L3 Info
Programmation avancée en C

Stealth

Antoine BASTOS

Yann ROUX—DAUGROIS

12/01/23

Sommaire

Approche du sujet....3
Problématiques rencontrées....4
La vue....4
Implantation de la salle....5
Déplacement....5
Gestion des collisions....6
Gestion du temps....6
Scores....6

Approche du sujet

Après lecture du sujet nous avons choisi d'utiliser une architecture MVC.

Notre implantation mettait en œuvre le graphe d'inclusion ci-contre. Cependant à mi-parcours nous nous sommes rendu compte que l'on faisait une erreur de conception.

En effet l'objet Room contenait l'ensemble des éléments du jeu. Ainsi les fonctions liées au déroulement du jeu, les événements, appartenait au module Room. Or la bonne méthode est de séparer les règles et les manières par lesquels les éléments du modèle interagissement entre-eux, du modèle lui-même.

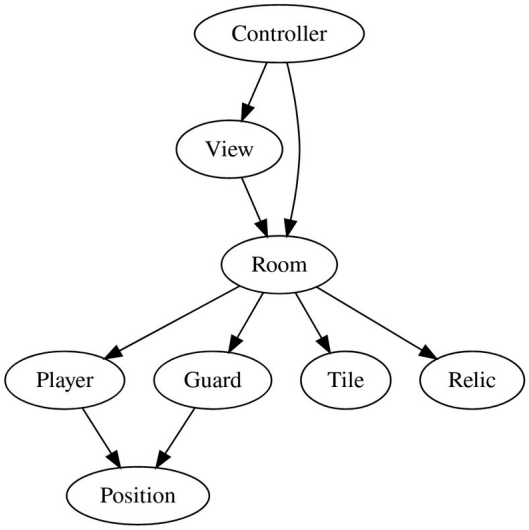


Figure 1: Graphe d'inclusion (non étendu) et synthétique du jeu (v1)

Nos modifications nous on données ce graphe d'inclusion :

Un structure GameData contient les données d'une partie. Elle permet au contrôleur de manipuler les données du jeu.

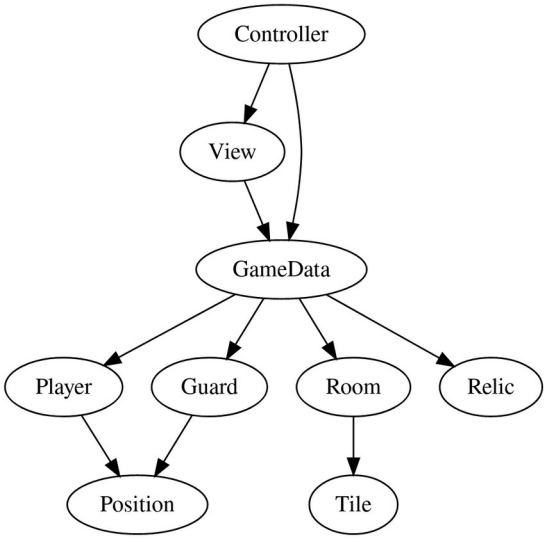
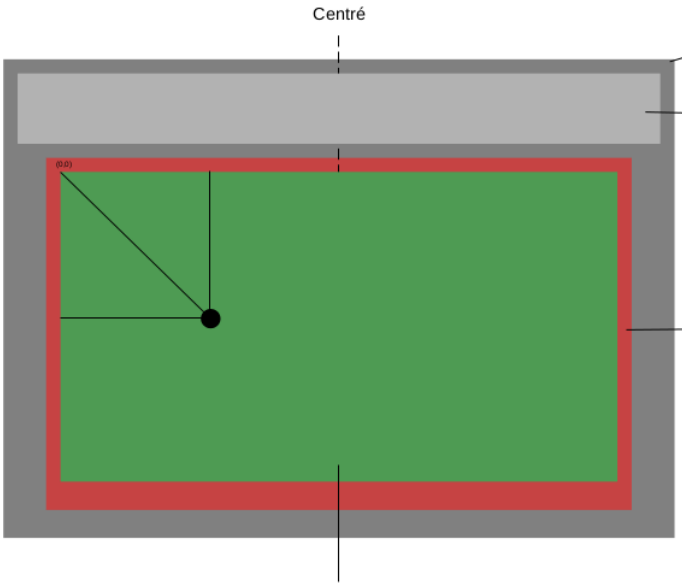


Figure 2: Graphe d'inclusion (non étendu) et synthétique du jeu (v2)

Problématiques rencontrées

La vue

Conceptualiser la vue avant de commencer à coder, a tout de suite permis de bien séparer le système de coordonnées du modèle de son affichage.



Zone disponible dans la fenêtre pour générer la zone de jeu. Cette zone dépend de la taille de la fenêtre. Ansi la zone de jeu est “responsive”. On
pourra donc faire varier la taille de la fenêtre et mettre le jeu en plein écran.

La zone de jeu pourra être positionnée n'importe où dans la fenêtre. Ici un “padding” est représenté entre la zone de jeu et les bords de la fenêtre, ou les autres éléments, pour mettre en évidence que cette zone n'est pas fixé. Cet espacement pourra à souhait être nul ou non.

Zone de jeu

La zone de jeu ne possède pas de position fixée. Elles est d'une aire inférieure ou égale à la zone disponible et centrée dans cette dernière.
Les position de toutes les entités de la zone de jeu sont calculées par rapport à cette origine relative, de la zone de jeu.

(En général on a évité de définir une position de façon absolue dans la fenêtre.)

Implantation de la salle

Nous avons utilisé un tableaux à deux dimensions pour stocker les tuiles. Il pourrait être judicieux d'utiliser un tableau de pointeurs plutôt que de structure. Ainsi seuls les tuiles comportant un intérêt (mur, mana, relic) seront sauvegardé en mémoire. L'intérêt majeur d'utiliser un tableau à deux dimensions plutôt qu'une liste est que, étant donné que l'on connaît les indices (i,j) de la tuile sur laquelle une entité se trouve à tout instant du jeu, on peut facilement gérer leur mouvement et prévenir les collisions.

Déplacement

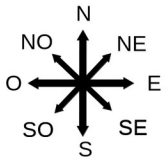
Les paramètres du jeu sont définit par des macros dans le fichier d'entête Settings.h

Du fait de l'architecture MVC la représentation du modèle est complètement décorrélé de la vue et la quantité de déplacement d'une entité dans le modèle n'a rien à voir avec la représentation de la quantité du déplacement à l'écran.

La fenêtre possède un taux de rafraîchissement de 60 FPS. Cette constante est stockée comme macro. On définie une autre variable globale, SPS (side (ou square) per second), pour définir la vitesse de base des entités quand v vaut 1.

Comme la boucle de jeu est réalisée 60 fois par seconde alors le déplacement d'une entité est un vecteur.

- de norme (v * SPS) / FPS

- de direction 

v la vitesse de l'entité

Si un garde à un déplacement de v = 0.3.
Et que SPS = 10

Sa vitesse est de 0.3 fois la vitesse de déplacement de base
du jeu, soit 3 case par seconde.

Donc le déplacement dans une frame de l'entité garde dans le modèle est de
v * SPS / FPS = (0.3 * 10) / 60

Gestion des collisions

Nous avons opté pour une résolution des collisions différente que celle proposée dans le sujet.

Dans notre cas c'est une réponse après-coup à la collision qui est réalisée. Quand un entité se déplace sur des coordonnées sur lesquelles sont disque la représentant chevauche un mur, on calcule la distance entre le point le plus proche de ce mur et le centre du cercle. On obtient un vecteur qui nous permet de déplacer le centre du cercle proportionnellement à son chevauchement sur le mur, en sens inverse.

Gestion du temps

A la place de manipuler le temps via clock_gettime on se sert des fonctionnalités de la lib MLV qui permettent de définir un taux de rafraîchissement et d'adapter la vitesse d'exécution de la même manière, avec *MLV_change_frame_rate* et *MLV_delay_according_to_frame_rate.*

Pour l'écoulement du temps nous avons repris un petit module Timer que Antoine avait écrit pour un tp de l'année passé qui nécessitait d'implémenter un chronomètre. En outre l'utilisation de clock_gettime avec le flag `CLOCK_REALTIME` est équivalent à gettimeofday de sys/time.h utilisé dans le module Timer.

Scores

Nous avons douté sur l'ordre par lequel trier les scores par l'utilisation de mana. Il était possible de trier dans l'ordre décroissant, ce qui témoigne que le joueur parvient à maîtriser l'environnement de jeu dans lequel il se trouve à tel point qu'il lui est possible d'utiliser des compétences lui permettant de gagner. Sinon de trier dans l'ordre croissant, visant à valoriser les joueurs parvenu à gagner en ayant utilisé le moins de mana possible, donc, dont le jeu à été le moins simplifié par l'utilisation de compétences. Sans réelle conviction pour l'une ou l'autre solution nous avons finalement choisi la seconde.

Amélioration ajoutées

Nous avons ajoutés des effets visuel et sonores. L'obtention de mana déclenche un son et la mise à jour de la barre de mana, dans la zone d'information. Les compétences active son aussi précisées dans cette zone au dessus de la barre de mana.

En mode panique la musique change et un filtre rouge dont l'alpha varie est appliqué sur la fenêtre.

De plus des images on été utilisées pour rendre le jeu plus attrayant.