Documentation Technique : iRover

13 décembre 2015

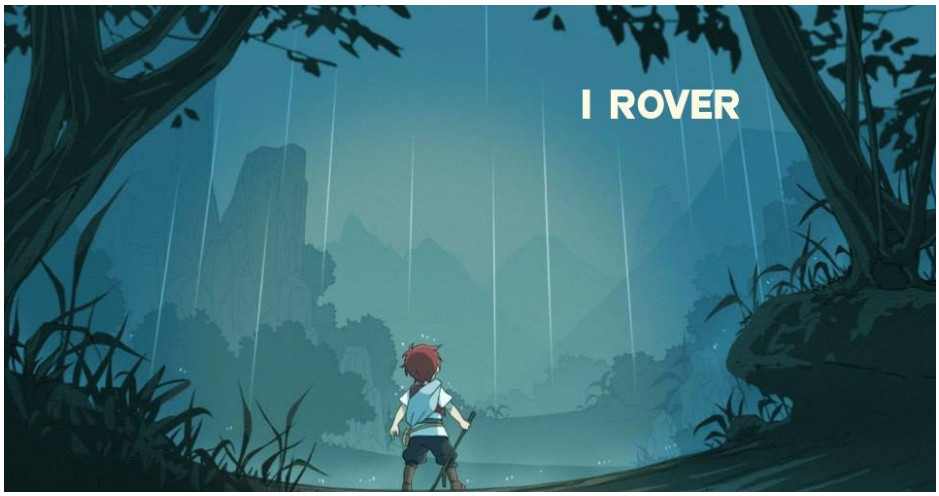

FIGURE 1 – iRover, l'histoire d'un héros qu'on appellait robot

Sommaire

1 Manuel Documentation Technique

Cette partie est dédiée aux développeurs et aux utilisateurs expérimenté ayant une culture informatique. Les divers éléments de l'application y sont décrit de façon plus mathématique et logique. Vous pouvez vous rapprocher de la doc Doxygen pour avoir encore plus de détails.

1.1 Génération Doc Doxygen

executer la commande pour générer le fichier de config

```
doxygen -g config_dox
```

modifier les champs suivants dans le fichier de config :

```


PROJECT_NAME           = "I-rover"

PROJECT_NUMBER         = 1.0

OUTPUT_LANGUAGE        = French
INPUT                  = CodeMain

FILE_PATTERNS          = *.hpp *.cpp


```

executer la commande :

```
doxygen config\_dox
```

2 Les personnages

Le jeu possède des personnages. Ceux-ci sont représentés par la classe personnage. Tous les personnages possèdent des attributs :

1. positionX : La position du personnage sur l'axe x.
2. positionY : La position du personnage sur l'axe y.
3. arme : L'arme qui possède le personnage.
4. armure : L'armure que possède le personnage.
5. puissance : La puissance du personnage, déterminée à partir de son arme.
6. robustesse : La robustesse du personnage, déterminée à partir de son armure.
7. isActive : Pour savoir si le personnage est encore en vie.
8. sprite : L'image du personnage dans le jeu.
9. collision map : Un vecteur qui détermine les endroits où le personnage peut aller.

Grâce à ces attributs, tous les personnages peuvent se déplacer d'une case grâce à la méthode deplacer(int x,int y). Les personnages peuvent aussi combattre d'autres personnages grâce à la méthode combattre(Personnage* personnage).

Le vainqueur d'un combat est déterminé par celui qui a la plus grande chance de gagner. Les deux adversaires tirent un nombre déterminé par l'algorithme suivant :

$$random() \times puissance \times robustesse \tag{1}$$

Ainsi on compare ces deux nombres. Celui qui a le plus grand gagne le duel.

2.1 Le robot, Rover

Le robot, en plus d'hériter des propriétés des personnages, possède un attribut *inventaire* qui représente le nombre de clefs qu'il possède.

L'objectif du robot est d'ouvrir tous les coffres de la carte et chaque coffre s'ouvre à l'aide d'une clef. Ainsi, le robot possède les méthodes *ramasser (Clef* clef)* ou *ouvrir(Coffre* coffre)*.

Le robot se déplace d'une certaine manière, grâce à un algorithme de pathfinding**??**.

2.2 Les ennemis

Un ennemi est un simple personnage qui a sa propre façon de se déplacer et de combattre. Un ennemi se déplace aléatoirement sur la carte, et continue tant que la partie n'est pas finie ou tant qu'il est encore en vie.

Lorsqu'un ennemi est à une case du robot, alors celui-ci affronte le robot.

3 Les coffres

Les coffres sont de simples objets qui possèdent les propriétés suivantes :

1. positionX : sa position sur l'axe x.

2. positionY : sa position sur l'axe y.
3. ouvert : booléen qui permet de savoir si le coffre est ouvert ou fermé.
4. sprite : l'image du coffre.

Une fois le coffre ouvert, son paramètre ouvert passe à true et ne peut donc plus s'ouvrir.

4 Les clef

Les clefs ont les mêmes attributs que les coffres excepté pour l'attribut ouvert. Une fois la clef ramassée, celle-ci disparait de la carte et l'inventaire du robot augmente de 1.

5 Les actuateurs

Le robot, tout comme les ennemis possèdent deux sortes d'actuateurs : les armes et les armures.

Ces armes et armures déterminent la puissance et la robustesse de chaque personnage du jeu.

Plusieurs armes existent dans le jeu, un bazooka qui augmente fortemment la puissance du possesseur, des mines que le personnage laisse derrière lui régulièrement...

Grâce à ces différents actuateurs, il est possible de choisir sa stratégie selon le nombre d'ennemis sur la carte, la forme de la carte...

Toutes les armes héritent de la classes Arme et les armures de la classe Armure. Ainsi, i lest facile de créer de nouveaux types d'armes et armures.

6 Les senseurs

Le robot possède un scanner. Celui-ci permet de scanner une map et de savoir quels sont les endroits où il peut aller et les endroits qui lui sont inacessibles.

7 L'environnement

Le terrain est représenté par la classe **Map**, qui contient les informations sur ce terrain (hauteur, largeur, taille des cases) et deux matrices contenant respectivement les types de cases de la carte, et la praticabilité de celles-ci. Cette dernière matrice est passée aux entités évoluant dans cette carte, pour connaître les cases passables ou non et adapter leurs algorithmes de recherche de chemin en conséquence. Une carte se construit à partir d'un fichier au format *.tmx*, un format de carte géré par le logiciel **Tiled**, un éditeur de cartes libre. Ce fichier est alors traîté comme un document DOM, grâce au module *ClanXML* de la bibliothèque *ClanLib 4.0*, pour en extraire les informations et les numéros des cases.

Pour gérer le dessin de la carte, la classe **Tileset** permet de représenter une feuille de textures consistant en une image regroupant toutes les textures des cases pouvant être dessinées. Ce tileset possède des métadonnées telles que le nombre de textures en lignes, colonnes, leur taille en pixels et l'espacement entre chacun dans l'image, métadonnées qui peut être récupérées également depuis le fichier de carte .tmx fourni. A partir d'un numéro de tile à une certaine position dans la carte, on peut alors déterminer quelle texture doit être dessinée à l'emplacement correspondant.

La classe **mapDisplay** est pour l'instant la classe principale de l'application, où sont affichés la carte et tous les éléments interagissant avec. Tout l'affichage est géré par le module *ClanDisplay* de la bibliothèque *ClanLib 4.0*, qui dessine au milieu de la fenêtre toute la carte case par case conformément au Tileset fourni, ainsi que tous les personnages et les objets.

8 La gestion des évênements

La gestion des évênemments concerne toutes les actions que peuvent faire les personnages dans le jeu.

8.1 Rencontre avec un ennemi

Lorsque le robot rencontre un ennemi, un combat se déclenche. Une des deux personnages en sort vainqueur, si le robot gagne alors la partie continue, sinon la partie s'arrête.

8.2 Ouverture d'un coffre

Lorsque le robot possède au moins une clef et qu'il se trouve à la même position qu'un coffre, alors il l'ouvre. Le statut ouvert du coffre passe donc à true.

8.3 Ramasser une clef

Lorsque le robot est à la même position qu'une clef, celui-ci la ramasse automatiquement. Son attribut inventaire est donc incrémenté de 1.

9 IA

9.1 path finding

Le code responsable du pathfinding est contenu dans *astar* défininissant la classe *Node* et la méthode *pathFind()*. Grâce à cette méthode on obtient le plus court chemin entre un objet et un couple de coordonnées sur la carte. Le chemin est obtenu sous la forme d'une chaîne de caractères, chaque caractère étant 0, 1 ,2 ou 3 et correspondant à un mouvement à effectuer comme suit :

1. 0 :Le personnage se déplace d'une case à droite
2. 1 :Le personnage se déplace d'une case vers le bas
3. 2 :Le personnage se déplace d'une case à gauche
4. 3 :Le personnage se déplace d'une case vers le haut

La fonction *goTo()* lit cette chaîne de caractères et réalise les différents mouvements à l'aide de la fonction *deplacer()* ;

9.2 découverte de la carte

L'exploration et le pathfinding sont testés dans la classe *MapDisplayTestPathfinding*, qui est une modification de la classe *MapDisplay*.

9.2.1 Exploration du robot

Le robot a accès aux coordonnées de l'ensemble des clefs et des coffres. S'il ne possède pas de clef, il cherche la clef la plus proche avec la fonction *findNearestKey()* et s'y dirige. Ensuite il cherche le coffre le plus proche avec la fonction *findNearestChest()* et s'y dirige. La clef étant normalement détruite après avoir ouvert le coffre, le robot peut effectuer ces actions en boucle.

9.2.2 Exploration des ennemis

Un ennemi reçoit des coordonnées aléatoires(contenues dans la carte) et s'y déplace avec la fonction *goTo()*.