Rapport de projet

- Incidence -

CHAMBONNET Kevin
GAUTHIER Silvère
MARTINEZ Thierry
MOKHRETAR Amin

19 mai 2014

Table des matières

Table des matières **2**

1 Remerciements **3**

2 Introduction **5**
2.1 Philosophie 5
2.2 Sujet 5

3 Cahier des charges **7**
3.1 Introduction 7
3.2 Mecanismes de jeu 7
3.3 Structure du jeu 8
3.4 Eléments graphiques 14

4 Gestion du projet **17**
4.1 Gestion de l'équipe 17
4.2 Découpage en tâches 17
4.3 Assignation 18
4.4 Gestion du temps 18
4.5 Profil de risques 19
4.6 Choix technologiques 19
4.7 Gestion des fichiers 20

5 Développement **23**
5.1 Moteur de jeu 23
5.2 Moteur de carte 24
5.3 Moteur multi-agent 28
5.4 Météo 31
5.5 Architecture 31

6 Post-Mortem **33**
6.1 Fonctionnalités non implémentées 33
6.2 Améliorations réalisables 33

A **35**
A.1 Diagramme de Gantt 35
A.2 Comparatif de performances 35
A.3 Incidences sur la carte 35

Chapitre 1

Remerciements

Un grand merci à Fabien Hervouet pour son encadrement.

Chapitre 2

Introduction

2.1 Philosophie

Du point de vue d'un observateur, un monde peut être qualifié de complexe sans que les individus y évoluant ne le soient forcément. Dans ce TER, il est question d'aborder cette thématique sous forme de jeu. Autrement dit, le problème qui nous intéresse ici est de déterminer comment faire émerger un comportement global en mettant en oeuvre des contraintes extérieures dans l'environnement pour influencer les interactions locales d'agents. De plus il faudra, pour le joueur, analyser le comportement des ces derniers afin d'optimiser la réalisation des buts demandés. Plus précisément, il s'agit de mettre en scène des agents (disposant d'un comportement précis et défini en terme d'interaction), évoluant dans un environnement modifiable par un joueur humain (qu'il s'agisse d'ajouter des ressources exploitables, de modifier le relief, etc.) devant atteindre un certain objectif (stabilité du système écologique, récolte d'une certaine quantité de ressources, développement d'un chemin).

2.2 Sujet

Ce travail impliquera de s'intéresser au paradigme de la programmation multi-agents. À partir d'une plateforme existante (type TurtleKit) ou d'une version simple programmée pour l'occasion, on proposera différents niveaux de jeu qui offriront par exemple : (1) la possibilité de modifier l'environnement permettant, en terme de contraintes, d'insuffler une dynamique globale aux agents pour les mener à réaliser un certain but ; (2) la possibilité de modifier le comportement local des agents pour atteindre des buts plus complexes. Le cadre plus précis sera à déterminer en fonction de faisabilités et des envies des étudiants : quelles actions pourront effectuer les agents ? quelles ressources à disposition ? quels types d'objectifs pour le joueur ? comment envisager la progression des niveaux ? D'autres évolutions sont possibles et seront traitées en fonction de l'avancement du projet : systèmes d'évènements (épidémies, catastrophe naturelle), outils simple pour la création de nouveaux niveaux, mode multi-joueurs, etc. D'un point de vue pragmatique on s'attachera en priorité à étudier l'adéquation de la mise en oeuvre des comportements réactifs des agents face aux différentes possibilités de contraintes permises pour le joueur, ainsi qu'à l'ergonomie de l'interface. L'aspect recherche de ce TER se concentrera

sur une familiarisation plus en profondeur avec le domaine de la modélisation et de la simulation multi-agents [1] ainsi qu'avec les méthodologies de conception de jeu [2].

Chapitre 3

Cahier des charges

Ce chapitre détaille toute la phase de conception du projet. Elle contient le Game Design Document que nous avons rédigé lors de l'étude préalable.

3.1 Introduction

Questions fréquentes

Qu'est-ce que ce jeu ?

C'est un jeu de type God-Like dans lequel on doit aider une civilisation à survivre le plus longtemps possible grâce à différents pouvoirs et actions divines.

Qu'est-ce que je contrôle ?

Vous ne contrôlez rien directement, tout se fait de façon indirecte grâce aux pouvoirs. Modifier l'environnement et aider les citoyens sont des actions qu'il faudra effectuer pour faire survivre la civilisation.

Quelles sont les conditions de victoire et de défaite ?

Il n'y aura pas de condition de victoire particulière dans le jeu de base, le but sera de survivre le plus longtemps possible.
La partie se terminera quand le joueur n'aura plus de citoyen.

3.2 Mecanismes de jeu

Cette section détaille succintement les possibilités du joueur pendant la partie.

Les Pouvoirs

Le joueur aura la possibilité d'interagir sur l'environnement via differents pouvoirs.

- **Placer un Arbre :** Fait apparaître un Arbre sur une case choisie. Coût : 3 PI.

- **Placer un Arbre Fruitier :** Fait apparaître un Arbre Fruitier sur une case choisie. Coût : 6 PI.
- **Placer de la Pierre :** Fait apparaître de la Pierre sur une case choisie. Coût : 5 PI.
- **Placer de l'Eau :** Fait apparaître de l'Eau sur une case choisie. Coût : 2 PI.
- **Placer une Falaise :** Fait apparaître une Falaise sur une case choisie. Coût : 7 PI.
- **Placer un Buisson :** Fait apparaître un Buisson sur une case choisie. Coût : 4 PI.
- **Placer de la Terre :** Fait apparaître de la Terre sur une case choisie, la Terre placée s'adapte en fonction des autres Terres autour. Coût : 2 PI.
- **Soigner :** Améliore l'état de santé d'un citoyen. Coût : 50 PI.
- **Naissance :** Crée un citoyen supplémentaire, hors naissances quotidiennes. Coût : 200 PI.

Choix de la nuit

Lors de la partie, il y aura plusieurs phases de jeu : le jour et la nuit. Chaque jour, le joueur peut effectuer les actions ci-dessus, et lors de la nuit, il a la possibilité de définir des paramètres pour le jour suivant :

- **Météo :** Le joueur peut choisir la météo du jour suivant (ensoleillée ou pluvieuse).
- **Métier :** Le joueur peut orienter la distribution des métiers, sans définir exactement la proportion de chaque métier.

Sauver et Charger une partie

Le joueur pourra sauvegarder sa partie à tout moment, mais la sauvegarde se fera au moment de la dernière nuit.
Le chargement d'une partie placera donc le joueur soit au début de la partie soit à la dernière nuit passée, avec la possibilité de modifier de nouveau les différentes options du jour (météo, métiers...).

3.3 Structure du jeu

Cette partie détaille succintement la structure globale du jeu, ce qui couvre les aspects du jeu non maîtrisables par le joueur.

Moteur multi-agent

La gestion multi-agent se fera à l'aide d'un appel au script de chaque catégorie d'entité pour chaque tick. Chaque catégorie d'entité aura donc un script propre, qui décrira son comportement et pourra faire appel aux primitives que fournira le moteur pour decider de l'action à faire.

Primitives

- Connaître les entités et cases alentours
- Connaître l'emplacement du village
- Changer de direction
- Connaitre sa santé
- Si leur sac est plein
- S'il est attaqué
- Par qui sont-ils attaqué

Actions

- Avancer
- Couper un Arbre
- Ramasser des Baies
- Casser de la Pierre
- Rentrer au village
- Attaquer une entité

Les Entites

Les métiers des citoyens

Chaque citoyen aura une tâche à accomplir durant la journée et ne pourra pas en changer avant la nuit. La nuit, un métier sera attribué à chaque citoyen selon les choix du joueur et les besoins des citoyens (cf. section 3.3, page 13).

- **Bûcheron :** Coupe les arbres et récolte les ressources associées (Le Bois en général mais aussi de la nourriture sur les Arbres Fruitier).
- **Mineur :** Casse les rochers et récolte la Pierre.
- **Chasseur-Cueilleur :** Chasse les animaux, cueille les baies ou cultive des champs pour récolter la Nourriture.

La santé des entités vivantes

Chacune des entités possède une gestion de la santé avec plusieurs états.

- **Bonne santé :** Si l'entité est un citoyen, elle offre de plus grands bonus de PI.
- **Normal :** L'entité est dans son état par défaut.
- **Blessé/Malade :** L'entité agit avec un léger malus de vitesse. Si l'entité est un citoyen, elle offre de plus petits bonus de PI.
- **Gravement blessé/malade :** L'entité agit avec un malus de vitesse plus important. Si l'entité est un citoyen, elle n'offre plus de PI.
- **Mort :** L'entité disparaît.

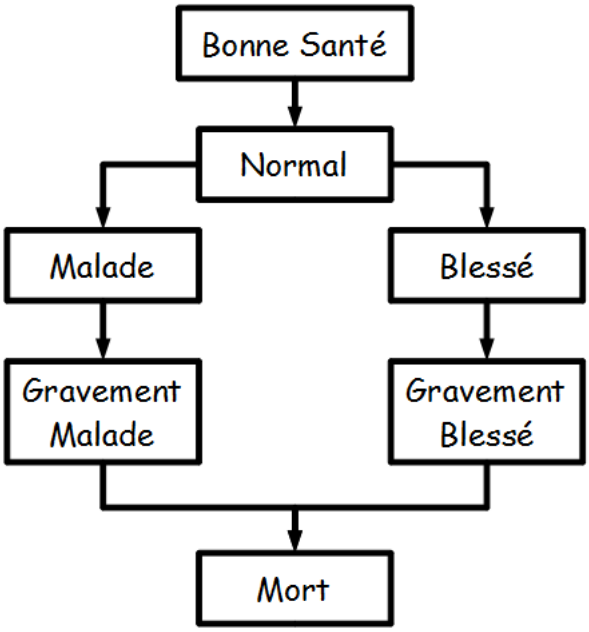


FIGURE 3.1: Diagramme de transitions entre les états de santé

Moteur de carte

La gestion de la carte se fera uniquement en c++. Il existera pour cela différentes classes comme TileMap, Ground, Element et Tileset.

La carte

La carte sera composée de 150x150 cases, mais ne sera affiché à l'écran qu'une vingtaine de cases en largeur sur une quinzaine en hauteur. Le déplacement sur la carte pourra se faire grâce aux touches fléchées mais aussi en plaçant la souris sur un des bords de l'écran.
Le joueur ne pourra pas voir au delà des limites des 150x150 cases composant la carte.

Les Cases

La carte sera découpée en cases. Chaque case aura un type de base en début de partie, qui pourra ensuite être modifié selon le déroulement du jeu (cf. figure 3.3, page 12). Un élément neutre est un type de case ne donnant pas lieu à une ressource quelconque.

Type de case	**Ressources**	**Franchissable**	**Description**
Terre	-	Oui	Type par défaut.
Terre Aride	-	Oui	Terre ne pouvant pas être cultivée.
Terre Innondée	-	Oui	Terre ne pouvant pas être cultivée.
Terre Fertile	-	Oui	Terre pouvant etre cultivée pour devenir un **Champs**.
Champs	Nourriture (1 à 3 unités)	Oui	Terre cultivée possédant plusieurs stades de maturité. Le maximum atteint, la récolte peut être effectuée et offrir de la nourriture.
Arbre	Bois (3 unités)	Non	Peut être coupé pour récolter du bois.
Arbre Fruitier	Bois, Nourriture (2 unités de chaque)	Non	Peut être coupé pour récolter du bois et de la nourriture.
Eau	-	Non	Des poissons peuvent s'y trouver permettant de récolter de la nourriture.
Pierre	Pierre (2 unités)	Non	Peut être cassée pour récolter de la pierre.
Falaise	-	Non	Elément neutre. Peut faire apparaître de la pierre à ses pieds.
Buisson	Nourriture (2 unités)	Non	La récolte de ses baies permet d'obtenir de la nourriture.

TABLE 3.1: Les différents types de case

Diagramme de transitions des différentes cases

Chaque type de case est compatible avec d'autres types. Les transitions entre les types sont détaillées dans ce diagramme :

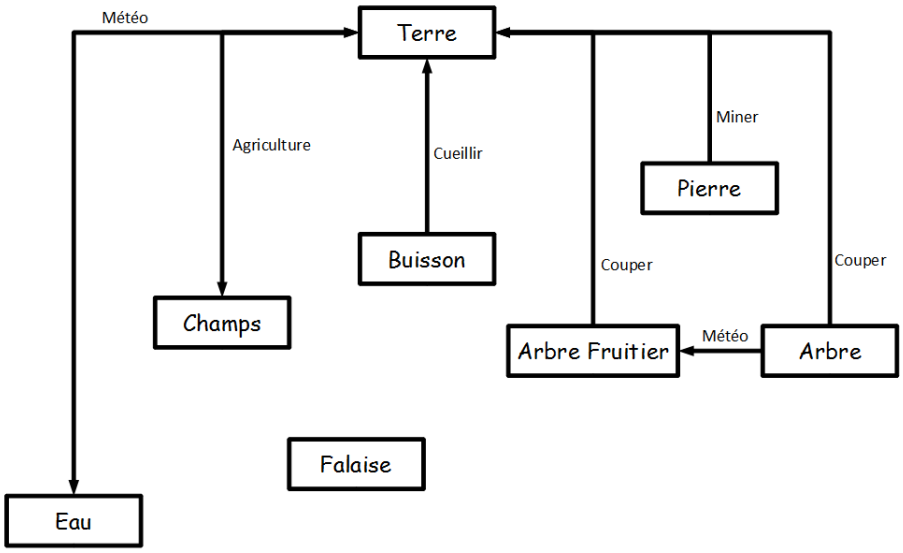


FIGURE 3.2: Diagramme de transitions entre les types de case

Les incidences sur l'environnement :

Lors de la nuit, les actions de la journée ont ce que l'on appelle des incidences sur les journées suivantes, c'est-à-dire des conséquences soit bénéfiques, soit néfastes. Par exemple :

- Une étendue d'eau peut faire apparaître des poissons.
- Une étendue d'eau peut faire apparaître de la végétation dans les environs.
- Une zone de végétation très dense augmente les chances de faire apparaître des animaux herbivores.
- Une grande concentration d'animaux herbivores peut faire apparaître des animaux carnivores.
- Une forêt très dense augmente les chances de faire apparaître des arbres fruitiers.
- Les falaises peuvent faire apparaître des pierres par éboulement.
- La météo peut modifier la taille des étendues d'eau, assécher ou humidifier la terre.

Les Ressources

Différentes ressources telles que le bois, la pierre ou la nourriture seront disponibles sur la carte, et d'autres comme les "points d'incidence", seront générées au cours de la partie.

Ressources utilisées par les citoyens

Ces ressources pourront être stockées dans une quantité illimitée, et les citoyens les utiliseront pour des constructions ou pour se nourrir.

- **Bois :** Utilisé pour la construction des bâtiments.
- **Pierre :** Utilisé pour la construction de certains bâtiments.
- **Nourriture :** Consommé par les citoyens chaque nuit pour se nourrir.

Comment ces ressources sont-elles récoltées ?
Elles sont récoltées par les citoyens, ainsi chaque métier correspond à la récolte d'une de ces ressources (cf. section 3.3, page 9).

Ressources utilisées par le joueur

Cette ressource est la seule que le joueur pourra utiliser, elle sera stockée dans une quantité illimitée.

- **Point d'Incidence (PI) :** Utilisé à chaque action ou pouvoir divin.

Comment cette ressource est-elle récoltée ?
Elle sera obtenue grâce aux citoyens qui nous en feront gagner une petite quantité tout au long de leur journée. La nuit, chaque citoyen rapporte des points bonus.

La météorologie

La météo sera présente et sera contrôlée par le joueur. Elle aura une incidence sur l'environnement et les citoyens. Elle sera basique : ensoleillée ou pluvieuse, chacune des deux aura une incidence différente.

- **Temps ensoleillé :** Améliore la pousse des champs mais un excès de soleil assèche les terres et récoltes, peut réduire les étendues d'eau et une sécheresse trop longue peut faire brûler certaines ressources.
- **Temps pluvieux :** Permet de faire pousser les champs. Un surplus de pluie innonde les terres et récoltes, augmente les probabilités de maladie et peut augmenter la taille des étendues d'eau.

Le cycle jour/nuit

Un cycle jour/nuit sera présent, avec des journées longues et des nuits courtes. Le Jour, les citoyens se vouent à leur métier, jusqu'au soir. La Nuit, tous les citoyens retournent au village, plus aucune action n'est possible. Lorsque la nuit tombe, toutes les actions du jour ont une incidence sur la nature et les entités, et seront visibles au début de la nouvelle journée.

- Le terrain est mis à jour, toutes les actions de la journée auront une incidence sur l'environnement.
- Tous les citoyens se nourrissent, la nourriture diminue en fonction du nombre de citoyen *(-3 de nourriture par citoyen)*. S'ils manquent de la nourriture, certains citoyens peuvent tomber malade.
- Certains citoyens tombent malade en fonction des anciennes météos.
- S'il y a assez de nourriture, de nouveaux citoyens peuvent naître.

- Gain des points bonus d'incidence en fonction de la taille de la population et de sa santé.
- Mise à jour du métier de chaque citoyen, choisi en fonction des choix du joueur et des besoins.

Scripts

Chaque catégorie d'entités déterminée selon leur classe d'appartenance aura un script propre. Ce script sera appelé à une fréquence variable en fonction de la situation dans laquelle se trouvera l'entité.

3.4 Eléments graphiques

Tous les effets graphiques du projet sont à base de sprites. Nous avons initialement choisi des tuiles de 32x32 pixels, mais il serait possible de changer cette taille dans le futur.

Les tuiles

Les tuiles seront des images de 32x32 ou 16x16 pixels au format PNG.

- **Le sol :** Il y aura 16 tuiles pour chaque type de sol, correspondant à toutes les possibilités de jonction avec le type voisin. En effet, chaque type de sol ne pourra être en contact qu'avec deux types différents suivant l'ordre hiérarchique suivant :
  Eau → Terre innondée → Terre fertile → Terre → Terre aride
- **Les ressources :** Il y aura 4 tuiles pour chaque ressource, correspondant au type de sol sur lequel elle se trouve.
- **Les bâtiments :** Chaque bâtiment sera constitué d'un ensemble de tuiles.

Chaque sol du tileset est affiché avec des bordures selon les types de sols environnant.

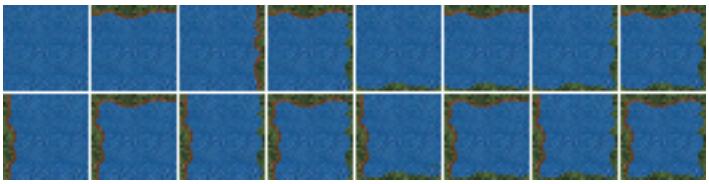


FIGURE 3.3: Exemple de sol avec différentes bordures

Chaque élément est affiché différemment selon le type de sol sur lequel il se trouve.

FIGURE 3.4: Exemple d'élément selon différents types de sol

Les entités

Chaque entité aura plusieurs positions possibles. Pour chacun d'elle, un ou plusieurs éléments graphiques seront créés.
Les positions basiques : Face, dos, profil droit, profil gauche.
Les positions spécifiques : Coupant du bois, ramassant des baies, chassant, cultivant, construisant, se déplaçant sans ressource, se déplaçant avec des ressources.
Chaque entité est animée selon sa classe, son sens de déplacement ou son action.

FIGURE 3.5: Exemple de fichier image pour une animation

Interface utilisateur

Nous aurons différents menus, tels que menu principal, menu de jeu ou menu de la nuit. Ils auront chacun un certain nombre de boutons que nous ne détaillerons pas pour l'instant. Il y aura également un bandeau d'affichage des quantités de ressources pendant le jeu ainsi que des boutons permettant au joueur d'effectuer les actions utilisateur.

Chapitre 4

Gestion du projet

Cette partie traite globalement de tout ce qui concerne l'organisation du projet, que ce soit au niveau de la conception, du développement, de l'équipe ou encore de la gestion des fichiers.

4.1 Gestion de l'équipe

Tous les membres se connaissant et étant supposés être capable de travailler en équipe, nous n'avons fait aucune élection de chef de projet.
Nous avons opté pour travailler de manière collégiale, et ainsi garder une cohésion de groupe sans pour autant avoir de hiérarchie instaurée au sein du groupe, qui pourrait au contraire déservir la réalisation de nos objectifs.
Chaque membre a donc autant de pouvoir que les autres, et peut donc participer activement au projet, autant lors de la conception que du développement. Toutes les décisions seront prises suivant la majorité lors de votes.

Pour ce qui est des réunions de projets, nous avons convenu avec notre tuteur d'une réunion, allant d'environ trente minutes à une heure, toutes les une à deux semaines, afin de mettre au point l'avancement du projet. En parallèle, tous les membres de notre équipe se retrouvent une fois par semaine afin de discuter des points clés effectués ou à venir, donner lieu aux votes pour les prises de décisions, ou encore, lors de la phase de développement, travailler en collaboration afin d'optimiser notre travail.

Au niveau du travail collaboratif, nous avons mis en place un dépôt sur github, contenant tant la documentation telle que ce rapport que les sources de notre jeu. Par ailleurs, nous mettrons sur ce dépôt uniquement les fichiers sources, les images et les sons, mais en aucun cas les fichiers temporaires ou les exécutables. Un fichier "makefile" sera disponible pour quiconque voudrait compiler le programme chez lui. Les seuls fichiers binaires disponibles seront les PDF de la documentation, pour un soucis de facilité d'accès.

4.2 Découpage en tâches

Afin de préparer le développement du jeu, il était nécessaire de séparer les fonctionnalités les unes des autres. Nous avons abouti à ce diagramme, qui

résume notre choix de découpage :

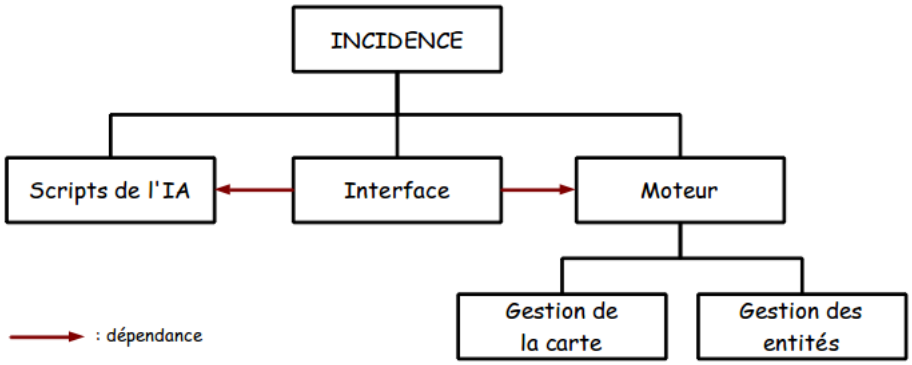


FIGURE 4.1: Diagramme des tâches du projet

4.3 Assignation

Le projet étant maintenant découpé en un certain nombre de modules, il ne restait plus qu'à assigner chaque tâche à un ou plusieurs membres de l'équipe. Nous nous sommes organisés comme ceci :

- **Scripts de l'IA :** MARTINEZ Thierry, MOKHRETAR Amin.
- **Moteur :**
  - **Gestion de la carte :** GAUTHIER Silvère.
  - **Gestion des entités :** CHAMBONNET Kevin.
- **Interface :** Tous les membres.

Bien entendu, cette répartition n'est pas totalement fixée, elle concerne en réalité l'affectation de responsables de parties, qui seront en charge de celle-ci mais pourront évidemment faire appel aux autres membres pour trouver une solution à un problème par exemple.
Le détail complet des tâches et assignations se situe dans la section Gestion du temps, page 18.

4.4 Gestion du temps

Afin de clarifier notre gestion du temps, un diagramme de Gantt est disponible en annexe et dans la documentation de notre projet, et sera mis à jour en fonction de l'avancée du projet.

4.5 Profil de risques

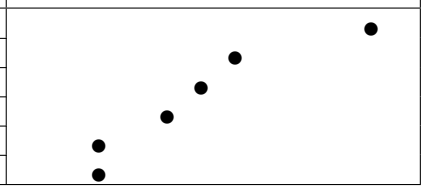
Nature du risque	Degré du risque pour le projet 0 1 2 3 4 5
Taille du projet	
Difficulté technique	
Degré d'intégration	
Configuration organisationnelle	
Changement	
Instabilité de l'équipe de projet	

TABLE 4.1: Profil de risque du projet Incidence

4.6 Choix technologiques

Afin de pouvoir développer correctement notre logiciel, il a fallu définir tout ce que nous allions utiliser en terme de langages et bibliothèques selon notre logique de conception.

Langages de programmation

Pour des besoins de performances, nous avons comparé différents langages. Pour réduire le temps de recherche et de comparaison, nous nous sommes appuyé sur des tests déjà effectués par d'autre.
Des tests de performances concernant un large panel de langages, comparés dans quatre contextes différents, sont fournis en annexe.
Nous pouvons observer que globalement, le langage le plus rapide est ici C++. L'utilisation de ce langage étant très fréquente dans les jeux vidéos, de part sa réputation d'un des langages les plus performants, et tous les membres de notre équipe sachant l'utiliser, nous avons fait le choix de programmer le moteur du jeu en C++.
Afin d'optimiser encore la rapidité du moteur, nous avons cherché à associer son coeur écrit en C++ avec un langage de scripting qui permettra de mettre en place les différentes actions du jeu.
D'après les graphiques ci-dessus, nous avons opté pour le langage LUA, performant et facile d'utilisation (syntaxe proche du C++). En effet, même si Python est très prisé et offre beaucoup plus de possibilités que LUA, nous l'avons estimé bien trop lourd pour l'utilisation que nous allons en faire.
Les deux langages C++ et LUA sont souvent associés dans les jeux vidéos, notre choix suit donc la tendance, ce qui nous offre une certaine assurance.

Bibliothèques

Pour la gestion graphique des tuiles composant la carte et des différentes entités, nous avons cherché une bibliothèque relativement simple d'utilisation mais surtout performante afin de garder la fluidité gagnée avec le choix des langages de programmation.
Connaissant la bibliothèque OpenGL, qui est bas niveau et performante dans les affichages deux et trois dimensions, nous nous sommes tournés vers deux bibliothèques utilisant OpenGL : SDL et SFML.

D'après plusieurs sites web et forums, les dernières versions (respectivement 2.0 et 2.1) de ces deux bibliothèques se valent en terme de performance.
En confrontant nos préférences personnelles quant au choix de l'une ou l'autre, nous nous sommes finalement mis d'accord pour utiliser la bibliothèque graphique SFML 2.1, qui paraît plus simple d'utilisation que la SDL. De plus, elle permet une gestion aisée des fichiers audio, ce qui sera un plus pour la finalité de notre jeu.

4.7 Gestion des fichiers

Nous avons beaucoup de fichiers à gérer dans ce projet, et nous devions établir des conventions ou des moyens afin de les gérer correctement.

Format des Fichiers

Le moteur de jeu étant écrit en C++, nous utiliserons des fichiers d'en-tête au format HPP et des fichiers de définition au format CPP. Pour les scripts d'IA des entités, le langage étant Lua, ceux-ci seront au format LUA.
Toutes les images nécessaires au jeu seront au format PNG afin de pouvoir utiliser la transparence et garder la pleine qualité d'image (contrairement à JPEG qui perd de l'information à la compression).
Les sons quant à eux seront au format WAV afin d'éviter toute gestion de la compression de fichier pour de petits fichiers qui n'en ont aucunement besoin.

Animation

```
Chemin/vers/image.png size_x_sprite size_y_sprite play?(1/0) loop?(1/0)
+Frame position_x position_y temps_affichage
+Frame position_x position_y temps_affichage
+Frame position_x position_y temps_affichage
```

Tileset

Le tileset sera une unique image au format PNG représentant l'ensemble des tuiles possibles. Chaque type de sol constituera une ligne du tileset, avec toutes les variantes dépendant des rebords entre tuiles (ce fichier sera présenté plus en détails dans la section 5.2, page 25).

Carte

Les différentes cartes sauvegardées seront enregistrées dans des fichiers IMS (Incidence Map Save), dans lesquels seront stockés le tileset utilisé et la conformation de la carte (dimensions, sols et éléments).

Sauvegarde

Un fichier sera créé pour la savegarde du jeu et un autre pour la sauvegarde de la carte, cela permettant dans le futur de créer des cartes réutilisables, comme par exemple dans un éditeur de carte.

Le fichier de carte contiendra uniquement le lien vers le tileset ainsi que les types de sols et éléments présents sur la carte. Le fichier sera donc de la taille : 2 * nombre_de_cases * sizeof(int) + sizeof(chemin_du_tileset).

Commentaires

Si une méthode ou fonction, voir même un bloc, dépasse une certaine taille (environ 10 lignes) ou devient trop compliquée, un commentaire sera ajouté avant celle-ci expliquant brièvement son processus :

```
/** Description :
*** Entrée : ...
*** Sortie : ...
**/
```

Marqueur spécifique	Signification
TODO	A mettre à la place du code d'une fonctionnalité à implémenter
RECODE	A mettre au dessus du bloc d'une fonctionnalité à refaire
FIXME	A mettre au dessus du bloc d'une fonctionnalité contenant un bug

Table 4.2: Forme et usage des commentaires

Convention de Nommage

Les conventions de nommages sont utiles pour la lecture du code par les autres membres du groupe, car elles permettent un cadre de travail respecté de tous.

Type de variable	Format du nom
Classe	Majuscule suivit de minuscules
Méthode et Fonction	Minuscules (pour les mots composés, chaque mots suivant est une majuscule suivit de minuscules)
Attribut de classe	m_
Variable globale	g_

Table 4.3: Convention de nommage

Gestion du code source

Afin de faciliter le travail collaboratif, nous utilisons un dépôt utilisant le gestionnaire de version GIT, hébergé sur le site :

`https://github.com/Incidence/Incidence`

Sur ce dépôt seront présents tous les fichiers sources nécessaires au développement du jeu ainsi que les documentations au format Latex et PDF (même si aucun fichier binaire ne devrait être présent, il est plus pratique de récupérer directement un tel fichier que de le compiler soit-même). De plus, y seront stockées toutes les données utilisées par le jeu telles que les images et les sons. Seuls les fichiers temporaires, exécutables et fichiers de sauvegarde ne seront pas stockés.

Chapitre 5

Développement

Ce chapitre est une sorte de carnet de bord. Il détaille tout ce qui concerne le développement de l'application. Au vu de la complexité de notre projet, nous détaillerons relativement peu les aspects techniques.

5.1 Moteur de jeu

Pour simplifier le développement et les réalisations de certaines taches, un moteur de jeu a été réalisé. Il servira surtout dans la gestion des différents états du jeu, de toutes les données utilisées (image ou son) mais aussi d'une interface utilisateur.

Gestion des états

Chacun des états possible du jeu est une classe héritant d'un class mère abstraite, State, qui défini le comportement de base d'un état, soit une méthode de mise à jour de l'état, une affichage des informations et deux dernieres gerant les evenements clavier/souris et les event lié à l'interface utilisateur (clique sur un bouton, ...).

Tout ces différents états sont stockés dans une pile et controlé par un gestionnaire d'état, le StateManager. C'est un singleton accessible partout, ce qui permet à n'importe quelle instance d'état d'ajouter ou supprimer un état.

Gestion de l'interface utilisateur

Différent widget permettant la création d'un interface utilisateur simple à été réalisé. On peut donc facilement ajouter des boutons et des barres de saisie de texte personnalisable qui seront lié à une intance d'UI contenu dans un état. Les événement associé aux boutons sera automatique envoyé à l'etat qui pourra traiter cet evenement pour réaliser une action.

Gestion des ressources et animations

Pour la gestion de toutes les ressources, tels que les nombreuses images, on a réalisé un gestionnaire de ressource qui s'occupe de charger toutes les données necessaires quand elles sont requises tout en évitant les doublons. Pour cela,

chaque ressource est stocké dans un tableau associé à son nom, ce qui permet une recherche plus rapide et une impossibilité d'avoir des doublons.

Une classe permettant de créer des animations facilement a aussi été faite, ce qui facilite grandement la gestion de celle-ci et leurs utilisations.

5.2 Moteur de carte

Le moteur de carte doit gérer tous les aspects techniques et graphiques de la carte. Il englobe les tuiles, les sols, les éléments et la carte en elle-même.

Les Tuiles

Les différents types de tuile seront tous recensés dans le tileset, qui se chargera de fournir les textures et attributs associés à ceux-ci. La carte n'utilisera que des pointeurs vers ces instances constantes de tuile, ce qui évitera le stockage inutile d'instances identiques.

Les Sols

Chaque sol aura différents attributs définis dans le fichier de configuration du tileset (cf section 5.2, page 25). Comme définis dans le cachier des charges, certains sols seront incompatibles avec d'autres.

Attribut	**Description**
Type	Entier définissant un identificateur du type de sol.
Coût	Entier définissant le coût en PI de la pose du sol.
Nom	Chaîne de caractères utilisées lors des affichages dans l'interface.
Comportement	Enumération de trois comportements : DEFAULT : comportement par défaut (tous les type de Terre). FLUID : les sols forment des zones (étendues d'eau). CLIFF : les sols sont fixes et forment des zones (falaises).
Franchissable	Booléen indiquant si le sol est franchissable ou non par les unités.
Rebords	Tableaux d'entiers indiquant tous les types de sol compatibles avec les bordures de l'image (cf figure 5.2, page 25).

Table 5.1: Attributs des sols

Les Eléments

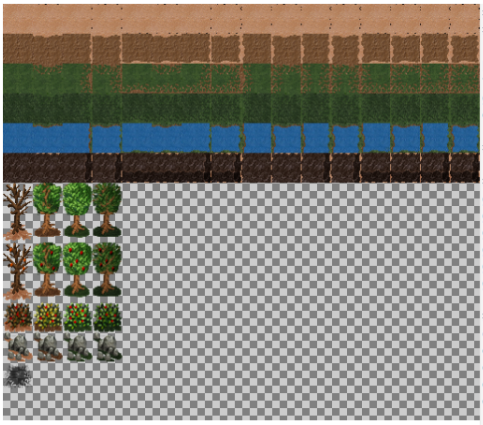
Chaque sol aura différents attributs définis dans le fichier de configuration du tileset (cf section 5.2, page 25).

Attribut	Description
Type	Entier définissant un identificateur du type d'élément.
Type de sol	Entier définissant un identificateur du type de sol sur lequel se place l'élément.
Coût	Entier définissant le coût en PI de la pose de l'élément.
Nom	Chaîne de caractères utilisées lors des affichages dans l'interface.
Comportement	Enumération de deux comportements : DEFAULT : comportement par défaut. FOREST : les éléments forment des zones (arbres).
Franchissable	Booléen indiquant si le sol est franchissable ou non par les unités.
Temps de récolte	Valeur de temps durant laquelle le citoyen doit agir afin de récolter les ressources.
Ressources	Ressources (et leur quantité) associées à l'élément.

TABLE 5.2: Attributs des éléments

Le Tileset

Un unique fichier image au format PNG contiendra l'ensemble des tuiles utilisées dans le jeu pour les sols ou éléments. Cette image sera utilisée comme texture et certaines parties seront extraites par le programme au gré des besoins. Dans le but de permettre à l'utilisateur d'aisément créer des mods de jeu ou changer de tileset, nous avons créé un fichier au format INI détaillant toutes les particularités du tileset.
Voici le tileset que nous utiliserons :



```
# Fichier de paramétrage du tileset :
# ==================================

path data/tileset.png
tilesize 32 32
groundColumnCount 16
elementColumnCount 4

# Chaque ligne correspond à un unique type de sol ou élément :

# ground type cost name behavior passable < borders >
ground 4 1 terre-aride default true < 5 >
ground 3 1 terre-defaut default true < 4 >
ground 2 2 terre-fertile default true < 3 >
ground 1 1 terre-innondee default true < 0 2 >
ground 0 2 eau fluid false < 1 2 >
ground 5 2 falaise cliff false < 4 >

# element type cost height name behavior passable pickingTime < ressources
quantities > < groundTypes >
element 0 1 2 arbre forest false 4 < wood 3 > < 4 3 2 1 >
element 1 1 2 arbre-fruitier forest false 4 < wood 2 food 1 > < 4 3 2 1 >
element 2 1 1 buisson default false 3 < food 3 > < 4 3 2 1 >
element 3 1 1 rocher default false 3 < stone 3 > < 4 3 2 1 >

# ashes
ashes
```

FIGURE 5.1: Texture et configuration du Tileset

Fichier de configuration :
Chaque ligne commençant par "#" ne sera pas lue, ce sont les commentaires. Les mots en vert sont les mots clés utilisés pour la lecture qui se fait ligne par ligne :

- path : indique le chemin du fichier de texture du tileset
- tilesize : est suivi des dimensions (en pixel, largeur puis hauteur) d'une tuile
- groundColumnCount : nombre de colonnes pour chaque sol dans le fichier de texture (toutes les associations de rebords)
- elementColumnCount : nombre de colonnes pour chaque élément dans le fichier de texture (tous les types de sols)
- ground : définit un sol et tous ses attributs
- element : définit un élément et tous ses attributs (height est la hauteur en nombre de cases dans la texture)
- ashes : fourni une texture pour les cendres (lorsque des éléments brûlent) (cf GDD)

Remarque : chaque ligne commençant par ground, element ou ashes incrémente un compteur afin de connaître la position de la zone de texture correspondante. L'ordre est donc important (définition des lignes de haut en bas).

Fichier image :
Chaque ligne correspond à un type de sol ou élément.

Pour chaque sol, il existe seize tuiles différentes qui correspondent à toutes les jonctions possibles avec d'autres types compatibles. Cela correspondra à l'attribut "bordures" des sols. Ils sont définis dans un ordre défini par un mot de 4 bits, avec pour chaque bit un booléen qui correspond à la présence ou non d'une bordure. En allant du bit de poids fort au bits de poids faible, on définit : gauche, bas, droite, haut.
Par exemple, une bordure correspondant au mot 1111 aura des bordures tout autour tandis que celle correspondant au mot 1010 aura des bordures seulement à gauche et à droite.

Pour chaque élément, il existe autant de tuiles que de types de sols compatibles. Elles sont définies dans l'ordre défini dans le fichier INI, sur une seule ligne.

Enfin, il reste une ligne transparent à la fin du tileset nécessaire pour l'affichage des éléments vides.

Incidences

Ici seront brièvement expliqués les principes des différents types d'incidences environnementales, qui seront utilisées dans différentes situations.

Dilatation

Les incidences basées sur la dilatation des zones prennent en compte les types de sols ou éléments afin d'élargir les étendues voulues en propageant les sols si besoin afin de garder les contraintes définies dans le GDD.
Quelques présentations des résultats de ces dilatations sont présentes en annexe, page 38.

Erosion

Les incidences basées sur l'érosion des zones prennent en compte les types de sols ou éléments alentours afin de choisir les types en propageant les sols si besoin afin de garder les contraintes définies dans le GDD.
Quelques présentations des résultats de ces érosions sont présentes en annexe, page 38.

Aléatoire

Les dernières incidences concernant le territoire sont basées sur un choix aléatoire de cases. Il s'agit ici d'appartition ou disparition de certaines ressources.
Quelques présentations des résultats de ces incidences particulières sont présentes en annexe, page 39.

Gestion des contraintes

Comme expliqué dans la section 5.2, le fichier de configuration du tileset implique une certaine quantité de contraintes de jonctions entre les sols. Par exemple, l'eau ne peut pas être juste à coté d'une falaise.
Cet aspect que nous avons choisi de ne pas "coder en dur" a posé beaucoup de problèmes lors de l'utilisation de fonctions telles que changer un sol, dilater une zone, etc... (cf sections 3.2, page 7 et 5.2 ci-dessus).
En effet, des fonctions ont dû être ajoutées pour comparer des sols ou connaître leurs compatibilités. De plus, chaque fonction changeant au moins un sol doit pouvoir propager les types de sols compatibles aux alentours, afin de garder une homogénéité des contraintes sur la carte. Ainsi, lorsque de l'eau est posée sur une falaise, autour de l'eau seront posés récursivement les types compatibles allant vers le type falaise :

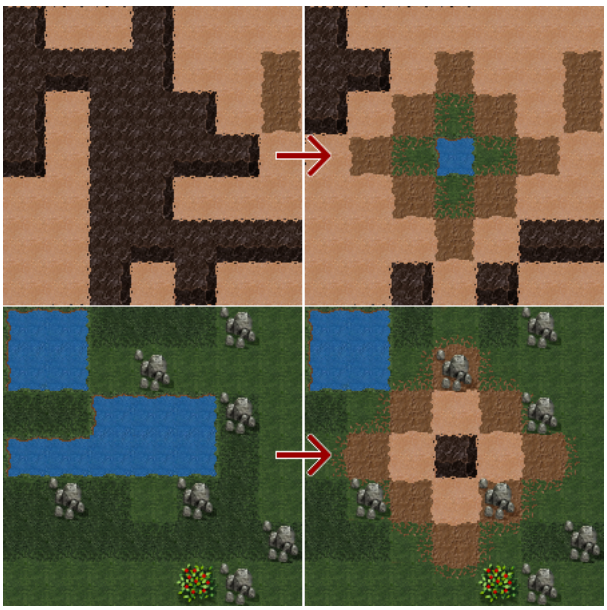


FIGURE 5.2: Exemple de propagation des contraintes sur les sols

Enfin, après tous les changements de sols, il faut rafraîchir les bordures de chaque case concernée afin de respecter les différentes possibilités du tileset (seize tuiles différentes, une pour chaque bordure possible ; cf section 5.2, page 25).

En conclusion, le plus difficile à gérer dans cette partie a été les précautions à prendre pour ne pas avoir de sols incompatibles entre eux, afin de garder non seulement les règles du jeu mais également l'esthétique.

5.3 Moteur multi-agent

Gestion des entités

La santé des entités vivantes

Chacune des entités possède une gestion de la santé avec plusieurs états.

- **Bonne santé :** Si l'entité est un citoyen, elle offre de plus grands bonus de PI.
- **Normal :** L'entité est dans son état par défaut.
- **Blessé/Malade/Fatigué :** L'entité agit avec un léger malus de vitesse. Si l'entité est un citoyen, elle offre de plus petits bonus de PI.
- **Gravement blessé/malade :** L'entité agit avec un malus de vitesse plus important. Si l'entité est un citoyen, elle n'offre plus de PI.
- **Mort :** L'entité disparaît.

Incidences

La météo influe sur les entités en fonction de la valeur de la jauge.

- **Jauge positive (un ou plusieurs jours de temps ensoleillé consécutifs) :** Améliore la pousse des champs mais un excès de soleil assèche les terres et récoltes, peut réduire les étendues d'eau, une sécheresse trop longue peut faire brûler certaines ressources et augmenter la probabilité de fatigue .
- **Jauge négative (un ou plusieurs jours de temps pluvieux consécutifs) :** Permet de faire pousser les champs. Un surplus de pluie innonde les terres et récoltes, augmente les probabilités de maladie et peut augmenter la taille des étendues d'eau.

L'état de santé joue un rôle capital au niveau de la vitesse de déplacement, du temps de récolte et aussi sur la fréquence à laquelle une entité va vous apporter des points d'incidences(PI).
Un état de santé faible , une maladie ou un état très faible peut entraîner le décès d'une entité avec une probabilité de plus en plus forte dans ce même ordre de citation.
Aussi chaque entité a une légère chance de donner naissance à un petit.

Gestion des scripts

Les scripts LUA implementent les différents comportements de toutes les catégories d'entités. Nous avons utilisé une librairie LUA permettant de redéfinir des méthodes de classe implémenté en C++ et de définir des methodes dans les classes C++ qui pourront etre appelé par les script LUA.
On utilise donc un script LUA pour redifinir une methode de la classe "Entity" qui sera appelé pour simuler l'intelligence de l'entités.
Par contre, cette librairie et les mécanismes de c++ nous ont imposé certaines contraintes. La plus importante est qu'il est impossible que 2 instances d'une même classe posséde 2 définissions différentes d'une de leurs methodes. Il nous a donc fallu implémenter une sous-classe pour chaque catégorie d'entités afin d'avoir un comportement different pour toutes ces categories. Cette problematique empeche les joueurs de pouvoir créer de nouvelles catégories d'entite, il pourra cependant modifier celle deja existante.

Liste des scripts

Le comportement des agents, en ce qui concerne le déplacement, a soulevé le problème de la gestion des obstacles étant donné que certains éléments du décor ne sont pas franchissables.
En effet, certaines actions nécessitant un trajet spécifique utilisent le résultat d'un algorithme de recherche de chemin et se déroulent sans interruption sur plusieurs ticks, c'est-à-dire sans rappeler le script de l'agent.
Par conséquent, les directives des scripts sont en partie des objectifs à atteindre et non de simples actions isolées.

Les citoyens alliés

- **Récolteurs(bûcherons, cueilleurs, mineurs) :**
  Si un bûcheron faible, un mineur faible ou un cueilleur perçoit un animal

agressif ou un citoyen ennemi celui-ci va fuir.
Si un bûcheron ou un mineur se fait attaquer, il va répliquer en attaquant en priorité l'entité face à lui si celle-ci l'attaque.
A chaque fois que l'entité récolte une ressource elle remplit son sac puis se dirige vers la base pour la déposer.
Lorsque son sac est vide, elle regarde dans son rayon de perception si une ressource est disponible. Si au moins une ressource est repérée, elle se dirige vers la plus proche pour la récolter. Sinon, elle se dirige de manière aléatoire.

- **Chasseurs :**
  Le chasseur reste relativement proche de la base et adopte un rôle de défenseur.
  S'il est attaqué, il va regarder le nombre de chasseurs et le nombre d'entités hostiles autour de lui , s'il y a moins d'entités hostiles que de chasseurs auxquels on ajoute 3, il va attaquer l'entité qui l'attaque avec dans l'ordre de priorité : l'entité en face de lui, un citoyen ennemi et un animal agressif. Sinon il adopte la fuite.
  S'il n'est pas attaqué, tout en gardant une distance proche de la base ,il va chercher une cible autour de lui dans cet ordre de priorité, un citoyen ennemi, un animal agressif puis un animal passif.
  Encore une fois s'il n'y a pas trop d'ennemis aux alentours, le chasseur va aller attaquer la cible sinon il va fuir.
  S'il n'y a ni citoyens ennemis ni animaux autour de lui et s'il n'est pas trop loin de la base, il va avancer aléatoirement sinon il va revenir vers la base.

Les citoyens ennemis

Les citoyens ennemis ont un comportement agressif.
S'ils sont attaqués, il vont adopter le même comportement qu'un chasseur sauf que dans l'ordre de priorité ils vont attaquer : l'entité en face d'eux, un chasseur et un récolteur.
S'ils ne sont pas attaqués, même démarche que le chasseur en changeant l'ordre de priorité par un chasseur puis un récolteur.
Si aucun chasseur ou récolteur n'est perçu dans leurs rayons de perception, ils vont avancer aléatoirement.

Les animaux

- **Agressifs :**
  Dès qu'un animal agressif perçoit une entité excepté un animal agressif il va l'attaquer, en attaquant en priorité l'entité qui est en face de lui. Si aucune entité n'est perçue ou s'il ne perçoit que des animaux agressifs , l'animal va se déplacer aléatoirement.
- **Passifs :**
  L'animal passif va tout simplement fuir toutes les entités à l'exception des animaux passifs.

Dans certaines situations vues précédemment, l'entité se déplacera de manière aléatoire et pourra marquer un arrêt de temps à autre afin de simuler un comportement naturel.

5.4 Météo

La météo est gérée comme une jauge comprise entre cinq et moins cinq. Au commencement d'une partie la jauge est à zéro.
A la fin de chaque journée nous comparons le temps du jour terminé avec le temps du jour précédent, si celui-ci est le même et est ensoleillé la jauge augmente , si par contre le temps est pluvieux la jauge descend.
En revanche, si la jauge est entre -3 et -5 et que la journée fut ensoleillé ou inversement si la jauge est entre 3 et 5 et qu'il fit un temps pluvieux la jauge revient respectivement à -1 ou 1.
Plus la valeur de la jauge est grande positivement ou négativement, plus il y aura d'incidences en rapport avec la pluie pour les valeurs négatives et inversement pour le soleil.

5.5 Architecture

Machine à états

Main menu state

Le menu principal est le premier état de l'application, il sert d'écran d'accueil et dispose de trois boutons :

- "Start" qui crée une nouvelle instance de jeu et appelle l'état "Game State".
- "Load" qui appelle l'état "Load Menu State".
- "Exit" qui quitte l'application.

Game state

L'état de jeu est le primordial au sein de l'application, il sert de support à l'instance de jeu afin d'en gérer le déroulement. Il contient l'affichage de la partie visible de la carte du monde et des entités ainsi qu'une interface, permettant au joueur de gérer la partie, ayant deux fonctions. La première consiste à informer de la valeur de certaines variables inhérentes à la partie comme le nombre de points ou d'une certaine ressource. La seconde permet au joueur d'indiquer les actions qu'il souhaite réaliser via un ensemble de boutons.

Night state

L'état de nuit revient régulièrement au cours de la partie, pendant celui-ci ont lieu les incidences sur le territoire (cf section 5.2, page 26) et sur les entités (cf section 5.3, page 29).
De plus, cet état permet au joueur de déterminer directement la météo de la prochaine journée de jeu et d'influer indirectement sur les proportions des rôles des entités. En plus de cela, un bouton de validation supprime l'état courant afin que la partie puisse reprendre son cours dans l'état de jeu.

End state

L'état de fin de partie signale au joueur qu'il a atteint la condition de défaite, il dispose simplement d'un bouton :

- "Back" qui supprime l'état courant.

Game menu state

Le menu de jeu est l'état de pause du jeu, il dispose de quatre boutons :

- "Back" qui supprime l'état courant.
- "Save" qui appelle l'état "Save Menu State".
- "Load" qui appelle l'état "Load Menu State".
- "Quit" qui supprime tous les états jusqu'à l'état "Main Menu State".

Save menu state

Le menu de sauvegarde est l'état qui permet de sauvegarder son avancement, il dispose d'un champ de saisie du nom de sauvegarde et de deux boutons :

- "Back" qui supprime l'état courant.
- "Save" qui crée un fichier de sauvegarde à partir de l'instance de jeu en cours et le nomme d'après le texte entré.

Load menu state

Le menu de chargement est l'état qui permet de charger une partie précédemment sauvegardée, il dispose d'une liste de boutons, représentant les fichiers de sauvegarde répertoriés à partir desquels l'instance de jeu sera modifiée, de deux boutons permettent de parcourir la liste et d'un dernier bouton :

- "Back" qui supprime l'état courant.

Chapitre 6

Post-Mortem

Cette section liste toutes les étapes de la conception que nous n'avons pas réalisées (selon notre ordre de priorités), ainsi que toutes les améliorations auxquelles nous avons pensé lors de la phase de développements.

6.1 Fonctionnalités non implémentées

Au niveau des entités, nous n'avons pas permis aux citoyens alliés de contruire des bâtiments, ni d'utiliser les ressources telles que le bois et la pierre pour ces contructions. Les entités n'ont pas non plus d'animations pour les actions spécifiques de récolte.

Au niveau du système de jeu, nous n'avons pas créé d'arbre de technologie, permettant au joueur de spécialiser les citoyens ou réduire les coûts des contructions, augmenter la génération des PI... etc. Les pouvoirs divins ainsi que le système de karma auxquels nous avions pensé n'ont pas été implémenté (le karma serait un choix du joueur sur le taux d'agressivité ou de passivité des citoyens alliés).

6.2 Améliorations réalisables

Au niveau de la carte, il serait possible de créer un éditeur de carte, sachant que toute les fonctions nécessaires existent déjà. Nous pourrions également modifier la génération aléatoire de carte pour avoir des cartes plutôt arides ou plutôt humides par exemple, ainsi que faire un pré-affichage de la modification de la carte au passage de la souris lorsque le joueur veut changer un sol (par exemple les cases qui seraient modifiées en vert et non modifiables en rouge).

Au niveau des entités, il serait bon de diversifier les bâtiments, la faune et les comportements des animaux, comme par exemple avoir des troupeaux. Nous pourrions aussi créer des civilisations neutres et mettre en place un système d'interactions entre les civilisations, à l'intérieur ou en dehors des limites de la carte (nous pouvons imaginer des émissaires étrangers venant commercer par exemple).

Au niveau du système de jeu, nous pourrions imaginer créer des objectifs pour le joueur, comme atteindre un point de la carte, récolter certaines ressources, créer des "artefacts" et les placer sur la carte... etc.

Au niveau de l'interface, nous pourrions mettre en place du son (musiques et bruitages), avoir un système de statistiques pour comparer les résultats de différentes parties. Nous pourrions aussi

Annexe A

Dans cette partie seront placés les éléments trop volumineux pour être inclus directement dans le texte, tels que les images ou graphiques.

A.1 Diagramme de Gantt

Le diagramme est fourni page 36.

A.2 Comparatif de performances

Le graphique est situé page 37.

A.3 Incidences sur la carte

Les présentations sont présentes de la page 38 à la page 39.

	Nom	Durée	Début	Nom de la Ressource
1	Etude Préalable	30 jours	20/01/14 08:00	CHAMBONNET Kevin;GAUTHIER ..
2	**⊟Moteur**	**42 jours?**	**03/03/14 08:00**	
3	**⊟Gestion des Entités**	**42 jours?**	**03/03/14 08:00**	
4	Moteur multiagent	20 jours?	03/03/14 08:00	
5	Implémenter les Entités	5 jours	14/04/14 08:00	CHAMBONNET Kevin
6	Placement des Entités	1 jour	21/04/14 08:00	MOKHRETAR Amin
7	Gestion des Maladies	2 jours	25/04/14 08:00	MARTINEZ Thierry
8	Incidences de la Santé	1 jour	29/04/14 08:00	MARTINEZ Thierry
9	**⊟Gestion de la Carte**	**41 jours**	**03/03/14 08:00**	
10	**⊟Moteur de Carte**	**39 jours**	**03/03/14 08:00**	
11	Conception	5 jours	03/03/14 08:00	GAUTHIER Silvère
12	**⊟Implementation des Classes**	**26 jours**	**17/03/14 08:00**	
13	Gestion du TileSet	4 jours	17/03/14 08:00	GAUTHIER Silvère
14	Sols et Elements	3 jours	21/03/14 08:00	GAUTHIER Silvère
15	Génération de Carte	3 jours	26/03/14 08:00	GAUTHIER Silvère
16	Optimisations Diverses	1 jour	15/04/14 08:00	GAUTHIER Silvère
17	Méthode "freePlace"	1 jour	21/04/14 08:00	GAUTHIER Silvère
18	Affichages	1 jour	14/04/14 08:00	GAUTHIER Silvère
19	Gestion des Bâtiments	2 jours	22/04/14 08:00	MOKHRETAR Amin
20	Affichage des Bâtiments	1 jour	24/04/14 08:00	MOKHRETAR Amin
21	Incidences sur le Territoire	3 jours	16/04/14 08:00	GAUTHIER Silvère
22	Incidences sur les Entités	2 jours	22/04/14 08:00	GAUTHIER Silvère
23	Sauvegarder / Charger	3 jours	24/04/14 08:00	GAUTHIER Silvère
24	**⊟Cycle Jour/Nuit**	**4 jours**	**21/04/14 08:00**	
25	Jour	1 jour	21/04/14 08:00	CHAMBONNET Kevin
26	Nuit	3 jours	22/04/14 08:00	CHAMBONNET Kevin
27	Météo	1 jour	21/04/14 08:00	MARTINEZ Thierry
28	**⊟Points d'incidence**	**5 jours**	**22/04/14 08:00**	
29	Ajout	3 jours	22/04/14 08:00	MARTINEZ Thierry
30	Suppression	2 jours	25/04/14 08:00	MOKHRETAR Amin
31	Sauvegarder / Charger	2 jours	25/04/14 08:00	CHAMBONNET Kevin
32	**⊟Scripting**	**35 jours?**	**03/03/14 08:00**	
33	Lister les Scripts	5 jours	03/03/14 08:00	MARTINEZ Thierry;MOKHRETAR ..
34	Etude préalable	5 jours?	10/03/14 08:00	MARTINEZ Thierry;MOKHRETAR ..
35	Concevoir les Scripts	25 jours?	17/03/14 08:00	MARTINEZ Thierry[50 %];MOKH..
36	Implémenter les Scripts	25 jours?	17/03/14 08:00	MARTINEZ Thierry[50 %];MOKH..
37	**⊟Interface Graphique**	**12 jours**	**01/05/14 08:00**	
38	Widgets	3 jours	01/05/14 08:00	CHAMBONNET Kevin
39	Menus	5 jours	05/05/14 08:00	MOKHRETAR Amin
40	Gestion Animations	3 jours	07/05/14 08:00	MARTINEZ Thierry
41	Gestion Sons	2 jours	01/05/14 08:00	MARTINEZ Thierry
42	Informations contextuelles	5 jours	12/05/14 08:00	CHAMBONNET Kevin;MARTINEZ ..
43	**⊟Graphismes**	**45 jours**	**08/03/14 08:00**	
44	Elements de Carte	5 jours	08/03/14 08:00	GAUTHIER Silvère
45	Entites	2 jours	29/04/14 08:00	GAUTHIER Silvère
46	Elements d'interface	7 jours	01/05/14 08:00	GAUTHIER Silvère
47	Equilibrage du Jeu	5 jours	19/05/14 08:00	CHAMBONNET Kevin;GAUTHIER ..

FIGURE A.1: Diagramme de Gantt

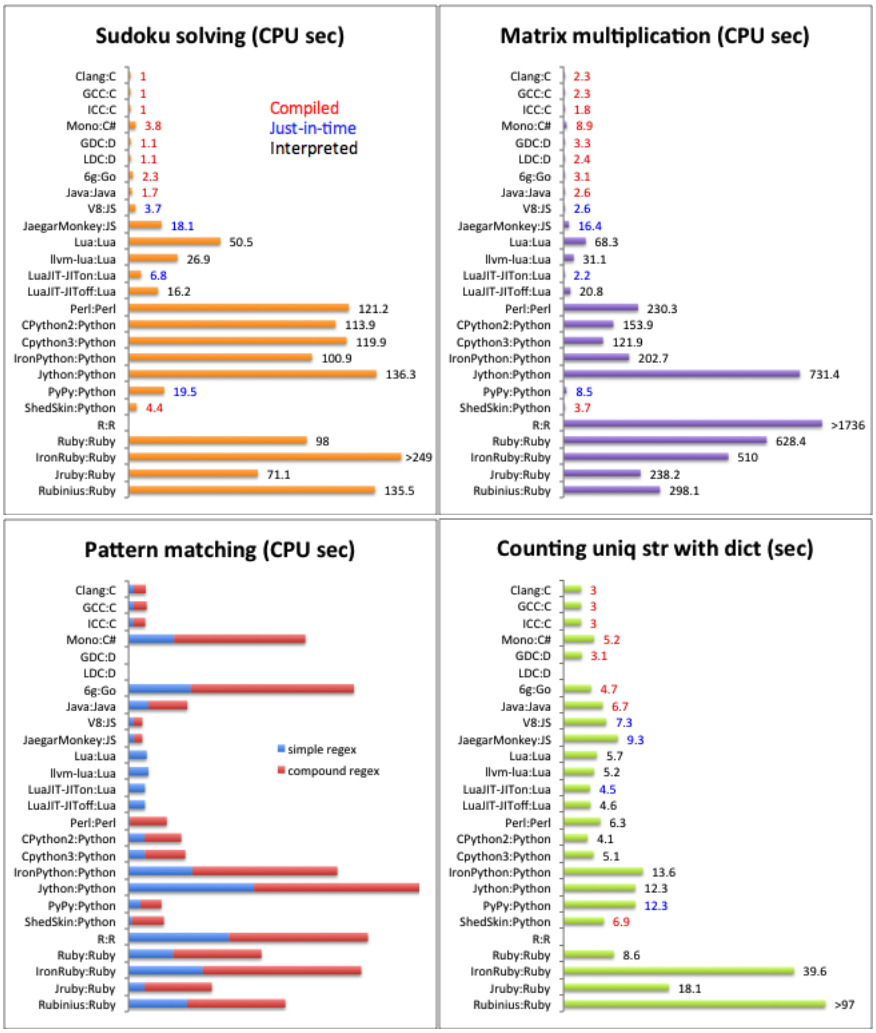


FIGURE A.2: Comparaisons de performances de divers langages dans des cas donnés

FIGURE A.3: Dilatations après trois itérations

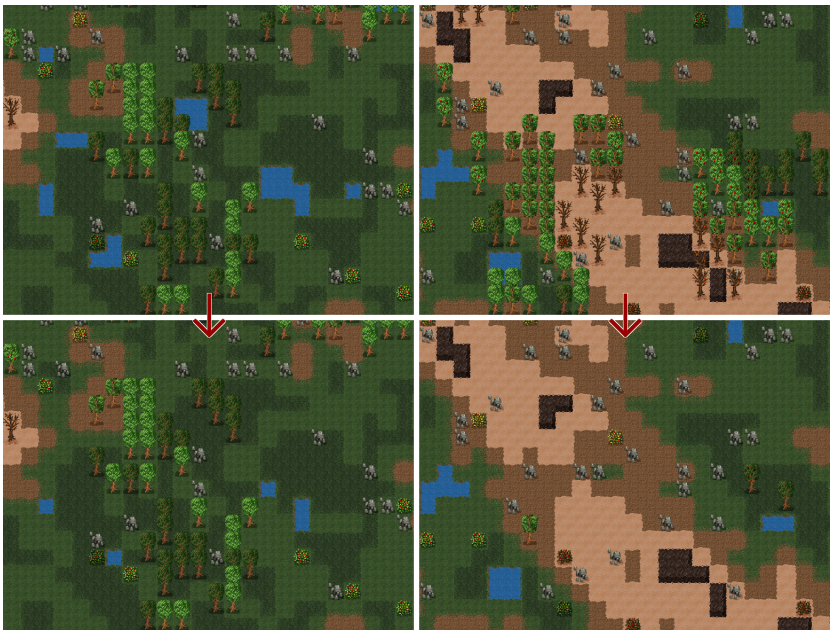


FIGURE A.4: Erosions après trois itérations

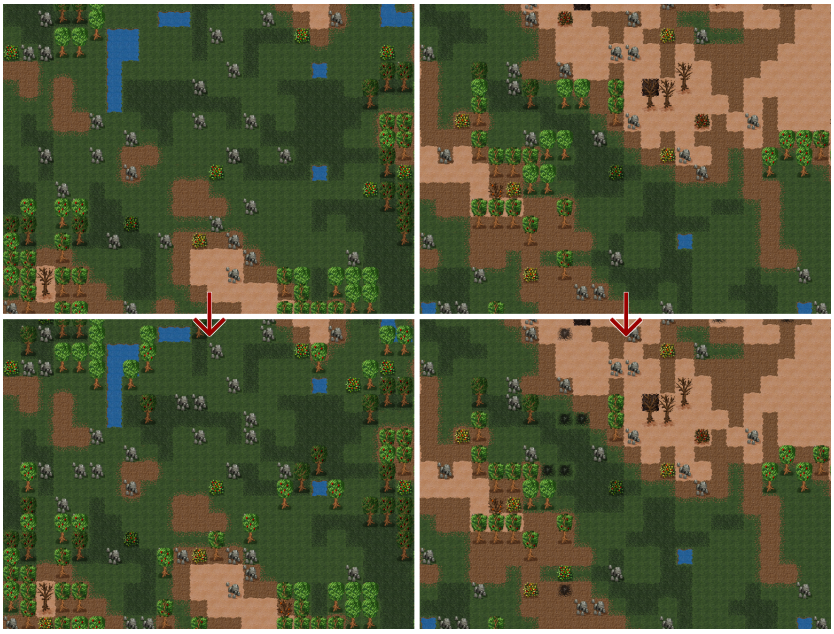


FIGURE A.5: Aléatoires après trois itérations