EPFL

Projet de semestre - C++ orienté objet
Dodgeball
COM-112(a)

Rafael RIBER - SCIPER: 296142
Valentin RIAT - SCIPER: 289121
EL-BA2 - Semestre de printemps 2018-2019

1 Architecture logicielle et description de l'implémentation:

1.1 Structuration des données

L'ensemble des joueurs (chacun une instance de la classe `Player`) est stocké dans un `vector` appelé `players`, attribut de la classe `Simulation`. Les balles sont stockées de manière identique dans un `vector` appelé `balls`.

La carte est est stockée comme une instance de la classe `Map`, attribut de la simulation. Au sein de `Map`, le `vector` 2D `obstacleMap` stocke les cases où un obstacle est présent, représentées par '`1`', et les cases libres, représentées par '`0`'.

1.2 Répartition des tâches entre les modules

À chaque mise à jour, on appele la fonction `simulate_one_step()` de `Simulation`. Cette fonction, par des sous-fonctions, détermine la cible de chaque joueur, et assigne l'addresse de la cible à l'attribut `target` des joueurs.

Elle déplace ensuite chaque joueur vers sa cible, et crée ensuite des nouvelles instances de `Ball` si des balles sont tirées, et les fait bouger selon leur trajectoire. Les tests de collision balle-joueur, balle-obstacle et balle-balle sont effectués au sein du module simulation, qui modifie les structures de données en conséquence. Le module simulation s'occupe ainsi du stockage et de l'intéraction des éléments de jeu. Les détails d'implémentation et les attributs de ces éléments de jeu sont traités par les modules correspondants (`Ball`, `Player` et `Map`).

La matrice des distances est calculée pour la première fois à la lecture du fichier, et est recalculée lorsqu'un obstacle est éliminé par une balle.

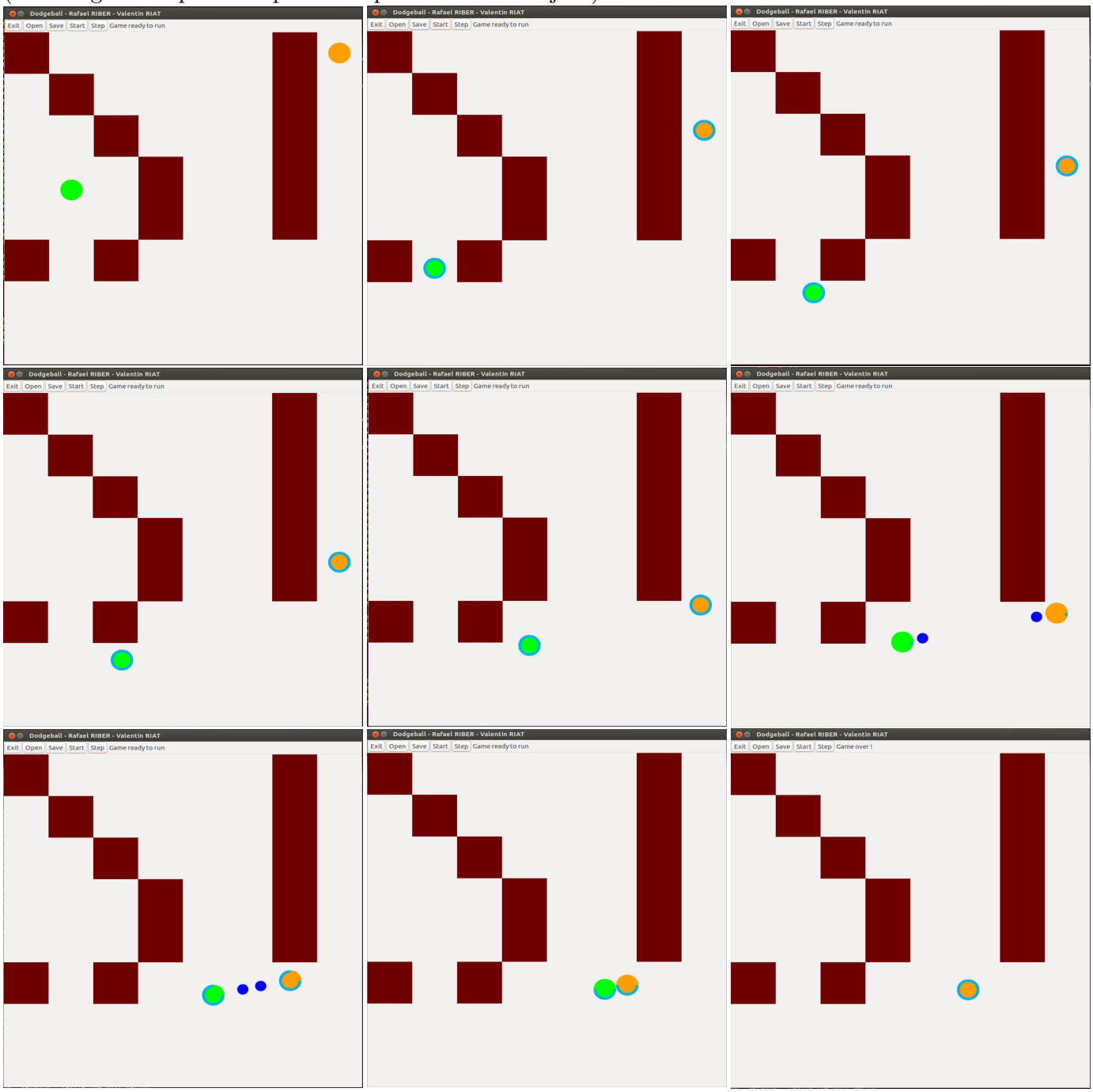
1.3 Coût calcul et coût mémoire

Dans le pire des cas, à savoir quand un obstacle est éliminé et on recalcule la matrice des distances, le coût calcul d'une mise a jour est d'ordre $\mathcal{O}(\text{nbSpot}^3)$.

La matrice des distances est de taille nbSpot^2. Comme on a $\text{nbSpot} = \text{nbCell}^2 - \text{nbObst}$, le coût mémoire d'une mise a jour est donc d'ordre $\mathcal{O}(\text{nbCell}^4)$.

1.4 Exemple de simulation

(Les images sont parfois séparées de plusieurs mises à jour)



2 Méthodologie

Nous avons géré le code à l'aide d'un dépot GitHub privé, en créant des branches pour chaque nouvelle fonctionnalité ajoutée. Cette méthode nous a permis de travailler simultanément sans soucis de versions de fichiers, et de suivre chaque modification du code, en testant chaque nouvelle fonctionnalité sans modifier la branche principale avant que la fonctionnalité soit prête à être implémentée.

Nous avons commencé par le module Simulation, en implémentant l'automate de lecture. La gestion de code par Git nous a permis de travailler chacun sur chaque module, avec toutefois une focalisation de Valentin sur la simulation et de Rafael sur l'implémentation du GUI.

Le bug qui nous a posé le plus de problèmes à été l'automate de lecture qui sautait des lignes, et que

nous avons du réécrire complètement afin d'arriver à une lecture correcte des fichiers de configuration.

3 Conclusion

Si nous devions refaire ce travail, nous aurions probablement ajouté plus de modules, afin de séparer les différentes tâches de simulation, par exemple. L'environnement mis à notre disposition nous a permis d'avoir des réponses rapides à nos questions, mais la machine virtuelle a parfois été difficile à utiliser du fait de sa lenteur.