RAPPORT DE PROJET

Ce document contient le rapport du travail effectué par l'équipe composé de: Andy Martel, Hong Ngoc Nguyen, Thibault Patois, Quentin Rollot et Mathieu Scala le chef de projet. Ce rapport est destinée au jury de notation pour nous assurer le A au projet

Table des matières

Description du projet :

Le but du projet était de programmer un jeu vidéo tactique au tour par tour de simulation de guerre le plus complet possible en JAVA.

Il devait comporter un menu de démarrage, un but, des éléments à débloquer, une campagne et des animations. D'autre part, le projet devait d'abord être programmé sur PC puis si le temps nous le permettait, sur Android.

Le menu du jeu devait comporter plusieurs parties :

- Mode Histoire
- Mode Combat
- Mode Tutoriel
- Options

Le mode histoire devait permettre au joueur de jouer à la campagne du jeu. Cette campagne devait être une succession de mission séparé par des dialogues entre protagonistes de l'histoire.

Le mode Combat devait permettre de jouer sur différentes cartes contre un ou plusieurs joueur (ou IA au choix) au tour par tour.

Le mode Tutoriel devait permettre de revoir les leçons et conseil vu au début du mode histoire et de voir des astuces débloquées au cours du jeu.

Le menu option permettra de régler quelques options comme le son par exemple.

Description du jeu :

Notre jeu se nomme Slatch. Il comporte une campagne scénarisée, composé de 16 missions, d'un mode de partie rapide et un tutoriel reprenant les 8 principaux aspects du jeu.
Lorsque le joueur lance le jeu, il effectue généralement une succession de partie.

Une partie est une bataille sur une carte quadrillée plus ou moins grande contre un ou plusieurs adversaires contrôlés par un autre joueur ou par une IA. Chaque joueur peut choisir sa factions au début de la partie (Militaire ou Robot), possédant chaque une unité spéciale.
Le but d'une partie est en général de détruire toutes les unités ennemies. Mais cette objectif peut changer (surtout dans le mode histoire) comme par exemple :

- Détruire la base ennemie en moins de 10 tours
- Survivre X tours
- Capturez la (les) base(s) ennemie(s)

Pour atteindre ces objectifs, chaque joueur a à sa disposition plusieurs bâtiments et unités avec leurs rôles.

Les bâtiments

Il existe trois types de bâtiments : les bâtiments, les usines et le QG.

Les bâtiments sont des bases qui rémunère le joueur à chaque tour. Plus le joueur possède de bâtiments, plus il gagnera d'argent au prochain tour.

Les usines sont des camps d'entraînements. Le joueur peut y construire toutes les unités du jeu (sauf l'unité spécial de la faction ennemi). Les usines rémunèrent également, à valeur égal avec les bâtiment, le joueur.

Le QG est le quartier général du joueur. Comme le reste des bâtiments du jeu, il rémunèrent le joueur. Si le QG du joueur est capturé, ce dernier a perdu la partie.

Les unités

Chaque factions possèdent 6 unités et une unité spéciale. Toutes les unités ont leur avantages et inconvénients face à un autre type d'unité. Par exemple, un démolisseur (infanterie équipé d'un mortier) est efficace contre les tanks mais très inefficace contre les infanteries.

D'autre part, toutes les unités ont leur propre type de déplacement et un nombre de cases de déplacements propre. Par exemple une infanterie peut se déplacer sur la montagne et la forêt, tandis qu'un tank ne peut ni se déplacer dans la forêt ni dans la montagne.

Chaque unité possède également une vision dans le brouillard de guerre. Cette vision dépend de son déplacement.

Enfin à chaque combat, une unité gagne de l'expérience en fonction des dégâts qu'elle a infligés. Si elle acquière suffisamment d'expérience elle monte de niveau et devient plus puissante.

Les terrains

Il existe 5 grands types de terrains : Les plaines, la forêt, la montagne, l'eau et la route. Il existe également des variantes de ces terrains dans l'environnement désert. Par exemple la montagne devient une dune dans un environnement de désert.

Chaque terrain possède une couverture et un coût de déplacement propre.

La plaine est le terrain de base du jeu. Le déplacement standard de chaque unité se base sur le coût de déplacement de la plaine. La couverture de ce terrain sert également de base pour les autres terrains.

La forêt est un terrain qui ne peut être traversé que par les unités à pied ou à roues (infanteries par exemple). Il offre une meilleur couverture que la plaine mais son coût de déplacement est plus élevé.

La montagne est un terrain qui ne peut être traversé que par les unités à pied. Il offre la meilleur couverture possible. Cependant son coût de déplacement est également le plus élevé.

L'eau est un terrain inaccessible par toutes les unités du jeu.

La route (ou les ponts) est un terrain dont le coût de déplacement est inférieur à celui de la plaine pour les unités mécaniques. Il permet un déploiement plus rapide des

troupes mais sa couverture est faible.

L’intelligence Artificielle

??

Les Graphismes

L'ensemble des images du jeu a été réalisé par notre équipe en utilisant le logiciel The Gimp.

Déroulement d'une partie

Le joueur a la possibilité de choisir parmi un grand nombre d'option avant de lancer une partie. Il peut par exemple choisir d'activé ou non le brouillard de guerre ou les animations. Il peut également créé des équipes.

Une partie se déroule généralement sur plusieurs tour. Le joueur achète ses unités et les envois ensuite vers la direction du camp ennemi. A chaque tour, le joueur gagne de l'argent et peut ainsi s'acheter des unités plus puissante. Au fur et à mesure des combats, les unités recrutés gagne de l'expérience et le joueur peut décidé de les évoluer.

Finalement lorsque qu'une équipe ou un joueur a gagné, le jeu s'arrête et le joueur arrive sur un tableau qui reprend les statistiques de tous les joueurs.

Récapitulatifs des Semaines

Toutes les vendredi matin, toute l'équipe se rassemblait dans une salle de réunion (comprendre sans ordinateur) pour faire le bilan de la semaine et discuter des fonctionnalités et correction à faire pour la semaine suivante.

Semaine 2

- Panneau descriptif réalisé et fonctionnel
- L'IHM est maintenant décomposé en 2 panels (panel du menu et pannel de la matrice). Création de 2 MouseListener pour ces panels
- Unification de la police pour la barre d'information
- Nouvel algorithme pour le déplacement (diminution de la complexité)
- Chargement de toutes les images au début d'une partie (pour éviter le chargement des images pendant la partie qui diminue la performance du jeu)
- La possibilité d'acheter des unités (+ le menu Shop)
- La possibilité de capturer un batiment
- Ajout de l'eau, route, démolisseur, ingénieur, UML, usine
- La possibilité de traverser les unités alliées
- Les bâtiments alliés soignent au début du tour
- L'ingénieur soigne (et n'attaque pas)
- IA se déplace vers un coordonnée X et Y prédéfini en plusieurs tours

Semaine 3

- Terminer l'ingénieur
- Fin de partie + Statistique
- Map Creator
- Déplacement plus fluide
- IA basique
    - Capture si elle est sur un bâtiment
    - Cherche l'unité la plus proche / bâtiment le plus proche (et unité à soigner le plus proche pour l'ingénieur)
    - Attaque / Capture / Soigne la cible
    - Achète l'unité la plus chère possible. Si elle a déjà suffisamment d'unité du type qu'elle peut acheter, elle économise
- Menu Principal (mais pas encore lié à Partie)
- Sauvegarder / Charger
- Correction de bugs pour pouvoir jouer à 4 joueurs
- Changement de condition de victoire (capture QG)
- Ajout de la moyenne distance
- Système de point d'expérience

Semaine 4

- Nouvelle IA :
    - Fonctionne avec une map d'influence
        - Capture
        - Defensif

        - Offensif
        - Menace
        - Retraite
    - Peut faire évoluer
    - Semble plus performante
    - Trop de temps de calcul sur une map trop grande
- IHM avec animations (non commit)
    - Avancement avec animations avec le Tick (80%) afin de remplacer les thread.sleep()
- Affichage des dégats/soins lors des combats
    - Mais ne reste que si on ne rafraichit pas !!
- Brouillard
- Equipe
- Debut de campagne
- Menu lié au reste (compléter à 45%)
- Unité Robot (80%)
- Affichage PVMax, Exp dans le menu descriptif

Semaine 5

- Animation finie
- Système du mode Campagne est fini, il ne reste plus qu'à remplir le scénario
- Portage Android à 50%
- Nombreuses nouvelles cartes
- Quelques améliorations de l'IHM

    - Affichage de tous tous les membres de l'équipe gagnante
    - Le statistique refait entièrement
    - Pause entre deux tours de joueur
- Possibilité de retour au Menu principal
- Menu principal fini à 99%

Semaine 6

- Correction d'un grand nombres de bugs
- Campagne terminé
- Menu principale terminé
- Tutoriel fini

Fonctionnement du Jeu

Intelligence Artificielle

Notre jeu possède deux intelligences artificielles distinctes. Elles représentent chacune une étapes de notre avancement. L'intelligence artificielle est notre première IA tandis que l'intelligence Artificielle Réfléchie est notre IA final.

Nous allons détaillé dans la suite le fonctionnement de ces deux intelligence artificielle.

Intelligence Artificielle Agressive

Intelligence Artificielle Réfléchie

L'intelligence artificielle réfléchie fonctionne par palier. Il y a 3 paliers avec pou chacun une fonction propre. Nous avons dans l'ordre des grades du plus élevés au plus faible : StrategieIA, puis OperationIA et enfin UniteIA. Leur complexité est inversement proportionnel au grade. Par exemple StrategieIA est plus complexe que UniteIA.

Nous allons détaillé ici le fonctionnement de chacune des classes qui constitue cet hiérarchie.

StrategieIA

C'est la classe mère de l'IA, c'est elle qui prend les grandes décisions comme par exemple attaquer l'ennemi ou défendre. C'est également elle qui va aider au placement des unités en fournissant à l'IA inférieur une matrice d'influence.

Cette IA procède de la manière suivante :

1. Elle actualise le mode dans lequel elle se trouve, qui décrira un comportement aggressif ou défensif.
2. Elle va remplir une matrice d'Influence que nous décrirons plus bas.
3. Après cela, elle donne l'ordre à OperationIA d'acheter des unités.

4. Ensuite, elle parcourt la liste des unités du joueur, et elle demande à OperationIA de jouer en lui passant une copie de la matrice en paramètre
5. Elle vérifie pour chaque unité si elle est morte, si oui elle la supprime de sa liste.
6. Elle lance ensuite les animations.

OperationIA

Cette IA se situe sous la StrategieIA. Elle s'occupe de donner des ordres au unités et d'acheté les unités. Elle procède de la manière suivante :

- Pour chaque unités de l'IA :

1. Elle adapte d'abord la matrice reçue à l'unité qu'on traite.
2. Elle va ensuite chercher une bonne case parmi les cases où il y a des bâtiments et des unités.
3. Elle regarde la case qu'elle a choisi, et selon son type elle va donner l'ordre à UniteIA d'effectuer une certaine action.

- Pour l'achat d'unité

1. Compte le nombre d'unités alliées et ennemies.
2. Parcourt les usines avec en priorité les usines proches d'unités ennemies.
3. Donne l'ordre à UniteIA d'acheter en fonction de l'argent disponible et des unités déjà possédées, ainsi que des unités ennemies.

UniteIA

Enfin UniteIA est IA la plus basse dans la hiérarchie. Elle n'effectue que les ordres demandé.

Elle reçoit donc les ordres de OperationIA, les décrypte et appelles des fonctions propres à moteur pour effectuer les ordres.

- Pour le déplacement, elle va simplement demander à Moteur de déplacer l'unité en question, et elle va la mettre en état déplacée(même si elle ne l'a pas été)
- Pour le soin, l'attaque et la capture, elle vérifie si la cible est déjà à portée. Sinon,

elle se déplace vers elle. Si elle n'est toujours pas à portée, elle ne fait rien, sinon, elle fait son action.

- Pour évoluer une unite, elle appelle la fonction evolue() de Moteur..

La classe Influence :

Elle contient cinq attributs entiers : offensif, defensif, capture, menace et retraite. Ce sont des scores qui pourront être pondérés selon le type de l'unité afin de calculer le score de la case.

Offensif est plus élevé si une unité se trouve à portée d'attaque, si elle est vulnérable à notre unité, si une unité ennemie se trouve sur une case quelconque.

Defensif sera renforcé par la couverture offerte par le décor, et les unités alliées.

Capture sera élevé pour les bâtiments ennemis, et encore plus sur les bâtiments à portée, c'est pondéré en fonction de la distance.

Menace sera généré par les unités ennemies, et est censé permettre aux unités d'éviter les cases trop dangereuses et de préférer d'autres cases plus « safe »

Retraite est comme défensif avec les bâtiments alliés en plus, utile pour retourner se soigner sur ses bâtiments.

Initialisation d'une phase de jeu

Partie

Partie est la classe qui permet de lancer une partie. Elle créé le terrain ainsi que toutes les unités. Elle s'occupe également de gérer les tours, de la sauvegarde du jeu ainsi que le joueur actuel (comprendre le joueur qui est actuellement en train de jouer). C'est également cette classe qui définit lorsque la partie est finie et désigne le joueur gagnant. Elle stocke des attributs essentiels au bon déroulement de la partie comme la liste des joueurs, la taille de la carte, ou le joueur gagnant.

Toutes les cartes disponibles dans le jeu sont sauvegardées dans l'enum Map qui contient les principales caractéristiques des maps comme sa taille, le nombre de joueur, la description, etc ... La map en elle-même est stockée dans un fichier texte. Ces fichiers

textes sont tous construit de la même façon. Chaque ligne est composé d'un identifiant (montagne, forêt), de ses coordonnées et du joueur a qui appartient l'identifiant.

Lorsqu'une nouvelle partie est instancié, on lit le fichier map correspondant à la carte souhaité et on instancie les équipes, les joueurs et le terrain.

Pour des raisons de simplicité, chaque type de partie (campagne ou partie rapide) possède sont constructeur. En effet, lors d'une partie rapide, le joueur doit pouvoir définir un certains nombres d'options comme sa faction, les équipes et des options de jeu (activé ou non le brouillard). Certaines de ces options imposés dans la campagne.

Campagne

La classe campagne est un ensemble de méthode qui permettent une succession de mission et de dialogue. Ce tout forme évidemment la campagne du jeu.

Chaque niveau de la campagne est précédé par un dialogue. Ces dialogues sont enregistrés dans des fichiers textes. Lorsque le joueur sélectionne la campagne dans le menu principale, les dialogues se lancent et se succèdent grâce à un clique sur l'écran.

A la fin du dialogue, on appelle une méthode qui initialise la partie correspondante au niveau actuel et remplace les dialogues par les panneaux de la partie, avec un écran spécial qui précise l'objectif de la partie.

A la fin de la mission, quand on clique sur le panneau qui affiche des statistique du niveau, une méthode suite est appelé. Elle analyse le résultat de la partie et décide en conséquence si le joueur peut continuer la campagne. Si c'est la cas, on affiche de nouveau les panneaux de dialogues avec les dialogues correspondant, sinon, on recommence le niveau.

La liste des missions de la campagne est stocké dans un liste, créé dans la classe mère Slatch.

Quand on arrive à la fin des niveaux du jeu, un dialogue spécial est créé pour féliciter l'utilisateur et lui permettre de retourner au menu principale.

IHM

Conception

Au début du projet, l'interface présente lors d'une partie était réalisée sur un seul panel : l'affiche ainsi que les coordonnées

Dans un souci de simplification, l'interface graphique a été modifiée afin de composer 2 ensembles :

- une barre d'information située en haut de la fenêtre
- le plateau du jeu

Ces deux parties ont été réalisées sur deux panels différents et sont disposés en utilisant un BorderLayout de façon à s'adapter au changement de taille de fenêtre. Cette conception permet d'utiliser deux mouse listener. Ainsi lors d'un click de sourie, les coordonnées récupérées sont celle du repère local du panel.

Affichage du plateau de jeu

L'affichage du plateau de jeu est réalisé en utilisant les apports de la programmation orienté objet. En effet, Thibault et Ngoc ont réalisé des objets « Unite » correspondant aux différentes unités et les objets « Terrain » correspondant aux différents terrains.

Les objets « Terrains », qui possèdent un objet « Unite » en attribut, sont réunis dans une matrice. Ainsi pour afficher la carte, chaque objet Unite et Terrain n'a plus qu'a ce dessiner « tous seul ». L'interface dessine dans un premier temps tous les terrains, puis toutes les unités.

Les autres éléments essentiels sont les différents panneaux :

- le panneau d'action d'une unité (pour se déplacer, attaquer, soigner, capturer ou évoluer)
- le panneau de description du terrain et/ou de l'unité sélectionné
- le panneau de l'usine : pour acheter des unités

Affichage de la barre d'information

Le rôle de cette barre est de fournir au joueur les informations importantes relative à la partie. Ses informations sont fournies par le travail de Thibault qui à créé la classe Partie. Cette barre d'information contient également le bouton SUIVANT qui permet de passé le tour. Lorsque l'on clique sur ce bouton, le passage du tour est réalisé par la classe partie du jeu que si le joueur actuel n'est pas une Intelligence Artificielle.

Utilisation de la souris

Utilisation de deux MouseListener pour récupérées les coordonnées dans le repère local du panel. Ses MouseListener permettent de le clique ou le relâchement du bouton de la

sourie.

Moteur

Algorithme de déplacement

L'algorithme de déplacement est appelé à chaque fois qu'une unité doit se déplacer. Cette algorithme procède de la manière suivante :

1. remplitPorteeDep est appelé. Il appelle d'abord initialiseTabDist
2. Dans cette dernière méthode, toutes les cases sont initialisées à -1(signifie que la distance est pour le moment infinie) sauf celles qui contiennent un ennemi, initialisées à -2(signifie que la cases est inatteignable). La case où se situe l'unité est également initialisée à -2.
3. Ensuite, remplitPorteeDep apelle le fameux algoDeplacement(Unite unite,final boolean porteeComptee).

   Ce dernier va créer une file de priorité de Triplet(possédant la méthode compareTo, cf plus bas pour voir la structure du Triplet) : Les Triplet de distance la plus courte se trouveront dans le devant de la file.
4. Il y ajoute d'abord la position de base de l'unité, puis on boucle tant que la file n'est pas vide
5. On récupère le premier triplet de la file, et on regarde chacun de ses voisins. Si un voisin ne dépasse pas la map, on calcule la distance avec la case en ajoutant la distance actuelle et le coût de déplacement sur la case voisine. Si la portée est comptée, on vérifie si la distance ne dépasse pas la portée de déplacement de l'unité. On regarde ensuite si cette distance est plus petite que celle déjà présente dans tabDist(rappel : -1<=>+infini).
6. Si c'est le cas, on ajoute le point actuel à la case du voisin dans le tableau des prédécesseurs. Si on est en bordure de déplacement et qu'un allié se trouve sur la case, on la rend inaccessible pour éviter le chevauchement d'unités sur la même case.

   Si ce n'est pas le cas, on actualise tabDist pour le voisin et on ajoute le Triplet correspondant dans la file.
7. Si on a une unité ennemie présente sur la case du voisin, on lui met comme prédécesseur la case actuelle si on a trouvé une distance plus courte

qu'avant. Cela permet à l'IA de se diriger vers une unité ennemie.

8. On revient dans remplitPorteeDep, on met les coefficients de tabDist correspondants aux unités alliées à -2 pour qu'on voie bien que les cases sont inaccessibles.

Structure d'un Triplet :

- 3 Attributs, d pour la distance, x pour la coordonnée x de la case correspondante, x pour la coordonnée x de la case correspondante
- Plus d est petite, plus un Triplet est grand. compareTo va donc comparer les d de deux triplets.

Attaque

Lorsqu'une unité se trouve a proximité d'une unité ennemi, le joueur peut décider de l'attaquer. L'attaque est un point important du jeu. En effet, en détruisant toutes les unités ennemies, le joueur peut gagner. Cette méthode est donc le lieu de nombreuses vérifications et appelle de méthode.

Tout d'abord, l'appelle de la méthode attaque, appelle une première méthode qui va calculer le nombre de dégâts sur la victime.

Ce calcul se fait sur plusieurs paramètres. La formule est la suivante :

dégât = dégât de base de l'unité * efficacité de l'unité contre l'unité ennemi * bonus de Terrain * malus de vie.

Explication de chaque facteur :

- dégât de base de l'unité : Chaque unité possède un attribut correspondant aux nombres de dégâts qu'il peut infliger. Cette valeur peut changer en fonction du niveau de l'unité.
- Efficacité de l'unité : Chaque unité est plus au moins efficace contre un autre type d'unité. Ces caractéristiques sont contenu dans un Hashmap.
- Bonus de terrain : Chaque terrain offre une couverture à l'unité qui se trouve dessus. La plaine est le terrain de base.
- Malus de vie : Lorsqu'une unité possède tous ses points de vie, elle inflige 100 % de ses dégâts de base. Plus son nombre de point de vie est faible plus ce

pourcentage diminue.

Une fois cette étape terminé, on vérifie que l'unité attaquée est toujours en vie. Si c'est le cas, cette dernière contre-attaque en utilisant la même formule que précédemment avec un facteur 0,7.

Si une unité meurt pendant ce processus, on appelle une méthode qui vérifie si le joueur possède encore des unités. Le cas échéant, le joueur meurt.

Capture

La capture de bâtiment est également un élément essentiel de notre jeu. Cela permet notamment un gain d'argent pour chaque joueur.

La capture des bâtiments n'est disponible que pour un certains nombres d'unités : seul les Commando et les démolisseurs le peuvent (ainsi que les ingénieurs).

Pour capturer un bâtiment, il faut réduire son nombre de point de vie à 0. Pour cela, lorsque l'unité placée sur le bâtiment appelle capture, elle lui enlève un certains nombres de point de vie.

Ce nombre de point de vie est calculé en fonction de la vie de l'unité et du type de l'unité. Par exemple, une unité qui possède 100 % pour ces points de vie aura un facteur 1. Si son nombre de point de vie diminue, le facteur diminuera en conséquence.

A ce facteur, on multiplie un coefficient : 10 pour un commando et un ingénieur et 15 pour un démolisseur.

Finalement, un commando avec 100 % de ses points de vie, enlèvera 10 points de vie à un bâtiment.

Map Creator

Pour créer les différentes cartes du jeu qui servent aux parties rapides et à la campagne, nous avons décider de créer un créateur de cartes, qui est représenté par la classe MapCreator.

Pendant la phase de développement, un bouton Map Creator est présent dans le menu principal pour permettre aux développeurs de constituer leurs cartes facilement. Pour la version finale, ce bouton est supprimé (le code est conservé), ainsi rendre cette fonctionnalité accessible uniquement aux développeurs, dans le but de commercialiser des extensions de cartes supplémentaires dans une future prochaine.

Dans la version Android du jeu, cette fonctionnalité n'est pas présente.

Quand on instancie l'objet MapCreator, un dialogue est créé qui demande le nom de la carte, la taille de cette carte et le nombre de joueurs. Quand ces informations sont remplies, une nouvelle fenêtre est créée, avec à gauche une carte vide et à droite une liste de joueurs, une liste des bâtiments et des terrains disponibles, ainsi qu'un bouton “Fini”.

Après le dialogue, un fichier texte est créé comportant les informations de la hauteur, la largeur et le nombre de joueurs de la carte. On lit ensuite ce fichier texte et créer une nouvelle Partie avec les paramètres inscrites. Il n'y a aucune information sur les terrains et les bâtiments, donc la carte est remplie de plaine par défaut.

On choisit un joueur sur le menu à droite (si aucun joueur n'est choisi, le joueur neutre est sélectionné par défaut), puis on choisit un terrain ou un bâtiment, puis on clique sur un endroit sur la carte. Le terrain sur cet endroit, représenté par un attribut dans Partie, est ainsi changé. On peut aussi effectuer un déplacement de la souris tout en cliquant, toutes les cases sur le chemin de la souris seront changées.

Quand on appuie sur le bouton “Fini”, on appelle la méthode dans la classe Partie pour sauvegarder cette partie dans le fichier texte créé au début du processus.

Portage Android

Conception

View -> SurfaceView avec des callback

Les principales adaptations

Utilisation d'une interface différente : une custom view : Activity + View ou Activity + SurfaceView & Callback

La gestion du multi-touch : gestion du zoom (du fait de la taille d'un écran de Smartphone)

Une IA simplifiée

Travail d'Équipe

Constitution de l'équipe

Notre équipe s'est d'abord constitué en fonction de nos affections puis sur nos compétences. En effet, beaucoup d'entre nous avaient déjà travaillé plusieurs fois en binôme et nous nous connaissions bien. De plus, le projet se portant sur le développement d'un jeu en JAVA, nos bonnes notes a tous en JAVA (IGI-3003) nous n'a conforter dans le choix de constitué cette équipe.

Enfin, nous avons élu naturellement Mathieu Scala en chef de projet. En effet, ce dernier avait présenté de bonne aptitude chef de projet lors d'atelier comme le management de projet et une bonne capacité à motiver ces collègues.

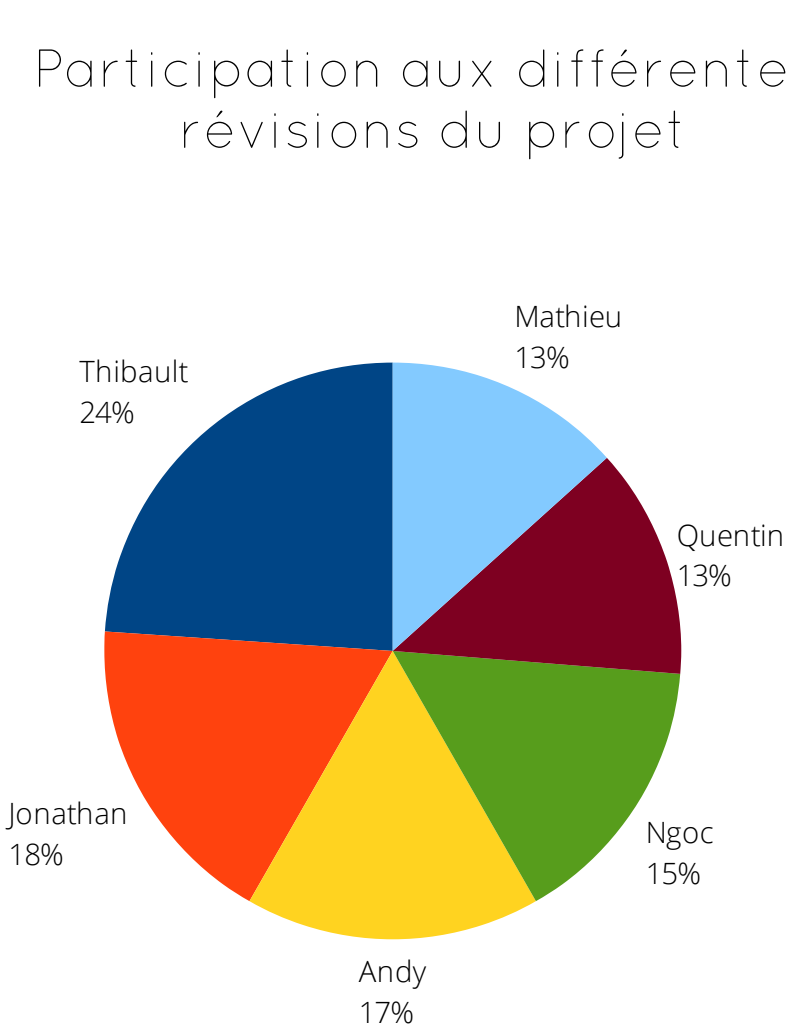
Répartition des taches

Au début du projet, nous nous sommes rassemblé pour définir les éléments essentiels au fonctionnement d'une partie. Nous avons donc dégagé plusieurs briques, composé de multiples classes. Nous avons donc réparti la création de ces classes à chaque membre de l'équipe. Ainsi chaque membres avait un travail à fournir pour la fin de la semaine.

Nous avons alors procédé de la même manière pour les semaines suivantes. Chaque membres de l'équipe travaillait sur une classe, ou un ensemble de classes correspondants à une (des) fonctionnalité(s) du jeu.

D'autre part, pour développer le projet, nous avons appliqué la méthode Agile, c'est à dire une approche incrémentale du projet. Nous travaillons toujours sur une version fonctionnelle pour rajouté des fonctionnalités chaque semaine.

Une telle méthode appliqué sur une équipe aussi nombreuse que la notre, nous a obligé à utiliser des outils de synchronisation comme SVN. De plus, dans un soucis de simplicité, nous nous donnions rendez-vous tous les jours en salle d'info (5006) à 8h pour coder ensemble. Ainsi tous les membres de l'équipe étaient au courant des changements de dernière minute et chacun pouvait communiquer simplement avec tous les autres membres de l'équipe en cas de problème.



En ce basant sur les commits effectuer à l'aide de SVN on peut alors créer le camembert ci-dessus. On peut ainsi remarquer que chaque membres de l'équipe à un pourcentage supérieur à 13 % de participation. Ces statistiques ne sont évidemment pas représentatives du travail effectué par chacun des membres de l'équipe mais permettent de prendre conscience que chaque membres a travaillé activement à l'avancement du projet.

Enfin comme souligner dans la constitution de l'équipe, beaucoup d'entre nous avaient déjà travaillé en binôme. Ainsi lors de phase de codage importante, nous nous retrouvions en binôme pour travailler plus efficacement.

Problème rencontrés

Discussion

Truc bien

Aspect à améliorer

Conclusion