Joseph Lemire

Nicholas Bourré

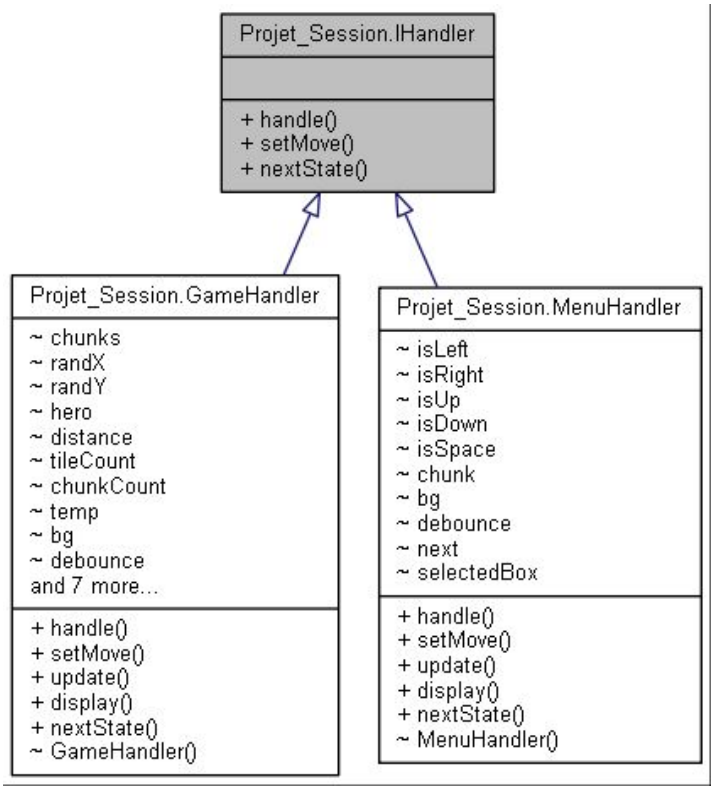
Projet de Session

5 novembre 2019

Side Runner

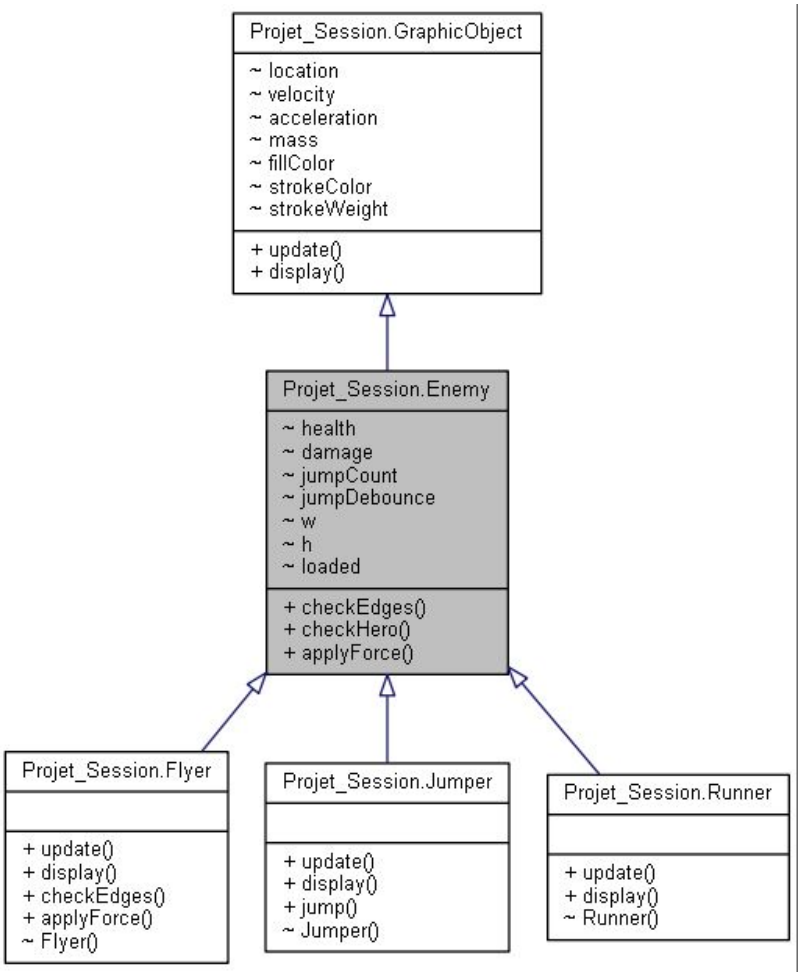
State

J’ai choisi le State design pattern pour faciliter le fonctionnement de mon menu. Mon interface IHandler est mon current state. Au début du programme il est mis à MenuHandler. handle() et setMove() sert au fonctionnement du programme. Le state change selon le retour de fonction nextState(), soit pour débuter la partie ou lorsque le héro meurt.



Template Method

J'ai choisi le Template Method pour le fonctionnement de mes ennemies. Chaque types d'ennemies héritent de l'abstract class Enemy qui a son tour hérite de GraphicObject. La class Enemy possède des fonctions générales qui s'appliquent à tous mes ennemies tel que checkEdges et checkHero() mais mes ennemies possèdent aussi des fonctions partagées mais a comportement différent, plus spécifiquement, update(). Le flyer n'est pas affecté par la gravité et a une vitesse de 6, le jumper a une vitesse de 2 et saute et, le runner, a tout simplement une vitesse de 4. Tout cela me permet de passer tous mes ennemies dans une boucle qui appelle les mêmes fonctions mais qui ont tous un comportement différent.



Builder

J’ai choisi le Builder pour générer mon monde. Il commence par générer une String qu'il convertit ensuite en paramètres qu’il envoie au ChunkConstructor, un sous-builder. Avec ces paramètres, le ChunkConstructor génère un array de tuiles ainsi qu’un array d'ennemies (celui-ci avec l’aide du EnemyFactory, un sous-sous-builder). Il les stores dans un chunk qu’il renvoie au Generator qui le ensuite dans un array. Lorsqu’il a fini de générer le monde, il renvoie l’array de chunks au jeu. Suite à cela, Generator et ses sous-constructeurs ne servent plus à rien.