INSA RENNES

Projet Analyse, Conception et P.O.O.
Quatrième année - Informatique

Éric Anquetil, Département Informatique
eric.anquetil@insa-rennes.fr
Arnaud Blouin, Département Informatique
arnaud.blouin@insa-rennes.fr
Alexandru Costan, Département Informatique
alexandru.costan@insa-rennes.fr
Aurélien Le Gentil, Département Informatique
aurelien.le_gentil@inria.fr
Maud Marchal, Département Informatique
maud.marchal@insa-rennes.fr

Institut National des Sciences Appliquées
Ecole publique d'ingénieurs

Table des matières

Jeu Civilisation

1 Principes et But du Jeu

Il s'agit d'un jeu tour-par-tour où chaque joueur dirige une civilisation. Le but du jeu est de détruire les autres civilisations. Pour ce faire, chaque joueur doit développer sa civilisation en créant des villes qui, à leur tour, peuvent créer des unités militaires qui vont pouvoir aller conquérir les villes des autres civilisations. Le jeu se déroule sur une carte du monde sur laquelle des villes sont créées et des unités se déplacent.

1.1 Règles du Jeu

Les civilisations

Il existe deux civilisations [1] [2] : INFO et EII. Les civilisations ont les mêmes unités mais possédant des caractéristiques différentes :

1. Directeur de département (coût : 200 de minerai ; caractéristiques : 0 attaque, 2 défense, 3 mouvement, 5 vie) : il s'agit d'une unité unique pour chaque civilisation (INFO : Directeur Leplumey et EII : Directeur Bedat). On ne peut en avoir et en créer qu'un à la fois. S'il meurt, il est tout de même possible de le recréer dans une ville. De par sa vaillance et son abnégation, le directeur augmente de 50% les caractéristiques de défense et d'attaque des unités combattantes se trouvant sur la même case (ce bonus n'affecte pas les caractéristiques des directeurs).
2. Étudiant (coût : 100 de minerai) : il s'agit d'une unité combattante au corps à corps (il faut se trouver sur une case à côté de l'unité à attaquer). Ses caractéristiques différent en fonction de la civilisation. INFO : 4 att, 2 déf, 2 mvt, 10 vie ; EII : 3 att, 3 déf, 2 mvt, 10 vie.
3. Enseignant (coût : 60 de minerai) : les enseignants dispensent le savoir-faire et le savoir-être dans le monde en créant des villes, développant ainsi la civilisation. Leur création requière une quantité non négligeable de nourriture. De ce fait, la croissance de la ville est stoppée lors de sa production. INFO : 0 att, 1 déf, 3 mvt, 1 vie ; EII : 0 att, 2 déf, 2 mvt, 1 vie.

À chaque tour, toutes unités peuvent être déplacées et réalisées une action (déclencher une attaque, créer une ville, *etc.*).

La Carte du Monde

La carte du monde se compose de cases carrées. La largeur d'une case est de 50 pixels. Il existe différents types de case fournissant différentes ressources :

1. Ce jeu est une œuvre de pure fiction. Par conséquent toute ressemblance avec des situations réelles ou avec des personnes existantes ou ayant existé ne saurait être que pure coïncidence.
2. Vous pourrez étendre le jeu à six civilisations (INFO, EII, SRC, GCU, GMA et SGM) si vous en avez le temps.

1. Montagne. Fournit 3 ressources en minerai.
2. Plaine. Fournit 3 ressources en nourriture et 1 en minerai.
3. Désert. Fournit 2 ressources en minerai.

Chaque case peut posséder une et unique ressource spéciale :
1. Fer : fournit 2 ressources en minerai supplémentaire.
2. Fruit : fournit +2 nourriture.

Indice de modélisation : utilisez *poids-mouche* pour minimiser le nombre d'instances de cases. Utilisez également *décorateur* pour modéliser les cases de ressources (*cf.* exemple du cours).

Il existe deux types de cartes :
1. Petite : 2 joueurs. 25 cases × 25 cases.
2. Normale : entre 2 et 4 joueurs. 100 cases × 100 cases.

La Ville

Une ville est créée par un enseignant sur une case donnée. La création de la ville détruit l'unité enseignant. Une ville occupe une case de la carte. Elle peut utiliser les ressources se situant dans un périmètre de 3 cases. À sa construction, une ville utilise uniquement les ressources de la case où elle se trouve.
La population. Une ville possède des habitants. Le nombre d'habitant à sa création est de 1. Une ville peut voir sa population grandir. Pour cela, à chaque tour elle accumule les ressources en nourritures des cases qu'elle utilise (et qui ne sont pas dépensées à la création d'un enseignant). À la création d'une ville, il faut accumuler 10 ressources en nourriture pour passer à 2 habitants. Ensuite, il faut respecter la formule suivante : $nbRes_n = nbRes_{n-1} + nbRes_{n-1}/2$. Il faut donc 15 ressources en nourriture pour passer de 2 à 3 habitants. Chaque augmentation d'un habitant permet à la ville d'utiliser les ressources d'une nouvelle case de son périmètre (tant qu'il en reste). Le choix de cette case, sera réalisé par un algorithme que vous développerez.
La production. À chaque tour, une ville peut dépenser les ressources en nourriture et minerai qu'elle produit grâce aux cases qu'elle utilise. Ces ressources sont d'abord dépensées pour la production de l'unité en cours. Tout excédant sera toujours perdu.

Les Combats

Seules les unités dotées d'au moins 1 en attaque peuvent lancer une attaque contre une unité d'une autre civilisation. Pour cela, les deux unités doivent se situées sur des cases juxtaposées. Lorsqu'une unité attaque une case contenant plusieurs unités, la meilleure unité défensive est choisie. Une unité attaquée possédant 0 en défense meurt immédiatement. Sinon, un certain nombre de combats a lieu. Ce nombre est choisi aléatoirement à l'engagement (entre 3 et le nombre de points de vie de l'unité ayant le plus de points de vie + 2 points). Le combat s'arrête lorsque ce nombre est atteint ou lorsque l'une ou autre des unités n'a plus de vie. Chaque combat calcul les probabilités de perte d'une vie de l'attaquant. Par exemple, Si l'attaquant a 4 en attaque et l'attaqué a 4 en défense (en tenant compte des bonus de terrain et du nombre de points de vie restant), l'attaquant à 50% de (mal-)chance de perde une vie. S'il a 3 att. contre 4 déf., le rapport de force est de 75% : 3/4 = 25%, 25% de 50% = 12.5%, 50%+12.5%=62.5% chance pour l'attaquant de perdre une vie. Explications du calcul : par défaut 2 unités égales ont 50% de gagner. Puisque dans le cas présent un écart de 25% est constaté entre les deux unités, il est nécessaire de pondérer le 50% par ces 25% ce qui donne 62.5% contre 37.5%. S'il a 4 att. contre 2 déf., le taux baisse à 25% (2/4 = 50%, 50%

de 50% = 25%, 25%+50% = 75% pour l'attaqué, 100%-75%=25% pour l'attaquant). Évidemment, lorsque l'attaquant gagne cela signifie que l'adversaire perd un point de vie.
Les points de vie entrent en compte dans le calcul des probabilités : si une unité attaquante ayant 4 en attaque possède 75% de sa vie, alors son attaque sera au final de 4*75% = 3. L'unité attaquée suit le même calcul pour sa défense.

À la fin d'un combat gagné par l'attaquant et si la case du vaincu ne contient plus d'unité, ce dernier se déplace automatiquement sur cette case. Si cette case est une ville, celle-ci est considérée comme conquise et fait partie intégrante de la civilisation de l'attaquant.
Lorsqu'une civilisation n'a plus de ville, elle est détruite et le joueur a perdu.
Si une unité n'est pas utilisée durant un tour, sa vie remonte automatiquement d'un point.

La Vue

Un joueur ne voit pas la carte, les ressources ainsi que les autres civilisations dans sa totalité. Toute les civilisations voient la cartes, ses ressources et les villes des autres civilisations. Cependant les unités des autres civilisations ne sont visibles que lorsqu'elles sont dans le périmètre d'une unité ou d'une ville du joueur courant. Les villes voient à 3 cases et les unités à 2.
Contrairement au jeu *Civilization* qui propose une vue de la carte en biais, le jeu doit permettre de voir la carte du dessus.

Début de Partie

Au début du jeu, chaque joueur choisi sa civilisation. Chaque civilisation débute la partie avec un enseignant (pour fonder une première ville) et un étudiant sur un endroit de la carte choisi de manière à ce que les civilisations ne soient pas trop proche. L'ordre de jeu est choisie aléatoirement en début de partie. Les joueurs jouent chacun leur tour sur leur même ordinateur.

Tour de jeu

À son tour, chaque joueur peut :

1. déplacer toutes ses unités suivant leur nombre de déplacements (un déplacement sur une case coûte un point de déplacement). Il est possible pour chaque unité de passer son tour (généralement par le biais de la touche *espace*). Un colon peut bâtir une ville s'il lui reste au moins un point de mouvement. De même, une unité combattante peut engager un combat s'il lui reste au moins un point de mouvement.
2. choisir quoi produire dans ses villes si une production est terminée. Il est possible de modifier une construction en cours (l'ancienne production et les ressources utilisées sont alors perdues).

Lorsqu'un joueur a fini son tour, il clique sur le bouton correspondant ("Fin tour"). C'est alors au joueur suivant de commencer son tour.

2 Modélisation (séances 1 et 2)

2.1 Description de l'étape de modélisation

Les thèmes essentiels que vous devez aborder lors de la phase d'analyse et de conception sont les suivants :

1. modélisation des données du jeu (joueur, case, carte, unité, vue, *etc.*) à l'aide de différents diagrammes de classes ;
2. modélisation du comportement du jeu à l'aide de diagrammes d'interaction, d'états-transitions (déroulement des combats, choix de la case de ville lors de l'agrandissement de la population, fonctionnement d'un tour, *etc.*) ;
3. utilisation des patrons de conception suivants (cela est lié aux trois points précédents) :

1. *Fabrique abstraite*, pour gérer les différentes civilisations.
2. *Monteur*, pour la création d'une partie.
3. *Observateur*, pour le lien entre le modèle et la vue.
4. *Poids-mouche* et *décorateur* pour la modélisation de la carte.
5. *Stratégie*, pour la création des différents types de carte.

Il existe également de nombreux autres scénarios que vous pouvez développer (scénarios nominaux ou bien gestion des erreurs notamment).

2.2 Tâches à réaliser

Votre modélisation du jeu à l'aide de diagrammes UML sera fondée sur votre analyse. Votre rapport pourra comporter les parties suivantes (à titre indicatif) :

1. 2 illustrations des fonctionnalités du jeu à l'aide de cas d'utilisation.
2. Les diagrammes de classe représentant :

1. (a) la modélisation globale du jeu (carte, joueurs, jeu, civilisation, *etc.*) ;
   (b) les patrons de conception utilisés.

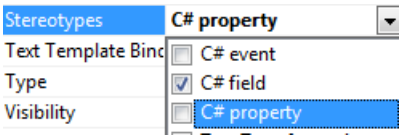
1. 1 diagramme d'états-transitions pour modéliser le fonctionnement des différents objets (par exemple le cycle de vie d'une des unités).
2. 2 diagrammes d'interaction pour vous aider à définir les diagrammes de classes (création d'une partie, déroulement globale d'une partie jusqu'à la victoire d'un joueur, déroulement d'un tour pour 1 joueur, *etc.*).

Tout diagramme supplémentaire, correct et ayant une utilité sera considéré positivement lors de l'évaluation du projet.

2.3 Aide

1. Débuter la modélisation d'un projet est souvent fastidieux et déroutant ("Par où commencer ?", "Que dois-je modéliser ?", *etc.*), Il est généralement recommandé de commencer un diagramme, voire plusieurs en même temps, sur une feuille de papier.
2. Les diagrammes de séquence aident à identifier les opérations des classes de vos diagrammes de classes. Essayez de faire en parallèle ces deux types de diagrammes.
3. Vous pouvez utiliser les classes issues de la librairie .NET comme type d'un attribut, *etc.* Pour cela, il vous suffit d'écrire dans le champ type le nom complet de la classe. Par exemple, pour utiliser la classe *Color*, il faut mettre dans le champ type *System.Drawing.Color*. Le nom complet de chaque classe peut se trouver sur Internet.
4. Dans le cadre de ce projet il vous est conseillé de définir les relations entre les classes avec des attributs. Normalement, il faut utiliser des associations et des compositions mais la génération

de celles-ci en C# posent problème actuellement. Vous devrez également associer le stéréotype "*C# field*" à ces attributs comme l'illustre la figure suivante :



1. Il vous est fortement conseillé d'en créer plusieurs diagrammes de classes. Par exemple, un diagramme pour les interfaces du modèle, un autre pour l'implémentation du modèle, un autre pour la vue, *etc.* Vous pouvez utiliser dans un diagramme de classes les classes définies dans un autre en allant dans l'onglet "*explorateur de modèles UML*" et en glissant-déposant la classe voulue dans le diagramme.
2. N'oublier pas de modéliser les constructeurs des classes ainsi que leurs paramètres. Pour cela, dans les propriétés de l'opération associez le stéréotype "*create*".
3. Si vous voyez pas comment modéliser la vue, il est conseillé de regarder de plus près comment développer un IHM en C# et plus précisément comment définir un élément graphique qui s'ajoute à une IHM. Par exemple, vous pouvez faire hériter vos vue de la classe *Canvas*.
4. Comme présenté en cours dans les exemples WPF, C# gère propose un moyen pour implémenter le patron de conception *observateur*. Pour cela il faut utiliser du côté du modèle l'interface *INotifyPropertyChanged*. L'implémentation de cette interface requière la déclaration d'un attribut d'un type spécial, un *event*. Les *events* permettent à une classe de notifier d'autres classes (*cf.* http ://msdn.microsoft.com/en-us/library/aa645739.aspx). Vous pouvez étudier et vous inspirer du code suivant :

```
public event PropertyChangedEventHandler PropertyChanged;
protected void OnPropertyChanged(string name) {
  PropertyChanged(this, new PropertyChangedEventArgs(name));
}
```

Ainsi, lors d'une modification d'un attribut, il est possible de notifier la vue écoutant le modèle modifié :

```
public void setPosition(Point pos) {
  this.pos = pos;
  OnPropertyChanged("position");
}
```

Mais cela implique que la vue s'abonne au modèle, par exemple :

```
model.PropertyChanged += new PropertyChangedEventHandler(update);
override public void update(object sender, PropertyChangedEventArgs e){...}
```

Dans ce code tiré d'une vue, on abonne la vue au modèle *model* : la méthode *update* sera appelée à chaque appel de *OnPropertyChanged* dans le modèle.

3 Génération du code et implémentation (séances 3 et 4)

3.1 Description de l'étape

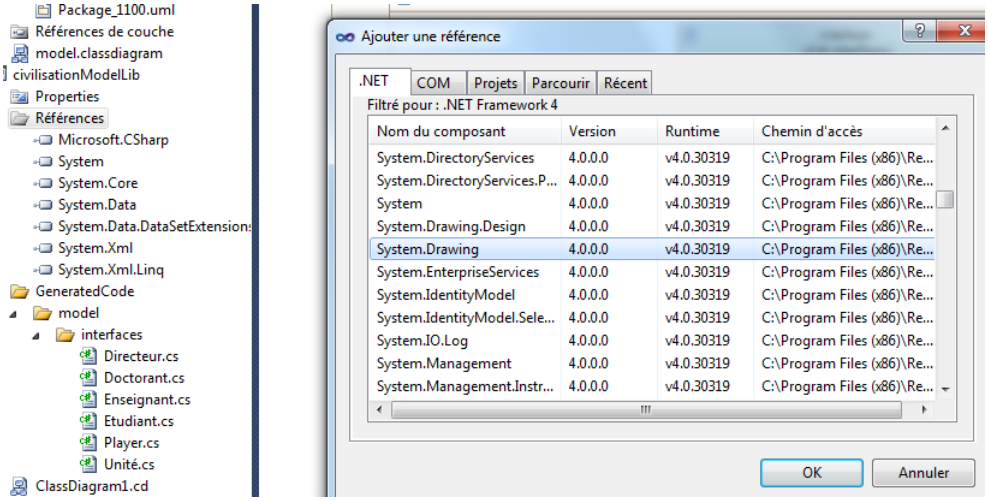
Votre application sera développée en C# et en C++ : C++ pour les algorithmes requérant des calculs (par ex. gestion des combats, création de la carte) ; C# pour le reste dont une partie sera automatiquement générée grâce aux diagrammes de classes.
La génération automatique de code permet de convertir un projet de modélisation en code source C#. Pour généré le code C# d'un diagramme de classes, allez dans l'onglet "*Explorateur de modèles UML*", cliquez-droit sur la racine et générez le code. Un nouveau projet sera créé. Lors de la première génération, une fenêtre vous demandera si vous voulez configurer la génération. Validez, puis dans la nouvelle fenêtre validez de nouveau.
Attention : la régénération du code écrasera toujours l'ancien !

3.2 Tâches à réaliser

Lors de ces séances, vous devrez générer le code à partir des diagrammes de classes puis développer les méthodes générées.

3.3 Aide

1. N'oublier pas de compiler au fur et à mesure du codage pour corriger plus facilement les erreurs.
2. Vous découvrirez très certainement des erreurs de modélisation lors du codage (c'est généralement le moment où l'on se dit que l'on aurait dû faire un diagramme de séquence pour éviter ce genre de problème). Il faudra donc modifier les diagrammes et éventuellement regénérer le code mais attention : Visual Studio n'est pas très malin en la matière et écrasera tout le code que vous aurez écrit des classes regénérées !
3. Sous Visual Studio, utiliser un type issu d'une bibliothèque .NET nécessite l'ajout de celle-ci dans les références du projet généré. Pour cela, allez dans l'explorateur de solutions, cliquez-droit sur le dossier *Références* de votre projet généré, sélectionnez ajout d'une référence et allez dans l'onglet *.NET* comme le montre l'image suivante (*System.Drawing* pour utiliser la classe *Color*, *WindowsBase* pour utiliser la classe *Point*, *etc.*) :



4 Test Logiciel (à partir de la séance 3)

Il est important de tester les différentes parties d'un logiciel *au fur et à mesure du développement.* Dans le cas présent, vous devez tester les librairies C++ et votre code C#. Pour simplifier ce processus de test, vous testerez votre code C++ au travers de tests écrit en C#.
Effectuer des tests sur du code C#. Il faut ajouter un projet de test C# à la solution. Il faut ensuite aller dans le code C# que l'on veut tester et sélectionner une opération ->clic droit ->créer des tests unitaires. Puis choisir le projet de test C# créé.
Le but de chaque opération de test est de vérifier certaines propriétés de votre code en utilisant des assertions :
http ://msdn.microsoft.com/fr-fr/library/ms182532%28v=vs.80%29.aspx
Par exemple voici un test vérifiant que l'ajout d'un forme à un dessin, dans le cadre d'un éditeur de dessins, est correct :

```
[TestMethod]
void TestAjoutFormeDessin() {
  Rectangle* rec = new Rectangle();
  dessin->ajouterForme(rec);
  Assert.AreEqual(0, dessin->getNbFormes());
  Assert.AreEqual(rec, dessin->getFormeAt(0));
}
```

Pour exécuter des tests, allez dans le fichier de tests, cliquez-droit dans l'éditeur de texte, puis "exécuter les tests".

4.1 Aide

1. N'oubliez pas que si les diagrammes de séquences, et compagnies, ne sont pas utilisés pour générés du code, ils sont très utiles pour définir des cas de test. Par exemple, vérifier le bon fonctionnement du changement d'état d'une unité.

5 Développement de la librairie C++ (séance 5)

Différents algorithmes devront être codés en C++. Ils seront utilisés sous la forme d'une librairie dans votre projet. Pour ce faire, vous aurez besoin de développer un *wrapper* faisant le lien entre le C# et le C++. Vous pouvez utiliser l'une ou l'autre des deux méthodes vues en cours.

5.1 Tâches à réaliser

Développer une librairie C++ réalisant les algorithmes suivants :

1. Création de la carte et placement des joueurs. La création d'une carte peu s'avérer extrêmement complexe. C'est pourquoi il vous est fixé un certain de contraintes visant à encadrer le fonctionnement d'un tel algorithme [3] :
   (a) Une carte doit contenir tous les types de terrain et de ressource.
   (b) Les types de terrain doivent être regroupés en blocs d'au moins 4 cases. Il peut avoir plusieurs groupes de blocs d'un même type. Cela permet d'obtenir des chaînes de montagnes, des déserts, *etc.*
   (c) Une carte de dimension $n \times n$ (une carte est forcement carrée), le nombre de ressources doit être de $n/2 = 5$.
   (d) Les joueurs doivent être placés le plus loin des uns des autres.
2. Suggestion des cases où un enseignant pourrait fonder une ville. Lorsqu'un enseignant se trouve sur la carte, il a pour but de fonder une ville au meilleur endroit possible. Cet algorithme doit analyser la carte (ressources, terrains, civilisations ennemies, *etc.*) afin de suggérer 3 emplacement (cases) d'une possible ville.

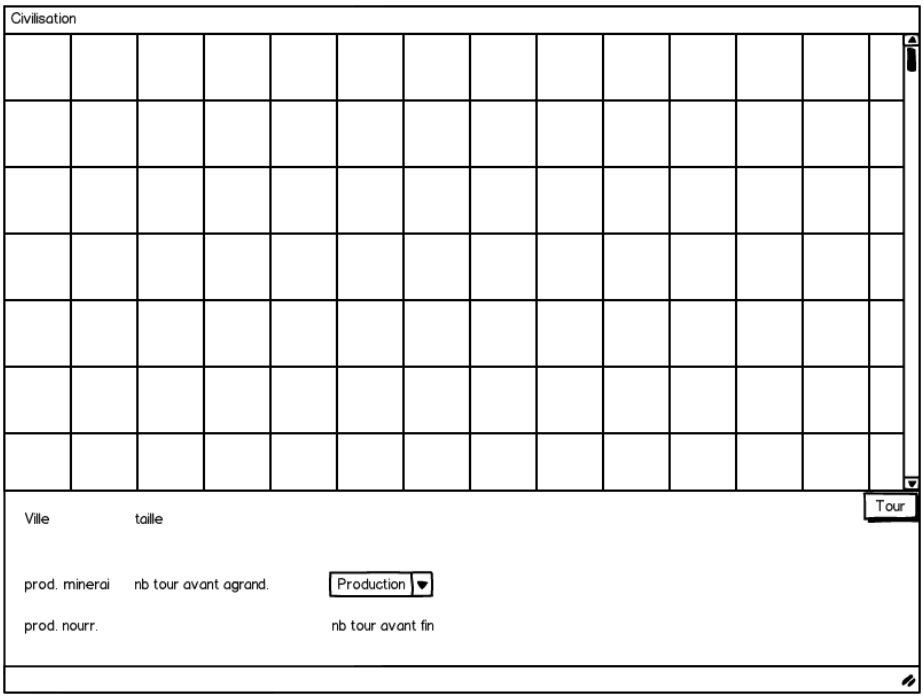
5.2 Aide

1. Si vous avez des problèmes de lien avec votre librairie C++ lors de la compilation en mode "Release" de votre problème, vérifier cette solution : allez dans les propriétés du projet wrapper (mode release) -> éditions de liens -> entrée -> dépendances supplémentaires -> ajoutez le chemin vers le .lib (entre guillemets).

3. Ne perdez pas trop de temps dans cet algorithme qui peut vite devenir chronophage. Commencez par une version basique que vous pourrez améliorer si vous en avez le temps à la fin du projet.

6 Aspects interactif et graphique (séances 6, 7 et 8)

L'interface graphique possède une vue affichant une partie de la carte, un panneau contenant les informations sur la ville ou l'unité sélectionnée et un bouton de fin de tour. Le mockup suivant vous donne une idée d'une possible IU :



6.1 Tâches à réaliser

Essayez de progresser de manière itérative :

1. Affichage de la carte et de ces ressources. La carte étant trop grande pour être vue dans sa totalité, des moyens (par ex. des ascenseurs : *ScrollViewer*) doivent être mis en place pour déplacer la vue. Pour afficher les terrains et les ressource, vous pouvez soit les faire dessiner de manière basique (p. ex. *DrawRectangle*) dans des méthodes *OnRender(DrawingContext dc)*, soit utiliser les textures que l'on vous fournies et les afficher avec *DrawImage*.
2. Affichage des unités et des villes.
3. La sélection d'une ville ou d'une unité s'effectue en cliquant dessus.
4. Création d'un panneau pour : voir et d'éditer la production et les caractéristiques d'une ville ; voir les caractéristiques d'une unité sélectionnée.
5. Une unité peut être déplacée ou attaquer à l'aide du clavier (barre espace pour passer son tour) ou éventuellement de la souris. Nous vous conseillons d'utiliser le pavé numérique pour bouger les unités (*Key.NumPadX*).
6. La fin d'un tour peut s'effectuer en cliquant sur un bouton *fin de tour* ou en appuyant sur la touche *entrée*.
7. Sauvegarder et charger une partie. Cela vous sera fort utile lors de votre démonstration.

6.2 Aide

1. Si vous utiliser les textures fournies, faites attention à utiliser un poids-mouche pour ne les charger qu'une seule fois.

7 Fonctionnalités avancées (facultatif et bonus)

Pour ceux disposant de temps, il est possible de développer des fonctionnalités plus avancées, qui seront considérées comme bonus lors de l'évaluation, dont voici une liste non-exhaustive :

1. Ajout de nouvelles unités ;
2. Ajout de nouvelles civilisations ;
3. Ajout de nouvelles ressources et de nouveaux types de terrain ;
4. Ajout de bonus d'attaque et de défense en fonction du type de terrain ;
5. Jouer en réseau ;
6. Jouer contre l'ordinateur ;
7. Type de terrain supplémentaire : la mer, il faudra alors penser à rajouter de nouvelles unités marines et à revoir l'algorithme de construction de la carte ;
8. Possibilité de construire des bâtiments dans la ville apportant des ressources ou des améliorations diverses (par ex. une caserne, une usine, *etc.*).

8 Consignes générales pour le projet

8.1 Fonctionnalités de base attendues

Les fonctionnalités de base attendues sont toutes décrites dans les sections "Tâches à réaliser" de cet énoncé.

8.2 Rapport de conception

Vous devrez rendre votre rapport de conception **au plus tard le 8 novembre 2012 avant minuit sur la plate-forme moodle (aucun rapport envoyé par mail ne sera pris en compte pour la notation**).

8.3 Soutenance de projet

Vous devrez aussi préparer une soutenance de 10 min. Les soutenances auront lieu le lundi 7 janvier 2012. Cette soutenance devra se diviser entre : une démonstration de votre jeu (prévoyez plusieurs cas d'utilisation) ; une présentation. Il est aussi impératif qu'au plus tard le vendredi 4 janvier 2012 à minuit, vous ayez déposé sur la plate-forme moodle les documents suivants :

1. la documentation utilisateur de votre jeu ;
2. le code source (en entier) ;
3. un exécutable qui fonctionne.

Chaque jour de retard entraînera un point de pénalité sur la note finale.

8.4 Notation

Le barème suivant vous est donné à titre indicatif :

1. Rapport de conception $\approx$ 8 pts ;
2. Documentation utilisateur, code (commentaires, propreté, *etc.*) et fichiers exécutables $\approx$ 6 pts ;
3. Présentation et démonstration $\approx$ 6 pts.