

Rapport de projet

Alexis Laouar, Rémi Oudin, Kévin Le Run

1 Vue d'ensemble

L'architecture du moteur de jeu est une variante du Modèle-Vue-Contrôleur où les modèles ne sont pas purs et ont une certaine part de contrôle.

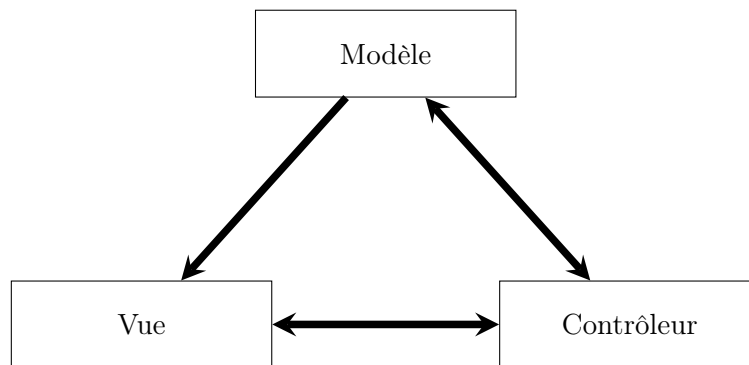


Figure 1: Modèle-Vue-Contrôleur

Les contrôleurs sont placés dans le package `runtime`, les modèles dans le package `game_mechanics` et les vues dans le package `gui`.

2 Les contrôleurs

Le contrôleur principal est l'objet `Controller`. Il contient la boucle `update-render` ainsi que l'ensemble des modèles actifs : les tours, les ennemis et les projectiles. À chaque tour de boucle, tous ces modèles sont mis à jour et dessinés à l'écran. La mise à jour des mouvements se fait par méthode d'Euler (plus de détails en section 3).

3 Modèles

Pour les modèles, nous avons distingués 3 modèles principaux.

3.1 Les ennemis

Les ennemis, qui possèdent leur chemin et vont principalement avancer en recalculant la trajectoire au dur et à mesure, à l'aide de la méthode d'Euler. Les ennemis ont les attributs suivants:

- HP, qui représente leurs points de vie
- Shield, qui représente la réduction de dégâts
- Speed, qui représente leur vitesse de déplacement
- Reward, la récompense.

Pour bouger, l'ennemi connaît la position de la case suivante dans laquelle il doit aller. Le mouvement qu'il effectue est continu, c'est-à-dire qu'il va de case en case mais de manière continue, ce qui l'amène à devoir recalculer son mouvement à chaque nouvelle case atteinte. Pour l'instant, il n'a besoin que de sa case de départ (pour l'apparition) et de sa case d'arrivée. À chaque mise à jour par le contrôleur, les monstres vérifient leur nombre de HP, et demandent au contrôleur de l'oublier s'ils atteignent un nombre de HP négatif.

3.2 Les tours

Une tour a une position fixe, et diverses caractéristiques, dont la portée, les dégâts, le coût, et le cooldown. Pour tirer, la tour prend à chaque nouveau tir l'ennemi qui est le plus avancé vers le core, et crée un objet projectile dont la cible est le l'ennemi visé. Il transmet les propriétés de dégâts et de vitesse du projectile à celui-ci.

3.3 Les projectiles

Un projectile est un objet créé par une tour. Il a pour attributs:

- Une cible qui est un ennemi,
- une vitesse,
- une position courante,
- des dégâts.

Il suit son ennemi, c'est-à-dire qu'à chaque étape, il va récupérer la position courante de sa cible, et se diriger vers lui. On considère que si après avoir bougé, un projectile a dépassé sa cible, il l'a touché. Alors, le projectile demande à sa cible de se retirer des HP.

4 Vues

5 Autres fonctionnalités développées

5.1 Génération automatique de vagues

Encodage des vagues

Les vagues sont stockées dans des fichiers `.csv` de la forme suivante :

```
0.000000, Bunny
0.249445, Bunny
0.498891, HeavyBunny
0.748336, Bunny
0.997781, Hare
1.247226, Hare
1.496672, HeavyBunny
1.746117, Bunny
1.995562, Bunny
2.245007, Bunny
2.494453, Bunny
2.743898, HeavyBunny
2.993343, Bunny
3.242788, Bunny
...
```

Le flottant est le temps (en secondes) d'apparition de l'unité, et le string qui suit est son identifiant.

Génération des vagues

Les fichiers de vagues sont générés automatiquement par un script `CamL` qui reçoit en argument le numéro de la vague. Chaque unité dispose des attributs suivants :

1. Son nom.
2. Sa difficulté, c'est-à-dire à quel point cette unité sera difficile à vaincre.
3. La première vague possible d'apparition.
4. Sa rareté, qui dépend du numéro de la vague. Par exemple, l'unité de base sera commune en début de jeu, puis sera progressivement remplacée par des unités plus difficiles.

Etant donné un numéro de vague, le programme génère une vague aléatoire comme suit.

- A partir du numéro de la vague, un indice de difficulté est calculé selon une fonction donnée. Actuellement, il s'agit d'un polynôme de degré trois.
- Encore à partir du numéro de la vague, un temps d'apparition dt est calculé. Actuellement, ce temps décroît d'une seconde à un dixième de seconde avec une décroissance en $\frac{1}{x^{0.7}}$.
- A partir des raretés des unités, une unité est choisie aléatoirement pour apparaître à l'instant présent. On compare ensuite sa difficulté à la difficulté de la vague : soit $d = \text{diff}_{\text{vague}} - \text{diff}_{\text{unité}}$.
 - Si $d \geq 0$, l'unité peut apparaître ; on écrit donc dans le fichier de sortie l'instant présent ainsi que le nom de l'unité, et on recommence cet algorithme à partir d'une date incrémentée de dt .
 - Si $d < 0$, l'unité est trop puissante pour apparaître. Si la difficulté est telle qu'une des unités peut apparaître, on recommence ; sinon, l'algorithme se termine.

L'intérêt de cette méthode de génération est que l'utilisation de fonctions pour la difficulté d'une vague, la rareté d'une unité et le temps d'apparition des unités permet très facilement d'effectuer des modifications d'équilibrage d'une part en ne modifiant que ces fonctions et non le programme, et d'autre part nous laisse une grande liberté d'action sur l'allure des vagues tout au long du jeu.

6 Conclusion

Ca marche.

