

Technical Design Document

- Incidence -

CHAMBONNET Kevin

GAUTHIER Silvère

MARTINEZ Thierry

MOKHRETAR Amin

February 23, 2014

Contents

I	Organisation	3
1	Gestion du projet	3
1.1	Gestion de l'équipe	3
1.2	Découpage en tâches	3
1.3	Assignment	4
1.4	Gestion du temps	4
2	Choix technologiques	4
2.1	Langages de programmation	4
2.2	Bibliothèques	6
II	Code Overview	7
3	Format des Fichiers	7
3.1	Animation	7
3.2	Tileset	7
3.3	Carte	7
3.4	Sauvegarde	7
4	Commentaires	7
5	Convention de Nommage	7
6	Gestion du code source	7
III	Développement	8
7	Le Moteur	8
7.1	Etat	8
7.2	Animation	8
7.3	Gestion des Ressources	8
8	Les Entités	8
8.1	Les citoyens	9
8.2	Liste des scripts	9
9	La Carte	9
9.1	Les Tuiles	9
9.2	Le Tileset	9
IV	Milestones	10

Part I

Organisation

1 Gestion du projet

1.1 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.

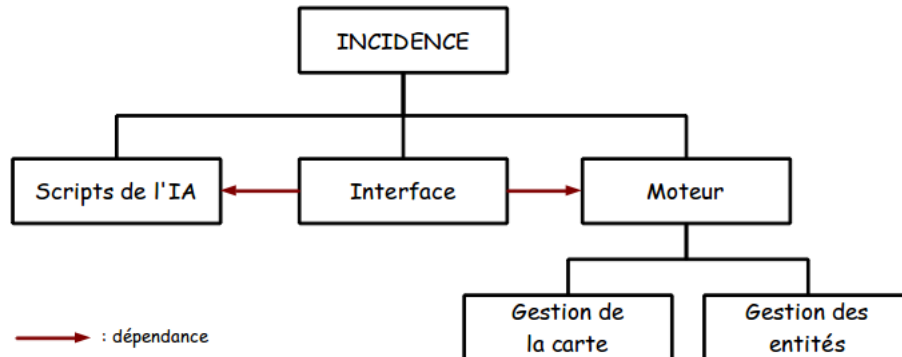
Chaque membre a donc autant de pouvoir que les autres, et peut donc participer activement au projet, autant lors de la conception que du développement. Toutes les décisions seront prises suivant la majorité lors de votes.

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 exécutable. 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.

1.2 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 :



1.3 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 4.

1.4 Gestion du temps

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

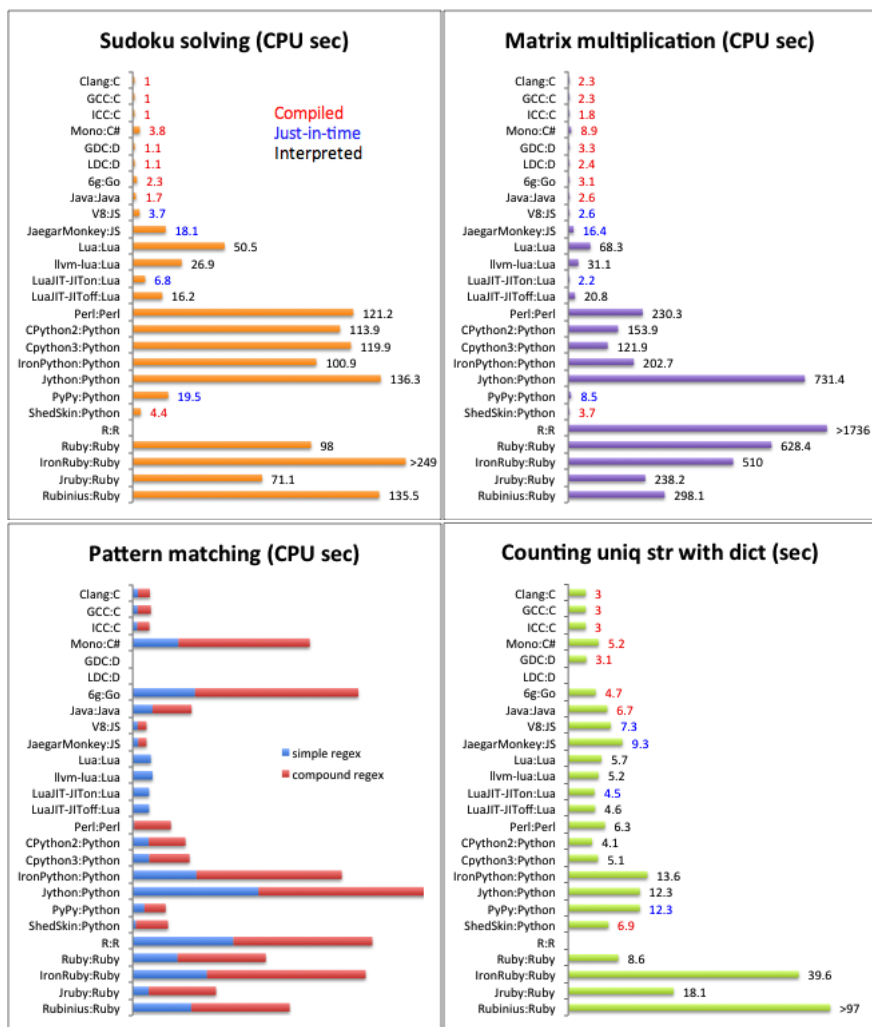
Voici tout de même une première estimation du temps nécessaire :

2 Choix technologiques

2.1 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.

Voici des tests de performances concernant un large panel de langages, comparés ici dans quatre contextes différents :



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.

2.2 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 son, ce qui sera un plus pour la finalité de notre jeu.

Part II

Code Overview

3 Format des Fichiers

3.1 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
```

3.2 Tileset

3.3 Carte

3.4 Sauvegarde

4 Commentaires

5 Convention de Nommage

6 Gestion du code source

Part III

Développement

7 Le Moteur

Le moteur du jeu est très basique et sert surtout à la gestion plus simplifiée des différents états du jeu ainsi que de toutes les ressources.

7.1 Etat

Le Moteur contient un gestionnaire d'état qui permet de contrôler et d'appeler le dernier état du jeu. Une classe abstraite Etat est la classe mère de tous les états, chaque état ainsi créé aura une méthode de mise à jour, d'affichage et de gestion des différentes actions clavier/souris du joueur.

7.2 Animation

Une classe Animation a été créée pour faciliter la création et le contrôle de toutes les animations du jeu. Toutes les animations peuvent être modifiées grâce aux fichiers d'initialisation de chacune d'elles.

7.3 Gestion des Ressources

Une classe de gestionnaire de ressources permet de gérer toutes les ressources du jeu pour éviter de charger en double ou plus certaines ressources. Ce gestionnaire gère toutes les textures, images et sons.

8 Les Entités

Les entités constitueront tout ce qu'il y a de mobile dans le monde : les Citoyens et les différents Animaux.

Chaque entité possèdera des méthodes communes utilisées dans les différents scripts de leur IA :

- **Connaître sa position sur la carte :** La position sera donnée en valeur entière correspondant à la case sur laquelle l'entité est présente.
- **Se déplacer vers une direction donnée :** L'entité se déplacera en ligne droite dans cette direction. Si des obstacles se trouvent sur le chemin, elle essayera, si possible, de les contourner.
- **Connaître les entités alentours :** Recupère une liste d'entités présentes dans le champs de vision de l'entité appelant la méthode.
- **Connaître les cases alentours :** Recupère une listes de booléen correspondants aux différentes cases présentes dans le champs de vision, avec pour valeur TRUE si la case est franchissable, FALSE sinon.

De plus, toutes les entités auront une méthode de mise à jour, qui appellera le script en cours, et une d’affichage, pour l’afficher sur la carte. Chaque entité possèdera aussi une liste d’attributs modifiables par les différentes méthodes :

- **Sa position sur la carte :** En pixel et non en case.
- **Son activité :** C’est à dire, le script en cours.
- **Son Animation en cours :** Le visuel de l’entité.

8.1 Les citoyens

Les Citoyens sont les entités que le joueur devra garder en vie et c’est eux qui auront un impact direct sur l’environnement. Pour cela, ils possèdent des méthodes supplémentaires :

- **Connaître les positions d’un type de case alentours :** Retourne une liste de positions correspondantes aux positions d’un type de case, passé en paramètre, dans le champs de vision du citoyen.
- **Interagir avec une case :** Le citoyen interagit avec la case passée en paramètre. Ex.: Pour un arbre, il le coupera.

Chacun des citoyens aura un métier assigné durant la nuit, et c’est ce métier qui définira le groupe de scripts que le citoyen suivra pour effectuer ses actions. Ce métier ajoute donc aussi de nouveaux attributs :

- **Le nom du métier :** Pour facilement connaître le métier mais aussi la ressource associée à celui-ci et surtout le groupe de scripts à appeler.
- **La quantité de ressources portée :** Chaque entité pouvant porter qu’un seul type de ressource, cette quantité est associée à la ressource, elle-même liée au métier. Ex.: Bois pour un Bûcheron.

8.2 Liste des scripts

9 La Carte

9.1 Les Tuiles

9.2 Le Tilesset

Part IV

Milestones

12 Mars 2013 : Gestion de la carte et des entités terminée + ajout des entités sur la carte + ajout des scripts aux entités

19 Mars 2013 : Finalisation des scripts et des entités

10 Avril 2013 : Fin de la première version :D