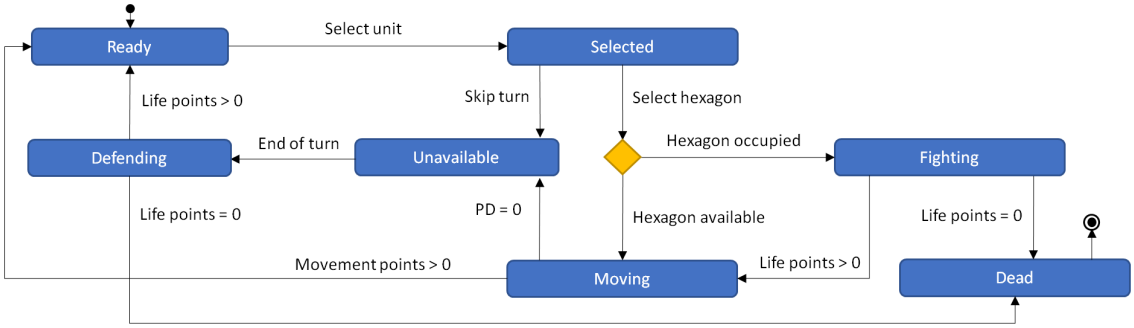


FIGURE 4 – Cycle de vie d'une unité.

3 Diagramme de classes

Nous allons maintenant détailler le diagramme de classes de *BedBihan* en expliquant les différents patrons de conception utilisés. Le diagramme complet est disponible en annexe [3].

3.1 Monteur : création d'une partie

Un joueur peut soit commencer une nouvelle partie, soit en charger une existante. Il existe donc différentes manières de construire une partie ; néanmoins, dans les deux cas, des étapes restent similaires (création de la carte, créations des unités, etc.). C'est pourquoi la création de la partie est assurée par le patron de conception **Monteur**. Ce patron de conception nous permet de séparer la conception de l'objet *Game* de sa représentation. Comme indiqué sur la FIGURE 5, la création de la partie sera dirigé par *GameCreator*, qui utilisera le monteur adéquat en fonction de la situation, minimisant ainsi le code nécessaire.

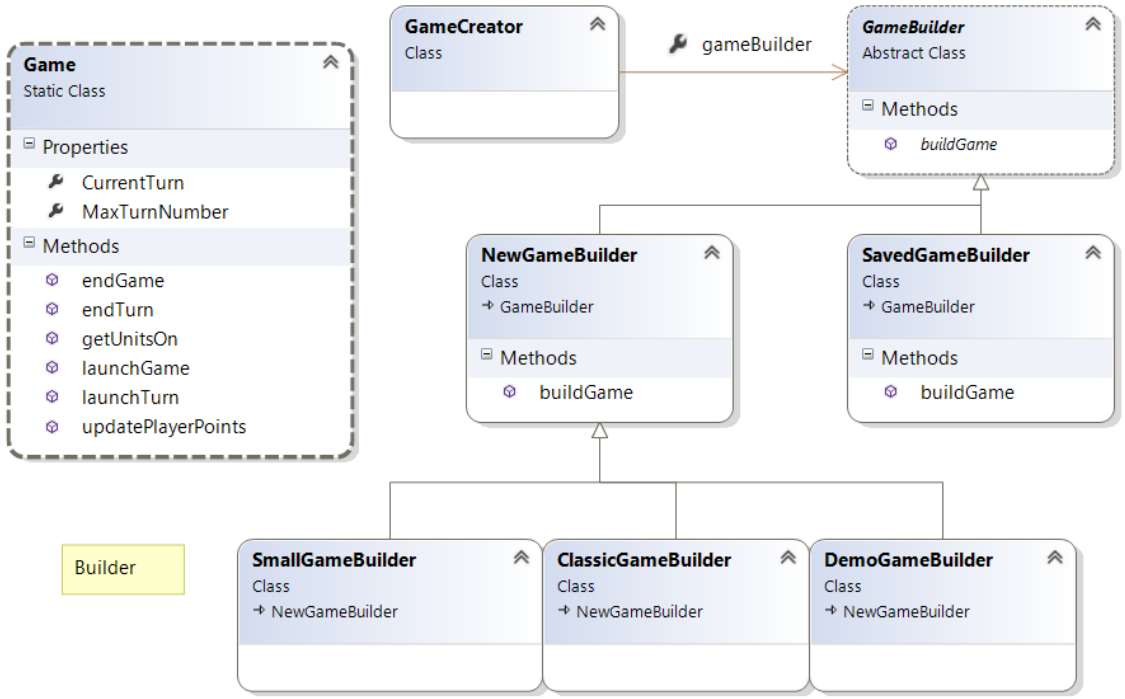


FIGURE 5 – Création de la partie : patron de conception Monteur.

3.2 Fabrique : création des unités de chaque peuple

Chaque joueur possède une armée composée d'un certain nombre d'unités d'un même peuple. Ces unités sont soit des Elfes, soit des Humains, soit des Korrigans. Cette règle se traduit par un objet *Player* qui possède une *Faction*, elle même composé d'objets *Units*. Cette modélisation est

3. Pour une meilleure lisibilité, le diagramme est également joint au format PNG avec ce rapport.

illustrée par la FIGURE 6. *Units* est une classe abstraite implémentée par trois classes concrètes représentants les trois peuples disponibles. La création de ces unité est assurée par le patron de conception **Fabrique**. Ce patron de conception nous permettra de créer l'armée adéquate en fonction du peuple choisit par le joueur au lancement de la partie. Il assure une modularité du code, car sa fonction *CreateUnit* retourne un type abstrait. Ainsi, l'appelant n'a pas besoin de se préoccuper de la classe concrète de l'objet retourné par cette méthode.

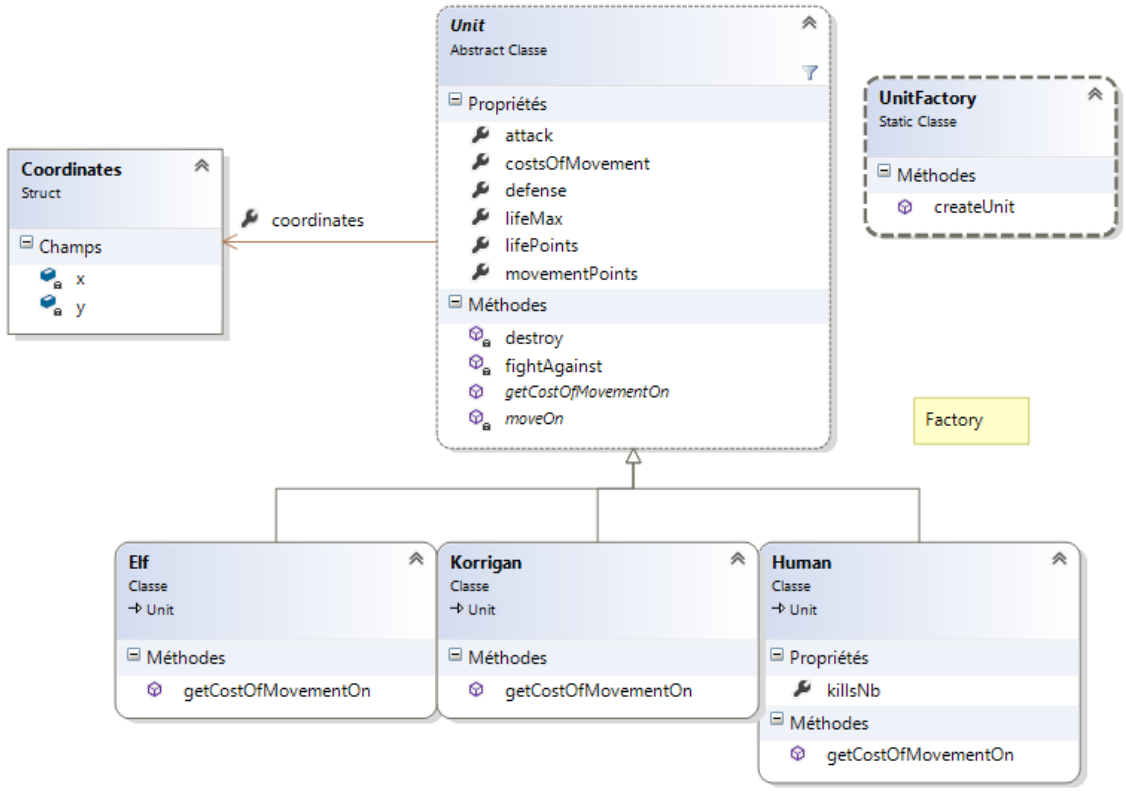


FIGURE 6 – Création des unités : patron de conception Fabrique.

3.3 Poids-mouche : création des hexagones

La classe *Game* possède un plateau *Board*, composé de cases *Hexagone*. Comme un plateau peut par exemple contenir 194 cases, l'ensemble des cases occuperait une grande quantité de mémoire si chaque case était instanciée une à une. Nous utilisons donc ici le patron de conception **Poids-mouche** représenté sur la FIGURE 7 pour minimiser ces instances d'*Hexagone*. La classe *HexagonFactory* se charge de fournir les quatre types de cases lors de la création de la carte. Ainsi, pour chaque case du plateau, s'il s'agit d'un type de case qui n'a pas encore été instancié, il la fabrique ; sinon, il fournit simplement l'objet déjà existant. De cette manière, on représente un nombre importants de cases par la répétitions de seulement quatre instances : *Woods, Mountains, Desert* et *Plain*. Cette modélisation nous impose de ne pas implémenter de paramètres à l'objet *Hexagone* car il serait similaire à toutes les cases de même type. Ainsi, ce seront les objets *Units* qui connaitront leur position sur le plateau, et non pas les cases qui sauront quelles unités sont sur elles.

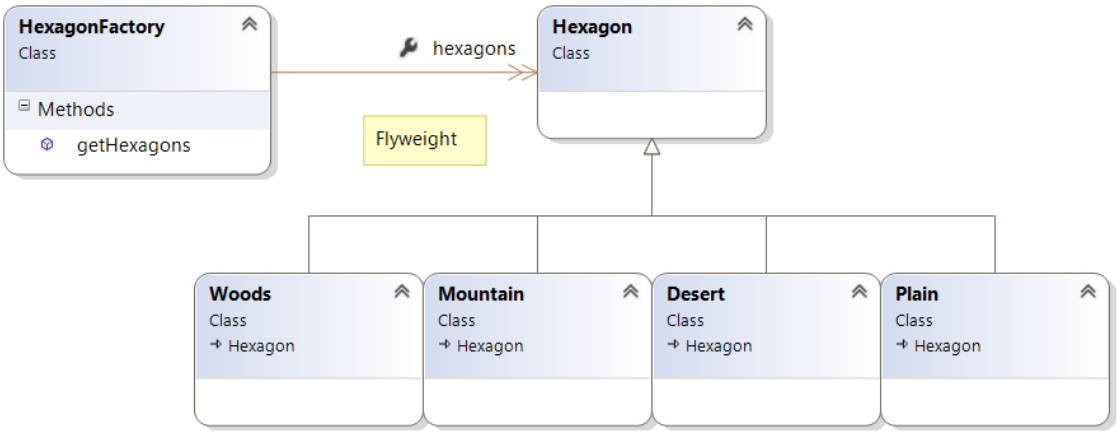


FIGURE 7 – Gestion des cases : patron de conception Poids-mouche.

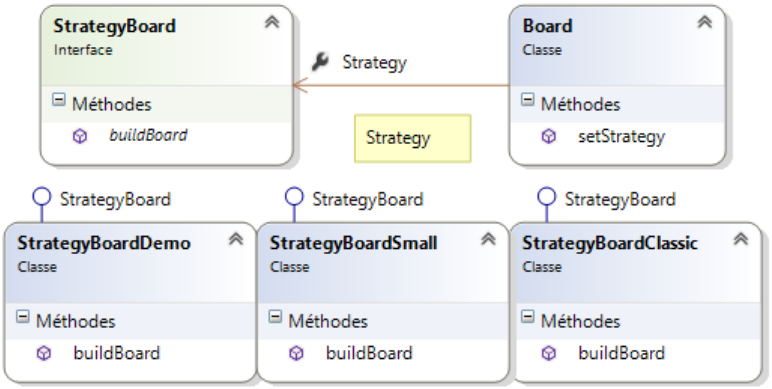


FIGURE 8 – Création des différents types de carte : patron de conception Stratégie.

3.4 Stratégie : création des différents type de carte

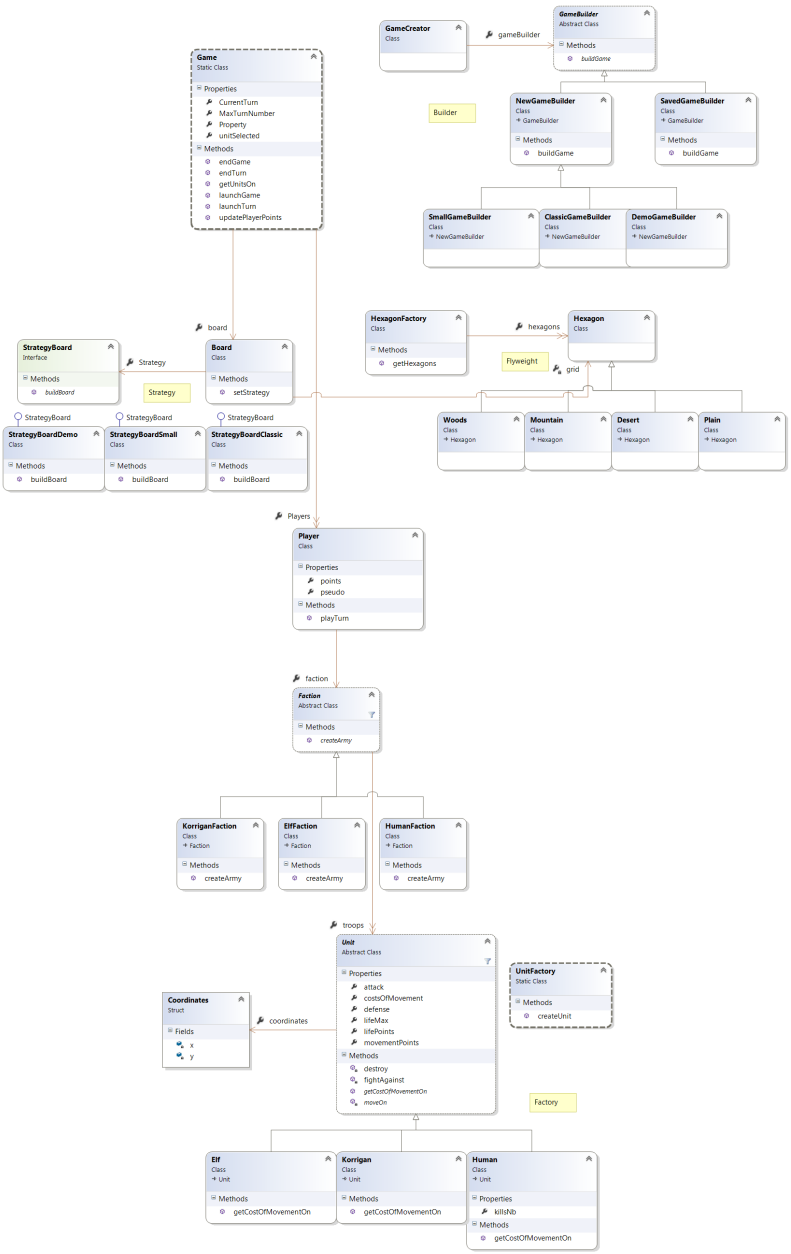
Comme indiqué dans la Section 1.2, il existe différentes tailles de cartes : Démo, Petite et Normale. La disposition des cases doit être décidée aléatoirement lors de la création d'une nouvelle carte, et ce indépendamment de sa taille. De plus, le nombre de case doit être un multiple de 4, afin d'avoir autant de cases de type Desert, Forêt, Montagne et Plaine. Nous allons donc utiliser le patron de conception **Stratégie** pour l'initialisation d'une nouvelle carte. Comme le montre la FIGURE 8, la classe *Board* possède un attribut *StrategyBoard* interchangeable qui définit la stratégie d'implémentation du plateau.

Conclusion

Maintenant que la conception de *BedBihan* a été effectuée, nous pouvons passer au développement réel du projet. Le code généré automatiquement par Visual Studio 2013 — le logiciel avec lequel nous avons réalisé notre diagramme de classe — nous servira de base pour l'implémentation de toutes les fonctionnalités du jeu.

Annexe : diagramme de classes complet

Ce diagramme illustre la structure globale de *BedBihan*. Toutefois, pour une meilleure lisibilité des détails de ce diagramme, la consultation du fichier PNG joint avec ce rapport est vivement recommandée.



INSA Rennes
20 Avenue des Buttes de Coësmes
CS 70839
35708 Rennes Cedex 7

Tél. +33 (0) 2 23 23 82 00
Fax +33 (0) 2 23 23 83 96

www.insa-rennes.fr

INSA

UNIVERSITÉ EUROPÉENNE DE BRETAGNE

Commission des Titres d'Ingénieur

MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE