17/12/2019

Perdaens Martin

Michotte Martin

Valentin Morgan

Rapport de projet : BattleGround

A close up of a logo

Description automatically generated

Table of Contents

[Cahier des charges 2](#_Toc27388534)

[Contexte du projet et objectifs: 2](#_Toc27388535)

[Contraintes techniques: 2](#_Toc27388536)

[Description des besoins fonctionnels : 3](#_Toc27388537)

[Use Case : 4](#_Toc27388538)

[Diagramme UML du modèle 5](#_Toc27388539)

[Choix d’implémentation 6](#_Toc27388540)

[Game-Flow general 7](#_Toc27388541)

[Difficultés rencontrées 8](#_Toc27388542)

[1. Maitrise de l’outil « Git ». 8](#_Toc27388543)

[2. Implémentation de la communication Client-Serveur avec le jeu 8](#_Toc27388544)

[3. Implémentation de l’architecture MVC 8](#_Toc27388545)

[Pistes d’amélioration 9](#_Toc27388546)

[ Revoir complètement la structure de notre architecture MVC 9](#_Toc27388547)

[ Ajouter le bonus “radar-discovery” 9](#_Toc27388548)

[ Améliorations du système de positionnement des unités 9](#_Toc27388549)

[ Amélioration de la gestion des tirs 9](#_Toc27388550)

[Conclusion individuelle 10](#_Toc27388551)

[Martin Perdaens 10](#_Toc27388552)

[Morgan Valentin 10](#_Toc27388553)

[Martin Michotte 10](#_Toc27388554)

# Cahier des charges

## Contexte du projet et objectifs:

Dans le cadre du cours de Développement informatique avancé orienté applications (java), il nous est demandé de réaliser une application utilitaire ou bien un jeu qui se présente à la fois sous forme d’interface graphique et en ligne de commande. Ceux-ci doivent comporter une communication réseau ou une interaction avec une base de données ou éventuellement un service web.

Nous avons opté pour la réalisation d’un jeu vidéo de type 1 contre 1 en réseau se basant sur le jeu populaire « bataille navale ».

Dans notre version du jeu, le terrain de jeu n’est plus l’océan mais la terre ferme, les bateaux sont remplacés par des bâtiments et des véhicules.

Certains bâtiments et véhicules offrent des compétences supplémentaires au joueur tel qu’un raid aérien depuis un aéroport ou un tir de rockets depuis un véhicule lance-roquettes. Ces compétences spéciales ne sont évidemment pas accessibles à tout moment :

* Une fois utilisée, une compétence doit être « rechargée » (elle redeviendra disponible après un certain temps).
* Si le bâtiment/véhicule est détruit, la compétence liée à celui-ci n’est plus disponible.

## Contraintes techniques:

* Le jeu doit pouvoir être joué de 2 manières :
* Par le biais d’une interface en ligne de commande
* Par le biais d’une interface graphique

Attention, les deux joueurs ne doivent pas spécialement être sur la même interface.

* Le jeu doit pouvoir être joué sur deux machines distinctes dans un réseau local.
* Les bonus sont utilisables uniquement si l’unité associée est toujours « vivante ».

**Déroulement d’une partie :**

1. Les deux joueurs se connectent.
2. Les deux joueurs placent leurs unités sur leur terrain de jeu.
3. Une fois les deux joueurs prêts, le joueur A débute son tour.
4. Une fois que le joueur A a fini son attaque, le tour est donné au joueur B.
5. Le point 4 est répété jusqu’à ce qu’un des deux joueurs :
   1. n’ait plus d’unités.
   2. quitte la partie.

## Description des besoins fonctionnels :

Nous décrivons ci-dessous les besoins fonctionnels du programme pour un joueur :

**Spécificité générale :**

**Interface de jeu :**

* 2 grilles de 13x13 cases
* Une grille sur laquelle le joueur place ces unités et voit les attaques infligées par son adversaire.
* Une grille sur laquelle le joueur peut placer ces propres attaques et en voir le résultat.

**Unités :**

Le joueur dispose de 6 unités (3 bâtiments et 3 véhicules) :

|  |  |  |
| --- | --- | --- |
| **Unité** | **Taille** | **Bonus** |
| Airport | 2x4 | Air-strike |
| Radar Tower | 2x3 | Radar Discovery |
| Headquarter | 2x2 | / |
| Railway Gun | 1x6 | Big-shot |
| MMRL[[1]](#footnote-1) | 2x2 | Rocket-strike |
| Tank | 1x2 | / |

**Bonus :**

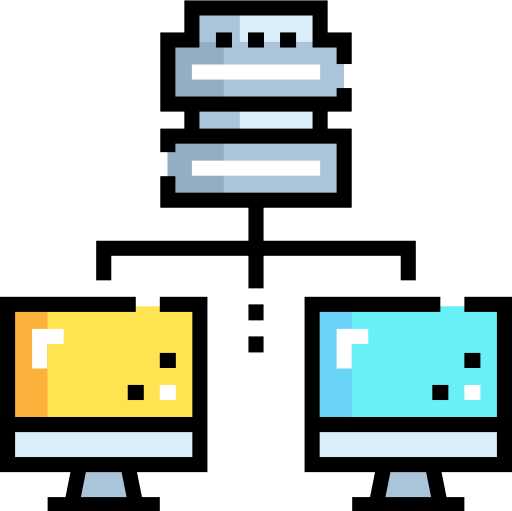
*Air-strike :* Un avion largue des bombes sur 7 cases en ligne.

*Radar Discovery :* 4 cases adjacente une à une peuvent être découverte.

*Big-shot :* Un obus de 800mm explose sur une case et détruit toutes les cases adjacentes.

*Rocket-strike :* 4 missiles tombent dans une zone de 7x7 de manière aléatoire.

## Use Case[[2]](#footnote-2) :



ServeurRRRR

Joueur 2

Joueur 1

Le serveur doit pouvoir :

1. Établir une connexion entre deux joueurs sur un même réseau.
2. Donner le tour à chacun des joueurs.
3. Recevoir les informations[[3]](#footnote-3) du joueur qui a le tour.
4. Traiter les informations reçues.
5. Renvoyer des informations aux deux joueurs. (Où est ce que le tir a été effectué, est-ce qu’une unité a été touchée, …)

Le joueur doit pouvoir :

1. Se connecter au serveur.
2. Avant le commencement de la partie, placer ces unités sur la grille de jeu.
3. Visualiser les deux grilles de jeu :
   1. Sa grille contenant ses unités et leur état. (non touchée, touchée, détruite)
   2. La grille de l’adversaire avec la position de ces propres tires et l’état de ceux-ci. (touché, non touché) (cette grille est vierge au début de la partie)
4. Lorsque c’est son tour, tirer sur une case de la grille de son adversaire ou utiliser une attaque spéciale.
5. Lorsque ce n’est pas son tour, le joueur ne peut rien faire hormis observe l’attaque de son adversaire.
6. Quitter le jeu sans avoir terminé la partie. (L’adversaire aura alors gagné par abandon)

# Diagramme UML du modèle

**A close up of a map

Description automatically generated**

# Choix d’implémentation

## Game-Flow general

# Difficultés rencontrées

## Maitrise de l’outil « Git ».

* C’était pas vraiment à proprement dit une difficulté, mais juste beaucoup de temps consacrés à une petite chose qu’un connaisseur (professeur) aurait pu nous expliquer en quelques minutes comme le traitement des branches, les branches distantes (remote branch) et comment fonctionne un merge.

## Implémentation de la communication Client-Serveur avec le jeu

* Dès que le diagramme UML était fini, nous nous sommes penchés sur la problématique de la communication entre 2 pc. On a vu plusieurs manières de l’implémenter sur internet mais le point commun avec toutes ces manières, c’est que le modèle repose sur le fait que le serveur est en quelque sorte « passif » : celui-ci attend toujours les requêtes/demandes d’un client. Et une fois qu’il reçoit une demande d’un client il lui répond.
* Pour notre jeu qui comprends 2 joueurs (donc 2 clients), il fallait trouver un moyen que lorsqu’un des 2 clients envoie une requête au serveur, celui-ci envoie une information aux **2 clients et pas juste à celui qui a fait la requête**.

## Implémentation de l’architecture MVC

* Nous ne sommes absolument pas contre MVC, au contraire c’est une très bonne idée /pratique. Le problème c’est qu’une fois que nous avons fini notre code pour la ligne de commande fonctionnel, une semaine avant la démonstration on reçoit les explications de ce qu’est MVC, l’utilité de celui-ci et comment l’implémenter (dans un code relativement simple).
* Notre grande difficulté, réside dans le fait de devoir revoir la structure du code pour implémenter MVC dans le peu de temps qui nous restaient.

# Pistes d’amélioration

## Revoir complètement la structure de notre architecture MVC

* On devrait en quelque sorte recommencer notre code à zéro en implémentant l’architecture MVC pour avoir un « vrai » MVC.
* (Martin tu sais mieux quoi dire ICI)

## Ajouter le bonus “radar-discovery”

* Une des armes (ou bonus) que nous n’avons pas réussis à implémenter c’est le bonus « radar-discovery » qui permettrai à un joueur de connaitre l’emplacement des unités ennemies (en sélectionnant une petite zone).
* Elle figure bien dans le code :
  + Dans le **gui** elle est grisée => bouton désactivée mais présent.
  + Dans la **ligne de commande** => Commentée mais présent.

## Améliorations du système de positionnement des unités

* ??

## Amélioration de la gestion des tirs

* Tir partiellement en dehors de la grille :
  + Lorsqu’un joueur tire en dehors de la grille (sauf le tire de base), par exemple le « *big-shoot* » où le joueur choisi une case et le tire touche la cache et toutes les cases adjacentes. Si le joueur choisi de tirer dans la case supérieure gauche (A1), il touchera 4 cases au lieu de 9 cases.
* Tir sur une case qui a déjà été tirée dessus :
  + On devrait peut-être implémenter un petit message disant au joueur que la case a déjà été visée afin que celui-ci ne perde pas un tour.

# Conclusion individuelle

## Martin Perdaens

* En ayant jamais programmer en Java, j’avais des craintes sur la manière de coder le projet. Au début je ne voyais pas comment faire mais par la suite avec les cours et le faites de chercher sur internet cela m’a permis de mieux comprendre.
* L’idée de faire un projet en groupe était chouette pour approfondir notre compréhension du langage mais j’aurais aimé avoir toute la matière avant de commencer le projet.

## Morgan Valentin

* Etant débutant en java, j’appréhendais un peu le langage. Cependant, ce projet m’a vraiment poussé à rentrer dans le langage et chaque difficulté que j’ai rencontré m’a permis d’aller plus loin dans la matière.
* Ce projet était une très bonne idée malgré que j’aurais aimé avoir aucun cours pratique et juste le projet **ou bien** un quadrimestre où l’on apprend le langage (comme pour le cours de programmation de 1ère année) et le 2ème quadrimestre où l’on a un projet.

## Martin Michotte

1. Mobile Multiple Rocket Launcher [↑](#footnote-ref-1)
2. Les numéros de puces indiquent l’ordre de priorité de la fonctionnalité [↑](#footnote-ref-2)
3. Par informations on entend : la position des unités, la position des tirs, … [↑](#footnote-ref-3)