

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 :	1
Description du jeu :	2
Les bâtiments.....	2
Les unités.....	3
Les terrains.....	3
L'intelligence Artificielle.....	4
Les Graphismes.....	4
Déroulement d'une partie.....	4
Récapitulatifs des Semaines.....	5
Semaine 2.....	5
Semaine 3.....	6
Semaine 4.....	6
Semaine 5.....	7
Semaine 6.....	8
Fonctionnement du Jeu.....	9
Intelligence Artificielle.....	9
StrategieIA.....	9
OperationIA.....	9
UnitelA.....	10
La classe Influence :.....	10
Initialisation d'une phase de jeu.....	11
Partie.....	11
Campagne.....	12
IHM.....	12
Conception.....	12
Affichage du plateau de jeu.....	12
Affichage de la barre d'information.....	13
Utilisation de la souris.....	13
Moteur.....	13
Algorithme de déplacement.....	13
Attaque.....	14
Capture.....	14
Portage Android.....	15
Conception.....	15
Les principales adaptations.....	15
Travail d'Équipe.....	16
Constitution de l'équipe.....	16
Répartition des tâches.....	16

Problème rencontrés.....	18
Discussion.....	19
Truc bien.....	19
Aspect à améliorer.....	19
Conclusion.....	20

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 débloquent, 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 débloquentes 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'activer ou non le brouillard de guerre ou les animations. Il peut également créer des équipes.

Une partie se déroule généralement sur plusieurs tours. Le joueur achète ses unités et les envoie ensuite vers la direction du camp ennemi. À chaque tour, le joueur gagne de l'argent et peut ainsi s'acheter des unités plus puissantes. Au fur et à mesure des combats, les unités recrutées gagnent de l'expérience et le joueur peut décider 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 panel 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 bâtiment
- 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
 - Défensif

- 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

L'intelligence artificielle de notre jeu fonctionne par pallier ou grade. Il y a 3 palliers en tout, et chacun de ces étages à sa propre fonction. Nous avons dans l'ordre des grades du plus élevés au plus faible : StrategiEA, puis OperationIA et enfin UnitEA. Leur complexité est proportionnel au grade. Par exemple StrategiEA est plus complexe que UnitEA.

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

StrategiEA

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 agressif 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 StrategiEA. 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 UnitelA 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 UnitelA d'acheter en fonction de l'argent disponible et des unités déjà possédées, ainsi que des unités ennemies.

UnitelA

Enfin UnitelA 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 appelle 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.

Défensif 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 crée 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 construits de la même façon. Chaque ligne est composée d'un identifiant (montagne, forêt), de ses coordonnées et du joueur à qui appartient l'identifiant.

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

Pour des raisons de simplicité, chaque type de partie (campagne ou partie rapide) possède son constructeur. En effet, lors d'une partie rapide, le joueur doit pouvoir définir certains nombres d'options comme sa faction, les équipes et des options de jeu (activé ou non le brouillard). Certaines de ces options imposées 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. Lorsque le joueur sélectionne la campagne dans le menu principale, un premier dialogue est appelé puis une mission se lance. A la fin de la mission, une méthode suite est appelée, elle analyse le résultat de la partie et décide en conséquence si le joueur peut continuer la campagne.

La liste des missions de la campagne est stocké dans une liste, créée dans la classe mère Slatch. A la fin de cette liste un dernier dialogue est appelé.

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 souris, 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'à 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 souris.

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 appelle 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 : $-1 \leq +\infty$).
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 rempliPorteeDep, 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, y pour la coordonnée y de la case correspondante
- Plus d est petite, plus un Triplet est grand. compareTo va donc comparer les d de deux triplets.

Attaque

Capture

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

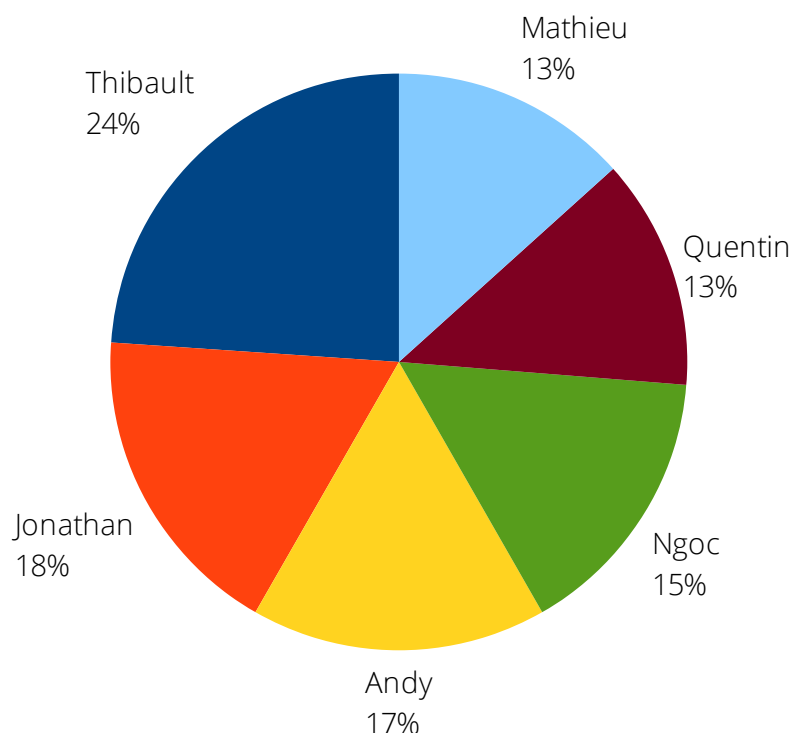
Au début du projet, nous nous sommes rassemblé pour définir les éléments essentiels au fonctionnement d'une partie. Nous avons donc dégagé plusieurs briques, composé de multiples classes. Nous avons donc réparti la création de ces classes à chaque membre de l'équipe. Ainsi chaque membres avait un travail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

Participation aux différentes révisions du projet



En se basant sur les commits effectués à l'aide de SVN on peut alors créer le camembert ci-dessus. On peut ainsi remarquer que chaque membre de l'équipe a 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 membre a travaillé activement à l'avancement du projet.

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

Problème rencontrés

Discussion

Truc bien

Aspect à améliorer

Conclusion