Tower Defense

Mise en œuvre d’un graphe interactif

Création d’une variante du célèbre jeu Tower Defense dans le cadre d’un cours d’algorithmique.

ASD2

Lazhar Farjallah / Aurélien Da Campo / Pierre-Dominique Putallaz

Heig-vd

ASD

TD

Table des matières

[1 Analyse préliminaire 3](#_Toc248740874)

[1.1 Introduction [LAZHAR] 3](#_Toc248740875)

[1.2 Organisation [LAZHAR] 3](#_Toc248740876)

[1.3 Objectifs [LAZHAR] 4](#_Toc248740877)

[1.4 Planification initiale [AURELIEN] 5](#_Toc248740878)

[2 Analyse 6](#_Toc248740879)

[2.1 Concepts algorithmique [LAZHAR] 6](#_Toc248740880)

[2.2 Etude de concurrentielle [AURELIEN] 6](#_Toc248740881)

[2.3 Etude de faisabilité [AURELIEN] 6](#_Toc248740882)

[2.4 Planification [AURELIEN] 6](#_Toc248740883)

[3 Conception 7](#_Toc248740884)

[3.1 Dossier de conception 7](#_Toc248740885)

[3.1.1 Systèmes d’exploitation [LAZHAR] 7](#_Toc248740886)

[3.1.2 Outils logiciels [LAZHAR] 7](#_Toc248740887)

[3.1.3 Librairies externes [LAZHAR] 7](#_Toc248740888)

[3.1.4 Architecture de l’application [AURELIEN] 7](#_Toc248740889)

[3.1.5 Interface graphique [AURELIEN] 7](#_Toc248740890)

[3.1.6 Schémas UML [PIERRE-DO] et [AURELIEN] 7](#_Toc248740891)

[3.1.7 Gestion de la concurrence [PIERRE-DO] 8](#_Toc248740892)

[4 Réalisation 9](#_Toc248740893)

[4.1 Dossier de réalisation 9](#_Toc248740894)

[4.1.1 Résultat de l’interface graphique [AURELIEN] 9](#_Toc248740895)

[4.1.2 Résultat de l’implémentation du maillage [PIERRE-DO] 9](#_Toc248740896)

[4.2 Description des tests effectués [LAZHAR] 9](#_Toc248740897)

[4.3 Erreurs restantes [LAZHAR] 9](#_Toc248740898)

[5 Conclusions *[LAZHAR]* 9](#_Toc248740899)

[5.1 Objectifs atteints / non-atteints 9](#_Toc248740900)

[5.2 Points positifs / négatifs 9](#_Toc248740901)

[5.3 Difficultés particulières 9](#_Toc248740902)

[5.4 Avenir du projet 10](#_Toc248740903)

[6 Annexes 11](#_Toc248740904)

[6.1 Sources – Bibliographie [LAZHAR] 11](#_Toc248740905)

[6.2 Journal de bord de chaque participant [LAZHAR] 11](#_Toc248740906)

[6.3 Manuel de TDA maillage [PIERRE-DO] 11](#_Toc248740907)

[6.4 Manuel d'Utilisation [LAZHAR] 11](#_Toc248740908)

[6.5 Archives du projet [LAZHAR] 11](#_Toc248740909)

# Analyse préliminaire

## Introduction [LAZHAR]

Ce projet de fin de semestre consiste à créer une application ludique mettant en œuvre des algorithmes et structures de données étudiées en cours. Nous avons choisi pour cela de créer une variante du célèbre jeu « Tower Defense », dans lequel des personnages se déplacent d’un point A à un point B selon un chemin optimal. En effet, on aura pour cela besoin d’une structure de graphe ainsi que les algorithmes associés, ce qui colle parfaitement avec la contrainte de départ car nous les avons étudiés en cours.

Le but est également ici de créer une application « didacticiel » qui sera présentée dans les futurs cours de cette unité d’enseignement. Ce projet permettra en effet de montrer une application réelle de l’utilisation d’algorithmes associés à des graphes (en particulier celui de recherche du chemin le plus court entre deux nœuds).

Nous commencerons par effectuer une analyse du projet, notamment en ce qui concerne l’organisation, les objectifs ainsi que la planification initiale. Puis nous détaillerons les étapes de conception et de réalisation. Enfin, avant de conclure, nous présenterons les fonctionnalités de l’application finale sous forme de mode d’emploi et nous présenterons des captures d’écran.

## Organisation [LAZHAR]

Les membres participant à ce projet sont :

**Etudiant 1** :

Aurélien Da campo, *aurelien.dacampo@heig-vd.ch*

**Etudiant** **2** :

Pierre-Dominique Putallaz, *pierre-dominique.putallaz@heig-vd.ch*

**Etudiant 3 (responsable de projet)** :

Lazhar Farjallah, *lazhar.farjallah@heig-vd.ch*

Nous prévoyons de nous répartir les tâches de manière suivante entre chaque entité du groupe :

1. *Responsable de projet*
   1. Rédactions, administrations
   2. Suivi des rendus (*deadlines*)
   3. Surveillance et coordination
   4. Développement
2. *Etudiant 1*
   1. Création de l’interface graphique
   2. Gestion de l’affichage
   3. Rendu graphique
   4. Interaction avec l’utilisateur
3. *Etudiant 2* 
   1. Algorithmique
   2. Implémentation des algorithmes de graphe
   3. Fournir les briques logicielles pour permettre la construction de la partie fonctionnelle de l’application

## Objectifs [LAZHAR]

Les objectifs de ce projet sont les suivants :

* Illustrer le concept de graphe de manière ludique et interactive.
* Acquérir de l’expérience dans la planification et l’accomplissement d’un projet conséquent.
* Utiliser et découvrir des librairies existantes implémentant le concept de graphe
* Apprendre à mettre en œuvre une interface graphique en Java.
* Séparer le travail en plusieurs niveaux d’abstraction pour faciliter l’élaboration et l’évolutivité de ce projet.
* Comprendre la nécessité d’utiliser des algorithmes complexes dans les applications informatiques.
* Mettre en œuvre un algorithme de recherche de chemin le plus court (ACPC).
* Respecter le design pattern MVC (*Model* – *View* – *Controller*) qui structure un programme en trois couches principales.

## Planification initiale [AURELIEN]

Le projet se déroulera du 18 novembre 2009 au 15 janvier 2010, ce qui représente un total de 25 périodes en classe (par personne). Nous prévoyons également de passer un total d’environ au moins 25 périodes par personne en dehors des heures encadrées.

Au total, c’est environ 150 périodes de travail que nous allons planifier comme suit :

*Ce chapitre montre la planification du projet. Celui-ci peut être découpé en tâches qui seront planifiées. Il s'agit de la première planification du projet, celle-ci devra être revue après l'analyse. Cette planification sera présentée sous la forme d'un diagramme de Gantt et/ou de PERT (l'utilisation de MS project est conseillée).*

*Ces éléments peuvent être repris des spécifications de départ.*

# Analyse

## Concepts algorithmique [LAZHAR]

## Etude de concurrentielle [AURELIEN]

## Etude de faisabilité [AURELIEN]

*Détailler les 3 a****s****pects de l'étude de faisabilité:*

* *risques techniques (complexité, manque de compétences, …).*
* *risques concernant le planning & les ressources humaines.*
* *risques concernant le budget.*

*Décrire aussi quelles solutions ont été appliquées pour réduire les risques (priorités, formation, actions, …).*

## Planification [AURELIEN]

*Révision de la planification initiale (Gantt et/ou PERT) du projet :*

* *planning indiquant les dates de début et de fin du projet ainsi que le découpage connu des diverses phases.*
* *partage des tâches en cas de travail à plusieurs.*

*Il s’agit en principe de la planification* ***définitive du projet****. Elle peut être ensuite affinée (découpage des tâches). Si les délais doivent être ensuite modifiés, le responsable de projet doit être avisé, et les raisons doivent être expliquées dans l’historique.*

# Conception

## Dossier de conception

### Systèmes d’exploitation [LAZHAR]

Le système d’exploitation sur lequel nous travaillons est Windows (XP et Seven). Cependant, grâce au choix qui a été fait d’utiliser un encodage de type UTF-8 ainsi que celui du langage portable Java, nous pouvons sans soucis travailler sur un environnement Linux ou Mac par exemple.

### Outils logiciels [LAZHAR]

Nous développons avec l’IDE Eclipse (version 3.4 et supérieure) intégrant tous les outils nécessaires au développement d’applications Java. Ce logiciel est très largement répandu dans le monde des développeurs et est très utilisé. Il possède de nombreuses fonctions spécialement conçues pour augmenter la productivité des développeurs et leur simplifier la vie (comme le *refactoring* par exemple). De plus, cette plateforme nous permet d’ajouter toute une série de plugins qui peuvent ajouter des fonctionnalités, telles que SVN (logiciel de gestion des versions du code). En ce qui concerne la génération des diagrammes de classe UML, nous utilisons le plugin ***eUML 2.0*** de chez ***Soyatec***.

### Librairies externes [LAZHAR]

Nous utilisons la librairie externe ***JGraphT*** codée en Java. En effet, cette librairie possède toutes les briques logicielles nécessaires à la création de graphes. Nous l’utilisons comme une boîte noire sans se soucier de son implémentation.

### Architecture de l’application [AURELIEN]

Nous avons choisi comme architecture du programme le fameux design pattern MVC (Modèle – Vue – Contrôleur).

### Interface graphique [AURELIEN]

### Schémas UML [PIERRE-DO] et [AURELIEN]

#### Maillage **[PIERRE-DO]**

#### Interface graphique (vue et controleur) **[AURELIEN]**

#### Gestionnaire du jeu (model) **[AURELIEN]**

### Gestion de la concurrence [PIERRE-DO]

Ici nous allons aborder un sujet qui nous a posé quelques soucis au niveau de la conception et de l’implémentation, la gestion de la concurrence en Java.

D’abord quelques mots sur le contexte dans lequel cette gestion s’est révélée problématique. La gestion des différentes parties est encapsulée dans différents threads Java.

*Fournir tous les documents de conception:*

* *le choix du matériel HW*
* *le choix des systèmes d'exploitation pour la réalisation et l'utilisation*
* *le choix des outils logiciels pour la réalisation et l'utilisation*
* *site web: réaliser les maquettes avec un logiciel, décrire toutes les animations sur papier, définir les mots-clés, choisir une formule d'hébergement, définir la méthode de mise à jour, …*
* *bases de données: décrire le modèle relationnel, le contenu détaillé des tables (caractéristiques de chaque champs) et les requêtes.*
* *programmation et scripts: organigramme, architecture du programme, découpage modulaire, entrées-sorties des modules, pseudo-code / structogramme…*

***Le dossier de conception devrait permettre de sous-traiter la réalisation du projet !***

# Réalisation

## Dossier de réalisation

### Résultat de l’interface graphique [AURELIEN]

### Résultat de l’implémentation du maillage [PIERRE-DO]

L’implémentation du TDA maillage est définie par le cahier des charges suivant (description des méthodes) :

1. Constructeurs : La liste des différents constructeurs permettant de créer le maillage en fonction de différents paramètres. L’unité utilisée par default est le pixel, depuis la zone graphique selon les standards. Une exception est levée si les paramètres ne sont pas probables.
   1. + Maillage (int,int,int,int,int) : ce constructeur public prend en paramètres dans l’ordre :
      1. La largeur totale en pixel du maillage.
      2. La hauteur totale en pixel du maillage.
      3. Le côté du nœud en pixel. Un nœud est toujours carré.
      4. La position de départ du maillage dans la zone graphique en x.
      5. La position du départ du maillage dans la zone graphique en y.
   2. + Maillage (int,int,int)*:* mêmes paramètres que le constructeur précédent, avec la position du maillage par default en (0 ;0) .
2. Méthodes d’éditions : ces méthodes permettent d’éditer la structure du maillage de manière dynamique après l’avoir crée. Si le rectangle définissant les zones à éditer n’est pas dans la zone, une exception est levée. Méthodes « thread-safe ».
   1. + void activerZone(Rectangle) : Permet d’activer une zone dans le maillage, c'est-à-dire de définir les nœuds contenus dans le rectangle passé en paramètre comme actifs et de les relier aux nœuds actifs adjacents. Le rectangle a une dimension en pixels relatifs à la zone de jeu.
   2. + void desactiverZone(Rectangle) : Permet de désactiver une zone dans le maillage, c'est-à-dire de marquer comme accessibles les nœuds contenus dans le rectangle passé en paramètre et de supprimer les arcs dans le graphe relatifs à ce nœud. Le rectangle a une dimension en pixels relatifs à la zone de jeu.
3. Méthode de calcul : Permet d’obtenir des informations sur le maillage, notamment le chemin le plus court d’un point à un autre. Méthode « thread-safe ».
   1. + ArrayList<Point> plusCourtChemin(int,int,int,int) : C’est la méthode principale du maillage. Elle permet à partir des coordonnées des points de départ et d’arrivée de calculer selon le maillage le chemin le plus court, qu’elle retournera sous forme d’un ArrayList de points représentant les différentes étapes du chemin. Dans l’ordre des paramètres :
      1. La coordonnée x du point de départ
      2. La coordonnée y du point de départ
      3. La coordonnée x du point d’arrivée
      4. La coordonnée y du point d’arrivée
4. Accesseurs : Cet ensemble des méthodes permet de consulter l’état du maillage à n’importe quel moment, par exemple de récupérer la liste des nœuds internes au graphe. Nous ne détaillerons pas ici l’ensemble des méthodes, pour les détails voir la JavaDoc.

La représentation interne de la structure se fait selon le principe suivant. L’ensemble des nœuds est stocké dans un tableau, puis passé au graphique pour qu’ils soient organisés logiquement et qu’on puisse leur ajouter des arcs. Comme il s’agit de références passées, on peut accéder de manière équivalente aux nœuds par le graphe (que les nœuds actifs) et par le tableau interne du maillage. Cela permet d’ajouter et de supprimer dynamiquement des nœuds dans le graphe sans à avoir à les régénérer.

Pour la gestion des arcs, c'est-à-dire l’ajout lors de l’activation dynamique d’une zone ou à la création du maillage, on utilise l’algorithme suivant :

[pseudo code, faut voir pour mettre le bon style]

Pour un nœud cible :

Ajout du nœud dans le graphe.

Marquer le nœud comme actif.

for tout ses voisins do

if le voisin est actif do

Ajout dans le graphe d’un arc cible <-> voisin.

Assignation du poids de l’arc, c'est-à-dire la distance en pixel qui sépare les deux points.

End if

End for

[fin du pseudo code]

Il est essentiel de retenir que le graphe n’est donc pas orienté. La figure ci-contre illustre l’algorithme, le nœud bleu est le nœud cible, il est relié aux nœuds actifs et lui-même marqué comme actif. Cet algorithme permet d’utiliser la même méthode lors de l’initialisation des nœuds que lors de la réactivation d’une zone après avoir été désactivée.

Donnons également un petit point sur les unités. En effet, c’est un détail qui nous a posé soucis. Nous avons distingué deux types d’unités :

* Le pixel : C’est l’unité avec laquelle est créé, édité et lu le maillage. Les classes qui implémente le maillage travaillent dans une zone de dessin en pixel, toutes les opérations prennent et retourne cette unité. Le maillage a conscience de sa dimension en pixel, de la largeur du nœud du pixel ainsi que du décalage par rapport à l’origine, toujours en pixel.
* La coordonnée nodale : C’est l’abstraction utilisée à l’intérieur de la classe Maillage pour représenter un nœud dans le tableau de nœud. Par définition une coordonnée nodale égale une largeur de nœud en pixel, à quelques subtilités prêt.

L’essentiel de la difficulté de travail avec des deux types d’unités (uniquement en interne), est qu’il a fallu non seulement définir une relation pixel->nodale, mais également la relation inverse. Nous avons utilisé l’algorithme suivant pour la conversion :

Pixel -> nodale :

Pour chaque coordonnée x et y en pixel do

Calcul de la coordonnée modulo taille du nœud.

Soustraction de cette valeur à la coordonnée.

Division de la valeur obtenue par la largeur du nœud.

Retour de x et y en nodal.

Nodale -> pixel :

Pour chaque coordonnée x et y en nodale do :

Multiplication de la coordonnée nodale par la largeur du nœud en pixel.

Retour de x et y en pixel.

En effet, la coordonnée nodale converti en pixel représente le centre du nœud, tandis qu’une coordonnée en pixel est converti vers la coordonnée nodale la plus proche.

Résumons donc les propriétés du maillage :

* Graphe non orienté (pour permettre une navigabilité dans les deux sens).
* Graphe pondéré (pour permettre de représenter la distance entre les points pour l’algorithme de Dijkstra).
* Toujours un chemin entre le point de départ et d’arrivée.
* Edition dynamique du maillage par activation ou désactivation d’une zone.
* Lecture sur le maillage par une liste des arcs actifs.
* Lecture sur le maillage par une liste de points représentant le chemin le plus court entre un point A et un point B donné.
* Utilisation de méthodes de conversions pour les différentes unités.

Afin de faire fonctionner la classe Maillage, il y a quelques classes annexes :

* Arc : Il s’agit d’un arc dans le maillage. Il étend la classe DefaultWeightedEdge (de Jgrapht) pour pouvoir être intégré dans le maillage et pour pouvoir jouer avec le poids de l’arc. Il contient également une référence vers le nœud de départ et le nœud d’arrivée.
* GenerateurDArcs
* Nœud :
* PathNotFoundException :

## Description des tests effectués [LAZHAR]

*Pour chaque partie testée de votre projet, il faut décrire:*

* *les conditions exactes de chaque test*
* *les preuves de test (papier ou fichier)*
* *tests sans preuve: fournir au moins une description*

## Erreurs restantes [LAZHAR]

*S'il reste encore des erreurs:*

* *Description détaillée*
* *Conséquences sur l'utilisation du produit*
* *Actions envisagées ou possibles*

# Conclusions *[LAZHAR]*

## Objectifs atteints / non-atteints

## Points positifs / négatifs

## Difficultés particulières

## Avenir du projet

# Annexes

## Sources – Bibliographie [LAZHAR]

*Liste des livres utilisé (Titre, auteur, date), des sites Internet (URL) consultés, des articles (Revue, date, titre, auteur)… Et de toutes les aides externes (noms)*

## Journal de bord de chaque participant [LAZHAR]

## Manuel de TDA maillage [PIERRE-DO]

## Manuel d'Utilisation [LAZHAR]

## Archives du projet [LAZHAR]

*CD, … dans une fourre en plastique*