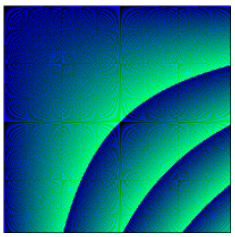
17 janvier 2014

PDG



Propagation Simulator

Rapport

Groupe 3

Decorvet Grégoire, Froger Hadrien,
Jaquier Kevin, Schweizer Thomas,
Sinniger Marcel

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Table des matières

Chapitre 1

Introduction

Le but de ce projet est de concevoir et d'implémenter un logiciel de simulation de propagation dénommé *Propagation Simulator*[1]. Ce simulateur en tant que tel n'a pas pour but de simuler un problème choisi lors de la conception. Ce logiciel permet de résoudre une large gamme de problèmes de simulation de propagation tel que le jeu de la vie, une épidémie, une migration, etc... En tant que simulateur, ce programme sera capable de simuler une situation selon les règles que l'utilisateur aura fournies avant le lancement du programme et de rendre à la fin des statistiques ciblées spécifiées par l'utilisateur.

1.1 Objectif du programme

Ce logiciel permettra de réaliser une simulation portant sur un problème de propagation quelconque sans devoir laborieusement écrire un programme spécifique pour simuler un problème précis avec ses propres règles. Pour paramétrer une nouvelle simulation, l'utilisateur aura à écrire des fichiers de configuration. Les données qu'il obtiendra de cette simulation seront spécifiées par l'utilisateur à l'intérieur des fichiers de configuration. Les résultats seront exportés durant la simulation sans aucun pré-traitement effectué par le programme. L'analyse des données devra se faire en dehors de ce logiciel.

1.2 Objectif du projet

Ce projet a d'une part pour objectif de continuer à accumuler de l'expérience dans la réalisation de projets logiciels conséquents en groupe restreint. D'autre part, nous souhaitons également utiliser ce projet pour expérimenter une gestion de projet agile[2].

Sur le plan technique, nous avons pour objectif d'utiliser un langage de programmation non étudié dans le cadre de nos études, et d'utiliser des librairies et technologies reconnues dans l'industrie pour lesquelles nous n'avons pas encore d'expérience.

1. *PropSim* en version abrégée.
2. Nous avons décidé de nous inspirer de la méthode agile SCRUM pour ce projet. Des informations supplémentaires sont données en 2.1 SCRUM

Chapitre 2

Gestion de projet

Ce chapitre explique la démarche que nous avons choisie pour la gestion de projet et la répartition des rôles.

2.1 SCRUM

SCRUM [1] est un ensemble de techniques pour la gestion de projet. Il permet d'aider les équipes à rendre des parties du produits finis dans des cycles courts (nommés *releases*), permettant ainsi un retour rapide des clients et des utilisateurs.

Il s'agit donc d'éviter une conception complète dès le départ, afin de se focaliser sur des résultats à courts termes. Le principe est de segmenter le rendu de projet en plusieurs release. A chacune d'elles, une démo peut être fournie au client et il peut potentiellement stopper le projet. Cela demande donc d'avoir des tests, un manuel utilisateur et une documentation à jour.

Une release est fragmentée en items. Chaque item correspond à un travail à faire allant de la documentation à la programmation. Ces items sont définis en début de release par tous les membres du projet puis affectés aux différents membres selon leur préférences.

SCRUM donne également de nombreux conseils sur le déroulement des rencontres, la fréquences des discussions. De nombreux retours d'expérience d'équipes SCRUM ont été publiés, qui donnent de nombreuses idées pour l'implémentation de la méthode. [2]

2.2 Organisation hebdomadaire

Ce projet était organisé par semaine. Commençant le lundi et finissant le dimanche. Le lundi, le chef de projet passait vers chacun des membres pour se tenir au courant de ce qui avait été fait depuis le dernier vendredi pour mettre à jour le journal de travail du projet. Le vendredi était consacré à la séance de groupe à laquelle tout le monde participait. Lors de cette séance, les membres présentaient le résultat de leur travail afin que tout le monde soit au courant de l'avancement et que le chef de projet puisse mettre à jour le journal de travail du projet. Chaque séance est consignée dans un procès-verbal qui contient le déroulement de la séance, les décisions prises et un bref résumé du travail des membres du projet. Cette séance servait également à effectuer de la programmation en binôme [3], planifier une release, mettre à jour une release ainsi que faire la synthèse d'une release.

1. http ://www.scrumalliance.org/
2. Henrik Kniberg, "Scrum and XP from the Trenches, How we do Scrum ?", Juin 2007
3. *Pair-programming* en anglais

2.2.1 Les réunions du vendredi

Les réunions du vendredi ont été très importantes pour le déroulement du projet. Nous nous sommes fortement inspirés des sprint-meeting de SCRUM. Ces réunions ont toujours eu lieu dans un lieu convivial, à la cafétéria. Pour faire un tour de table, nous accordions cinq minutes à chaque membre pour expliquer ce qu'il a fait depuis la dernière réunion, les éventuelles difficultés rencontrées, et l'état des items assignés (terminé ou non).

Ainsi, nous évitions de passer trop de temps sur des discussions à rallonge et restions focalisés sur l'essentiel. Après le tour de table, la pile était mise à jour, et d'éventuels items rajoutés, supprimés etou (ré)affectés. Le reste du temps était à disposition pour régler des problèmes spécifiques, se coordonner et programmer en binômes.

2.3 Responsables

Des responsables ont été nommés afin de servir de tête de pont pour un sujet précis. Cette personne devient alors le coordinateur du domaine dans lequel il a été affecté et en est aussi la référence pour tout autre membre du projet qui aurait une question au sujet du domaine dont le coordinateur à la charge.

La répartition est la suivante :

Rôle	Membre
Répondant projet	Thomas Schweizer
→ Répondant planning	Thomas Schweizer
Répondant analyse et développement	Kevin Jaquier
→ Répondant JSON / Structure de configuration	Hadrien Froger
→ Répondant interface graphique	Thomas Schweizer
→ Répondant déploiement	Marcel Sinniger
Répondant documentation	Grégoire Decorvet
Répondant tests	Hadrien
Répondant technique	Kevin Jaquier

Les rôles précédés d'un '→' sont des sous rôles du rôle de niveau supérieur (Le premier rôle au dessus du sous-rôle qui n'est pas précédé par une flèche.)

- Le répondant projet assure la liaison entre le client/investisseur et le projet. Il coordonne également le déroulement du projet et ses collaborateurs [4].
- Le répondant planning s'occupe de proposer et tenir à jour la planification.
- Le répondant analyse et développement s'occupe de coordonner le développement et d'être la personne de référence en cas de question dans ce domaine.
- Le répondant JSON [5] / Structure de configuration est la personne de référence pour tout ce qui a trait aux fichiers JSON et à l'architecture des données utilisateur de simulation.
- Le répondant interface graphique est la personne de référence pour tout ce qui a attrait à l'interface graphique.
- Le répondant déploiement est la personne de référence pour le déploiement du projet et sa livraison au client.
- Le répondant documentation est la personne qui supervise le format et la rédaction de la documentation.
- Le répondant des tests est la personne de référence pour les tests. Elle vérifie également que les tests sont effectués convenablement à l'aide des outils mis à disposition.

4. Dans le cadre SCRUM, cette personne tient le rôle de "scrum master".
5. JavaScript Object Notation - permet de représenter des données sous forme textuelle et structurée

- Le répondant technique est la personne de référence pour la configuration des dossiers, conventions de nommage et configuration des IDEs.

2.4 Releases

Du fait de la méthode agile qui a été choisie, le projet a été découpé en trois *releases*. A chacune de ces releases, des fonctionnalités ont été définies. Nous les rappelons dans ce chapitre après la planification. Nous avons choisi que nous aurions trois releases, chacune ayant une durée de trois à quatre semaines. Ce choix a été motivé par le temps à disposition pour le projet et la contrainte que nous ne voulions pas de releases plus petites que trois semaines. En effet, deux semaines auraient été trop courtes pour réaliser l'analyse, la programmation et la documentation d'une partie du programme.

2.4.1 Planification

Planification initiale

Cette planification a servi de fil conducteur pour toute la durée du projet. Nous n'avons pas eu à la modifier car le système agile permet d'être très flexible et de transvaser certaines fonctionnalités à travers les itérations. Nous pouvons observer que la découpe des quatre principales étapes du projet est bien visible et occupent chacune à peu prêt la même période de temps.

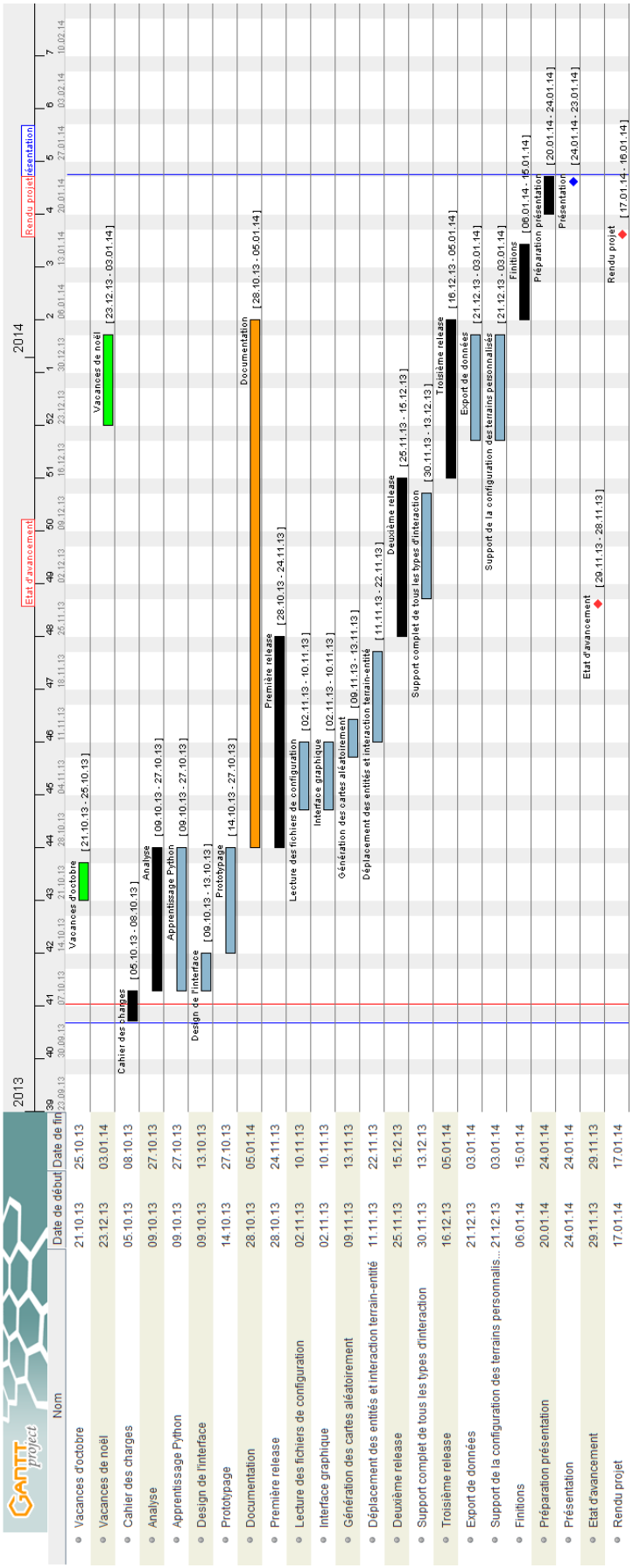


FIGURE 2.1: Planification initiale

2.4.2 Release 1

Le but de la première release était de mettre au point toute la structure de départ du programme et d'implémenter la gestion des actions d'une entité avec un terrain.

Les objectifs visés étaient :

- Génération de monde purement aléatoire pour avoir une base de travail rapidement.
- Affichage de l'interface graphique. Aucune configuration ne se fait dans l'interface. L'interface graphique sera composée des éléments suivants :
  - Un bouton "play" pour lancer la simulation.
  - Un bouton "pause" pour l'interrompre.
  - Un bouton "stop" pour l'arrêter définitivement.
  - Des informations sur la simulation, comme le numéro d'itération par exemple.
- Configurer une entité : exécution des actions. Mais pas des interactions entre entités.

L'utilisateur pourra effectuer les quelques tests suivants.

En tant qu'utilisateur je peux :

- Lancer l'application exécutable afin de pouvoir démarrer la simulation que j'ai configurée.
- Configurer un type de terrain "normal" où les entités peuvent se déplacer.
- Configurer une entité "humain" afin qu'elle puisse se déplacer dans le monde.
- Configurer une entité "moustique" afin qu'elle puisse se déplacer dans le monde.
- Configurer un type de terrain "infranchissable" afin que les entités "moustique" et "humain" ne puissent pas entrer dans le terrain.
- Voir les entités se déplacer sur le monde, et je vois les différents paramètres de ma simulation (temps passé, itération actuelle, nombre d'itérations totales, quantité de chaque entité, et noms des zones différentes).

Remarque : toutes les configurations se font dans des fichiers de configuration.

A la fin de la release, le résultat suivant à été obtenu :

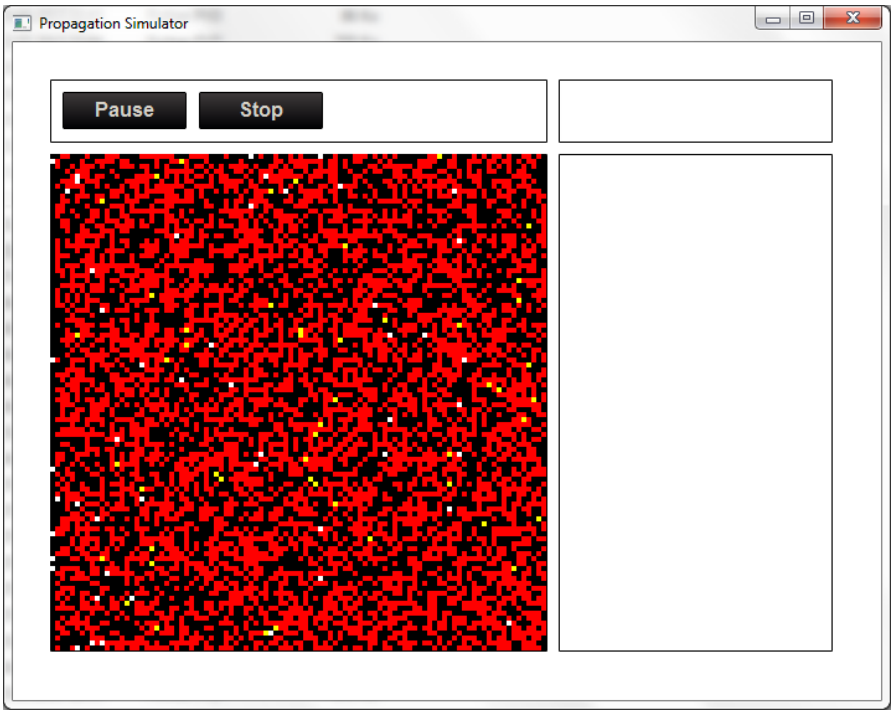


FIGURE 2.2: Illustration de l'interface graphique à la fin de la release 1.

2.4.3 Release 2

Le but de cette deuxième release était d'implémenter les interactions entre les entités du simulateur. Cette implémentation permettra à l'utilisateur de configurer des interactions entre les différentes entités du programme. Les objectifs initiaux pour cette release étaient :

- Les entités peuvent avoir des interactions de type entité-entité
- Les interactions sont entièrement configurables

L'utilisateur pourra effectuer ces quelques tests. En tant qu'utilisateur je peux :

- Définir autant d'interactions que je veux entre les moustiques et les humains. Par exemple, un humain peut "écraser" un moustique, et un moustique peut "infecter" un humain.
- Configurer un type de terrain "à risque" où les moustiques peuvent "infecter" les humains.
- Configurer un type de terrain "normal" où les moustiques ne peuvent pas "infecter" les humains.
- Configurer un type de terrain "infranchissable" afin que les entités "moustiques" et "humains" ne puissent pas entrer dans le terrain.

A la fin de la release, le résultat suivant à été obtenu :

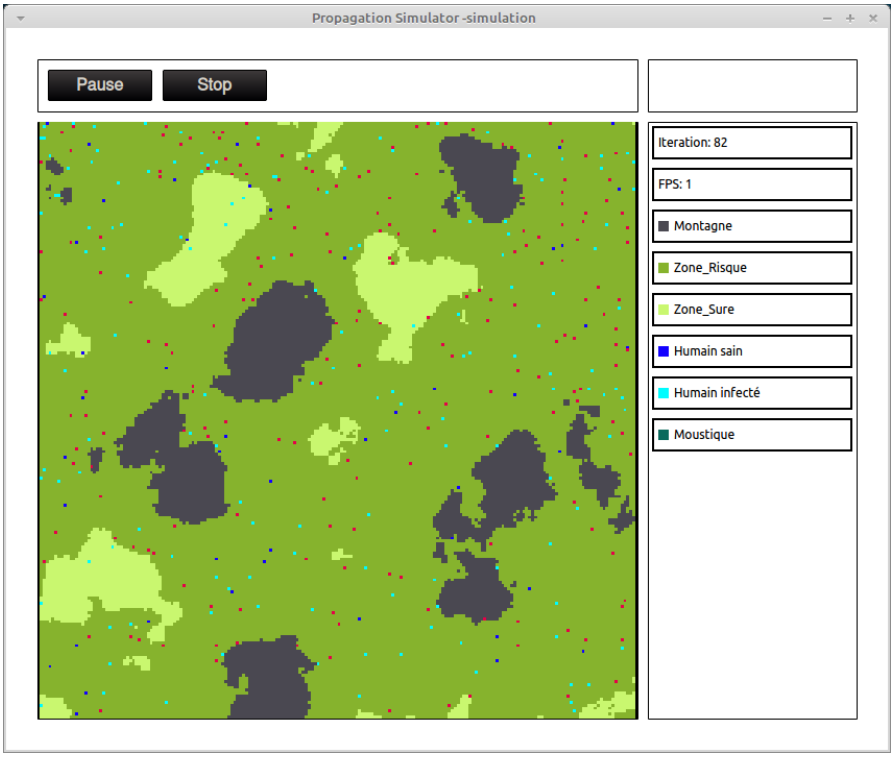


FIGURE 2.3: Illustration de l'interface graphique à la fin de la release 2.

2.4.4 Release 3

La troisième release avait pour objectif de permettre à l'utilisateur de récupérer les résultats fournis par notre programme. Les données rendues sont celles spécifiées dans les fichiers de configuration. Ces mesures sont exportées sans aucun pré-classement, elles sont sous forme brute. Objectifs visés :

- Exportation des données
- Utilisation d'un carte faite manuellement dans la simulation.

En tant qu'utilisateur je peux :

- Avoir accès à des données de simulation exploitable par la suite dans des logiciels statistiques (tels que le programme R [6] par exemple).
- Configurer le format d'exportation.
- Ajouter une carte de simulation que j'ai crée par mes soins.
- Configurer le mode de génération de carte afin d'utiliser une carte prédéfinie ou aléatoire.

A la fin de la release, le résultat suivant à été obtenu :

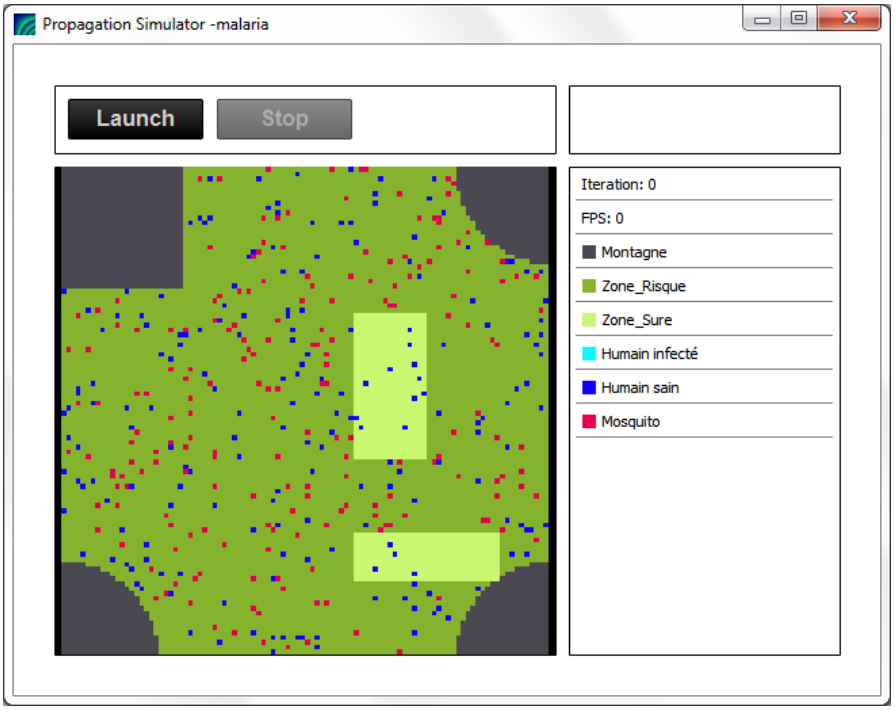


FIGURE 2.4: Illustration de la simulation d'exemple de la malaria à la fin de la release 3.

6. Voir http ://www.r-project.org/

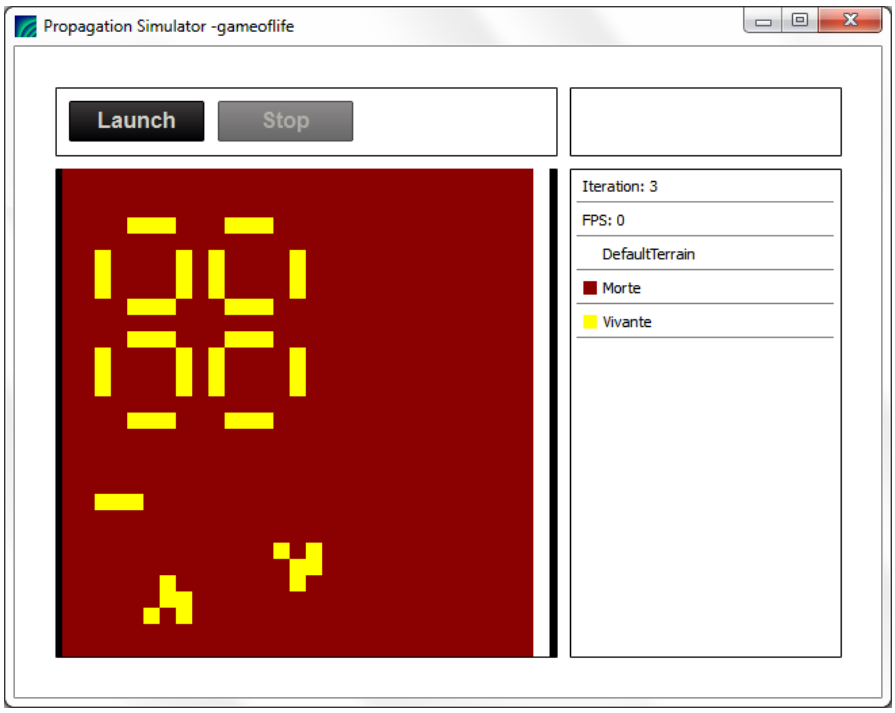


FIGURE 2.5: Illustration de la simulation d'exemple du jeu de la vie à la fin de la release 3.

Chapitre 3

Etude de faisabilité

Lorsque le groupe est arrivé à un consensus sur le sujet du projet à savoir la réalisation d'un simulateur de propagation, un bon nombre d'éléments étaient encore en suspens :

- Le choix du langage.
- La définition précise de ce que le programme simulera.
- La définition des limites du simulateur, à savoir ce qu'il ne permettra pas de simuler.
- Les fonctionnalités de l'interface graphique.
- Le choix de la librairie graphique.

Cette section explicite les choix que nous avons effectués pour mettre au clair ces éléments.

3.1 Choix des technologies

Le choix du langage s'est porté sur Python, pour les raisons suivantes :

- Le langage est très utilisé dans le domaine scientifique, duquel notre application fait partie.
- L'envie commune d'explorer d'autres technologies - habituellement, nous utilisons Java.
- La connaissance du langage par certains membres du groupe, permettant de vérifier la pertinence de son utilisation dans le projet.
- La librairie standard très riche ainsi que bon nombre de librairies tierces permettant de ne pas avoir à réinventer la roue.
- Sa syntaxe concise et extrêmement claire augmentant à la fois la productivité et la lisibilité du code.
- La relative simplicité du langage, qui le rend très accessible.
- Sa portabilité.

Ce choix était une première étape dans la considération des fonctionnalités du programme. Notamment, il a ouvert la porte à deux opportunités : la première étant la possibilité d'utiliser la librairie Qt, très répandue et extrêmement puissante bien que relativement aisée à prendre en main, et bien intégrée à l'environnement Python grâce aux *bindings* PySide et PyQt. La seconde opportunité est la possibilité de paramétrer l'application par le biais de fichiers de configuration ou même de scripts Python. En effet, la nature dynamique du langage se prête extrêmement bien à ce genre de traitements.

La décision d'utiliser Qt ne s'est pas seulement justifiée par l'usage de Python mais aussi car l'interface prévue pour l'application, bien que relativement simple, nécessiterait l'usage d'un canevas graphique suffisamment performant.

3.2 Analyse initiale

De là, un *brainstorming* a été effectué pour définir ce que le simulateur ferait et comment il le ferait. Il a été décidé que notre simulateur chargerait des fichiers de configuration qui caractérisent les règles et les entités de la simulation pour les transformer en classes à la volée que nous utiliserons par la suite dans le moteur de simulation. L'interface graphique, affichera simplement les entités de la simulation à l'écran sur une grille.

3.3 Prototypes

A partir de cette étape de décision, nous avons commencé une phase de prototypage afin de vérifier si la réalisation des composants clés du projet étaient faisables [1] tant au niveau de nos connaissances, du temps et de l'état de la technologie.

3.3.1 Parsing des fichiers utilisateurs

Le but de ce prototype était de créer une syntaxe simple et complète pour la création des fichiers de configuration utilisateur.

Critères à satisfaire

- L'utilisateur doit pouvoir configurer une simulation uniquement par le biais des scripts JSON, sans connaissance de python.
- Les scripts doivent permettre une configuration la plus avancée possible.
- La syntaxe utilisée doit être cohérente et simple à prendre en main.

Conclusion

Le prototype s'est réalisé correctement. Différents problèmes ont été relevés, comme la nécessité de faire de la vérification syntaxique au niveau des noms fournis, de créer un système de levée d'exception qui permette de tracer correctement les problèmes de chargement de scripts.
Cela n'a entraîné aucune modification des technologies à utiliser et nous a permis de créer une ébauche du manuel utilisateur pour la syntaxe d'écriture.

3.3.2 Construction d'entités

Après des recherches sur le sujet, il est apparu évident d'utiliser les métaclasses Python afin de créer des classes d'entités qui donneront à l'utilisateur les instances correspondant à leurs configurations.

La métaclasse est une classe qui crée une autre classe. Nous avons donc créé un squelette d'entité qui accueillera chacune des configurations. Au final, le procédé revient à ceci :

1. Prendre le squelette d'entité (`AnonymousEntity`).
2. Le dupliquer.
3. Lui ajouter des propriétés.
4. Lui ajouter des méthodes.

Les métaclasses ont uniquement le pouvoir d'ajouter des éléments statiques à la classe, étant donné qu'elles construisent les classes. Ceci nous a posé des problèmes lors de l'ajout d'attributs aux classes. Nous avons donc changé de méthode, en créant une propriété par attribut.

1. Typiquement, le chargement des fichiers de configuration.

Les propriétés en Python peuvent être vues comme des accesseurs *get*() et *set*() pour un attribut. Elles nous permettent ainsi de déclarer au niveau de la classe qu'un attribut répondant à un nom donné existe dans l'instance. La méthode get se devra d'ajouter l'attribut dans l'instance au moment du premier *get*() effectué, ce qui nous permet au final d'influencer les instances finales au niveau de la métaclasse.

Ce prototype nous a permis de concevoir par la suite le reste de l'application, car nous avons désormais un outil pour construire et préparer les classes de la simulation dynamiquement.

3.3.3 Threads et processus

Ce prototype a été réalisé dans le but de tester les possibilités de traiter les threads et processus en Python, ainsi que de vérifier leurs comportements.

Nous avons pu constater que Python mettait à disposition des outils pour la gestion des ressources concurrentes, tant pour les threads que pour les processus. Cet aspect ne poserait donc pas de problème.

Néanmoins nous avons pu constater quelques faiblesses. En effet, les threads se sont révélés peut efficaces de part le fait qu'il ne s'exécutent pas en parallèle, mais en alternance, dans le cadre de processeurs multicœurs. Ce qui n'est pas le cas des processus, qui eux sont bel et bien parallèles. En revanche l'échange d'informations entre différents processus nécessite un certain temps.

Selon ces constatations, nous avons opté pour l'usage de threads dans le cadre de l'affichage graphique. En effet les données sont plus rapidement transmises entre l'interface graphique et le simulateur. D'autre part, ces deux entités n'ont pas besoin de s'exécuter en vrai parallélisme, de part le fait qu'il n'y a que peut d'interactions avec l'utilisateur et que l'affichage ne peut se faire que lorsqu'une itération a été complètement calculée.

3.3.4 Interface graphique

Ce prototype avait pour but de vérifier que nous pourrions utiliser le *framework*[2] Qt avec le langage Python en utilisant la librairie PySide qui effectue les liens nécessaires pour pouvoir utiliser les fonctionnalités de Qt[3] en Python.

Il existe actuellement deux librairies majeures qui permettent d'utiliser Qt en Python : PyQt et PySide. Notre choix s'est porté initialement pour PySide pour plusieurs raisons :

- Respecte plus la philosophie de codage Python.
- PySide utilise une licence plus libre que PyQt pour la commercialisation.
- Contenu équivalent à PyQt.

Lors de la phase de recherche d'informations sur les librairies graphiques, nous avons décidé d'essayer la solution graphique QtQuick pour réaliser l'interface graphique au lieu de coder les fenêtre en utilisant les fonctionnalités de Qt. Ce choix s'est effectué suite à notre expérience de développement avec Swing de chez Java. Nous nous sommes rendu compte qu'il était laborieux de réaliser une interface avec Swing. Or, réaliser une interface avec du code Qt reviendrai à faire la même chose qu'en Java. QtQuick nous offrait une solution alternative et facile à réaliser car le language utilisé est le QML qui est un langage de description développé expressément pour réaliser des interfaces graphiques. QtQuick nous offre aussi un logiciel permettant de dessiner notre interface. Dans ce projet, nous étions partis sur une interface avec une seule fenêtre. Le choix de cette technologie se révélait donc particulièrement pertinente étant donné notre expérience inexistante avec cette technologie. Utiliser cette technologie avait aussi l'avantage de renforcer le modèle MVC[4] de notre logiciel en forçant la séparation de la vue du reste du programme avec l'utilisant un langage de programmation différent.

2. Terme anglais pour définir une solution logicielle fournissant des librairies, des programmes et un environnement
3. Qt est écrit nativement en C++
4. Pour Modèle-Vue-Contrôleur

Pour s'assurer que Qt et QtQuick suffiraient à nos besoins, nous avons réaliser plusieurs prototypes qui évaluaient séparément certains aspects capitaux de l'interface graphique comme la génération d'image et les listes. A la fin de la période d'analyse, ces prototypes ont étés amalgamés pour réaliser un prototype unique qui serait la base de l'interface utilisée pour le simulateur.

En conclusion, le prototypage de l'interface graphique a permis d'avoir une base sur laquelle travailler et aussi de nous permettre d'apprendre de nouvelles technologies utilisées dans l'industrie.

3.3.5 Déploiement

Un point de l'analyse était consacré à évaluer une méthode de déploiement (chercher une sorte d'emballage) pour pouvoir livrer le logiciel sous une forme agréable pour l'utilisation.

Critères à satisfaire

Les critères suivants étaient obligatoirement à satisfaire par la méthode de déploiement :

- L'utilisateur ne doit pas faire une installation de manière manuelle. Par exemple installer manuellement un environnement Python ou des bibliothèques de Python.
- Le logiciel doit être simple à démarrer. Autrement dit, un fichier exécutable (pour double-cliquer sous Windows) qui lance l'application.
- La méthode de livraison doit supporter des bibliothèques non-standard, en particulier :
  - QT 4.8 pour l'interface graphique
  - PySide pour la liaison entre Python et QT
  - Noise pour la génération de la carte
  - Numpy pour la meilleur gestion des tableaux et des fonctions mathématiques

Les critères suivants étaient facultatifs :

- Le client n'a rien à installer (même pas de manière automatisée). Il peut directement lancer le logiciel. Tout les composants nécessaires comme des bibliothèques supplémentaires ou des DLL sous Windows sont déjà intégrés dans le logiciel.
  - Le logiciel est livré sous la forme d'un dossier qui contient le fichier exécutable et toutes les dépendances.
  - Le logiciel est livré sous la forme d'un seul fichier exécutable qui contient toutes les dépendances.

PyInstaller

`PyInstaller` [5] était le premier candidat qui à été étudié en détail. Comme `PyInstaller` satisfaisait tous les critères, même les facultatifs, il a été choisi comme méthode de déploiement pour le `Propagation Simulator`. Par conséquent, nous avons renoncé à évaluer d'autre méthodes de déploiements.

3.3.6 Génération de bruit

Ce prototype avait pour but de tester plusieurs choses à la fois. Il consiste à afficher un bruit de Perlin animé dans une fenêtre graphique Qt.

Les aspects testés par ce prototype sont les suivants :

- Création d'un canevas graphique avec Qt, permettant d'afficher une grille de pixels colorés.
- Méthode efficace pour redessiner le canevas graphique.
- Création d'une carte aléatoire avec une bruit de Perlin, à l'aide du module `noise` et paramétrage du bruit généré.

5. http ://www.pyinstaller

Pour réaliser l'animation, on utilise un bruit en 3 dimensions plutôt que 2, afin d'ajouter la dimension temporelle pour l'animation. Les paramètres sont les mêmes, à l'exception du paramètre `base` qui n'existe que pour la dimension 2 et permet de spécifier un "seed" pour permettre la génération reproductible des nombres aléatoires.

Pour correspondre aux besoins de l'application, plutôt que d'utiliser des nombres flottants tels que générés par le bruit, on transforme ces valeurs en "clusters", soit en nombre entiers correspondant chacun à un type de terrain donné. Ici le type de terrain représente juste une couleur RGB 24 bits.

On stocke ces valeurs dans une matrice à deux dimensions. Les tableaux natifs du langage Python n'étant pas pratiques pour manipuler ce genre de matrices, on utilise une matrice de la librairie Numpy pour cela. Cette librairie facilite la manipulation des tableaux multidimensionnels, tout au augmentant significativement les performances de ces manipulations en laissant la possibilité d'effectuer certaines optimisations comme, entre autres, spécifier un typage statique à la création de la matrice.

La manipulation de l'image et son affichage à une fréquence élevée (plus de 10 images/seconde) était la partie critique de ce prototype. En effet, la première version du programme mettait plus de 2 secondes à recalculer la matrice et à l'afficher. Il a donc fallu effectuer un certain nombre d'optimisations pour vérifier la faisabilité d'un rafraîchissement efficace de l'image.

Du point de vue du calcul de la matrice, les changements les plus importants en terme d'amélioration des performances étaient d'utiliser le typage statique de la matrice, et de minimiser les calculs effectués à l'intérieur des boucles.

Pour ce qui est du rafraîchissement de l'image, les recherches de documentation sur internet ont montré que la façon la plus efficace de le faire était d'utiliser un `QImage` (optimisé pour les entrées/sorties) pour effectuer la mise à jour des pixels, et de charger cette image dans un `QPixmap` qui lui est optimisé pour l'affichage.

Il a été considéré d'utiliser une fenêtre QML pour le canevas graphique, mais les possibilités offertes par les version de QtQuick inférieures à la version 2.0 sont limitées et inefficaces pour ce genre d'opérations.

D'autres optimisations ont été faites : rafraîchir l'affichage à chaque changement d'une valeur dans la matrice plutôt que de reparcourir entièrement la matrice après sa mise à jour, et effectuer le rafraîchissement des pixels uniquement pour les pixels qui doivent changer, plutôt que pour l'image entière.

Après ces optimisations, les performances obtenues sont suffisantes pour les besoins de l'application, même si elles ne sont pas encore optimales, si on compare à un canevas OpenGL programmé en C, par exemple.

En conclusion, les objectifs du prototype sont atteints. Le module `noise` répond aux besoins pour la création d'une carte aléatoire, et le canevas graphique de Qt permet un affichage et un rafraîchissement suffisamment rapide. De plus, la librairie Numpy a pu être essayée et son intérêt pour l'implémentation de la simulation est confirmé.

Script de création des fichiers de configuration

Pendant la phase d'analyse, nous avons eu l'idée de proposer un outil sous forme de script en console, permettant de créer, à partir de modèles (templates), tous les fichiers de configuration nécessaires à l'utilisateur pour définir les paramètres de sa simulation. Ce script permettrait de créer une simulation, puis d'y ajouter des entités, et leur ajouter des actions et interactions. Cela créerait simplement des fichiers de configurations comportant déjà la structure et les champs nécessaires que l'utilisateur n'aurait plus qu'à remplir.

Cette fonctionnalité a été ajoutée aux prototypes en tant que fonctionnalité non prioritaire.

Un premier prototype a été réalisé à l'aide notamment du module `argparse` de la librairie standard de Python. L'outil possède une syntaxe puissante et extensible, un affichage de l'aide similaire aux

commandes UNIX et généré automatiquement, et plusieurs paramètres. La création des fichiers et des dossiers à partir de modèles est également fonctionnelle, grâce à un système de templating basique également fourni par la librairie standard.

Gestion des attributs d'entités

Nous avons conçu ce prototype afin de nous familiariser avec la gestion d'ajout d'attributs et des différents moyens d'ajout dynamique dans une classe.

Ce prototype s'est avéré utile pour appréhender les moyens d'ajouter certains attributs et en interdire d'autres.

Nous avons pu ainsi créer une classe dans laquelle nous autorisons l'ajout d'attributs, et une autre dans laquelle nous interdisons toute création.

Nous pouvons donc grâce à ce prototype :

- Ajouter des attributs à une instance.
- Permettre des alias (deux noms d'attributs représentent le même contenu).
- Gérer un dictionnaire d'attributs indépendants.

3.3.7 Conclusion

Tous les prototypes sont arrivés à une solution pour le défi qu'ils devaient résoudre. Le projet a été classé comme faisable. Les technologies suivantes ont été retenues pour sa réalisation :

- Python 2.7
- PySide 1.2.1 et Qt 4.8 (Utilisé à travers PySide)
- QtQuick 1.1
- PyInstaller 2.1
- noise 1.2.1
- nose 1.3.0
- numpy 1.7.1
- path.py 0.5.6

3.4 Analyse fonctionnelle

A la suite des prototypes, le programme a été découpé en fonctionnalités. L'utilisateur sera capable de :

- Configurer une simulation par le biais de fichiers.
- Définir les statistiques qui seront enregistrées par la simulation.
- Lancer une simulation en mode graphique.
  - Mettre en pause et reprendre la simulation.
  - Arrêter la simulation.
  - Exporter les données de simulation.
- Lancer une simulation en mode texte.

Chapitre 4

Conception

Ce chapitre se penche sur la démarche d'analyse et de conception effectuée tout au long du projet lors du développement des fonctionnalités.

4.1 Cas d'utilisation

Ce diagramme des cas d'utilisation montre les fonctionnalités proposées par *Propagation Simulator*. La conception ainsi que la réalisation de ces fonctionnalités seront décrites en détail dans les pages suivantes.

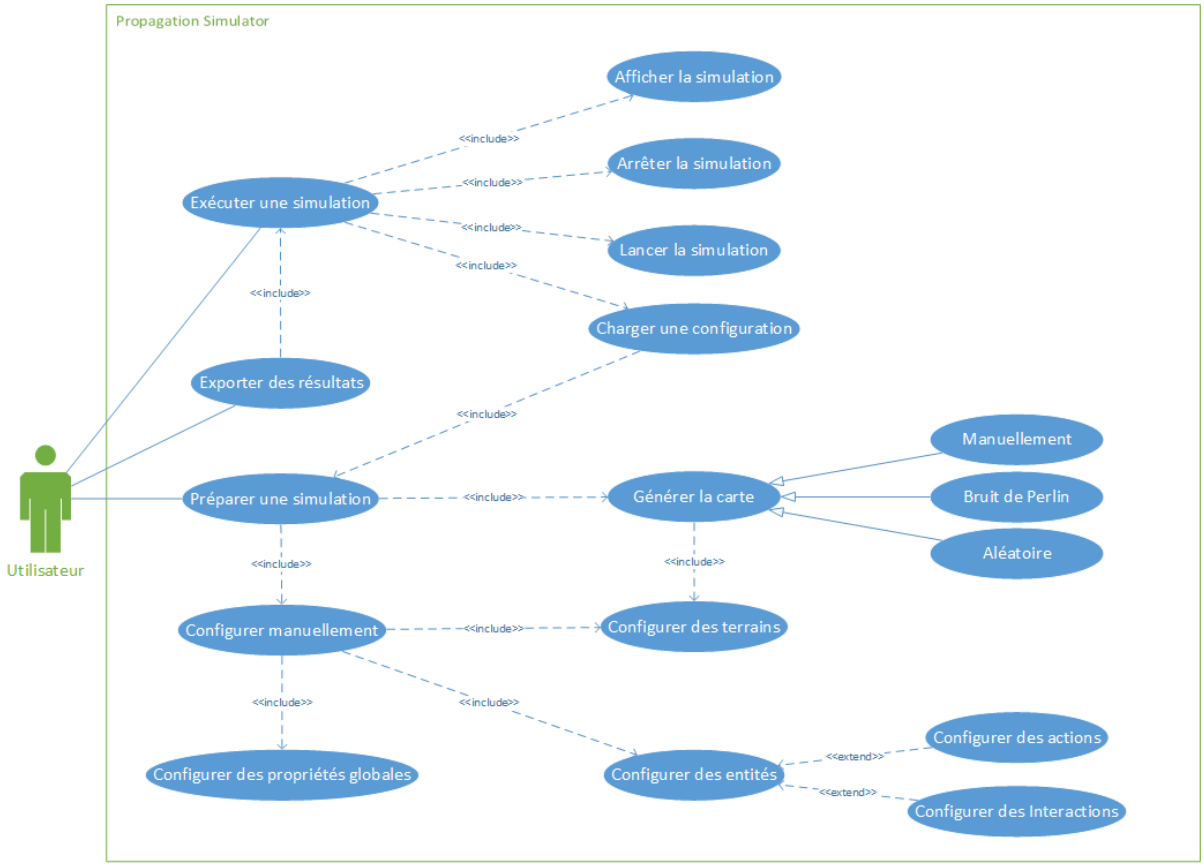


FIGURE 4.1: Cas d'utilisation

4.2 Vision d'ensemble

La simulation doit se dérouler en plusieurs phases :

1. Lecture des fichiers utilisateurs pour réaliser la simulation.
2. Construction des éléments nécessaires à la simulation.
3. Exécuter la simulation avec ou sans affichage graphique.

Dans le cas d'un affichage graphique, la fenêtre de l'application doit être affichée pendant la préparation, ce qui permet d'indiquer à l'utilisateur que le programme s'est bien lancé. Pour ce faire, l'interface graphique doit être indépendante et ne faire que demander les informations à afficher au simulateur, ce qui nous amène à la vision globale suivante :

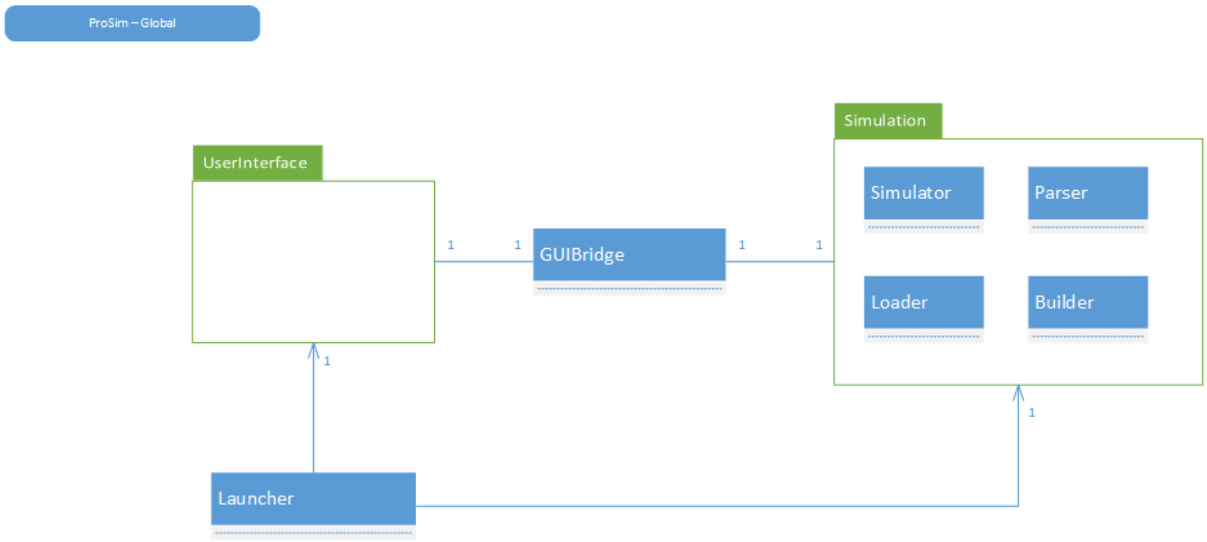


FIGURE 4.2: Schéma général

Il nous faut donc développer deux parties distinctes et aptes à communiquer l'une avec l'autre pour des échanges d'informations. La "passerelle" à utiliser sera détaillée dans la partie du simulateur (c.f. 4.6 Simulateur).

La simulation va devoir utiliser des entités au comportement régi par les scripts de l'utilisateur. Les entités comportent également des attributs obligatoires ou définis par l'utilisateur pour être utilisés par ses scripts.

Il nous faut donc établir une politique de traitement concernant les événements régissant les entités, ainsi que la gestion des attributs.

4.2.1 Les événements

Les simulations reposant sur des entités au comportement défini par l'utilisateur, nous avons distingué deux catégories d'événements, à savoir les actions et les interactions.

Les actions

Il s'agit d'éléments régissant le comportement de base d'une entité, comme par exemple le fait de se déplacer sur la carte de simulation. Les actions doivent être effectuées à chaque itération de la simulation (ou "pas" de simulation). Si une action devait être conditionnelle, il appartient à l'utilisateur d'écrire le test voulu dans le script de l'action.

Une action étant le comportement de base d'une entité, nous avons choisi de lier chaque action à type de terrain. De la sorte, l'utilisateur bénéficie d'un test automatique permettant de configurer une entité pour agir en fonction de son environnement extérieur, à savoir le terrain.

Les interactions

Les interactions sont des "actions" conditionnelles, pour lesquelles la condition d'exécution est la présence d'une autre entité située à une distance inférieure à une certaine valeur. Il n'est pas nécessaire d'établir des interactions, elles ne seront effectuées que si elles sont déclarées par l'utilisateur.

Une interaction survient pour deux entités, dont les types sont spécifiés par l'utilisateur. De plus, pour deux entités A et B, seule l'interaction A avec B doit être effectuée et non pas un doublon B avec A. Cela implique que le simulateur doit assurer l'unicité des associations d'entités dans le cadre des interactions. En outre, si l'entité A n'est pas de même type que B, l'appel à l'interaction A avec B doit se faire sur le même script que l'appel à l'interaction B avec A.

4.2.2 Les attributs

Chaque entité manipulée lors de la simulation comporte des attributs (variables) définis par les scripts de l'utilisateur. Ces attributs pouvant être utilisés et modifiés au cours de l'exécution d'actions ou d'interactions, il a fallu établir une politique de gestion quant à la valeur des attributs.

En effet, pour une itération de la simulation, l'état des variables doit être cohérent afin que chaque action ou interaction se base sur la même valeur, à savoir celle du début de l'itération. A savoir que si un attribut est modifié par un script, le script s'exécutant après doit pouvoir utiliser la même valeur originale que le script précédent. Cet aspect permet de simuler l'exécution en "parallèle" de chaque script, indépendamment de leur ordre d'appel.

Afin d'assurer cette cohérence, nous avons dupliqué les attributs. La consultation d'un attribut retourne donc la valeur du début de l'itération, à savoir l'état courant. La modification d'un attribut aura pour effet de n'altérer que son "doublon" qui représente son état futur. La mise en place des nouvelles valeurs ne devra se faire qu'en fin d'itération par le simulateur.

Évidemment, si plusieurs scripts modifient un même attribut, l'état futur de cet attribut ne pourra valoir que le résultat d'un seul script. Un script appliqué au début de l'itération sera donc toujours "masqué" par un script appelé après. Afin de gérer cet aspect, nous avons choisi que lors d'une modification d'un attribut, la première à survenir serait toujours effectuée et que les suivantes se substitueraient aléatoirement.

En effet dès lors que qu'un attribut a été modifié, toute modification ultérieure n'aura lieu qu'aléatoirement.

Nous avons choisi d'écraser la valeur qu'une fois sur deux, à savoir 50% de chance. Cette approche implique que les derniers scripts à être exécutés auront plus de chance de voir leur valeur affectée à l'attribut que les premiers.

Cette approche n'est de loin pas parfaite d'un point de vu de l'équiprobabilité entre les modifications. Cependant, assurer l'équiprobabilité nécessiterait de compter le nombre de modifications de chaque attribut, et donc d'ajouter à chaque attribut une variable de type compteur supplémentaire. Dans le but de modérer l'impact d'une entité en mémoire, nous avons préféré conserver une approche non équitable, mais non coûteuse.

4.3 Analyse syntaxique

Nous avons implémenté plusieurs parsers (ou analyse syntaxique), pour lire et gérer les différentes erreurs des scripts JSON de l'utilisateur.

Les différents parsers ont des rôles distincts et sont utilisés dans un ordre logique prédéfini.

En effet, l'utilisateur doit avoir des possibilités de configuration à tous les niveaux de la simulation, nous devions donc gérer la lecture des fichiers :

start.json : Pour configurer le lancement de la simulation.
- Configuration de la carte (type et paramètres de la génération de carte).
- Les éléments pour garantir la reproductibilité.
- Le nom de la simulation à lancer.
- Les paramètres d'export.
- Les paramètres graphiques (cadence d'affichage, nombre de simulation graphique etc.).

terrain.json : Pour configurer les terrains, leurs noms et couleurs.

attributes.json : Pour les attributs d'une entité.
- La couleur de l'entité.
- Le nom de l'entité au singulier et pluriel.

actions.json : Les actions de l'entité.
- La fonction de positionnement des entités sur la carte.
- La fonction d'initialisation de chaque instance d'entité.
- Les actions de l'utilisateur pour les types de terrains donnés.

interactions/*.json : Tous les scripts d'interactions entre deux entités.

Tous ces différents script JSON devront être lus, puis interprétés afin de charger les scripts Python des différentes méthodes, les paramétrer etc.

Certains scripts, comme le *start.json* demandent de renseigner une valeur cohérente pour certaines clés afin de pouvoir lancer la simulation. D'autres paramètres sont optionnels, c'est pour cela qu'une découpe en plusieurs parsers a été nécessaire.

Nous avons donc pris le parti de définir quatre parsers différents, afin de gérer le plus finement possible les différentes règles et grammaires.

4.4 Loader

L'objectif du Loader est de préparer les différentes classes d'entité selon les différents scripts de l'utilisateur. Ainsi, nous traitons la syntaxe et le parsing de la majorité des fichiers dans cette partie.

4.4.1 Préparation

4.4.2 Syntaxe des fichiers de configuration de l'utilisateur

L'objectif principal annoncé pour la syntaxe était de permettre une courbe d'apprentissage la plus douce possible.

Ainsi l'utilisateur pourrait configurer, au début, uniquement des petits détails en s'appuyant sur une simulation par défaut. Petit à petit, il pourrait introduire des lignes Python à exécuter, avant de faire le pas et de créer de vrais fichiers de scripts Python permettant une configuration complète.

La syntaxe des attributs

La syntaxe pour les attributs est très simple. Il s'agit au final d'un fichier de déclaration contenant les différents attributs accessibles par l'entité lors de la simulation. Ces attributs suivent les normes de déclaration JSON, tout en autorisant les commentaires courant en programmation.

Les noms des attributs sont vérifiés afin qu'il puisse être utilisés comme des identifieurs Python valides. Nous avertissons ainsi l'utilisateur des éventuels problèmes dès le chargement du script.

La syntaxe des actions

Cette syntaxe a été compliquée à mettre en place, car il s'agit de fournir une syntaxe polyvalente et cohérente en même temps. De ce fait, nous avons dû effectuer des changements de conception.

Ainsi l'utilisateur peut configurer ses actions en définissant :

- Une action par le biais d'un script par défaut.
- Une action par le biais d'un script utilisateur (lié de manière absolue ou relative).
- Une action définie par le biais d'un script "inline" au sein d'un fichier JSON. Cette définition demande deux fonctions, à savoir $is\_performed$ et $action\_performed$. Chacune de ces fonctions peut être définie ainsi :
  - En liant des noms de scripts "fonctions" par défaut, et préciser leurs paramètres.
  - En liant des scripts "fonctions" utilisateur (lié de manière absolue ou relative).
  - En écrivant des scripts Python "inline" directement dans le fichier JSON qui seront évalués par des fonctions pré-déterminées :
    - Pour $is\_performed$: nous appliquons un AND sur le résultat des expressions Python données.
    - Pour $action\_performed$: nous exécutons les expressions les unes après les autres.

Par ce biais, l'utilisateur peut commencer par utiliser uniquement les scripts d'actions et les fonctions par défaut, afin de modifier le comportement de ces entités simplement. Ainsi il est très facile de modifier le mouvement des entités, tout en gardant les comportements par défaut pour la simulation.

Diagramme de classes

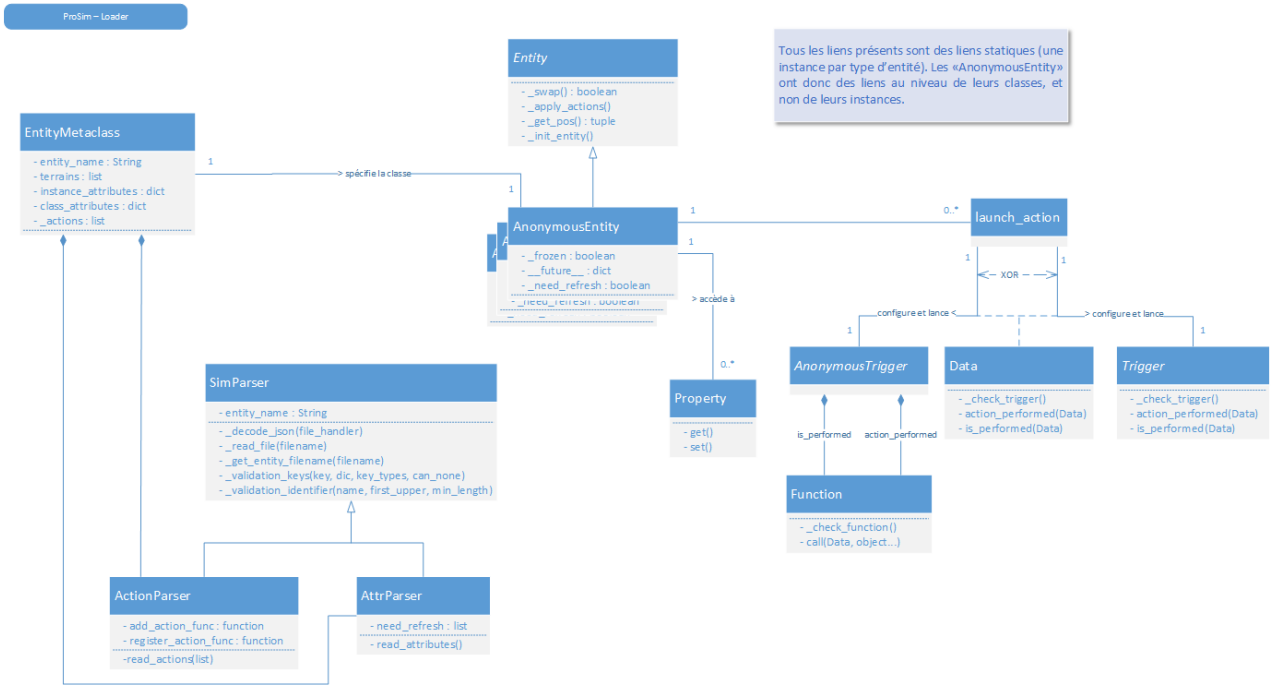


FIGURE 4.3: Conception du *Loader*

Cette conception permet une création définitive des entités (spécification des *AnonymousEntity*). Nous pourrons donc potentiellement libérer de la mémoire et détruire les métaclasses et les parseurs pour ne garder que les objets et instances qui seront effectivement utilisés.

4.5 Builder

Le *Builder* s'occupe de la préparation de la simulation. Il demande au *Parser* et au `Loader` d'extraire les configurations des fichiers JSON. Le *Simulator*, l'unité centrale du simulateur, a besoin du *Builder* pour accéder aux configurations sans se soucier du traitement des fichiers de configuration.

Les tâches principales du *Builder* :

- Créer les instances des entités.
- Récupérer les différents types de terrain.
- Générer la carte de la simulation.
- Réunir toutes les configurations dans un objet *Params*.

4.5.1 Génération de la carte

Une tâche du *Builder* consiste à construire la carte à partir de la configuration de l'utilisateur. Pour ce faire, il existe trois méthodes qui seront expliquées par la suite.

Aléatoire

La méthode purement aléatoire génère automatiquement une carte en alternant des différents terrains de façon uniforme.

Bruit de Perlin

La méthode à l'aide du *bruit de Perlin*[1] permet de générer une carte plus réaliste que la méthode précédente. Le *bruit de Perlin* est une texture procédurale pseudo-aléatoire. Le but de cet algorithme consiste à augmenter le réalisme, par exemple dans la génération des paysages. En Python le module tiers *noise* met à disposition les fonctions nécessaires.

Carte personnalisée

Cette méthode permet à l'utilisateur de créer manuellement sa propre carte.

Deux approches pour la modélisation de la carte ont été évaluées : l'approche *bitmap* consistant à spécifier la valeur de chaque case de la matrice, et l'approche *vectorielle* où l'on superpose des formes définies par leur type (rectangle, ellipse...), leur position et leurs dimensions. Dans les deux cas, on arrive à définir pour chaque case le type de terrain voulu.

La méthode par dessin de formes (des rectangles et des ellipses) a été choisie. Cette approche permet à l'utilisateur de créer plus facilement des champs d'un certain type de terrain. De plus, la possibilité de modifier une seule case reste possible en définissant un rectangle de taille $1x1$.

4.6 Simulateur

Le simulateur est au centre du programme, son rôle est d'utiliser les outils à disposition pour mener à bien la simulation de l'utilisateur. Il doit donc gérer les informations de la simulation de l'utilisateur pour produire un résultat, avec la possibilité d'obtenir un affichage graphique minimaliste.

Il s'agit donc, pour cette partie, de mettre à disposition de l'interface graphique des outils permettant d'afficher le monde de la simulation ainsi que ses entités. Il faut également permettre à l'interface un contrôle minimal, à savoir le fait de démarrer la simulation, de la mettre en pause ou de l'arrêter définitivement.

1. http ://www.noisemachine.com/talk1/index.html

Le simulateur doit également mettre à disposition de l'utilisateur des outils afin d'interroger le moteur sur le monde de la simulation, comme par exemple d'obtenir le type de terrain d'un emplacement spécifique.

4.6.1 Diagramme de classes

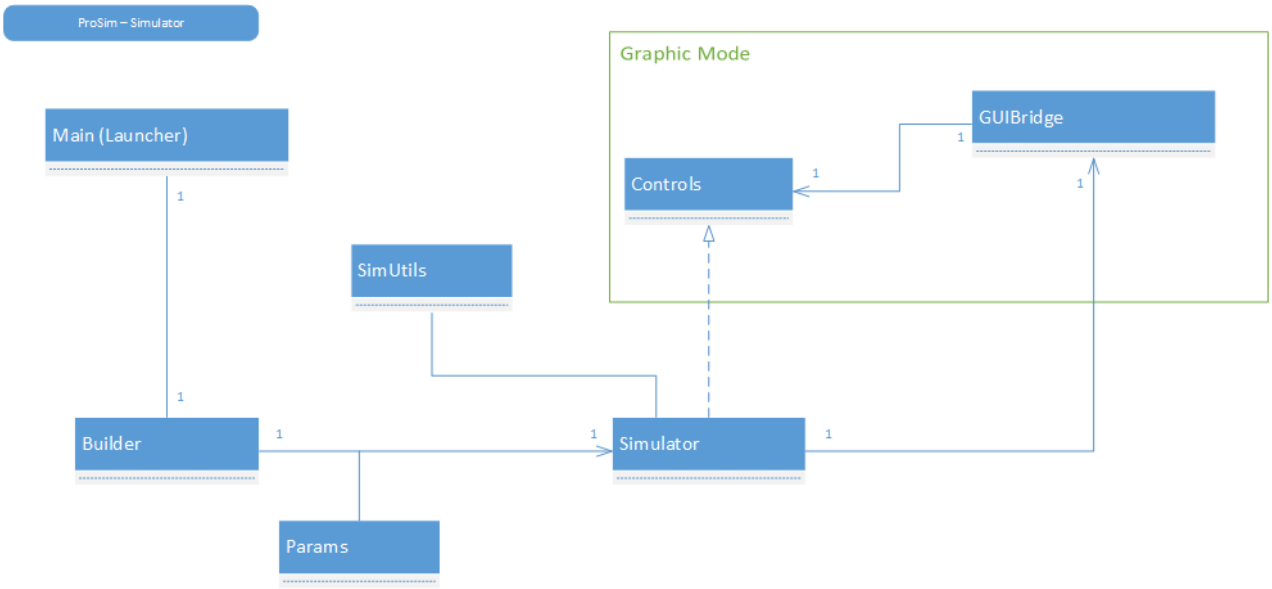


FIGURE 4.4: Conception du simulateur

Builder

Il s'agit de la partie générant le monde et les entités. Le simulateur n'utilisera que les informations données dans le conteneur `Params`. Pour plus d'informations, se référer à la section dédiée 4.5 Builder.

Params

`Params` est un conteneur, construit par le `Builder`. Il s'agit des informations nécessaires à la réalisation de la simulation de l'utilisateur. En plus de contenir le monde et ses entités, les informations suivantes sont requises :

- Le nombre d'itérations à réaliser pour la simulation.
- S'il faut ou non effectuer un affichage graphique des itérations, et si oui, la fréquence d'affichage.

SimUtils

Cet élément met à disposition de l'utilisateur des outils pour récupérer des informations sur le monde de la simulation. Par exemple, il est question de mettre à disposition les éléments suivants :

- Obtention du type de terrain d'un point précis du monde.
- Déterminer si un point donné est compris dans les limites du monde.
- Obtention des entités présentes à un point précis de la carte.
- Et divers autres outils à disposition de l'utilisateur pour ses scripts qui seront ajoutés au fil du développement.

Ces éléments ont pour but de permettre à l'utilisateur de préparer des comportements avancés. Par exemple cela rend possible de vérifier le type de terrain de destination avant le déplacement d'une entité.

Le mode graphique

Afin de permettre un affichage des itérations effectuées par le simulateur, il est nécessaire de réaliser une "passerelle" entre le simulateur et l'interface graphique.

Controls

Il s'agit de définir les actions possibles concernant le maniement du simulateur, à savoir :

- Démarrer la simulation.
- Mettre en pause et reprendre la simulation.
- Mettre fin à la simulation.

GUIBridge

Ce composant est utilisé comme passerelle entre l'interface graphique et le simulateur afin que ces deux éléments puissent se synchroniser. En effet, le simulateur doit cesser temporairement son activité le temps que l'interface affiche l'itération, de sorte que les données ne soient pas modifiées en cours d'affichage. Pour ce faire nous avons décidé d'utiliser un système de passage de témoin pour mettre en pause l'affichage le temps de préparer les données pour l'itération suivante et, réciproquement, de mettre en pause le simulateur le temps d'afficher les données. De la sorte, nous nous assurons d'un état cohérent des données de la simulation.

Simulator

Il s'agit pour cet élément d'utiliser les bons outils pour mener à bien la simulation. Son rôle consiste donc à réaliser chaque itération de la simulation jusqu'à son terme.

Dans le cas où l'utilisateur souhaite un affichage graphique, il a la charge de garder trace des toutes les entités modifiées afin de les transmettre à l'interface graphique. De plus il doit s'assurer de s'interrompre si l'interface demande la mise en pause de la simulation.

4.7 Interface utilisateur

Dès que la phase de prototype fût terminée, le développement de la structure de l'interface graphique de base a immédiatement commencé afin que le reste du simulateur sache comment se connecter avec l'interface.

Il a été décidé en collaboration entre le responsable interface graphique et celui de développement comment le transfert de donnée allait être fait afin que les deux parties puissent commencer à être programmées séparément.

L'interface devait fournir un point d'entrée et de sortie au simulateur ainsi qu'un autre point d'entrée et sortie pour la partie QML de l'interface qui gère uniquement l'affichage.

L'interface se devait également de gérer le rendu graphique de la simulation et la gestion des signaux que l'utilisateur pourrait envoyer.

4.7.1 Diagramme de classes

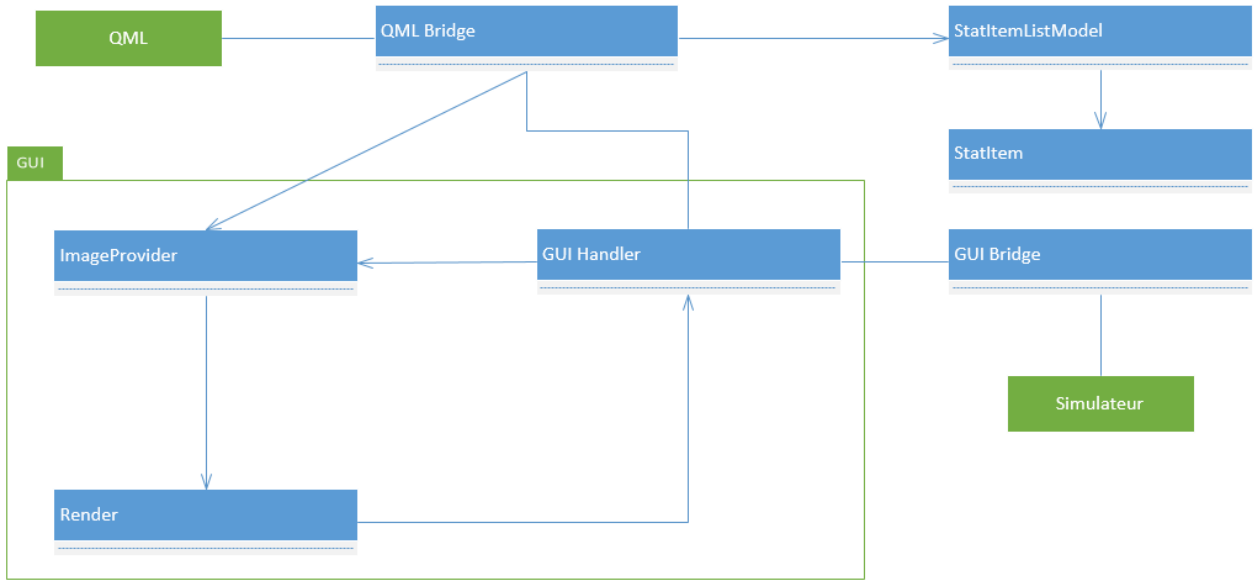


FIGURE 4.5: Diagramme de classes de la gestion de l'interface

Les parties QML et Simulateur sont reléguées à de simples rectangles verts car il n'est pas du ressort de la partie interface de les détailler ici. La classe *Gui Handler* est la classe principale de l'interface. C'est cette classe qui initialise toutes les autres classes vitales de l'interface. Les deux points d'entrée et sortie mentionnés avant sont les classes *QML Bridge* et *GUI Bridge*.

Le *QML Bridge* relaie les informations entre les interactions utilisateurs et le gestionnaire graphique. Cet échange va dans les deux sens.

4.7.2 Interface homme-machine

Le lien entre le programme et l'utilisateur est effectué par une interface graphique décomposée en quatre parties principales. Cette interface graphique est définie par la partie *QML* spécifiée dans la figure 4.5. Cette partie ne possède par de diagramme d'analyse car la conception de l'interface graphique repose sur une syntaxe de description basée sur le framework *QtQuick* qui fait partie de la technologie QML. Voici à quoi ressemble l'interface :

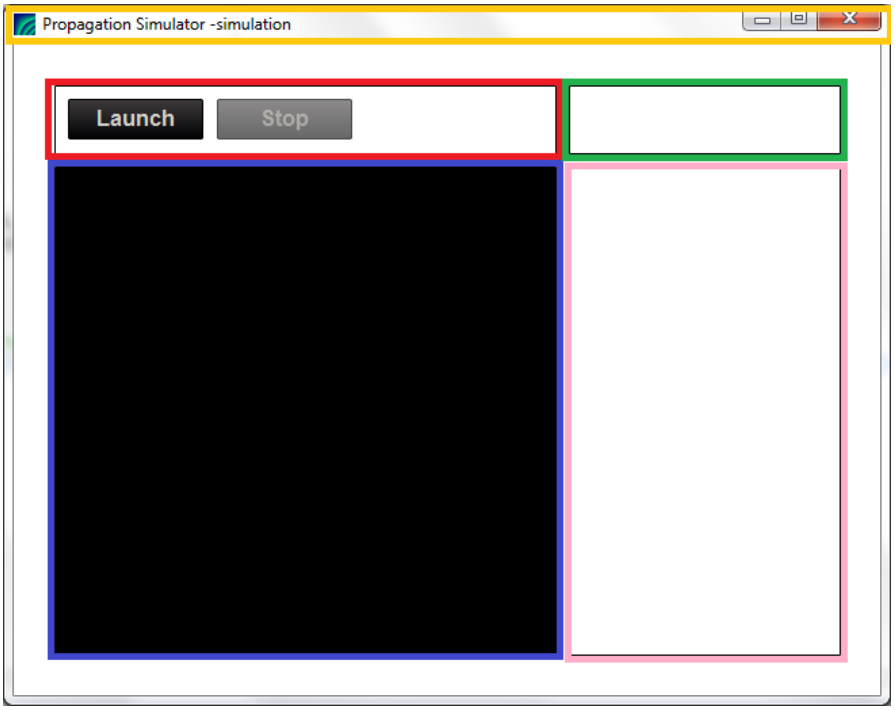


FIGURE 4.6: Conception de l'interface utilisateur

Nous pouvons voir qu'à l'intérieur de la fenêtre, nous avons quatre rectangles de couleurs différentes. Chacun englobe une partie spécifique de l'interface. La zone définie par le rectangle rouge comporte tous les points d'interaction entre l'utilisateur et le programme. C'est à l'aide de ces boutons que l'utilisateur peut modifier le flux d'exécution du programme.

La zone définie par le rectangle vert sert à rendre l'interface cohérente est a été prévue pour afficher des informations supplémentaires à l'utilisateur dans le cadre d'un développement ultérieur.

La zone bleue contient le rendu graphique de la simulation. C'est à cet endroit que l'utilisateur pourra observer le déroulement de la simulation qu'il a paramétrée.

La zone rose contient les étiquettes générées automatiquement servant de légende au rendu graphique de la simulation. Ces étiquettes fournissent aussi des informations telles que le nombre d'images par seconde pour le rendu graphique de la simulation et le pas de la simulation affichée.

Chapitre 5

Réalisation

Ce chapitre aborde les principales difficultés rencontrées lors de la réalisation et les choix internes au développement qui ont été effectués tout au long du projet.

5.1 Analyse syntaxique

5.1.1 Les exceptions

Les exceptions gèrent en interne une pile de message. L'objectif est de lever une exception ayant un nom explicite et de la faire remonter au fil du programme pour donner toujours plus d'informations explicites à l'utilisateur.

Les exceptions lancées sont donc alimentées en informations au fur et mesure de leurs remontées et sont, ensuite, affichées à l'utilisateur tout en bloquant la simulation. Donner une raison précise à l'exception n'a pas toujours été possible, car les erreurs Python des scripts sont parfois peu explicites. Cependant, en précisant le nom de fichier et son emplacement nous pouvons donner des pistes suffisantes pour un debug.

Afin de permettre à l'utilisateur de traiter l'évolution du chargement, nous avons indiqué le plus possible les différents chargements des scripts, ajoutant ainsi un meilleur suivit de l'erreur.

5.1.2 Les sucres syntaxiques

Des sucres syntaxiques ont été définis afin de créer une syntaxe avec une courbe d'apprentissage progressive.

Nous avons donc tout d'abord conçu une syntaxe complète, puis nous avons créé des sucres syntaxiques pour les débutants. Cet aspect implique que nous devons gérer dans le code les absences ou changements de type des différents éléments du JSON. Ceci a considérablement compliqué la gestion des exceptions, mais permet de rendre une syntaxe plus agréable.

5.2 Loader

Le paquetage *loader* doit effectuer de nombreuses opérations de chargement, et doit préparer toutes les entités, les actions et les interactions définies par l'utilisateur.

Pour cela, nous sommes tout d'abord passés par des métaclasses. Une métaclasse est une classe permettant de construire dynamiquement d'autres classes. Ainsi pour les entités, nous avons une métaclasse qui construit les différentes classes correspondant aux différents types d'entités.

Cette méthode a grandement facilité l'architecture du paquetage, nous avons ainsi deux métaclasses : une pour les types d'entités et une autre pour les interactions. Ces deux métaclasses ont pour but de :

- Charger les scripts utilisateurs.
- Gérer les erreurs de syntaxe et de conventions.

Le chargement des scripts utilisateur se fait dynamiquement. Un cache des scripts chargés permet une optimisation mémoire dans le cas où plusieurs entités ou interactions utiliseraient les mêmes scripts.

5.2.1 Robustesse

Le chargement des scripts utilisateurs est la seule porte d'entrée pour l'utilisateur, et étant donné qu'il peut ajouter du code Python, il peut potentiellement changer complètement le comportement du simulateur. Nous avons donc effectué quelques vérifications minimales pour garantir l'interfaçage correct entre le simulateur et les scripts utilisateurs.

Effets de bord

Les actions des entités peuvent être déclenchées dans une interaction. Nous avons donc considéré les actions comme des méthodes des entités. Cela peut entraîner de nombreuses modifications sur la même variable. Nous avons pris parti de rendre l'application de ces modifications aléatoires, afin de fausser le moins possible la simulation avec des effets de bords répétitifs. Ce choix entraîne plusieurs choses :

- La première modification d'une variable est toujours appliquée.
- De la deuxième à la énième modification, celle-ci est appliquée de manière aléatoire.

Ainsi, nous ne permettons pas à l'utilisateur de jouer avec la modification successive de la même variable, étant donné qu'il ne peut en prédire l'évaluation.

Reproductibilité

Un problème qui est survenu est l'utilisation des classes *Random*. En effet, dans le fichier de configuration *start.json*, nous utilisons un nombre pour initialiser le générateur de nombres aléatoires. Si l'utilisateur utilise par la suite sa propre instance de générateur, la reproductibilité n'est plus garantie. Nous ne pouvons pas empêcher l'utilisateur d'importer certains paquetages, nous devons donc nous reposer sur la documentation utilisateur, et demander d'utiliser exclusivement l'instance donnée par la façade *SimUtils*.

5.2.2 Performances

Nous avons tenté de rendre l'exécution de la simulation la plus rapide possible, au détriment du temps d'exécution du *Loader*. En effet celui-ci ne sera appelé qu'en début de simulation, mais construit les entités qui elles seront appelées n fois. Cela nous à mené à stocker des références sur les actions de nombreuses fois dans l'entité. D'une part, chaque action est un attribut de classe (accessible en méthode d'instance), d'autre part elle est stockée dans une liste indexée sur les terrains, et ce, pour chaque terrain où l'action s'applique.

Métaclasse

L'utilisation de métaclasses permet un gain certain de performances. Les premiers prototypes réalisés demandaient d'ajouter les attributs et les méthodes adéquates à toutes les instances d'entités. Avec l'utilisation des métaclasses, nous construisons d'abord des classes, qui se chargeront simplement de créer des instances, ce qui nous permet un gain en robustesse et en performance.

Évaluation paresseuse

Les attributs des entités sont ajoutés sous la forme de propriétés. Nous avions besoin d'adopter un comportement différent pour la lecture des attributs et pour leur modification. La lecture d'un attribut est toujours sur sa valeur en début de pas de simulation, alors que la modification s'exécute sur une autre variable tampon qui sera appliquée au changement de pas de simulation.

Nous avons donc exploité la notion de propriétés en redéfinissant les méthodes correspondant à la lecture et à la modification d'attributs, afin d'introduire ce comportement de façon totalement transparente. Au premier appel de lecture ou d'écriture de la propriété, un attribut s'ajoute au dictionnaire de l'objet. De la sorte, l'attribut n'est ajouté à l'instance qu'uniquement lorsqu'il est utilisé.

Cette évaluation paresseuse des attributs est uniquement due à l'usage des propriétés. Le gain de performances pourrait être garanti si certains attributs n'étaient pas évalués, mais il est plus probable que cela demande au final plus d'opérations, étant donné que tous les attributs risquent d'être évalués au premier pas de simulation.

Suivi des attributs au niveau graphique

Nous avons introduit la possibilité d'ordonner un rafraîchissement graphique lors de modifications de certaines variables. Par exemple les coordonnées horizontales et verticales sont par défaut "suivies" par l'interface graphique. Nous avons réalisé ceci en utilisant une fonction d'édition de propriété différente afin de mettre un fanion dans l'entité indiquant la demande de rafraîchissement.

De cette manière nous alourdissons très peu l'implémentation, car l'édition d'attribut passe simplement par une méthode différente.

Tampon de modifications

Afin de gérer les modifications des attributs, nous avons créé un tampon pour chaque attribut. Le système est simple : en début de pas de simulation, l'attribut possède une valeur disponible en lecture. Toute modification affectera une variable tampon, qui sera appliquée au changement de pas.

Afin d'alléger le système, nous nous basons sur un mécanisme de tampon à taille dynamique. Ainsi, seuls les attributs modifiés auront un tampon, permettant ainsi un gain de place évident.

5.3 Builder

La tâche principale de la classe *Builder* consiste à produire un objet *Params* qui englobe toutes les configurations de la simulation. C'est au lancement du programme qu'il est demandé au *Builder* de retourner un objet *Params* pour que la simulation puisse être démarrée.

Les éléments suivants expliquent les points les plus importants de l'implémentation du *Builder*.

5.3.1 Représentation des terrains

La représentation des différents terrains s'est réalisée avec un identifiant sous la forme d'un entier pour chaque type de terrain.

5.3.2 Représentation de la carte

La classe *SimulatorMap* représente la carte du simulateur avec un tableau à deux dimensions. Chaque case du tableau stocke un type de terrain. Comme Python ne permet pas de déclarer les types statiquement, la bibliothèque *Numpy* a été utilisée pour la déclaration d'un tableau d'entiers. Cette astuce permet d'éviter des évaluations de type à la génération de la carte pour chaque case (Python permet sinon d'avoir des tableaux avec des types hétérogènes, ce qui ralentit l'exécution).

5.3.3 Bruit de Perlin

La fonction *snoise2*[1] [2], mise à disposition par le paquetage *noise*, a été utilisée pour la génération d'un bruit de Perlin à deux dimensions.

5.3.4 Création des ellipses pour la carte personnalisée

Pour la création des ellipses à partir des coordonnées du point central et des dimensions horizontales et verticales, l'algorithme[3] de Geoff Hagopian[4], mathématicien et professeur au *College of the Desert* aux États-Unis, a été implémenté. Notre implémentation s'effectuait à partir d'une généralisation écrite en JavaScript[5] par l'usager *izb*[6] de *stackoverflow.com*.

5.4 Simulateur

La classe *Simulator* a pour tâche d'utiliser les paramètres donnés par le *Builder* afin de mener à bien la simulation. Son rôle est donc d'effectuer tous les appels aux scripts nécessaires à la réalisation d'une itération (un pas de simulation) avant de mettre à jour les valeurs et de passer à la suivante. Pour plus de détails, se référer au chapitre de conception en 4.6 Simulateur.

5.4.1 La passerelle avec l'interface graphique

La réalisation de l'interfaçage entre le simulateur est la partie graphique a été délicate et se fait à l'aide de la classe *GUIBridge*. Nous avons opté pour une approche par passage de témoin, à savoir que lorsque le simulateur a terminé une itération, il attend que l'interface graphique ait récupéré toutes les informations relatives aux modifications apportées par l'itération avant d'œuvrer à la résolution de l'itération suivante. De la sorte, nous avons économisé de l'espace mémoire dans la mesure où il n'est pas nécessaire de sauvegarder toutes les entités modifiées le temps de l'affichage.

L'inconvénient est que le simulateur se trouve donc bloqué à la fin de chaque itération le temps que l'écran graphique soit rafraîchi, ce qui ralentit son exécution. Nous avons accepté ce défaut dans la mesure où il est supposé que l'activation de l'affichage graphique ralentit d'une façon ou d'un autre la simulation, et donc que cette fonctionnalité est à réserver dans un but de visualisation ou de test. On préférera ainsi utiliser le mode textuel pour l'exécution de simulation avec de grosses quantités de données.

5.4.2 Ajout et suppression d'entités

Cette partie a été effectuée en utilisant une liste intermédiaire. En effet, la liste des entités étant parcourue pour appliquer les actions, modifier directement cette liste entraînerait des effets de bords. Si au cours d'une itération une entité était ajoutée, celle-ci ne devrait exister qu'à la prochaine itération pour garder un état cohérent avec l'itération "figée" actuelle. Ce constat nous a conduit à utiliser une liste intermédiaire, afin de n'ajouter les nouvelles entités qu'une fois l'itération terminée. De la sorte, la liste principale reste intacte pendant les calculs.

Concernant la suppression, le principe reste le même. Néanmoins, l'entité doit pouvoir être communiquée à l'interface graphique (si elle est utilisée) afin que celle-ci puisse supprimer l'affichage de cette entité. Ainsi une entité ne doit être supprimée qu'au cours de l'itération suivante. Afin d'indiquer qu'une entité est "supprimée", un attribut spécial a été ajouté afin de fournir une méthode de test pour savoir

1. http ://www.nullege.com/codes/search/noise.snoise2, 13.12.2013
2. https ://github.com/caseman/noise, 13.12.2013
3. http ://geofhagopian.net/sablog/Slog-october/slog-10-25-05.htm30.12.2013
4. http ://geofhagopian.net/, 30.12.2013
5. http ://stackoverflow.com/questions/15474122/is-there-a-midpoint-ellipse-algorithm, 30.12.2013
6. http ://stackoverflow.com/users/974/izb, 30.12.2013

si une entité est active ou non. Une entité non-active signifie qu'elle sera supprimée avant la prochaine itération.

5.4.3 Les interactions

Déterminer si deux entités sont à distance suffisante pour déclencher une interaction et éviter les doublons a été la partie délicate de la réalisation du simulateur.

Une approche simple aurait été de parcourir la liste des entités et de vérifier toutes les entités suivantes afin de tester la distance les séparant pour appeler ou non le script d'interaction.

Malheureusement, en supposant un grand nombre d'entités, cette approche en $O(n^2)$ impliquerai un nombre excessif de comparaisons.

Nous avons donc opté pour un pré-triage des entités concernées. Pour ce faire, la carte est découpée est petites zones dont les dimensions dépendent de la distance maximale requise pour qu'une interaction ait lieu. Au début de l'itération la liste des entités est parcourue une fois afin d'affecter chaque entité à la zone lui correspondant.

Il ne reste alors plus qu'à comparer les entités d'une zone avec celles de la même zone, ainsi que des zones adjacentes. En parcourant la matrice des zones de gauche à droite et de haut en bas, les zones adjacentes à consulter sont uniquement celle de droite et celles de la ligne suivantes étant donné que les zones précédentes auront déjà vérifié les interactions possibles avec la zone en cours de traitement.

En supposant les zones 1 à 5 déjà traitées et la zone 6 comme étant celle en cours de traitement :

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$

Alors il suffit de comparer les entités de la zone 6 avec celles de la zone 6, 7, 9, 10 et 11. Il est même possible d'effectuer un court-circuit afin que si une entité se trouve dans le cadran supérieur gauche de la zone courante, seules les entités de la même zone soient vérifiées. En effet, si une entité se trouve dans ce cadran, alors les seules entités potentiellement à distance valide seront forcément dans la même zone, pour peu que la taille de la zone soit supérieure au double de la distance maximale d'interaction.

Afin d'illustrer ce constat, supposons une distance d'interaction de 1, avec une taille de zone de 3, nous obtenons les positions possibles suivantes au sein de la zone :

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Une entité en position 1, 2, 4 ou 5 ne pourra avoir une éventuelle interaction qu'avec les entités aux positions 1 à 9.

Grâce à cette approche et au fait que les entités soient affectées à une zone précise en début d'itération, nous avons pu mettre à disposition un outil permettant de demander au simulateur la liste des entités présentes à une position précise, et ce, sans avoir à parcourir la liste de toutes les entités, mais uniquement celles de la zone correspondant au point recherché.

5.5 Interface utilisateur

Dans cette partie, nous reviendrons sur les particularités de réalisation de l'interface graphique. Cette partie est séparée en deux. La première moitié portera sur toute la réalisation ayant trait à la partie cachée de l'interface qui n'est pas visible mais qui relie la fenêtre visible et le simulateur. La seconde moitié portera sur la réalisation de la fenêtre en elle-même et l'agencement des composants.

5.5.1 Gestionnaire de l'interface graphique

La réalisation de cette partie s'est déroulée globalement sans heurts mais certains points ont été particulièrement durs à résoudre comme l'interfaçage entre le simulateur et la partie graphique de la simulation. Cela provient, entre autre, du fait que nous utilisons une librairie graphique qui demande d'avoir une tâche séparée. Ainsi donc, cette partie de l'application travaille avec des threads en Python (pour communiquer avec le simulateur) et un thread graphique géré entièrement par Qt qui affiche la fenêtre. La tâche graphique ne tolère pas que l'on s'adresse directement à la partie QML du programme en étant dans une tâche Python. Pour palier à ce problème, nous devons utiliser le système de signaux et de slots fournis par Qt.

Un autre point est la fermeture de toutes les tâches du programme lorsque l'utilisateur met fin à l'exécution. Ce problème a été corrigé en tout deux fois dans le projet car ce il fût fixé une première fois mais suite à une modification du programme, il est réapparu. Nous ne sommes toujours pas sûr de ce qui était à l'origine de ce problème.

Le dernier point particulier est la gestion de la liste des statistiques et légendes affichées à droite de la fenêtre graphique. En effet, le prototype sur lequel nous nous étions basés pour faire cette liste provenait d'un exemple qui ne fonctionnait pas dans le cas de mise à jour du contenu d'un item dans la liste : la vue de la liste n'était pas mise à jour. Pour résoudre ce problème, il a fallut envoyer un signal spécial à l'interface pour que cette dernière repeigne la liste sur la fenêtre, provoquant ainsi la mise à jour voulue. Cependant, le signal utilisé correspond à la ré-initialisation du modèle. Ce signal n'est pas adapté mais il s'agit du seul que nous ayons trouvé qui fonctionne.

Modifications

Deux modifications majeures de l'interface ont été réalisées par rapport au cahier des charges initial.

La première est la suppression de certains labels d'information sur la partie droite de l'interface. Ce choix a été motivé par le peu de pertinence qu'apportaient ces informations à l'utilisateur. Certaines informations n'ont pas été supprimées mais déplacées à un autre endroit dans l'interface comme le nom de la simulation qui est visible dans la barre de titre au lieu d'être dans la liste.

La deuxième modification est survenue lors de l'implémentation de la fonctionnalité d'exportation. En effet, il était prévu d'avoir un bouton "*Export*" dans l'interface graphique permettant d'exporter les statistiques enregistrées lors de la simulation. Nous nous sommes rendu compte qu'un tel bouton signifiait de stocker toutes les statistiques en mémoire lors de l'exécution puis de les écrire dans des fichiers lors de l'appui sur ce bouton. Comme les éléments enregistrés pouvaient prendre une place considérable en mémoire, nous avons décidé d'écrire pas à pas les données dans les fichiers d'exportations afin de soulager la mémoire. La décision d'exporter les résultats ou non est désormais dans le fichier de configuration du programme *start.json* au lieu d'être incarnée par le bouton "*Export*" qui a disparu.

5.5.2 QML

La réalisation de cette partie a été plutôt mouvementée car c'était une technologie totalement nouvelle. Heureusement, la courbe d'apprentissage de cette technologie est bien pensée est permet très vite de réaliser des fenêtres fonctionnelles.

La partie la plus difficile à faire rejoint le problème énoncé dans le chapitre précédent concernant la mise à jour du rendu. Comme nous n'avons pas pu résoudre ce problème du côté du code python dans l'immédiat, nous avons forcé le rafraîchissement du côté QML en utilisant un *timer* qui met à jour le rendu toutes les 100 millisecondes. Cette solution n'était pas propre et nous perdions une partie des performances. Après plusieurs tentatives, nous avons réussi à mettre au point une solution qui permet de supprimer ce *timer* QML et d'utiliser un système de coordination géré par le gestionnaire graphique.

Un autre problème de résolution est survenu au moment où nous voulions ajouter une barre de défilement dans la zone des légendes et statistiques afin de pouvoir afficher un nombre illimité d'objets. La version que nous utilisons de QML ne dispose pas d'un élément permettant de faire une liste déroulante. Il existe des solutions développées par des enthousiastes mais elles ne permettent pas de résoudre notre problème de façon optimale. Nous avons donc décidé d'abandonner cette amélioration et de la faire uniquement si nous avions du temps supplémentaire pour réaliser ceci.

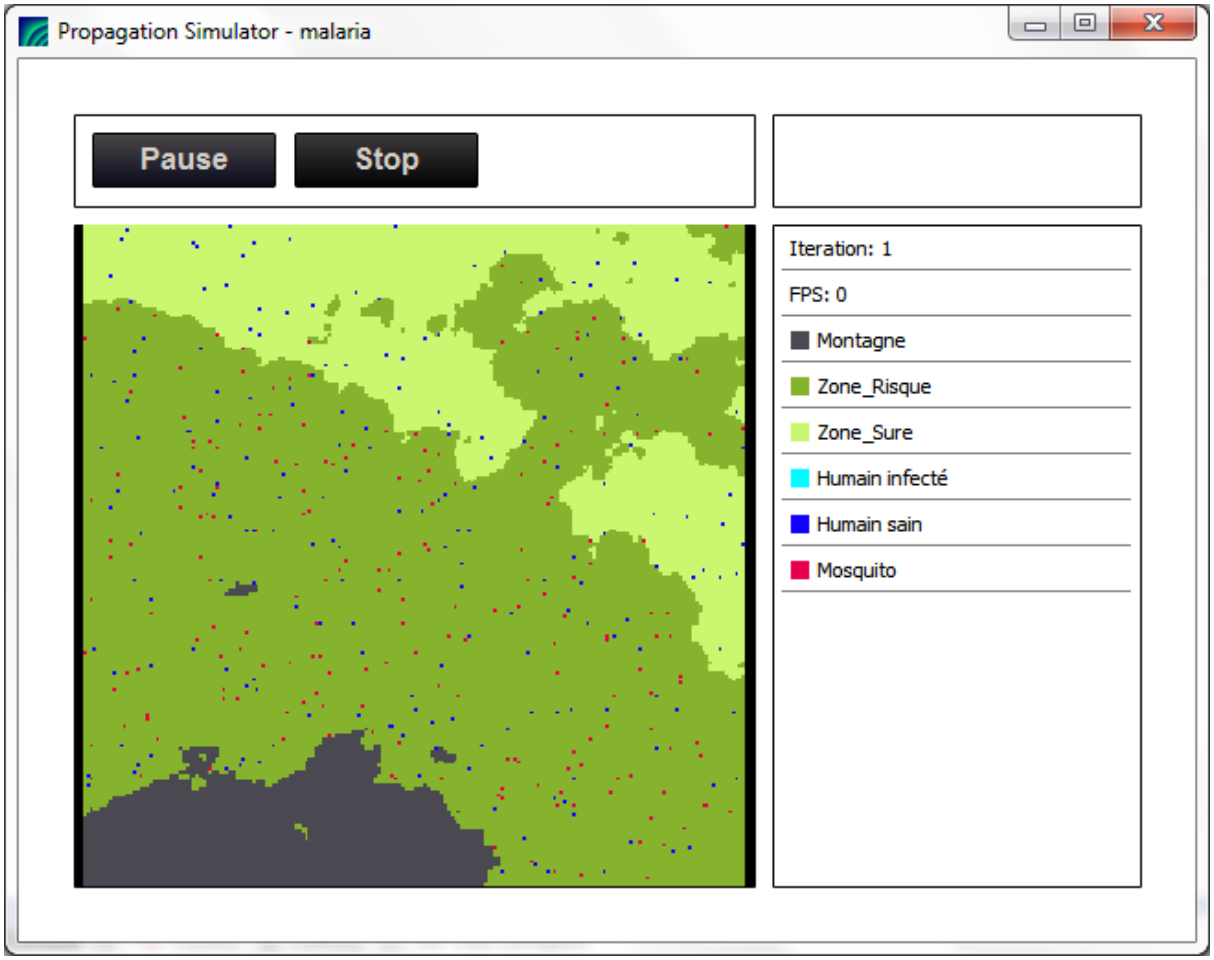


FIGURE 5.1: Apparence finale de *Propagation Simulator* en exécution (Simulation malaria).

5.6 Difficultés des parties mineures

5.6.1 SimUtils

Cette classe à été conçue pour servir de boîte à outils pour l'utilisateur. En effet, l'utilisateur doit pouvoir accéder un certain nombre d'éléments de la simulation afin d'avoir une plus grande liberté dans la création de ses scripts.

Les utilisateurs peuvent ainsi avoir accès à ces éléments :

- Un moyen de générer des nombres aléatoires.
- Un moyen de connaître le type de terrain d'une position donnée.
- Un moyen de connaître si une position donnée est sur la carte.
- Un moyen d'ajouter une entité à la simulation.
- Un moyen de supprimer une entité de la simulation.
- Un moyen de connaître les entités à une position donnée.

Nombre aléatoire

Afin de permettre des résultats reproductible, nous offrons une méthode permettant d'utiliser un générateur de nombres aléatoires initialisé d'après le script *start.json*.

Informations sur la carte

Afin d'éviter les effets de bords en transmettant la grille en entier, nous permettons d'accéder uniquement à certaines portions de la carte. Nous proposons ainsi uniquement deux méthodes pour accéder à la carte, à savoir une pour connaître le type de terrain à une position donnée et une autre pour savoir si un point défini n'est pas en dehors de la carte.

À noter que si l'utilisateur demande à une entité de sortir de la carte, l'opération sera simplement ignorée. Les limites de la carte font donc office de "mur", et l'utilisateur n'a pas à traiter les erreurs liées à ce genre d'opérations.

Ajout et suppression d'entité dans la simulation

Ces deux outils peuvent sembler étranges. Cela est dû au fait que nous proposons à l'utilisateur de passer par des scripts pour le positionnement des entités sur la carte. Ce positionnement demande donc une liste d'entités déjà instanciées, avec des positions définies.

Ainsi, par le biais de *SimUtils*, nous pouvons facilement ajouter des entités à la simulation, même en cours de route (par exemple, la reproduction de deux entités est possible).

Chapitre 6

Tests

Tout au long du projet des tests ont été effectués afin de vérifier le fonctionnement unitaire des différents éléments. Il est évident que chaque élément du projet ne pouvait être testé de manière unitaire. Concernant ces éléments, comme par exemple le simulateur, nous avons du faire des tests de fonctionnement du programme, d'un point de vue général.

6.1 Tests unitaires

6.1.1 Tests unitaires pour les entités

Les tests unitaires sont effectués pour le comportement des entités. Toutes les entités doivent permettre de fournir un comportement de base, sur lequel le simulateur pourra s'appuyer.

Swap des attributs
Comme nous l'avons déjà décrit, les entités possèdent tous leurs attributs à double. Une version de l'attribut concerne sa valeur au début de l'itération, l'autre servant uniquement pour l'écriture. Nous employons une méthode paresseuse, c'est-à-dire que les attributs sont doublés uniquement dès qu'une modification se porte sur lui.

Voici une liste des sujets sur lesquels les tests unitaires portent :

- Un attribut modifié s'applique uniquement après un swap.
- Un attribut modifié n'est modifié que dans l'instance concernée.
- Un attribut modifié demandant un rafraîchissement de la vue à l'édition retourne `True` lors du swap.
- Un attribut non modifié n'est pas affecté par un swap.
- Tous les types que le JSON peut fournir effectuent un swap correct.

Sur ce dernier point, nous n'avons pas réussi à effectuer de swap sur des listes et sur des dictionnaires. Nous avons donc des risques importants d'effets de bord, dans le cas où plusieurs modifications de la liste se feraient en même temps. Ce défaut pourrait être corrigé en rendant constant le contenu des listes (récursivement), mais cela diminuerait l'intérêt pour l'utilisateur d'utiliser les listes.

Ce problème pourrait être corrigé dans d'éventuelles versions futures, en mettant un système de double tampons sur tous les éléments de la liste par exemple.

Tous les autres tests sont passés avec succès.

6.1.2 Tests unitaires pour les exceptions utilisateurs

Nous n'avons pu que partiellement implémenter des tests unitaires pour les exceptions utilisateurs. En effet, lorsqu'un utilisateur utilise le logiciel afin de créer ses scripts, il peut et va commettre des erreurs. Il doit donc être informé le plus clairement possible. Nous avons donc pu vérifier par des tests unitaires que l'exception est bien levée, mais la qualité du message de l'exception a du être jugée au cas par cas. Notamment les simulations d'exemples nous ont beaucoup aidé à implémenter des messages d'erreur plus parlant.

Les messages d'exception sont cependant parfois trop précis, et affichent notamment des portions de codes appartenant au cœur du simulateur. Des améliorations futures pourront protéger l'utilisateur de messages trop précis, qui lui fournissent des informations dont il n'a pas besoin.

Malgré ce problème, nous avons pu assurer que la cause de l'erreur est dans la majorité des cas transmise dans le message d'erreur. Quelques problèmes subsistent dans les messages d'exception, mais ils sont mineurs. Par exemple, dans le cas où la personne définit deux fois le même attribut, une erreur fatale survient avec un message très peu explicite sur la cause de l'erreur.

Le problème que nous avons rencontré avec la gestion des messages d'erreur est que la précision est difficile à atteindre, car différents cas d'erreur peuvent mener au même problème. Il a donc été compliqué d'isoler les cas pour fournir une information cohérente à l'utilisateur.

6.1.3 Builder

Les méthodes de base du *Builder*, en particulier la mise à disposition des terrains et des informations de base des entités ont été testées à l'aide des tests unitaires. Ces méthodes ont passé les tests unitaires avec succès.

Pour ce qui concerne la génération de la carte, le bon fonctionnement a été validé à l'aide de l'interface graphique. La carte générée par le *Builder* était bien visible dans l'interface graphique. Ceci permettait de contrôler toute la chaîne du code pour la génération de la carte personnalisée, la génération à l'aide du bruit de Perlin ainsi que pour le mode purement aléatoire.

6.2 Tests de comportement

6.2.1 Simulations d'exemple

Pour chaque release de *Propagation Simulator*, des simulations d'exemple ont été livrées avec. Ces exemples utilisent toutes les fonctionnalités mises à disposition par l'application dans son état actuel. Ceci permettait de tester et de valider le bon fonctionnement des fonctionnalités implémentées pendant une release.

Malaria

Il s'agit de la simulation donnée en exemple dans la documentation du projet. Elle simule la propagation de la malaria dans une population humaine. La carte comprend une zone montagneuse infranchissable, une zone risquée peuplée de moustiques porteurs du virus, et une zone sûre peuplée uniquement d'humains. Les moustiques peuvent piquer les humains et les humains peuvent ensuite se retransmettre la maladie.

Bien que simpliste, elle est un exemple typique de simulation visée par le programme. Il illustre l'utilisation attendue des terrains, des entités, des actions et des interactions. Il comprend des scripts "inline" définis au sein des fichiers JSON, des scripts déportés dans des *Function* et des *Triggers* implémentés en Python. Les paramètres de la carte peuvent être changés pour faire usage d'un des trois modes de génération de terrains.

Fourmis tandem

Il s'agit de la simulation décrite en détail dans le manuel utilisateur sous forme de tutoriel. Elle simule de façon réaliste les déplacements d'une colonie de fourmis cherchant un nid pour y construire une fourmilière. Il s'agit d'une simulation assez complète permettant d'illustrer l'utilisation du programme pour une simulation relativement complexe du point de vue du nombre de règles à définir.

Là encore, c'est une simulation qui entre parfaitement dans le cadre de ce pour quoi le programme est conçu.

Game of Life

Cette simulation implémente le fameux automate cellulaire de J. Conway [1]. Contrairement aux précédentes, il ne s'agit pas du type de simulation pour lequel le programme a été conçu. Cependant, elle montre que le système de script est suffisamment souple pour permettre une utilisation différente du programme.

L'implémentation est faite en plaçant une entité sur chaque case de la carte, et en définissant une action permettant de compter les entités vivantes alentours et décider si l'entité doit survivre ou pas. Les performances se dégradent donc très vite, mais l'intérêt ici était de tester et illustrer l'utilisation de certains mécanismes du simulateur et certaines fonctions de *SimUtils*, ainsi que la définition de règles très précises et vérifiables.

6.2.2 Interface graphique

Les tests de l'interface graphique sont particuliers car leur résultat est uniquement visible à l'écran. Pour mesurer et quantifier ces tests, le seul moyen à notre disposition était de constater manuellement que l'interface produisait bien les résultats attendus autant que nous puissions en juger.

1. http ://en.wikipedia.org/wiki/Conway%27s_Game_of_Life

Chapitre 7

État des lieux

7.1 Fonctionnalités réalisées

Les fonctionnalités correspondant au découpage des releases une, deux et trois ont étés effectuées à savoir :

- Génération de monde entièrement aléatoire.
- Génération du monde avec un bruit de type *Perlin*.
- Génération du monde manuellement.
- Exportation des données.
- Affichage de l'interface graphique. Aucune configuration ne se fait dans l'interface. Les fonctionnalités de l'interface sont les suivantes :
  - Une zone de contrôle permettant de lancer, de mettre en pause et de stopper la simulation.
  - Une zone de légende de la simulation.
  - Le rendu graphique de la simulation.
- Configurer une entité : exécution des actions.
- Configurer une entité : interactions entité-entité.
- Toutes les interactions sont configurables (on peut ajouter de nouvelles interactions).

7.2 Points non réalisés

Cette partie regroupe les fonctionnalités qui n'ont pas été réalisées et certains éléments qui ne font pas partie des fonctionnalités mais qui fonctionnent partiellement ou qui n'ont pas été terminés. Ces éléments ne sont pas finis à cause d'un manque de temps lorsque survient un problème inattendu qui prend plus de temps que la moyenne pour être résolu.

Toutes les fonctionnalités ont été réalisées. Cependant, les points suivants ne sont pas complets :

- Appels à *new_ entity* et *remove_ entity* pendant la simulation [1].
- La reproductibilité n'est pas garantie si l'utilisateur importe le module *random* dans un script de configuration.
- Des erreurs de syntaxe produisent, dans certains cas, des résultats incompréhensibles.

Nous estimons qu'il faudrait une semaine à temps plein à notre équipe pour résoudre ces problèmes. Le premier défaut mentionné empêche simplement de créer ou supprimer de nouvelles entités en cours de simulation. Dans le cadre d'une simulation par propagation, on recherche plutôt à observer l'évolution d'une population déjà existante face à certains événements. Ce défaut est donc mineur car il ne fait pas partie des spécifications initiales et n'empêche pas l'exécution du programme. Les deux

1. Initialement, il n'était pas prévu formellement de donner cette possibilité.

derniers défauts ne portent pas préjudice au programme tant que l'utilisateur respecte les directives de configuration de fichiers données dans le manuel utilisateur et utilisées dans les simulations d'exemple.

7.3 Améliorations

Par rapport à la conception originale, les améliorations ci-dessous ont été implémentées. Ces améliorations font suite à des discussions en interne dans lesquelles nous avons convenu d'ajouter ces améliorations en fonction du temps qu'elles prenaient pour être implémentées et de leur utilité.

7.3.1 Generate

Il s'agit d'un outil en ligne de commande permettant à l'utilisateur de créer l'arborescence par défaut d'une simulation et d'y ajouter des entités, des terrains, des actions et des interactions. Cette fonctionnalité additionnelle est extérieure au programme mais apporte un gain de temps conséquent pour l'utilisateur.

Ce programme ne permet pas de compléter les fichiers de configuration, mais uniquement de générer la structure et tous les fichiers de configuration nécessaires pré-remplis.

Plus de détails sur l'outil sont donnés dans le manuel utilisateur. L'outil est également auto-documenté : le seul fichier de code qu'il contient comprend les explications nécessaires, et il comporte des pages d'aides grâce à l'option `--help` qui expliquent tous les paramètres.

7.3.2 Quick Reference Card

Toujours dans l'optique de faciliter l'écriture des scripts et fichiers de configuration, nous avons conçu, en plus du manuel utilisateur, une "feuille de triche" comprenant de façon concise une référence rapide de la structure des fichiers et des méthodes utilisables dans les scripts, dans le même style que les *Cheat Sheets* ou *Quick Reference Cards* que l'on peut trouver pour résumer la syntaxe des langages usuels.

7.3.3 Scripts utilisateur

Nous avons embelli la syntaxe JSON utilisée pour une partie des scripts utilisateur en y ajoutant des sucres syntaxiques qui simplifient la vie à l'utilisateur. Nous avons également créé des procédures et des attributs par défaut que l'utilisateur peut surcharger dans ses fichiers de configuration.

7.3.4 Redimensionnement de la fenêtre

Le redimensionnement de la fenêtre permet à l'utilisateur de modifier la taille de la fenêtre en cours d'exécution. Le rendu graphique de la simulation sera lui aussi affecté le grossissant ou le réduisant selon l'action de l'utilisateur tout en gardant le même rapport hauteur/largeur.

7.3.5 Performances

Nous avons réalisé certaines améliorations de performances dans le programme afin de permettre à l'utilisateur d'avoir des simulations plus complexes en terme de taille de carte et de nombre d'entités. Les modifications que nous avons réalisées se situent dans la méthode de rendu pour l'affichage ainsi que dans la gestion de la position des entités lors du calcul des interactions lors d'un pas de simulation.

7.3.6 Scénarios de simulation

Nous avons rajouté trois scénarios de simulation prêts à l'emploi afin de tester l'application et de permettre à l'utilisateur de se familiariser avec le logiciel à l'aide de modèles pré-existants. L'élaboration

de ces scénarios nous a également permis de tester différents aspects du simulateur afin de s'assurer que tout fonctionnait comme nous l'attendions.

Chapitre 8

Conclusion

Ce chapitre fait la synthèse de l'état actuel du programme et conclu le déroulement de projet avec une analyse critique.

8.1 Etat du programme

Le programme est fonctionnel et permet de visualiser le déroulement d'une simulation selon le cahier des charges.

L'équipe est heureuse et fière du résultat obtenu et de son travail avec les technologies que nous avons utilisées. Le programme a pu être fini malgré un semestre académique très chargé car nous devions concilier le temps pour le projet et les autres cours qui ont demandé énormément de temps ces dernières semaines.

Nous estimons à environ une semaine le temps supplémentaire qu'il nous faudrait avec la même équipe et à plein temps pour régler les derniers détails restants et apporter quelques améliorations (voir chapitre 7.2). Cela nous permettrait aussi de simplifier la syntaxe de configuration et améliorer les messages d'erreurs sur les scripts utilisateur.

8.2 Gestion de projet

La gestion du projet s'est bien déroulée. Procéder par une révision du travail à effectuer chaque semaine et un partage de l'état d'avancement hebdomadaire a permis que chaque membre du projet soit au courant de ce que les autres faisaient. Ces réunions du vendredi étaient bien gérées et réglaient les choses de manière efficace. A cela s'ajoutait une mise au point individuelle opérée entre le chef de projet et chaque membre individuellement le lundi afin de faire le point sur l'avancement. Ce procédé à permis de récolter beaucoup d'informations sur le déroulement du projet et d'être en permanence aux aguets pour un éventuel problème ou avancement soudain afin de re-répartir le travail au mieux.

Le système de pile d'items emprunté à la méthode SCUM permettait à chacun de savoir exactement ce qu'il avait à faire, ce que les autres avaient fait et où en était le programme globalement. Cependant, cet outil a été sous-utilisé car il était parfois malaisé d'utiliser ce document par les membres du projet (format, logiciel, etc.) et certaines personnes n'aiment pas consigner ce qu'elles font dans un document.

8.3 Planification

Deux fois par semaine, nous mettions à jour le nombre de tâches (ou "items") effectuées par les membres du projet. Nous obtenions ainsi une mesure des tâches restantes à effectuer et celles qui ont déjà été effectuées.

Dans l'ensemble la planification a bien été suivie car elle était effectuée pour chaque release séparément, donc environs toute les trois semaines. Nous sommes arrivés au résultat que nous nous étions fixés. Nous pouvons donc dire que la planification a été correctement maîtrisée. Cela est dû notamment à la planification des étapes du projet et la phase de prototypage nous a permis d'éliminer les barrières technologiques et diminuer les surprises en cours de développement.

Il y a bien eu quelques points non terminés à la fin de certaines releases, mais à aucun moment le projet n'a fait l'objet d'un retard important et/ou non maîtrisé, chose suffisamment rare pour être mentionnée. Les points terminés de façon partielle seulement au terme d'une release concernaient en général la documentation, qui avait était faite mais pas aussi rigoureusement que l'on serait en droit d'attendre. Dans la mesure où au moins un brouillon était produit, cela restait sans conséquences sur le déroulement du projet.

8.3.1 Release 1

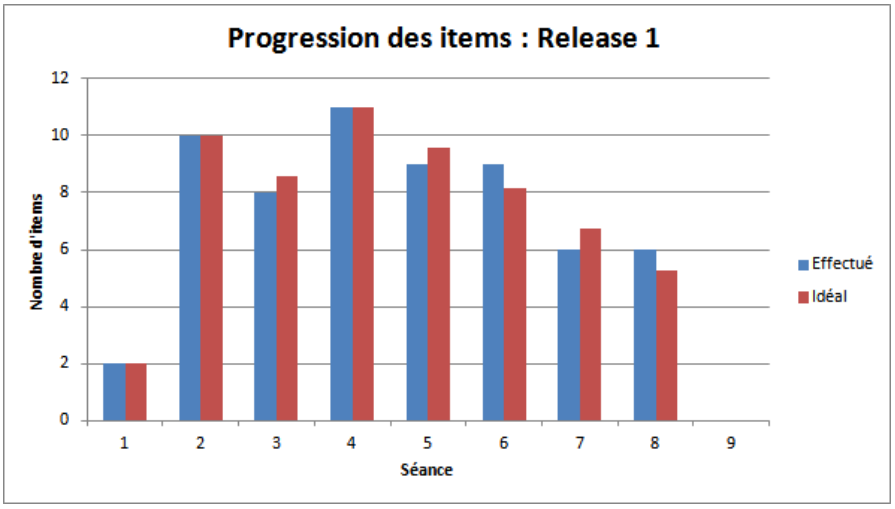


FIGURE 8.1: Progression des items de la release 1.

Cette première release était la plus longue et a commencé le 28 octobre 2013 pour finir le 24 novembre 2013. A chaque séance, nous pouvons voir le nombre d'items effectués et le nombre d'items qu'il aurait fallu effectuer pour cette séance [1]. Le nombre idéal d'items est calculé à partir du nombre total d'items à faire et le nombre de séances encore à disposition.

Pour cette release, nous pouvons voir qu'il y a eu peu de surprises et que la charge de travail a été correctement estimée à cause des faibles différences entre le nombre d'items souhaités et ceux effectués.

1. Chaque numéro de séance impair correspond à un lundi et chaque numéro pair correspond à un vendredi.

8.3.2 Release 2

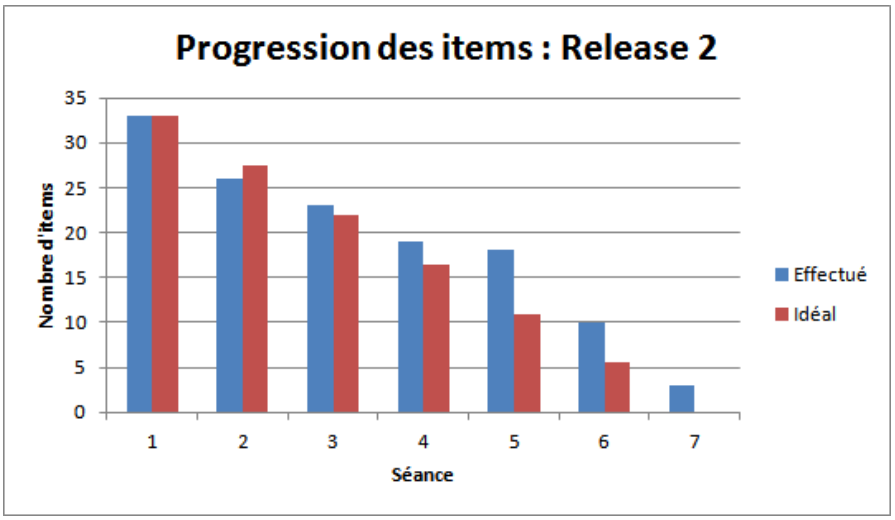


FIGURE 8.2: Progression des items de la release 2.

Cette deuxième release a commencé le 25 novembre 2013 pour finir le 15 décembre 2013. Contrairement à la release précédente, peu d'items ont étés rajoutés en cours de release car si nous traçons une ligne sur le sommet des histogrammes pourpres, nous avons une droite qui descend. Nous pouvons remarquer qu'entre les séances 4 à 7 nous avons eu quelques items de retard dus à une surcharge de travail à cause des autres cours du semestre. Nous n'avons malheureusement pas pu tout finir ce qui était prévu à la fin de la release. Les items restants ont étés transférés sur la release suivante de façon totalement contrôlée.

8.3.3 Release 3

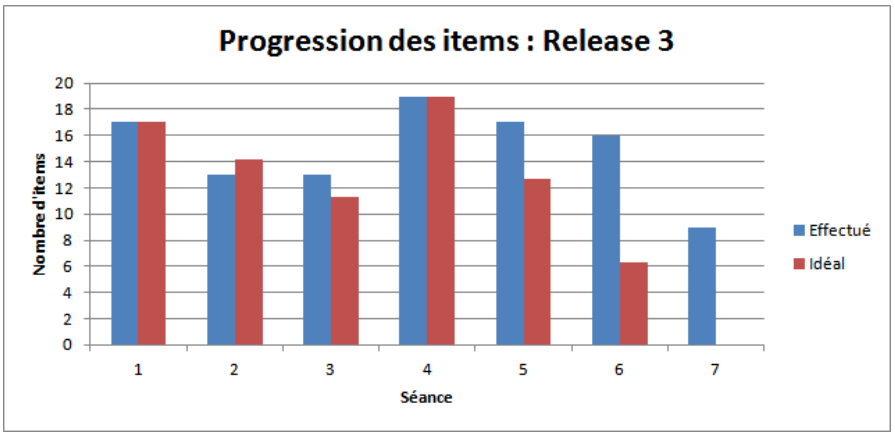


FIGURE 8.3: Progression des items de la release 3.

Cette troisième release a commencé le 16 décembre 2013 pour finir le 5 janvier 2014. Nous pouvons observer qu'elle fut largement plus erratique que les précédentes. Cela est du au fait que cette release se passait majoritairement pendant les vacances. Nous avons eu une séance par *Skype* mais ce n'est pas aussi efficace qu'un contact direct. Nous avons accumulé un peu de retard à cause de la période pendant laquelle la release a été faite. Heureusement, une semaine supplémentaire après la release 3 avait été prévue pour palier à ce genre de situation.

8.4 Améliorations possibles

8.4.1 Interface

L'interface est assez simple dans ce programme et elle peut être améliorée en lui rajoutant un affichage des erreurs des scripts utilisateurs dans une partie dédiée afin de gagner en lisibilité. Pour l'instant, les erreurs sont affichées dans une console qui s'affiche s'il y a une erreur lors de l'exécution. Nous pourrions également améliorer la liste des légendes à droite de l'interface pour lui rajouter une barre déroulante.

Dernièrement, nous pouvons rajouter un lanceur de programme [2] qui permettrait de supprimer le fichier de configuration *start.json*. Cela contribuerait à rendre le programme plus accessible aux utilisateurs.

8.4.2 Performances

Les limites de notre programme sont atteintes relativement rapidement malgré nos efforts pour le rendre le plus performant que nos capacités techniques le permettaient. Il y a donc encore des améliorations dans les performances qui sont possibles, notamment dans les parties suivantes du programme :

- Chargement d'une simulation.
- Calcul d'un pas de simulation.
- Calcul du rendu de la simulation.
- Gestion des entités de la simulation.

Une piste pour améliorer les temps de calculs est la séparation du travail en tâches ou processus afin de répartir la charge de travail sur les différents processeurs. Pour améliorer le temps de chargement d'une simulation, nous pourrions également tenter de minimiser le nombre d'accès aux disques dur.

8.4.3 Exportation

Pour l'instant, le programme se contente d'exporter les statistiques que l'utilisateur aura choisi. Il est possible dans le futur d'ajouter un traitement poussé des résultats comprenant des graphiques, des temps de calculs, des statistiques avancées, etc.

8.4.4 Generate

Il s'agirait d'améliorer l'outil de génération de simulation en lui rajoutant plus d'options comme le choix du type des attributs d'une entité, création de lien vers des actions/interactions prédéfinies, renommer une entité, créer une simulation en partant d'une simulation déjà existante, etc.

8.4.5 Scripts utilisateur

Pour les scripts utilisateurs, nous pourrions en améliorer l'utilisation en simplifiant la syntaxe et en donnant plus d'informations sur la classe *SimUtils* qui met à disposition des fonctions pour l'utilisateur.

Une amélioration majeure, serait de pouvoir paramétrer complètement une simulation par l'interface graphique, simplifiant ainsi le travail de l'utilisateur.

Un dernier point serait de rendre possible l'importation d'entités depuis un fichier. Cela permettrait à l'utilisateur de définir précisément la population de départ d'une simulation.

Nous pouvons aussi permettre l'ajout de constantes dans les scripts de l'utilisateur, des attributs globaux à toutes les entités de la simulation et des variables disponibles dans toute la simulation (des compteurs par exemple). Dans le même ordre d'idée, nous pourrions permettre à l'utilisateur de choisir parmi plusieurs configurations de lancement pour une même simulation.

2. *Launcher* en anglais.

8.4.6 Simulation

Ajouter le possibilité d'avoir des interactions entre plus que deux entités. Cela permettrait de créer des effets de groupe de manière simple pour l'utilisateur.

8.5 Conclusions personnelles

8.5.1 Grégoire Decorvet

Le projet s'est bien déroulé compte tenu de la nouveauté technique. En effet, tant le style de l'application que les outils utilisés pour la réaliser sont nouveaux pour les membres de l'équipe.

Les prototypes ont été rapidement réalisés, ce qui nous a permis d'avoir une bonne vision globale du projet. L'analyse a été efficacement réalisée, ce que nous avons pu constater dans la mesure où le développement du produit s'est bien passé. Nous avons une bonne base sur laquelle de nouveaux éléments peuvent se greffer pour de nouvelles fonctionnalités.

L'utilisation d'éléments de la méthode SCRUM est une expérience supplémentaire par rapport à la seule réalisation du projet. Ce fut fort instructif de pouvoir essayer quelques uns de ces concepts. Néanmoins son utilisation est restée limitée de part le fait que nous ne travaillions pas à plein temps sur ce projet, la faute en incombe aux autres cours.

Travailler avec mes collègues fut également enrichissant. Chacun ayant une vision légèrement différente quant à la résolution d'un problème technique ou d'une politique à prendre concernant certains détails, ce qui au final permet d'affiner nos avis et de tendre vers une solution intéressante. Cet aspect est un point fort de par la bonne ententes entre les membres du groupe.

Je tire de ce projet un bilan positif, tant au niveau de l'expérience avec des technologies que je n'ai pas eu l'occasion d'expérimenter avant, qu'au niveau du produit réalisé.

8.5.2 Hadrien Froger

Ce projet à été très enrichissement pour moi. J'avais déjà appliqué SCRUM dans un plus grand projet, et je me suis servis de ces connaissances pour inspirer monsieur Schweizer et le motiver à concevoir ensemble une gestion de projet Agile adaptée au projet.

Concernant la programmation, je connaissais un peu Python et j'ai pu apprendre de nouveaux concepts, comme ceux de métaclasses. Même si les métaclasses semblent utiles dans un petit nombre de situations, cela m'a permis de comprendre plus en profondeur la notion d'orienté-objet.

Beaucoup de principes Python m'étaient inconnus et j'ai apprécié découvrir les notions de signaux, itérateurs et pointeurs de fonctions dans un cadre orienté-objet.

En m'attaquant à la création de la syntaxe utilisateur, j'ai du me mettre le plus possible dans la peau de l'utilisateur, ce qui était très intéressant. Créer une syntaxe claire, cohérente et suffisamment expressive n'a pas été aisé, et le travail en équipe à été très profitable.

Je suis globalement très satisfait du résultat final, nous avons réussi à bâtir une structure propre tout en restant souple dans notre démarche de développement. De plus, le processus itératif nous à permis de faire de véritables sprints qui nous ont permis un rendu plus calme. Nous avons donc pu travailler sans pression dans une bonne ambiance de travail.

8.5.3 Kevin Jaquier

Je tire un bilan positif du projet jusqu'ici. Tout d'abord, il m'a permis de découvrir et mettre en pratique beaucoup de choses nouvelles, aussi bien du point de vue méthodologique que technique.

Pour ce qui est de la méthodologie, la méthode SCRUM que nous avons utilisée semble bien fonctionner et j'en suis très satisfait. Elle permet de faire avancer le projet en favorisant efficacement les aspects

prioritaires. Les releases régulières et la gestion des tâches nous forcent à obtenir des résultats concrets et à maintenir la documentation à jour. Les réunions hebdomadaires permettent au groupe de faire le point sur ce qui a été fait par chacun et planifier le travail des semaines à venir.

Du côté technique, je connaissais déjà le langage Python (à peu près aussi bien que les autres langages que nous utilisons en cours), et ce projet m'a permis d'approfondir encore ces connaissances. Le seul fait d'utiliser ce langage dans un projet de cette taille est déjà une expérience enrichissante. J'ai aussi eu l'occasion de mettre en pratique certains aspects que je ne connaissais pas ou mal : métaprogrammation, scripting et templating (avec l'outil de génération de fichiers), outils de gestion de package, déploiement, etc. Il y a aussi évidemment la librairie Qt avec laquelle j'ai eu du plaisir à travailler. J'ai beaucoup apprécié sa puissance et la simplicité de sa conception.

En bref, ce projet m'a apporté beaucoup de connaissances pratiques qui pourront être utiles à l'avenir.

8.5.4 Thomas Schweizer

Il s'agit de la troisième fois que j'endosse le rôle de chef de projet. Cette fois ci est particulière car nous utilisons une méthode agile de notre plein gré et nous avons essayé de nous tenir à son application tout au long du projet. Ce n'est pas facile et demande de la rigueur de la part de tous les membres du projet. Je suis cependant satisfait de la tournure qu'a pris le projet à la fin. En effet, le logiciel fonctionne correctement. Je suis aussi content de voir que le système agile a permis de réaliser une grande partie de la documentation au fur et à mesure du projet et a contribué à une organisation efficace du travail.

Sur le programme lui-même, je suis satisfait du résultat. Nous avons un programme peu commun qui fonctionne bien et qui permet de résoudre des problèmes divers. Pour un premier essai dans le domaine, je trouve le résultat satisfaisant.

Sur le plan technique, j'ai pu apprendre à utiliser QML et (ré)apprendre à utiliser Qt. J'ai encore travaillé sur les interfaces graphiques ce qui ne me dérange pas car ce projet et tellement différent du précédent qui consistait à développer une application de gestion de budget. J'ai pu aussi expérimenter quelques surprises avec l'utilisation de Qt et de Python notamment le fait que je ne puisse pas faire communiquer un thread python et le thread Qt.

En conclusion, c'était un très bon projet sur lequel j'ai appris un nouveau langage et de nouvelles technologies qui me serviront sûrement plus tard dans ma carrière. Mes collègues étaient très compétents et ce fut un plaisir de travailler avec eux sans conflits.

8.5.5 Marcel Sinniger

Le projet *Propagation Simulator* m'a permis d'apprendre un nouveau langage de programmation, d'utiliser d'autres technologies pour la rédaction du rapport et des notes et à découvrir une autre méthode agile.

Dans le cadre de *Propagation Simulator* j'ai pu découvrir le monde de Python. Le fait que Python se base sur l'indentation du code m'a beaucoup fasciné. Au début, j'ai vraiment apprécié le typage dynamique de Python, mais au fur et à mesure, j'ai rencontré les limites et les inconvénients. Surtout en lisant le code des autres, j'avais un peu de peine à comprendre leur code parce que les types n'étaient pas visibles, ni dans les déclarations des méthodes ni dans les définitions des attributs. Il fallait plonger dans le code, même dans d'autres fichiers, pour pouvoir comprendre à quel type un objet correspondait.

Les technologies LaTeXet *Markdown* étaient deux outils que je ne connaissais pas auparavant. Grâce à ce projet, j'ai eu l'occasion de découvrir ces outils. J'ai constaté que les deux permettent d'augmenter la productivité étant donné que la rédaction et la mise en forme sont fortement séparées. Néanmoins, j'ai malheureusement dû constater que les correcteurs d'orthographe ne sont pas aussi sophistiqués que dans *Microsoft Word* par exemple.

La collaboration de notre groupe m'a bien plu. Chaque membre de notre groupe était très motivé. Les réunions hebdomadaires ont fortement influencé la communication et la motivation du groupe. Les rétrospectives, surtout sur la coopération de notre groupe, après chaque release nous a permis de proposer des améliorations pour les prochaines releases. Ceci augmentait la satisfaction puisque tout le monde pouvait s'exprimer et mettre en évidence des points non optimaux.

Depuis le début du projet je me suis occupé du déploiement. Après chaque release j'avais la responsabilité de créer un paquetage du logiciel. Pour la première release, c'était un vrai défi de faire tourner l'application parce que toutes les ressources dont le logiciel avait besoin n'étaient pas toujours automatiquement inclues par PyInstaller. De plus, PyInstaller ne marchait pas avec quelques raccourcis de syntaxe de Python. Ce n'était donc pas aussi facile que PyInstaller le fait croire sur son site web. Comme nous supportons Windows et Linux, j'ai aussi pu toucher le monde Linux que je ne connaissais pas très bien auparavant.

Le fait que nous n'avons pas eu de grand problème technique pendant la phase de développement grâce à la phase d'analyse m'a fasciné. Nous avons réussi à éliminer les incertitudes avec les prototypes. A l'avenir je serai conscient que l'investissement dans la phase d'analyse et la phase de prototypages apporte un très grand gain au projet en éliminant les inconnues au niveau des technologies.

Pour conclure, ce projet m'a permis de toucher plusieurs mondes que je ne connaissais pas auparavant et ce fut un grand plaisir.

Annexe A

Procès verbaux

Ci-joint, la liste des procès verbaux effectués dans le cadre du projet pour synchroniser et planifier les séances de groupe. Elles sont transcrites telle quelles du document original et utilisent un vocabulaire informel. Les procès verbaux étaient écrit par M.Schweizer durant la réunion. C'est pour cette raison que la description de ce qu'il a fait n'est pas toujours dans les procès verbaux et qu'elle réfère à son journal de travail : Il ne pouvait parler aux autres membres et écrire en même temps. Son journal de travail est inclus à la fin des procès verbaux.

A.1 Procès verbaux

A.1.1 PV - 18.10.2013

Avancement individuel

Grégoire
Pas eu le temps de faire quoi que ce soit.

Kevin
Modèles de prototypes, écritures des conventions de codage. Il annonce que nous pourrons rajouter des choses à mesure du développement.

Hadrien
Pas fini formellement les tests unitaires mais sinon tout est fait. Le prototype de structure est fini.

Marcel
Bien avancé sur le déploiement il a trouvé une solution qui marche pour les 3 grandes distributions actuelles.

Thomas
Fini l'item d'environnement de dev avec un peu d'essais avec le QML.

Changements de planification

Nous avons décidé d'ajouter deux prototypes : appel fonction et outil en ligne de commande pour ajouter simulation, entités, terrains, etc... Le premier aura pour but de mettre en place le moyen que nous allons utiliser pour effectuer l'appel aux actions que l'utilisateur va définir. Le deuxième a pour

objectif de réaliser un petit utilitaire en ligne de commande afin de créer la structure d'une simulation, entité, terrain ou autre dans le programme. Deux items ont étés mis dans la pile "Done".

Autre

Nous avons définit les répondants de chaque partie du projet afin de savoir en interne qui doit s'adresser à qui pour proposer quelque chose.

A.1.2 PV - 1.11.2013

Ordre du jour

- Qui a fait quoi ?
- Affectation des items de la release 1.
- Démonstration des prototypes (2min/pers/proto)

Avancement individuel

Hadrien
Prototype structure programme et configuration => Fini avec élaboration d'un pseudo langage de script pour définir une action + manuel.

Tests unitaires => Tests unitaires de la classe des méthodes et de l'utilisation de la classe avec Nose test.

Cela l'a amené à plusieurs trucs dont le moteur de simulation : lit les fichiers, parse => classe entité. => permet la surcharge d'autres classes et manipulation !

Kevin
Conventions, prototypes canevas et bruit. Script de création des fichiers. Tout est fini. Chaque membre du projet doit lire les conventions python. Prototype bruit/canevas => Voir si le nous arrivons à afficher et rafraîchir pour que ce soit utilisable. Le bruit marche correctement => réglage des paramètres. Paramètre de seed.

Script => tout ok, la syntaxe doit être adaptée en fonction de notre structure finale. 5 heures effectuées sur 3 heures de base.

Grégoire
Documentation => tout est bon et utilisable, les modèles et commandes sont sur le readme avec les indications comment utiliser les commandes.

Threads et processus => comparaisons et exemples pour l'illustration. Possible d'utiliser des signaux et events. Il s'est avéré que les threads ne sont pas parallèles mais les processus le sont mais cela prends longtemps à communiquer entre les processus.

Marcel
Variantes d'installation avec les packages : pyinstaller et pxfree. Il y a regardé pyinstaller en premier et testé l'installation avec les prototypes que nous avons utilisé. Des problèmes ont été rencontrés lors de l'installation sur Windows lorsque nous utilisons d'autres packages.(numpy, noise)

Il a trouvé une solution pour l'installation.

Problème avec les threads et process pour Windows mais résolu en ajoutant une ligne.

Thomas
Comprendre technologie QML. Plusieurs prototypes voir journal.

Autre

Définition des items pour la release en cours (release 1).

- Langage Python (base) et conventions.
- Environnement de documentation.
- Release github.

Nous utilisons Python 2.7, Nose, Qt 4.8, numpy. UTF8 Tout le monde !

A.1.3 PV - 8.11.2013

Ordre du jour (faire vite l'administratif !)

- Qui a fait quoi ?
- Démos éventuelles
- Ajouter les items de programmation
- Concertations d'analyse, pair programming.

Avancement

Marcel
PyInstaller. Cas d'utilisations. Pseudo code.

Grégoire
Diagramme fait communément, lancement de simulation en interface graphique ou en console (arrière plan)

Gestion des variables lorsqu'une est utilisée par plusieurs actions.

Hadrien
Item parser et révision du modèle pour être plus léger. + Tout ce qui est commun.

Kevin
Structure faite pour fichiers et dossiers. Mise en place de l'architecture de dév (dossiers et fichiers user)

Thomas
Voir journal de travail.

Autre

Définition des items pour la suite de la release 1.

A.1.4 PV - 15.11.2013

Ordre du jour

- Avancement,
- Nouveaux items.

Avancement

Marcel
Travail sur le builder mais rencontre de beaucoup de problèmes.

Hadrien
Portage du prototype sur l'application, commentaires, rendre plus léger, tests (mais pas fini), système d'exceptions.

Kevin
Interfaçage : OK Swap variables : OK Getter/setter dynamiques : OK Nouvel item d'intégration de templates.

Grégoire
Pont GUI - Sim et définition de la base du simulateur.

Thomas
Voir journal de travail.

Autres
Discussion du format de rendu pour les diagrammes : nous avons adopté Visio.

A.1.5 PV 22.11.2013

Ordre du jour

- Avancement individuel
- Ajout de l'item de création de release.

Avancement

Marcel
Correction et avancement de la liaison. 3h.

Hadrien
Les tests marchent ainsi que les chemins. Documentation. Manuel utilisateur. Pré requis. en tout 3h.

Kevin
Nada.

Grégoire
Mise en ordre pont simulateur / interface. Stat items. Modèle de domaine simulateur. Doc : 1h30 Code : 7h30

Thomas
Voir journal.

Autre

- Ajout item de release.
- Tout doit être prêt et nettoyé.
- Pour Lundi qui vient, il faut que chaque membre choisisse les items de la release 2.

Items choisis au 25.11.2013

Thomas

- Interaction entité-entité.
- Toutes les interactions sont configurables.
- Récupération des données de simulation pour les intégrer sur l'interface
- Redimensionnent de la fenêtre.
- Échelle de rendu.
- Préparation de l'interface pour un affichage complexe des pixels. (grandeur, etc..)
- Documentation.
- Lorsque la fenêtre est fermée, cela stoppe aussi le simulateur.

Hadrien

Syntaxe interaction

- Créer le parser pour les interactions
- Mettre en place la syntaxe (idem que les actions)

Syntaxe générale

- Actions surchargées ne devraient pas avoir de clause with
- Sucres syntaxiques : with : "*", pas de with == with : null, appel de fonction sans paramètre simplifiées.

start.json

- Dossier d'export
- Mode de map : alea ou perlin
- Seed pour la map

Installation

- Déploiement du user_data si il n'existe pas (avec simulation d'exemple)
- Logos pour exécutable :D

Efficacité

- Nettoyer la mémoire après le parsing : del Builder, del Parsers, del Metaclasses

Sécurité

- Renforcer les contrôles de syntaxe
- Messages des exceptions plus potables
- Vérifier les attributs obligatoires du start.json
- Bloquer les double lancements du simulateur.

Terrains

- Couleur de terrains (Les terrains deviennent une classe ou un dict)

Entité

- Couleur des entités
- Méthode de génération des entités surchargeable (surcharge de méthode statique)

Outil en ligne de commande

- Création de simulation
- Création d'entité
- Création de actions
- Création d'interactions
- Création de Trigger (fichier .py)

Marcel

- Cas d'utilisation
- Documentation

- Ajouter la possibilité (dans le code) d'avoir des interactions entité-entité configurable.
- Améliorer la génération automatique de la carte (comme la figure 6 du cahier des charges)
- Ajouter les stats de la simulation dans le GUI (colonne à droite, selon Figure 6 du cahier des charges)
- Configuration de la simulation
- Légende de types d'entité
- Légende de types de terrain
- Ajouter les stats de la simulation dans le GUI (en bas, selon Figure 6 du cahier des charges)
- Ajouter le temps écoulé
- Ajouter le Framerate Configurer (dans les fichiers de config) la simulation de malaria.
- Ajouter terrain "à risque"
- Ajouter un terrain du type "infranchissable"
- Tenir compte du terrain à risque dans l'action d'infections des humains
- Tenir compte du terrain infranchissable dans les actions de déplacement (humain et moustique)
- Ajouter une interaction "écraser un moustique" dans l'humain -Taille de la carte configurable à l'aide d'un ficher de config
- Parser/Builder : Permettre la configuration de la taille de la simulation (x * y) dans un fichier de config et passer cette info dans Params
- GUI : récupérer la taille de la simulations depuis l'objet Params

Grégoire

- Amélioration scripts simulation moustiques (release 1 + inclure 2). Déplacement intelligent, interactions.
- Gérer correctement les StatItems
- Amélioration des couleurs par défaut (terrain en sombre-terne, entités en couleurs vives)
- Affichage adaptatif, mettre à l'échelle le rendu.
- Gestion des couleurs par l'utilisateur
- Ajouter les interactions dans les scripts => parser / loader (interactions entité-entité, intérationc entité-terrain = action)
- Gestion positions des entités plus poussées pour appeler rapidement les interractions sans parcourir la liste trops souvent.
- La doc...encore :D
- Mettre de l'ordre dans la doc actuelle.
- Fichier de config (tailles map, nom simulation)
- Nom simulation en barre de titre. If time...
- Préparer les logs de données

Kevin

- Mettre à jour script de création de fichiers avec structure release 1 + interactions.
- Gestion des statistiques
  - Côté simulateur
  - Côté scripts utilisateurs (?)
- Gestion des interactions
  - Définition par JSON
  - Traitement côté simulateur
  - Définition par script Python
- GUI
  - Régler affichage des stats
  - Afficher les entités selon couleurs paramétrées

  - Afficher légende des couleurs des entités et terrains -Faire une fermeture propre du programme (message de confirmation si simulation en cours + fermeture des processus enfants)
- Autres items
  - Paramétrer couleur des entités
  - Initialisation de la map par Perlin avec seed
  - Séparer seed de la map et seed de la simulation
  - Nettoyer mémoire de builder et loader au start
  - Compléter la simulation de démo (avec les interactions)
  - Doc

A.1.6 PV 29.11.2013

Ordre du jour

- Retour sur la release 1. (Ce qui était bien et ce qu'il faut corriger.)
- Tri et affectation des items de la release 2.

Retour sur le release 1

Manque de documentation finale.

Pile et diagrammes => les membres Tests unitaires ? Utilisable dans certains cas. => Le simulateur et l'interface ne sont pas forcément testables : faire des tests manuels simples mais documentés. Documentation => README à lire.

20 minutes de présentation.

Tri des items

Tout le monde a fourni des items. Bien joué. Tout est trié.

A.1.7 PV 06.12.2013

Ordre du jour

- Vérifier pour le nombre d'heures inscrites
- Avancement individuel

Avancement

Marcel
Rien

Kevin
A fait un truc pour Marcel.

Hadrien
Syntaxe interaction + début parseur et méthodes.

Grégoire
Fermeture propre.

Thomas
Nom de la simulation Fermeture propre. Affichage des légendes.

A.1.8 PV 13.12.2013

Ordre du jour

- Affecter les chapitres de documentation.
- Avancement
- Contrôle des durées.

Avancement

Hadrien
Interactions + refactoring Ajout de paramètres dans start.json. Surcharge d'attributs (couleurs, nom) Modification de la vue pour afficher la couleur des entités.

Kevin
Couleur des terrains. Documentation.

Marcel
Logo pour le simulateur (développement et pyinstaller) Essais avec le garbage collector et les import. 1 heure.

Grégoire
Relecture de documentation et segmentation de la structure du rapport. Début de l'implémentation des interactions.

Thomas
Voir journal de travail.

A.1.9 PV 20.12.2013

Ordre du jour

Raclette, n'aura pas lieu.

A.1.10 PV 27.12.2013

Ordre du jour

Avancement Affectation des items

Avancement

Grégoire
Suppression des entités.

Hadrien
Ajout de la création des entités. Ajout de deux simulations utilisateurs. => découverte de problèmes graphiques => entités immobiles et ordre des labels. Réglage des problèmes de droits Exceptions utilisateurs. => plein de problèmes.

Marcel
Rien

Kevin
Rédaction du game of life. Détails à régler dans la structure du document.

Thomas
Voir journal perso

Autre

Ajout des items

A.1.11 PV 2.01.2014

Ordre du jour

Avancement Re répartition des items.

Avancement

Grégoire
Tests sur l'écrasement des attributs. => reviens au même que faire un écrasement 1/2. Pas de lissage. => Refaire un écrasement 1/2 dans le code.

Problème de doublons d'entités.

Hadrien Froger
Réécriture des tests unitaires. Rendu des exceptions plus joli.

Marcel Sinniger
Génération de la carte manuellement Syntaxe faite avec Kevin.

Kevin Jaquier
Game of life. => presque fini. Gros problèmes sur certains bouts de syntaxe. Problèmes avec les cases adjacentes.

Thomas Schweizer
Voir journal de travail.

Autres

A.1.12 PV 10.01.2014

Ordre du jour

- Avancement / Pile
- Modalités de fin du projet.

Ce que doivent faire les membres

- Lister les améliorations qu'ils ont ajoutées au programme par rapport au cahier des charges initial.
- Les améliorations qu'ils souhaiteraient pour améliorer le projet.
- Retour sur la gestion de projet.
- Les fonctionnalités non finies.

Avancement

Marcel
Génération complète d'une installation.

Kevin Jaquier
Rien

Grégoire Decorvet
Rien

Thomas
Voir journal.

A.1.13 Journal de travail

Thomas Schweizer

Semaine 0
Rendu cahier des charges

Semaine 1
Travail sur les documents de gestion de projet mais surtout sur l'environnement de travail pour le développement.

Semaine 2
Travail sur le document de gestion de projet avec les piles => rédaction PV + mise à jour doc xlsx avec des graphes suite à la discussion avec Hadrien.

lundi

PV effectué

Préparation de l'espace de graphes pour les autres releases ainsi que la structure des piles.

22.10 : Débroussaillage de qt et qml avec pyside, c'est pas facile pour un nouvel arrivant... 1heure

23.10 : Ajout de l'exemple sur les threads en python. Ajout d'un exemple avec mise à jour dynamique du contenu.

Prototype de grille qui m'a pris un temps fou (2heures) pour accéder à un élément enfant car c'est pas la même chose que dans Qt de base et que la documentation n'est pas explicite la dessus !.

Okay, la génération dynamique d'éléments n'est pas une bonne technique. C'est déconseillé par les devs.

Design avec colibri, il me reste des erreurs.

Suppression des erreurs, ajout de menus dynamiques dans le code.

Semaine 3
Ajout dans le proto colibri un exemple de modification dynamique d'un élément.

—

PV et mise à jour des items de la release 1. Structure du dossier pour le programme et fichiers de projet. Ajout des environnements virtuels.

Ajout d'un exemple de test à la racine, je le mettrai dans le bon dossier demain.

Semaine 4
Ajout d'un fichier batch pour lancer la série de tests nose sur Windows. Essais pour faire fonctionner nose lorsque le test est lancé depuis un sous dossier.

Modification de la structure des dossiers d'environnement virtuel pour avoir un dossier Windows et Linux.

Modification des buildsystem sublimetext pour le projet afin d'indiquer les variantes de plate-forme.

Essai d'installation de nose et numpy

Résolution du problème pour les tests et leur exécution depuis test.bat grâce à Hadrien.

Mise à jour de l'environnement virtuel pour Windows.

Avance sur l'intégration de l'image sur le QML.

Tentatives d'intégrations.... (2 heures). Suppression partielle de l'environnement virtuel.

Affichage d'une fenêtre avec du vert ! Linkage entre QML - Python complet

Avance encore du protographique. L'image provider répond correctement maintenant et Ce que j'affiche est bien la fenêtre !

—

Mise en place du rafraîchissement de la fenêtre à l'aide d'une solution temporaire.

Création d'un diagramme Visio pour l'analyse de l'interface et de son interconnexion avec le reste du programme. 0.5h

Création des fichiers et mise en place du dossier de dev pour les interfaces graphique, mise à jour Visio. 0.2h

Semaine 5
Discussion et établissement de l'interface entre le simulateur et l'interface graphique.

—

Implémentation de l'interface. 2 heures. PV. Implémentation de l'interface #2 1.5 heures.

—

Implémentation de l'interface dans le train. Je suis arrivé à une solution mais j'utilise un Timer fourni par le QML. C'est transparent au niveau Python mais je me suis rabattu sur cette solution car la solution avec les signaux me faisait un débordement de stack à cause des signaux échangés pour avertir qu l'un doit être mis à jour et que l'autre à fini de load. 1h

Ajout de la désactivation du bouton stop si la simulation n'est pas en train de tourner.

Ajout de la console ! (Fonctionne seule : à tester sur l'entier du programme) (1 heure)

Semaine 6
Continuation du travail sur la console. 0.5h Modification de la politique de gestion du QApplication. Il a été déplacé dans le fichier d'amorce. 0.5h Finition de la console et ajout d'un canal supplémentaire. 0.5h

—

Liaison des commandes start, stop et pause de l'interface au simulateur. 0.5h Ajout des statistiques à l'interface. 0.1h

Travail sur la rédaction de l'introduction et remaniement de la structure de la conclusion. 1h

Rédaction et structure de la conception globale. 0.5h

Liaison du simulateur au gui lorsque l'image est chargée.

Rédaction du chapitre de gestion de projet. 0.5h

Logique simulateur avec interface 0.5h

—

Travaille sur le rendu 3h. Nettoyage et tentative de mise à jour du panel de droite (echec) : 0.5h

DOCUMENTATION !!!!! 3h.

Semaine 7
Ajout d'un tag sur git et suppression de la branche. 0.5h Les statItem s'updatent dans la vue. Je suis pas content avec ma solution mais au moins elle marche. 3h

Groupement des items proposés. 0.5h

Travail sur la fermeture brutal de la fenêtre. 0.5h

—

Semaine 8
Traitement pile du lundi. Fermeture propre du simulateur Nom de la simulation dans la fenêtre. Légendes 0.8

—

Amélioration de l'algo de rendu, utilisation de la couleur des terrains Les entités qui ne bougent pas proviennent du rendu à cause d'une condition mal faite. 0.5h Essais sur l'affichage pour enlever le timer. 2h Afficher les légendes pour les entités. 0.1h

—

Redimensionnement de la fenêtre avec bandes noires. (2h) Ajout du framerate(1h)

Doc Introduction (0.1h) Doc Gestion de projet (0.5h) Doc conception (0.5h) Doc Réalisation (0.5h) Doc Etat des lieux 0.5h (conflits !!!!!) Doc Conclusion 0.5h

Semaine 9
— Relecture document 0.5 Modification du comptage de frames 0.1 Ajout bouton d'exportation. 0.1 Modification aspect de la liste des légendes 0.4 Travail chef de projet 0.5h

Semaine 10
— Travail chef de projet 0.3h Suppression timer de rafraîchissement + nettoyage interface graphique. 1h

Semaine 11
Tri des labels Console pour l'affichage des exceptions lors de l'exécution du simulateur. Commentaires de la partie graphique.

Semaine 12
Arborescence du dossier de rendu. Documentation