

Rapport de Projet : iRover

R. Joachim CLAYTON, Geoffrey DESBROSSES, Clément BAUCHET, Jean-Christophe GUILLON

13 décembre 2015

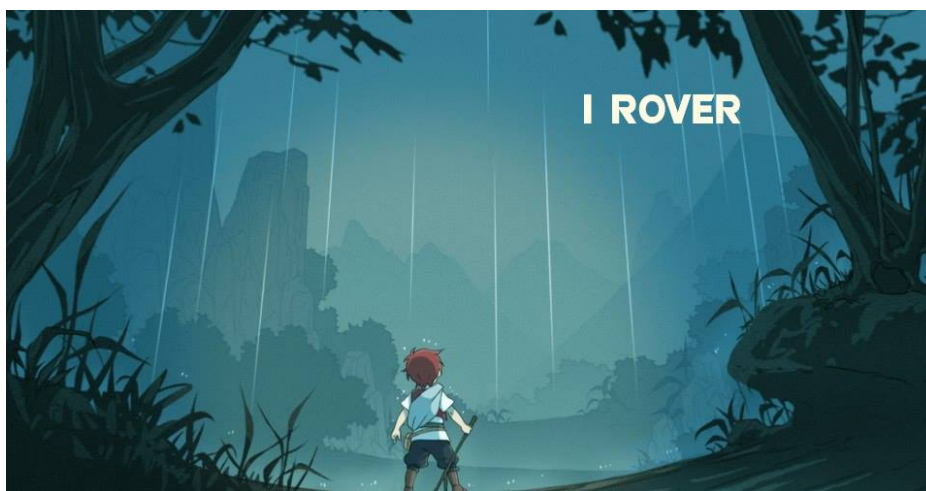


FIGURE 1 – iRover, l’histoire d’un héros qu’on appelait robot

Sommaire

| | | |
|----------|--|----------|
| 1 | Rapport de Projet | 3 |
| 1.1 | Introduction | 3 |
| 1.2 | Membres du groupe et tâches de chacun | 4 |
| 2 | Définition de la carte | 5 |
| 3 | Définition des objets | 6 |
| 3.1 | Les objets existants | 6 |
| 3.2 | Les objets manquants | 6 |
| 3.3 | Gestion des événements | 6 |
| 4 | IHM | 6 |
| 5 | IA | 7 |
| 5.1 | Path finding | 7 |
| 5.2 | Parcours de la carte | 7 |
| 6 | Documentation | 8 |
| 6.1 | Documentation Utilisateur | 8 |
| 6.2 | Documentation Technique | 8 |
| 6.3 | Le rapport | 8 |
| 7 | Problèmes rencontrés et idées non aboutis | 9 |
| 7.1 | Définition de la carte : Clan Lib | 9 |
| 7.2 | makefile | 9 |
| 7.3 | Documentation : FAQ | 9 |
| 7.4 | Documentation : bib | 9 |
| 7.5 | Svn | 9 |
| 8 | Conclusion | 9 |

1 Rapport de Projet

1.1 Introduction

Le projet a pour objectif la création d'une application graphique où un petit personnage (robot) réaliserait une tâche particulière.

Pour cela nous avons imaginé un scénario de mini jeu semblable au jeux en deux dimensions où notre robot prendrait la forme d'un héros sans peur et sans reproche.

L'autre aspect de ce projet est de pousser notre groupe à utiliser, comprendre et intégrer certains outils de développement, à savoir :

1. Utilisation des autotools.
2. Génération d'une documentation technique avec Doxygen.
3. Utilisation d'une plateforme svn pour la réalisation de notre projet.
4. Rédaction de documents avec L^AT_EX.
5. Réalisation de quelques test avec CppUnit.

1.2 Membres du groupe et tâches de chacun

| Responsable | Tâche | Description |
|------------------------|------------------------------|--|
| Clément Bauchet | Création graphique | La création de l'environnement graphiques et des images |
| | Parsing de la carte | L'interprétation entre les objets graphiques et les objets C++ |
| Geoffrey Desbrosses | Création des objets | Création de chaque objet de l'application et de leurs actions |
| | Documentation du code source | Gestion de la documentation Doxygen |
| | IHM | Interface utilisateur de l'application. |
| Jean-Christophe Guerin | IA | Définitions des algorithmes de pathfinding de déplacement et de découverte |
| Joachim Clayton | Documentation utilisateur | Redaction de la documentation utilisateur |

2 Définition de la carte

Pour commencer, il fallait générer un ou plusieurs terrains à l'aide du logiciel *Tiled*, un éditeur de cartes. La carte est créée avec une certaine *tilesheet* (une image contenant le dessin de toutes les cases possibles), et enregistrée sous le format **.tmx**, un format de type XML.

Pour pouvoir afficher le terrain dans notre application, nous avons d'abord fait appel au module *ClanXML* de la bibliothèque *ClanLib* qui est utilisée pour notre application, notamment pour l'affichage. Une classe **Map** a été créée, représentant le terrain et ses informations : les dimensions, les types de ses cases et la praticabilité de celles-ci par le robot ou d'autres entités. A la création d'une carte, ces informations sont récupérées directement depuis le fichier de carte **.tmx** fourni, en le parsant en tant que document DOM comme permis par *ClanXML*.

N.B. : A l'heure actuelle, il n'est pas possible de déterminer par seule lecture du fichier **.tmx** quels types de cases sont praticables par le robot. Ceci est déterminé au cas par cas dans le code. De plus, la praticabilité des cases se résume simplement à la possibilité ou non d'aller dessus. Nous pourrions définir un coût de déplacement pour chaque type de case, qui serait alors pris en compte dans l'évaluation du chemin optimal du robot.

Une fois les données de la carte obtenues, il nous faut afficher cette dernière dans la fenêtre principale de l'application. Comme nous ne disposons que de données brutes sur les cases (un nombre indiquant quelle texture sera utilisée pour dessiner la case), nous avons créé une autre classe, **Tileset**, qui représente l'image regroupant toutes les textures utilisables pour dessiner les cases (la *tile-sheet* décrite précédemment), ainsi que ses informations : ses dimensions, la taille des cases en pixels et l'espace entre chaque texture dans l'image. Là encore, ces données peuvent être retrouvées directement depuis le même fichier **.tmx** utilisé pour créer la carte. En connaissant le numéro d'une case de la carte, on peut alors retrouver quelle texture utiliser pour la dessiner. Il est donc désormais possible d'afficher toute la carte.

Enfin, nous avons défini la classe **mapDisplay**, qui sert actuellement de classe principale pour l'exécution de l'application. C'est ici que toutes les entités intervenant dans le déroulement du jeu sont affichées : la carte, le robot, ses ennemis et les objets.

Une possibilité d'extension serait de permettre au robot d'interagir directement sur son environnement et modifier le terrain au fur et à mesure de son exploration (*e.g.* en construisant des ponts, ou détruisant des murs...), ce que la structure de Map permet car la carte n'est pas en lecture seule après l'analyse du fichier **.tmx**.

3 Définition des objets

La définition des objets comprend la création de tous les objets Personnage, Coffre, Clef, Arme...et toutes les actions entre ces objets.

3.1 Les objets existants

Le robot possède une fonction pour se déplacer et modifier ses coordonnées. Il possède également une méthode *Combattre* propre à notre sujet. Des actuateurs *arme* et *armure* ont été implémentés. Le robot peut interagir avec d'autres objets tels que les ennemis, les clefs ou les coffres. Deux types d'armes ont également été créés (bazooka et mine) et les classes Arme et Armure permettent d'implémenter facilement de nouvelles armes et armures ayant leurs propres caractéristiques.

3.2 Les objets manquants

Le robot ne possède pas de senseur. L'idée était d'implémenter un scanner qui pouvait scanner une carte et savoir quels étaient les cases du terrain où le robot pouvait aller. Aucun type d'armure n'a été implémenté, cependant les personnages peuvent tout de même avoir une armure par défaut avec une robustesse et un nom. Ainsi ça ne modifie pas le comportement des personnages.

3.3 Gestion des événements

Certains objets peuvent interagir entre eux, par exemple la mine peut exploser au contact d'un ennemi, deux personnages peuvent s'affronter, le robot peut ramasser des clefs et ouvrir des coffres. Cependant, ces actions ne sont pas encore visibles sur l'interface graphique, excepté le déplacement du robot.

4 IHM

5 IA

5.1 Path finding

Pour réaliser le pathfinding nous avons utilisé l'algorithme A*, que nous avons trouvé parfaitement adapté à notre projet. Nous avons choisi sa version la plus simple qui permet de trouver le chemin le plus court sur une carte 2D représentée par une matrice contenant les coordonnées accessibles et inaccessibles. Le coût de déplacement n'est pour l'instant pas intégré mais il pourrait être ajouté sans modification lourde. Pour plus d'informations sur l'algorithme A* : https://en.wikipedia.org/wiki/A*_search_algorithm Pour plus d'informations sur notre implémentation, veuillez consulter la documentation technique.

5.2 Parcours de la carte

L'objectif initial était de doter les personnages de senseurs permettant de découvrir leur environnement et d'une mémoire pour stocker ces informations, au fur et à mesure de la découverte. Nous n'avons pas eu le temps d'intégrer cette fonctionnalité et en réalité chaque personnage a connaissance de l'entièreté de la carte. Les différents personnages sont donc dotés d'une IA de parcours, plutôt que d'exploration et de découverte. Les ennemis ont le comportement le plus simple : ils se déplacent constamment, à des coordonnées choisies aléatoirement. Le robot, quant à lui, suit une routine simple :

- S'il ne possède pas de clé, il va sur une case contenant une.
- S'il possède une clé, il va sur une case contenant un coffre. . . .

Le robot pour l'instant n'évite pas les ennemis.

6 Documentation

Au sujet de la documentation nous avons rédigé trois documents que nous liions à la doxygen du projet.

6.1 Documentation Utilisateur

La Documentation Utilisateur contient une approche scénaristique afin de donner envie à l'utilisateur d'entrer dans un monde fantastique afin de le divertir. Il est important dans un jeu de réfléchir à l'histoire qui entoure une application, un univers, une mythologie pour éventuellement faire évoluer et atteindre les générations. Toujours dans ce même document, la deuxième grande partie est un descriptif explicatif destiné à l'utilisateur. Ce document doit être compréhensible par tous et reprendre les éléments importants dont a besoin un utilisateur.

6.2 Documentation Technique

La Documentation Technique contient une approche comme son nom l'indique, beaucoup plus technique. Elle est destinée à un public programmeur ou à des informaticiens, dû au nombre de termes techniques employés. Elle est illustrée par Doxygen.

6.3 Le rapport

Le document rapport est le résumé de notre travail ayant pour lecteur notre professeur de module "Concept et Outils de développement". Nous avons souhaité chacun ajouter un descriptif de nos parties mais aussi faire part des problèmes rencontrés et des améliorations que l'on souhaitait ajouter.

7 Problèmes rencontrés et idées non aboutis

7.1 Définition de la carte : Clan Lib

Malgré un début difficile en raison de problèmes à la compilation et l'édition des liens de ClanLib, cette bibliothèque s'est avérée assez facile et intuitive à utiliser, ce qui a permis de vite obtenir un affichage de l'application.

7.2 makefile

Nous aurions aimé rajouter d'autres options au makefile ou au .configure plutôt pour fournir une partie plus complète sur ce sujet.

7.3 Documentation : FAQ

Nous avons pensé à ajouter une FAQ à nos documentations technique et utilisateur, pour permettre aux utilisateurs de ces documentations d'avoir des réponses aux questions récurrentes qui se posent aux premiers abords d'une application de ce type.

7.4 Documentation : bib

Nous voulions également créer une bibliothèque de référence pour permettre un archivage et une indexation plus efficace de nos documents, sources, programmes et illustrations.

7.5 Svn

(commentaire personnel Joachim Clayton) Je me suis souvent demandé s'il était nécessaire ou bon d'utiliser les branches et les tags et faire du vrai versioning. Il est vrai que si le projet était arrivé à une version finie on aurait dû l'utiliser, pour pouvoir travailler sur une v2.0 et y rajouter des fonctionnalités.

8 Conclusion

Notre objectif était de pouvoir créer une petite application et surtout l'utilisation des autotools linux, de la plateforme svn et la création de la documentation. Nous avons eu des difficultés mais avons pu dans l'ensemble utiliser ces outils :

1. Utilisation et rédaction d'un makefile
2. Création d'une documentation Doxygen grâce aux commentaires dans le code.
3. Utilisation de SVN via linux ou tortoise sur les machines Windows
4. Rédaction de documents avec \LaTeX .