Baurberg
Sharon
EISE4

Rapport du Mini-projet de C++

Description de l'application

Qu'est-ce que le monde d'après ? Le monde d'après est un univers où plus rien n'a aucun sens. Ce sont les poissons qui pêchent les hommes. Ce sont les carottes qui essaient de nous manger. Enfin, les chats sont désormais au pouvoir.

L'application est donc un jeu de type plateforme où le personnage principal doit se battre contre les carottes et les chats qui veulent envahir le monde. Le but est simple, se débarrasser des ennemis sans que nos Points de vie tombent à zéro !

L'application se base sur une macro classe, Plateformer, qui va gérer l'affichage des obstacles et des ennemis. Elle va aussi permettre de vérifier s'il y a un contact entre un personnage et un obstacle par exemple ou de lancer les fonctions propres aux obstacles ou aux ennemis afin de blesser le personnage principal.

Les deux autres classes de l'application sont la classe Perso (dans laquelle on distingue le personnage principal et les ennemis) et la classe Obstacle de laquelle dérive la classe Pic (matérialisant les obstacles qui blessent le personnage principal).

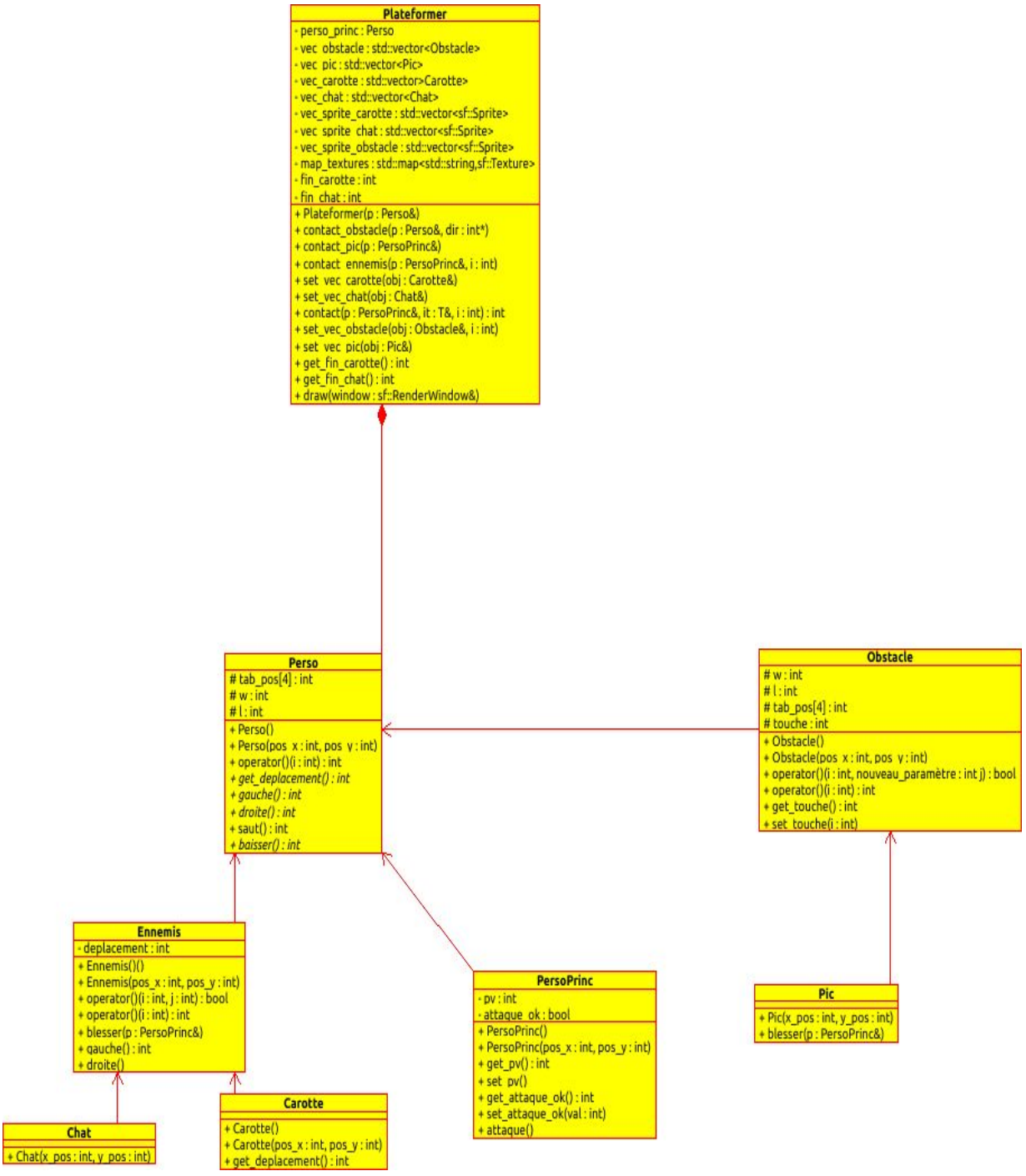
Le personnage principal (classe PersoPrinc) peut se déplacer à gauche, à droite,sauter (ou plutôt voler dans les airs, le monde d'après est bien étrange!) et a une possibilité de revenir sur le sol mais ne peut pas traverser des obstacles. Il peut être traversé par un ennemi. A cette occasion, des poins de vie lui sont enlevés. Dans le case où le personnage touche le bout d'un pic, des PV lui sont également enlevés.

Les personnages de la classe Ennemis se déplacent à une vitesse différente (leur attribut déplacement) tout en évitant aussi les obstacles. Ils ont aussi la possibilité de blesser le personnage principal.

Un Obstacle lambda n'a pas d'autre action dans le jeu que de bloquer le passage du joueur.

L'Obstacle Pic permet, lui, de causer des dégats au personnage principal s'il s'appproche trop près.

Voici le diagramme UML de l'application:



Pourquoi cette organisation semblait-elle être l'organisation optimale?

Le mécanisme d'héritage est logique. L'héritage permet d'ajouter des fonctionnalités en plus à un objet sans répéter de façon redondante celles déjà accordées. Ici, un Obstacle bloque le personnage principal. En cas de présence d'un Pic et non d'un Obstacle (block) normal alors on peut lui rajouter la méthode blesser. De même, pour les personnages, il était interessant de mettre le personnage principal de façon séparée des ennemis. Entre les deux, peu de fonctionnalités sont partagées et celles-ci sont consignées dans la classe Perso dont les deux classes PersoPrinc et Ennemis héritent.

Les contraintes du jeu et de l'organisation de classes choisie

Les surcharges d'opérateurs définies dans les classes Perso et Obstacle sont très utiles pour économiser des lignes de code. Néanmoins, l'utilisation de deux surcharges du même opérateurs dans la partie Perso, les paranthèses, avec un nombre différent de paramètres n'est pas bien accepté par le compilateur, il a fallu redéfinir la surcharge à un endroit qui n'était pas forcément utile en temps normal.

Le jeu de Plateformer utilise beaucoup d'objets en même temps. Il faut pouvoir gérer tous les ennemis et tous les obstacles.

Pour cela, les conteneurs tels que le vector permettent de regrouper ces objets et de pouvoir les monitérer plus aisément. Le plus gros problème lors de l'utilisation de ces conteneurs est que si l'on crée des vecteurs de Perso alors, même si l'objet rentré dans la vecteur pouvait être d'une classe héritant de Perso, l'objet était limité à être un Perso et ne pouvait pas retrouver ses fonctiannalités antérieures (comme celle ajoutées dans les classes Ennemis (Chat ou Carotte)).

Gérer ses conteneurs prend beaucoup de temps. Néanmoins, cela est nécessaire car cela permet de créer des objets à la volée et à terme de créer des nouveaux niveaux à chaque fois avec un choix au hasard des différentes objets présentés dans le niveau (solution que j'avais en tête dès le début du Projet).

Les maps sont très utiles sur la partie graphique. Cela m'a permis d'indexer mes différents textures pour automatiser et regrouper un peu de l'affichage des sprite après la mise à jour des éléments graphiques.

Au final, deux grosses contraintes ont pu être remarquées lors de ce projet:

- Un manque de temps: Il y a beaucoup de fonctionnalités que j'aurais aimé intégrer dans mon application. Néanmoins, par manque de temps pour tout réaliser jusqu'au bout j'ai préféré me concentrer sur la façon de gérer les diférents objets du Jeu Plateformer plutôt que sur l'aspect graphique pur. Cela m'a permis de devenir plus à l'aise avec l'utilisation des conteneurs.

- Un manque d'espace: développer sans optimiser son code dans ce genre de programmation peut devenir très compliqué.
  On peut vite arriver à une situation où le code devient confus et où rien ne marche.
  J'ai eu plusieurs versions de ce projet où mon code finissait inévitablement par devenir confus.
  Ce projet a donc permis de trouver des moyens de ne pas se préciter et organiser un peu mieux la manière de coder également.
  J'ai également pu utiliser les template afin de raccourcir le code que je produisais.

L'utilisation d'une librairie graphique provoque des erreurs sur valgrind. Néanmoins, en parallèle, aucun malloc n'a été écrit spécifiquement dans le code. Cela réduit grandement les chances de ne pas libérer la mémoire quand on en a plus besoin.

Processus d'installation

Pour pouvoir lancer le code, il faut avoir préalablement installé la librairie SFML sur son ordinateur. Pour les distributions Linux, il suffit d'utiliser la commande sudo apt-get install libsfml-dev.
Une fois installée, il faut compiler les codes sources.Il faut rajouter -lsfml-graphics -lsfml-window -lsfml-system comme option lors de la compilation.

Ces options ont été déjà intéressés par le sujet.

Toutes les images du jeu sont dans le dossier prévu à cette effet, il ne faut pas y toucher.

Pour compiler, il suffit en réumé de lancer la commande make puis de lancer ./sfml-app pour ouvrir le jeu.

Si l'on veut regarder les tests unitaires alors il faut lancer ./test après le make.

Les sprites des perso ont été trouvées majoritairement sur ces deux sites libres de droit:
www.GameBuildingTools.com,
https://www.gameart2d.com/free-platformer-game-tileset.html

Parties de l'implémentztion dont je suis la plus fière

Comme évoqué précdemment, j'ai bloqué sur la manière de gérer tous ces objets en même temps.

Une fois que je me suis mise à utiliser des template, j'ai vu la nette différence. Le traitement est bien plus facile.

Je suis contente également d'avoir implémenté les surcharges d'opérateurs comme je l'ai fait. Cela rend le code bien plus compact et clair. Les fonction qui analysent le fait qu'il y ait ou non des contacts entre les objets sont plutôt courtes comparé à mes premières versions.