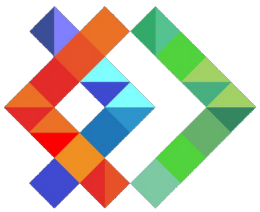

Projet d'algorithmique et d'image n°1

Cours d'algorithmique avancée - Synthèse d'Image I

— IMAC première année —

Imac Tower Defense

Ce projet a pour but :

- De vous familiariser avec les structures d'arbres et de graphes.
- De vous faire utiliser les fonctions de dessin 2D OpenGL.
- De vous faire coder les algorithmes fondamentaux des graphes
- De vous faire manipuler des images.

Nombre de personnes par projet : 2 ou 3
Date de rendu (non définitive) : 24 mai

---

1 Introduction

Ce projet a pour but de vous faire réaliser une application SDL permettant de visualiser une carte composée d'un chemin où des monstres se déplacent d'une entrée vers une sortie. L'utilisateur pourra placer des tours pour empêcher ces derniers d'atteindre une sortie. Ce projet sera rendu pour le XX mai (la date sera précisée ultérieurement) et accompagné d'un rapport (cf. travail à fournir). Vous enverrez donc avant cette date, le rapport, le Makefile, les fichiers .c et .h ainsi que l'exécutable aux adresses `steeve.v91@gmail.com` et `olivier.f4lconnet@gmail.com` le tout taré et zippé. Les projets seront réalisés en binômes, la masse de travail étant trop importante pour une seule personne. Les trinômes sont acceptés mais devront réaliser du travail supplémentaire. Enfin votre projet devra fonctionner sur les machines linux des salles de TD. Aucun code ni exécutable fonctionnant exclusivement sous Windows ou Mac ne sera accepté.

Ce projet est commun aux matières algorithmique avancée et synthèse d'images I. Le projet en lui même possède deux aspects : une partie algorithmique et une partie infographie. La notation du projet se dédoublera donc en deux notes distinctes, une pour chacune des matières concernées. Le rapport de ce projet comptera pour les deux notes et n'est donc pas à négliger.

2 Présentation du projet

Votre groupe est chargé, au sein d'une entreprise de jeu vidéo, de créer un prototype de logiciel permettant, d'une part, le chargement d'une carte, et d'autre part, la construction sur cette carte de différents batîments pour empêcher des vagues de monstres tentant de trouver la sortie d'atteindre cette dernière, le tout dans un jeu nommé `Imac Tower Defense`.

2.1 Travail à fournir

Pour le projet :

- Réalisation d'un Makefile, et d'une structure ordonnée de projet (cf. section 2.2) *(partie algorithmique)*

Initialisation d'une carte (cf. section 3) :

- Chargement d'une carte. *(partie infographie)*
- Création du graphe des chemins. *(partie algorithmique)*
- Vérification de la jouabilité de la carte. *(partie algorithmique)*

Création des différents élements (cf. section 4) :

- Sprites de bâtiments, de tours, de monstres.*(partie infographie)*
- Création des structures de bâtiments, de tours, de monstres. *(partie algorithmique)*

ITD : le jeu (cf. section 5) :

- Ajout, suppression et édition de bâtiments et de tours. *(partie infographie)*
- Déplacement des monstres. *(partie algorithmique)*
- Gestion des actions des tours et bâtiments. *(partie infographie)*
- Gestion du temps. *(partie algorithmique)*

Rapport contenant notament :

- Introduction.
- Présentation de l'application.
- Description globale de l'architecture de votre application (justification des découpages, etc.).
- Description des structures de données utilisées.
- Description détaillée des fonctionnalités de l'application interactive avec notamment les règles du jeu (Guide utilisateur).
- "Résultats" obtenus par votre application (capture d'écran, temps de calcul...).
- Conclusion

2.2 Structure de votre programme

Dans ce projet complexe, il est hors de question de n'utiliser qu'un seul fichier contenant toutes les fonctions nécessaires à l'application ! Il vous faut donc séparer les traitements en différents fichiers `.c` et `.h`. La découpe des fichiers est laissée à votre appréciation mais doit être logique. Globalement, le projet étant scindé en deux parties, il serait logique que la partition des fichiers en tienne compte. Afin de compiler le projet, un Makefile devra être réalisé. D'autant que vous aurez certainement à utiliser des bibliothèques (pour les listes et les files). Si vous utilisez des bibliothèques, n'oubliez pas de les inclure (ainsi que leur sources) lorsque vous m'enverrez le projet. **Note importante :** Tout `#include "unfichier.c"` ou tout rendu de projet sans Makefile entrainera un 0 !

Le programme sera donné sous la forme d'un exécutable nommé `itd`. Il prendra 1 argument : le fichier `.itd` décrivant la carte de jeu.

Enfin votre projet sera mis dans un repertoire nommé aux noms des 2 binômes (ou des 3 trinômes le cas échéant). Dans celui ci vous suivrez la structure suivante :

```
Nom1_Nom2/
  \-- bin/
  \-- include/
  \-- src/
  \-- lib/
       \-- include
       \-- src
       Makefile
  \-- data/
  \-- images/
  \-- doc/
  \-- Makefile
```

Le répertoire `bin` contient l'exécutable `itd`. Le répertoire `include` contient les fichiers d'entête `.h` de vos programmes tandis que `src` contient les fichiers sources `.c`. Le répertoire `lib` (optionnel) contiendra les fichiers `.a` ou `.so` de vos bibliothèques ainsi que tout le nécessaire pour les compiler : un Makefile, un répertoire `include` contenant les `.h` et un répertoire `src` contenant les `.c`. Le répertoire `data` contient les fichiers `.itd` et `images` contient tous les `.ppm`. Le répertoire `doc` contiendra toute la documentation, dont votre rapport. Enfin un `Makefile` permettera de compiler `itd`.

Trinôme uniquement : Les trinômes devront permettre à l'exécutable de ne prendre aucun argument. Dans ce cas, l'utilisateur pourra choisir une carte parmi celles présentes dans le répertoire `data`.

3 Les cartes Imac Tower Defense

Les éléments constitutifs d'une carte sont regroupés dans un fichier de configuration simple de suffixe `.itd`.

3.1 Fichier de description .itd

Ce fichier indique les différents paramètres nécessaires pour charger une carte dans le jeu. C'est un simple fichier texte au format suivant :

```
@ITD 1
#premiere ligne de commentaire
carte fichier.ppm
energie 100
chemin 255 255 255
noeud 0 0 0
construct 255 200 80
in 0 200 0
out 200 0 0
5
0 1 10 20 3 2 4
1 2 454 103 2 4
2 4 300 103 3 1 0
3 3 200 103 0 2
4 4 200 206 0 2 1
```

La première ligne est constitué d'un code (pour ce projet ce sera toujours `@ITD`) indiquant qu'il s'agit d'un fichier de description de carte pour ITD suivi du numéro de version de ce fichier de description (ici ce sera `1`).

Ensuite il y a une ligne de commentaire. Cette ligne est suivie de plusieurs lignes. Chaque ligne, **dans un ordre quelconque**, est constitué de deux éléments : un mot clé et une valeur.

Mot clé	Valeur	Type de la valeur
`carte`	Nom, **ne contenant pas d'espace**, de fichier image au format `ppm` représentant la carte proprement dit (cf. section 3.2)	chaine de caractères
`energie`	Nombre d'unité d'energie produite par les centrales	entier ≥ 0
`chemin`	Couleur clé des chemins dans l'image	Triplet R,V,B (0-255)
`noeud`	Couleur clé des noeuds	Triplet R,V,B (0-255)
`construct`	Couleur clé des zones 'constructibles'	Triplet R,V,B (0-255)
`in`	Couleur clé de la / des zones d'entrée(s)	Triplet R,V,B (0-255)
`out`	Couleur clé de la zone de sortie	Triplet R,V,B (0-255)

Figure 1: Description des parametres du fichier

Enfin, le fichier contient, sur une ligne, un entier indiquant le nombre de nœud des chemins suivi d'autant de ligne décrivant chaque nœud. Chacune de ces lignes décrit ensuite un noeud : elle présente tout d'abord un entier positif ou nul indiquant l'indice de ce nœud, suivi d'un entier positif indiquant la nature de ce noeud, suivi des deux coordonnées largeur, hauteur de la position de ce noeud dans l'image, et enfin une liste d'indices représentant les successeurs de ce noeud. En effet, un noeud est, dans la carte ITD, soit une zone de départ des monstres, soit un coude dans un chemin, soit une intersection de chemins, soit la zone d'arrivée. Aussi le type des noeuds seront indiqués par les entiers suivants :
1 - Zone d'entrée des monstres
2 - Zone de sortie
3 - Coude
4 - Intersection
Ces nœuds sont représentés dans la carte ppm par un unique pixel d'une couleur particulière (cf. section 3.2).

3.2 Fichier image représentant la carte

Pour représenter la carte, nous utilisons une image en couleur au format `ppm` afin d'indiquer où sont situées les routes, les zones constructibles et non constructibles, la/les zones d'entrées ainsi que la zone de sortie. Certains pixels dans cette carte représentent également les noeuds du graphe représentant le chemin. Les couleurs de ces différentes zones sont décrites dans le fichier `.itd`. Tout pixel ayant une couleur différente de celles décrites dans ce fichier appartient à la catégorie des zones constructibles.

Comme vous pouvez le constater, l'essentiel de la carte est constitué de chemin et de zone constructible. En fait, les chemins sont connectés via les noeuds. Les noeuds sont représentés dans l'image par un unique pixel de couleur. Les chemins relient ces noeuds **de manière rectiligne**. Les chemins ont une "largeur" de 30 pixels. Ainsi tout coude dans un chemin est situé sur un noeud. Les zones d'entrée et de sortie sont également symbolisées par un noeud. Ces noeuds sont entourées par des disques de couleur (comme spécifiées dans le fichier `.itf`) de diamètre 30 pixels également.

3.3 Validation d'une carte

Plusieurs conditions doivent être remplie pour qu'une carte soit valide pour le jeu ITD. Votre application devra être capable, lors du chargement d'une carte ITD, de vérifier si une telle carte est valide ou non.

Voici les différentes exigences :

- Fichier `.itd` au format correct : Présence sur la première ligne du bon code, ligne de commentaire présente, chacun des 8 paramètres existe et a une valeur "correcte", nombre de noeud correspondant au nombre de lignes restantes (chaque noeud a une description), description des noeuds valide (bon type de noeud).
- Bonne validité des noeuds : Les coordonnées de chaque noeud dans le fichier .itf doit correspondre à un pixel noir dans l'image.
- Bonne validité des chemins : Le long d'un chemin entre 2 noeuds on doit rester sur le chemin (ne pas croiser d'autre pixel que des pixels de type chemin). **Pour ce faire, utilisez l'algorithme de parcours de segment de Bressenham**.
- Existence d'au moins une zone d'entrée et de sortie.
- Existence d'au moins un chemin entre la zone d'entrée et de sortie : c'est un parcours en profondeur simple du graphe des chemins (cf. section 4).

4 Les éléments du jeu et leur représentation informatique

4.1 Résumé des règles et liste des éléments du jeu

Le jeu se déroule comme suit. Après le chargement de la carte, le joueur dispose d'une certaine somme d'argent initiale pour construire, sur les zones constructibles de la carte, des tours de défense. Ces tours sont là pour détruire des monstres, arrivant par vagues de 10, et qui cherchent à atteindre la zone de sortie. Si un monstre parvient à la zone de sortie, la partie est perdue. Si, au bout de 50 vagues de monstres, aucun n'est parvenu à la zone de sortie, alors la partie est gagné.

Chaque monstre détruit par les tours rapporte de l'argent au joueur. Cet argent permet au joueur de construire des nouvelles tours ... L'ensemble des éléments du jeu est donc :

- La carte décrite section 3 avec ses chemins
- Les monstres
- Les tours
- Les batiments

Les batiments comprennent d'une part des centrales d'énergie qui alimentent les tours de défense et d'autres parts des installations qui apportent des améliorations aux tours. Si une tour n'est pas complètement alimentée, elle ne peut pas tirer.

4.2 Les chemins

Les chemins de la carte forment en fait un graphe que nous allons utiliser pour indiquer aux monstres leur chemin. Dans ce graphe, chaque sommet représente une intersection, un coude ou encore la zone de sortie ou une zone d'entrée. Les arcs représentent les chemins rectilignes reliant coudes, intersections, zones d'entrée/sortie.

Vous devrez donc, dans votre application, construire un graphe représentant l'ensemble des chemins de la carte. Cela vous permettra entre autre de vérifier l'existence d'un chemin entre la/les zones d'entrée et celle de sortie. Les données associés aux sommets de ce graphe sont simplement le type de sommet (coude, intersection, zones d'entrée/sortie) ainsi que les successeurs de ce sommet.

Les arcs disposent également de données et notamment une valeur représentant le risque pour les monstres d'emprunter cette partie de chemin. Cette valeur est calculée comme suit :

$$\text{valarc} = \text{nbPixel} \times \Sigma_{tour}\text{recouvrement}$$

nbPixel est le nombre de pixels en 8 connexité de l'arc entre les 2 sommets du graphe. recouvrement indique, pour chaque tour de défense, le nombre de pixel de l'arc situé dans la zone d'effet de la tour (voir section suivante). Cette pondération des arcs du graphe sera utile pour le calcul du chemin des monstres.

4.3 Les monstres

Les monstres disposent de points de vie et d'une résistance propre à chacun des types de tour (voir section suivante). Les monstres arrivent par groupes de dix. À chaque nouvelle vague, les monstres voient leur point de vie et/ou leur résistance augmenter. C'est à vous de définir cette augmentation. De même, chaque monstre tué rapporte de l'argent au joueur. Initialement, les monstres de la première vague rapporte 5 unités d'argent. À chaque nouvelle vague, les monstres rapporteront plus d'argent.

Vous devrez implémenter au moins 2 type différents de monstre ayant des caractéristiques distinctes notamment de vitesse.

Graphiquement, les monstres seront représentés dans le jeu par ce qu'on appelle des sprites. Les sprites sont des images rectangulaires possèdant bien souvent de la transparence. Affichés sur une image de fond, ils permettent de représenter un élément (ici les monstres ou les tours) sur un fond (ici la carte).

4.4 Les tours

Le joueur peut créer des tours de défense qui tireront sur les monstres. Les tours ont quatre caractéristiques : la puissance, la portée, la cadence et la consommation d'energie. La puissance indique les dégats effectués par la tour. La portée indique en pixels la distance à laquelle la tour peut tirer. La cadence indique en nombre de dizième de secondes l'intervalle entre deux tirs. Enfin la consommation indique la quantité d'énergie nécessaire à la tour pour fonctionner.

Les tours sont de quatre types :

- Les tours de type rouge : Type rocket. Ces tours infligent beaucoup de dégats mais ont une cadence de feu faible.
- Les tours de type vert : Type laser. Elles tirent très rapidement mais ont une faible portée et occasionne des dommages moyens.
- Les tours de type jaune : Elles occasionnent peu de dégat, ont une portée très limité mais une bonne cadence de tir et tirent sur tous les monstres à leur portée.
- Les tours de type bleue : Elles ont une très bonne portée et une bonne cadence de tir mais occasionnent peu de dégats.

Les différentes paramètres de ces différentes tours (dégats, portée, cadence, consommation) sont laissés à votre appréciation. Chaque tour a également un coût d'achat pour pouvoir être placée dans la carte.

Les tours sont représentées graphiquement par des sprites mais ont une forme de disque d'un rayon d'au moins vingt pixels (laissé à votre appréciation). De plus les tours ne peuvent être construites que sur une zone constructible, c'est à dire pas sur un chemin, ni sur une autre tour ou un autre batiment. Afin d'optimiser le placement de la tour, on souhaite éviter de vérifier si chaque pixel sous la tour (le sprite) à poser est de type constructible. En effet, on connait la liste des noeuds, des chemins et des autres tours et batiments. Il faut alors **vérifier un par un si le placement de la tour n'empiète pas sur un chemin (intersection disque / segment large), sur une autre tour (intersection disque / disque) ou sur un autre batiment (intersection disque / carré)**.

4.5 Les batiments

Il existe dans notre jeu, deux types disctincts de batiment : les centrales et les installations. Graphiquement, les batiments sont également représentés par des sprites **mais à base carrée** d'au moins vingt pixels (taille laissée à votre appréciation). Notez que ces carrés ont une largeur et une hauteur impaire ce qui permet de définir sans ambiguïté un centre à celui-ci. C'est à partir de ce centre que les portée sont calculées.

Comme pour les tours, on doit veiller à ce que le placement des batiments soient valides c'est à dire ne recouvre pas ni un chemin (segment large de 30 pixel), ni une tour (disque). Mêmes remarques que pour le placement des tours : on souhaite éviter un test pixel à pixel.

4.5.1 Les centrales

Les centrales sont extrèmement importantes pour le jeu car ce sont elles qui fournissent l'énergie aux tours de défense. En effet, sans énergie, les tours de défense ne tirent pas. On vous laisse le soin de déterminer le cout des usines mais la quantité d'energie qu'elles produisent est indiquée dans le fichier `.itd`. Ces centrales transmettent leur énergie à toutes les tours situées dans leur portée. Les centrales possède donc en plus de leur production d'énergie, une caractéristique de portée exprimée en pixels. Toutes les tours dont le centre est situé dans la portée de la centrale sont ainsi alimentées. De plus, les tours alimentées peuvent elle même servir de relai d'énergie pour toutes les tours situées à leur portée.

Afin de gérer au mieux la distribution d'énergie, il convient donc de réaliser un graphe représentant les centrales (il peut évidemment y en avoir plusieurs), les tours et leur relation de portée. Ainsi chaque sommet de ce graphe représentera soit une centrale, soit une tour. Et un arc reliera deux sommets si la tour (ou la centrale) du sommet destination est à portée de la tour (ou de la centrale) du sommet d'origine. Pour connaitre les tours qui sont alimentées, il convient de faire un parcours en largeur de ce graphe à partir de l'ensemble des centrales. Sur un même "niveau" dans le parcours en largeur, la répartition de l'énergie restante à distribuer est laissé à votre appréciation.

4.5.2 Les installations

Trois installations peuvent être construites par le joueur : le radar, l'usine d'armement, et le stock de munitions. Ces batiments sont eux aussi munis d'une portée. Le radar permet d'augmenter de 25% la portée de toutes les tours situées ... à portée du radar. L'usine d'armement permet d'augmenter de 25% la puissance des tours à portée. Enfin, le stock de munitions permet d'augmenter de 25% la cadence de tir des tours à portée. Le coût et la portée de ces installations est laissé à votre appréciation.

4.6 Résumé des structures de données à utiliser

Au delà de la définition de structures pour les tours, batiments, monstres... vous aurez besoin, pour l'implémentation de ce jeu, de deux graphes : le graphe des chemins et le graphe d'énergie (les liens énergétiques centrales/tours). Vous avez toute latitude sur le choix de l'implémentation du graphe des chemins. Notez par ailleurs qu'il s'agit d'un graphe non orienté. Par contre, pour le graphe d'énergie, **vous devrez utiliser une structure de donnée pointée** car il y aura beaucoup d'ajout et de modification de ce graphe. Il risque d'être également très utile de représenter les vagues de monstres sous forme de listes.

5 Le jeu ITD

5.1 Interface générale

L'application sera réalisée à l'aide d'OpenGL et de la SDL. Votre application contiendra au minimum une fenêtre d'affichage contenant la carte et une fenêtre interface SDL contenant au moins un bouton permettant de quitter l'application. De plus, une option dans les arguments du programme, ou une action de l'utilisateur (appui sur un bouton par exemple) affichera une aide pour l'utilisation du jeu.

5.2 Déroulement du jeu

L'application commence tout d'abord par charger la carte i.e. le fichier `.itd` (section 3) et vérifier que celle-ci est valide (voir section 3.3). Une fois la carte chargée le jeu peut commencer après une action de l'utilisateur.

Le jeu fonctionne en "continu" c'est à dire que c'est le temps qui rythme la succession des évènements. L'unité de temps est le dizième de seconde. Durant un dixième de seconde, les tours peuvent tirer au mieux une fois et les monstres peuvent se déplacer d'un pixel. En fait chaque tour tire toutes les n dizièmes de secondes où n est la cadence de tir de la tour. Les tours tirent sur le monstre le plus proche d'elle.

L'utilisateur peut par contre construire tour ou batiment quand il le souhaite. Lorsqu'il place une tour ou un bâtiment, l'application doit veiller à ce que ce placement soit valide : la tour ou le batiment ne doit pas recouvrir un chemin (voir section 4.4 et 4.5). Le joueur doit pouvoir également mettre en pause l'application.

Votre interface devra permettre de :

- Voir la carte, les monstres et les tours
- Indiquer l'argent restant disponible
- Sélectionner une tour ou un batiment à construire
- Placer cette tour ou ce batiment dans la carte
- Déplacer les monstres
- Sélectionner une tour déjà construite et afficher ces propriétés : puissance, cadence, consommation, portée ainsi que si elle est alimentée ou non.

Si un monstre arrive à une zone de sortie, le jeu est perdu. Si 50 vagues de monstres ont été éliminées alors le jeu est gagné.

5.3 Comportement des monstres

Si il y a plusieurs zone d'entrée, l'application choisira aléatoirement une zone d'entrée pour une vague de monstre donnée.

Lorsqu'il arrive en jeu, tout monstre doit construire son trajet pour atteindre la sortie. Afin de rendre un peu plus malin nos monstres, nous allons utiliser le graphe de chemin (dont les arcs sont pondérés par leur distance et l'influence des tours). Chaque monstre devra alors lancer l'algorithme de plus court chemin sur le graphe de chemin pour rejoindre la zone de sortie. L'utilisation de Dijkstra est indiquée. Ainsi chaque monstre connaitra la liste des noeuds du graphe de chemin constituant son trajet. Entre chaque noeud le parcours est rectiligne.

6 Travail supplémentaire pour les trinômes

En plus de la mention de la section 2.2, vous devrez réaliser les fonctionnalités suivantes :

- Format de fichier : Votre application devra pouvoir lire un fichier `.itd` au format 1 ou 2 (représenté dans la première ligne par `@ITD 2`). Le format de fichier `.itd` version 2 inclus les couples mot clé valeur suivants :

Mot clé	Valeur	Type de la valeur
`powerX`	Puissance pour les tours de type X (X à remplacer par R (rouge) V (vert), B (bleu) et J (jaune))	entier ≥ 0
`rateX`	Cadence pour les tours de type X (R,V,J,B)	entier ≥ 0
`rangeX`	Portée pour les tours de type X (R,V,J,B)	entier ≥ 0
`costX`	Coût d'achat des tours de type X (R,V,J,B)	entier ≥ 0
`costC`	Coût d'achat d'une centrale	entier ≥ 0
`rangeC`	Portée d'une centrale	entier ≥ 0
`costI`	Coût d'achat d'une installation (radar...)	entier ≥ 0
`rangeI`	Portée d'une installation	entier ≥ 0

- Zones non constructibles : Afin d'embellir la carte, tout pixel qui ne soit pas de la couleur d'une zone constructible, d'un chemin ou d'une zone d'entrée sortie est considéré comme une zone non constructible. Ainsi, lors de la création d'une tour ou d'un batiment, l'application devra vérifier, **en plus des intersections avec chemin et autre tour/batiment**, que les pixels recouvrant le sprite (disque ou carré) ne sont pas de type zone non constructible.

- L'éclair choquant : Les centrales alimentent les tours. Mais en contrepartie toute tour alimentée redonne à chaque tour, si c'est possible, une unité d'energie à chaque centrale à qui elle serait connectée (connection à partir de la tour donc). Au bout de 20 unités d'énergie accumulées de la sorte par centrale, une centrale peut lancer un éclair choquant. Celui-ci est constitué d'éclairs qui recouvrent l'ensemble des chemins et infligent de fort dégats (quantité laissé à votre appréciation) à tous les monstres. Pour savoir à tout moment, combien de tour sont à la fois alimentée et capable de réalimenter les centrales, il faut réaliser l'algorithme de detection des composantes fortement connexe sur le graphe d'énergie (voir section 4.6). Le nombre de tour de chaque composante fortement connexe indiquera le nombre d'unité d'énergie gagné par les centrales situées dans cette composante fortement connexe.

7 Des bonus

Tout outils, spécifications, fonctionnalités supplémentaires seront récompensés. On peut penser notamment à :

- Déplacement stochastique des monstres : déplacement aléatoire à chaque noeud rencontré.
- Différents types de terrain pour les chemins ralentissant ou accélérant les monstres.
- Visualisation des tirs des tours sur les monstres.
- Sélection des monstres et affichage de leur propriétés.
- Affichage du trajet prévu par un monstre sélectionné.
- Projectiles : Faire en sorte que les tours rouges envoient des projectiles. Ceux ci se déplacent en ligne droite vers le monstre ciblé.
- ...

Néanmoins, si une nouvelle fonctionnalité modifiait même de manière légère les spécifications, notamment le format des fichiers `itd`, présentes dans cette description de projet alors cette nouvelle fonctionnalité devra être validée par M. Nozick ou M. Biri. Elle devra aussi être indiquée clairement dans le rapport.

8 Conseils et remarques

- Ce sujet de projet constitue un cahier des charges de l'application. Tout changement à propos des spécifications du projet doit être validée par M. Nozick ou M. Biri.
- Pour l'intersection disque ou carré / segment large, vérifier **mathématiquement** que le disque est à une distance de 15 (= 30/2) de la représentation mathématique du segment.
- Le temps qui vous est imparti n'est pas de trop pour réaliser l'application. N'attendez pas le dernier mois pour commencer à coder.
- Il est très important que vous réfléchissiez **avant de commencer à coder** aux principaux modules, algorithmes et aux principales structures de données que vous utiliserez pour votre application . Il faut également que vous vous répartissiez le travail et que vous déterminiez les tâches à réaliser en priorité.
- Ne rédigez pas le rapport à la dernière minute sinon il sera baclé et cela se sent toujours.
- Le projet est à faire par binôme. Il est **impératif** que chacun d'entre vous travaille sur une partie et non pas tous "en même temps" (plusieurs qui regarde un travailler). sinon vous n'aurez pas le temps de tout faire. C'est encore plus vrai pour les trinômes.
- Utilisez des bibliothèques ! Notamment pour les types abstraits de files et de piles.
- N'oubliez pas de **tester** votre application à chaque spécification implémentée. Il est impensable de tout coder puis de tout vérifier après. Pour les tests, confectionnez vous tout d'abord de petites cartes (taille 5 par 5 par exemple) avec un chemin extrêmement simple. Si cela marche vous pouvez passer à plus gros ou plus complexe.
- Vos chargés de td et cm sont là pour vous aider. Si vous ne comprenez pas un algorithme ou avez des difficultés sur un point, n'attendez pas la soutenance pour nous en parler.