Sprint review — Sprint 1

Escape

Le backlog en détail : https://github.com/victord54/escape/tree/main/docs/sprints/1

Rappel backlog sprint 1 :

- ✅ Pathfinding : Victor & Claire
- ✅ Déplacement du joueur GUI : Théo
- ✅ Relier GUI au jeu : Antoine
- Sprites : Dan
    - état actuel : backlog (pas commencé)

Modifications du backlog :

- ✅ [feature] Améliorations GUI : Antoine
- ✅ [hotfix] Plantage de la CLI : Antoine
- [bugfix] GUI lignes entre cases : Antoine
    - état actuel : en attente de la fonctionnalité “Sprites”

Problèmes rencontrés :

→ **Pathfinding** : Transformer le monde en graphe a été un peu compliqué. On avait des flottants pour les x, y avec beaucoup de chiffres après la virgule donc pour simplifier les calculs, on a transformé les flottants en entier en prenant 4 chiffres après la virgule. L’un des autres problèmes que l’on a rencontré, c’était le nombre de nœuds que l’on pouvait faire pour que le pathfinding fonctionne sans que l’algorithme ne prenne trop de temps à s’effectuer.

→ **Pathfinding** : Problème au niveau de la vitesse d’exécution de l’algorithme de pathfinding pour tous les monstres. On s’est très vite rendu compte que quand il y avait plusieurs monstres, les FPS descendaient beaucoup dû au fait que le temps d’exécution pour chaque monstre, les uns après les autres, était un peu long. Pour régler ce problème, on a lancé le pathfinding pour chaque monstre sur un Thread différent afin de paralléliser les calculs.

→ **Améliorations GUI** : La modification d’un paramètre du jeu entraîne différents changements. Dans le but de simplifier le code existant, et de pouvoir ajouter d’autres évènements de ce type plus tard facilement, on a réfléchi à comment notifier certains composants sans se prendre la tête. Le patron Observable/Observer de Java étant déprécié, on a pensé à PropertyChangeSupport, abordé en cours, mais c’est un peu pénible à utiliser. Cela impliquait de devoir indiquer dans une chaîne de caractère, sans se tromper, le nom de la propriété à surveiller (ou utiliser des constantes partout), et caster dans l’écouteur l’objet reçu. Cela a demandé beaucoup de réflexion et d’essais. Ayant en tête le fonctionnement proposé par certains frameworks dans d’autres langages, nous sommes finalement partis sur du PropertyChangeSupport, mais où notre implémentation cache la lourdeur de son utilisation, tout en étant très souple, typé et avec des méthodes explicites.

→ **Déplacement du joueur GUI** : En fonction des machines et des écrans utilisés par ces machines, le jeu n'a pas un taux de rafraîchissement identique. De ce fait, appliquer un déplacement aux personnages à chaque frame rendrait l'expérience du jeu inégale pour chaque joueur (les personnages seraient plus rapides dans le cas où le framerate est plus élevé). Après réflexion, nous avons donc remédié à ce problème en ajoutant un paramètre nommé "deltaTime" à la méthode de déplacement des personnages. Ce paramètre représente la différence de temps entre le dernier appel à la méthode et celui fait à l'instant présent. Ce paramètre peut donc être utilisé par le personnage pour rendre la vitesse de ses déplacements proportionnelle au temps réel et non au taux de rafraîchissement. À noter que ce *deltaTime* est déduit par le moteur du jeu lorsqu'on demande à déplacer les personnages.

Diagramme de classes général :

Consulter avec une meilleure qualité (permalink), Consulter la version du sprint précédent

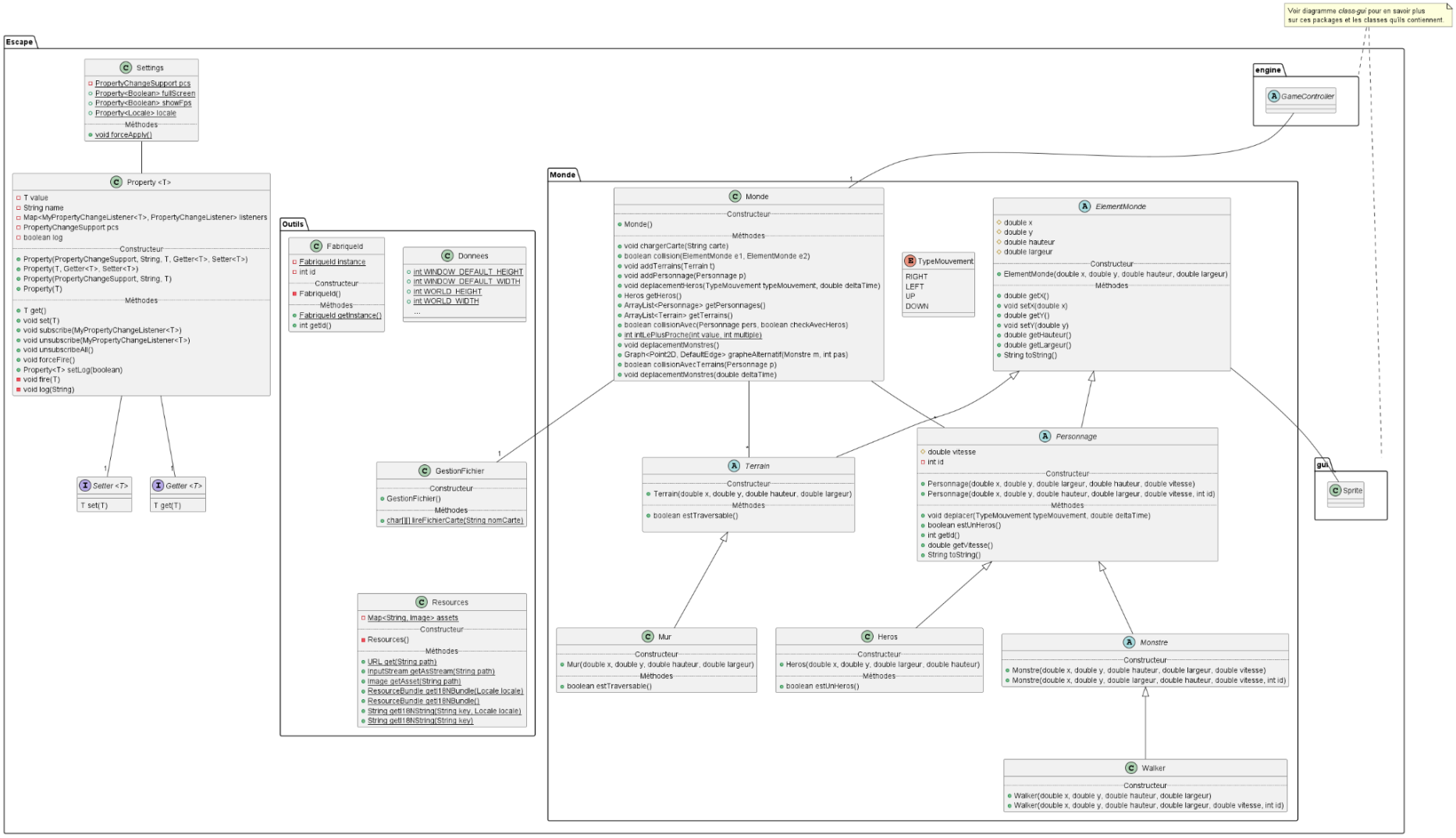

Diagramme de classes de l'interface graphique :

Consulter avec une meilleure qualité (permalink), Consulter la version du sprint précédent