BESSON Léonard
CORNAT Jacques
LUC Aymeric
RAULOT Adrien

15/01/2014

Dossier de Conception - Version 5

Dans ce dossier de conception, nous allons détailler le projet à sa version 5.

Voici les différentes parties de cette version :

- Ejection (CORNAT Jacques)
- Menu (BESSON Léonard)
- Timer (RAULOT Adrien)
- Modification de Gameplay (LUC Aymeric)

EJECTION

Pour la mise en place de l'éjection d'un personnage lorsqu'il subit un dégât, nous allons commencer par montrer un schéma explicatif, puis un diagramme de classe, puis enfin des explications sur les changements à effectuer.

Tout d'abord, voyons différentes situations :

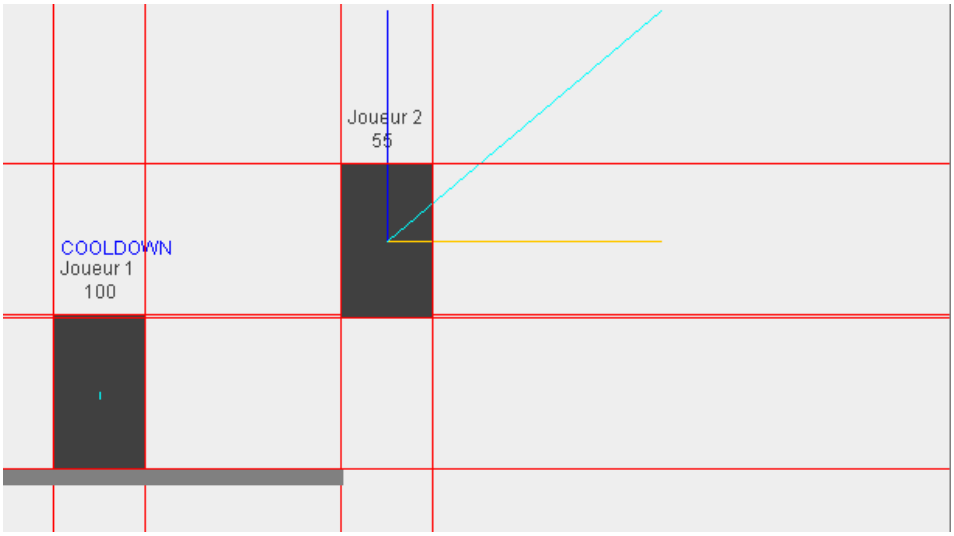
Le joueur 1 se prépare à éjecter le joueur 2 en le frappant :

Les 2 joueurs sont immobiles, et le joueur 1 cast son attaque.

BESSON Léonard
CORNAT Jacques
LUC Aymeric
RAULOT Adrien

15/01/2014

Voici la situation quelques frames plus tard :

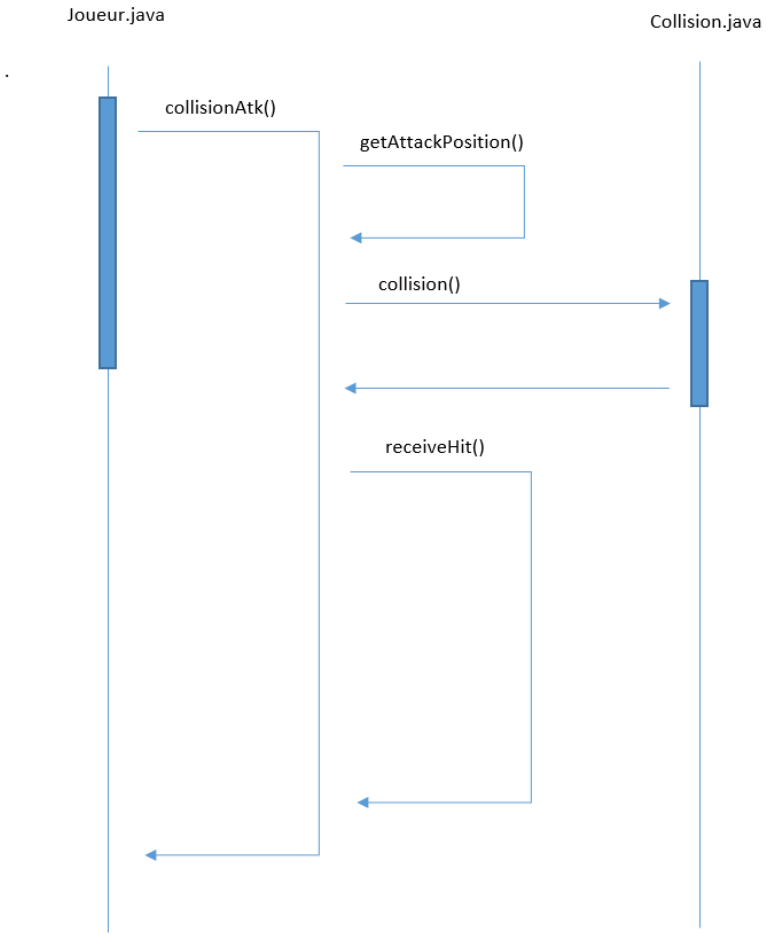


Le joueur 2 se fait éjecter par le coup du joueur 1.

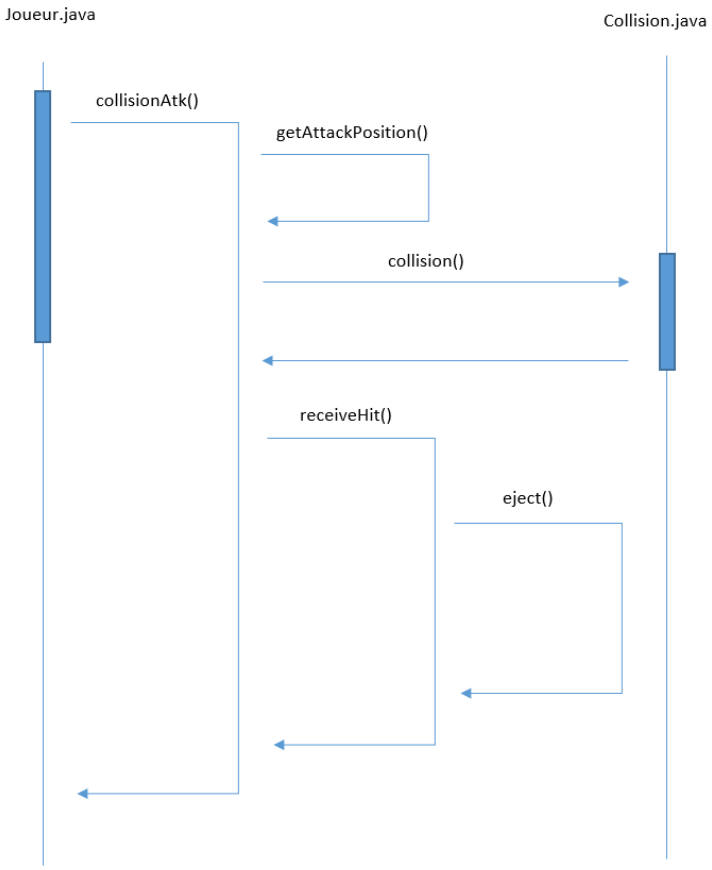
On peut voir que les forces semblent s'appliquer correctement au joueur touché.

Dans le cas où le joueur 2 bouge, la force appliquée est sensiblement la même.

Voici maintenant le diagramme de classe anciennement utilisé :



Puis le nouveau :



Explications :

Nous avons dû modifier les attaques des joueurs pour qu'elles appliquent une force au joueur qui sera touché par cette dernière.
Nous avons donc seulement ajouté 2 attributs à la classe Attaque : powerX et powerY.

Il est important que prendre en compte le sens du joueur pour que l'éjection reste cohérente :

En effet, si une attaque possède un powerX de 200, et un powerY de 100, le joueur touché verra donc sa vitesse actuelle remplacée par ces valeurs, et ce, quelle que soit la direction de l'attaquant, donc il faut aussi penser à inverser le powerX lorsque l'attaquant attaque vers la gauche. S'il attaque vers la droite, il n'y a rien besoin de changer.

Après implémentation, nous avons pu constater que le moteur physique (collision, gravité…) fonctionnait toujours aussi bien, en dehors d'un cas :

Dans l'algorithme de gestion de la vitesse du joueur, si ce dernier désire aller dans une direction, et que son personnage à une vitesse supérieure à sa vitesse cap, sa vitesse sera remise à la limite cap.

(Si le joueur ne touche pas les commandes de direction, alors il perdra tout seul de la vitesse, comme prévu dans le moteur de déplacement.)

Cela a pour effet concret :

Si le personnage est éjecté vers la droite, et que le joueur souhaite aller vers la gauche, alors il perdra de la vitesse normalement, jusqu'à atteindre la vitesse nulle puis repartir dans le sens inverse. Pas de problème ici.
Le souci survient lorsque le joueur souhaite aller vers la droite, lorsqu'il est éjecté vers la droite, (respectivement pour la gauche bien sûr), l'algorithme va donc appliquer la vitesse cap au joueur, ce qui aura pour effet de ralentir brusquement le joueur, cet effet étant indésirable.
Nous avons donc imaginé résoudre ce problème en appliquant 1 état au joueur : NORMAL ou EJECTED. Dès que la fonction ejected() est appelée sur le joueur, alors ce dernier passe en EJECTED, puis lorsque sa vitesse passe en dessous de la vitesse cap, alors il redevient en état NORMAL.
Ainsi, dans l'algorithme de la gestion des mouvements, le joueur souhaitant aller vers la droite, s'il est en état EJECTED, alors la limite ne s'appliquera pas, et seule l'inertie s'appliquera au joueur, pour le ralentir, jusqu'à ce que sa vitesse soit inférieure à celle de la vitesse cap, condition à laquelle le joueur pourra continuer à aller vers la droite.

MENU

Pour concevoir le menu, nous devons utiliser quelque chose qui nous permet de superposer les affichages que nous avons et les interchanger quand on en a besoin. Java propose plusieurs solutions pour ce genre de besoin, Le JLayeredPane et le CardLayout semblent les plus adaptés à notre besoin. Après avoir regardé des exemples d'utilisations, le CardLayout nous a semblé le plus simple à utiliser.

Voici la situation de notre programme pour le moment :

La classe Game possède un attribut JFrame à laquelle nous ajoutons notre VueGraphique.

Quand nous lançons le jeu, la fenêtre affiche directement le jeu grâce à la méthode paint() de la Vue.

Voici comment nous allons procéder pour intégrer le CardLayout dans notre programme :

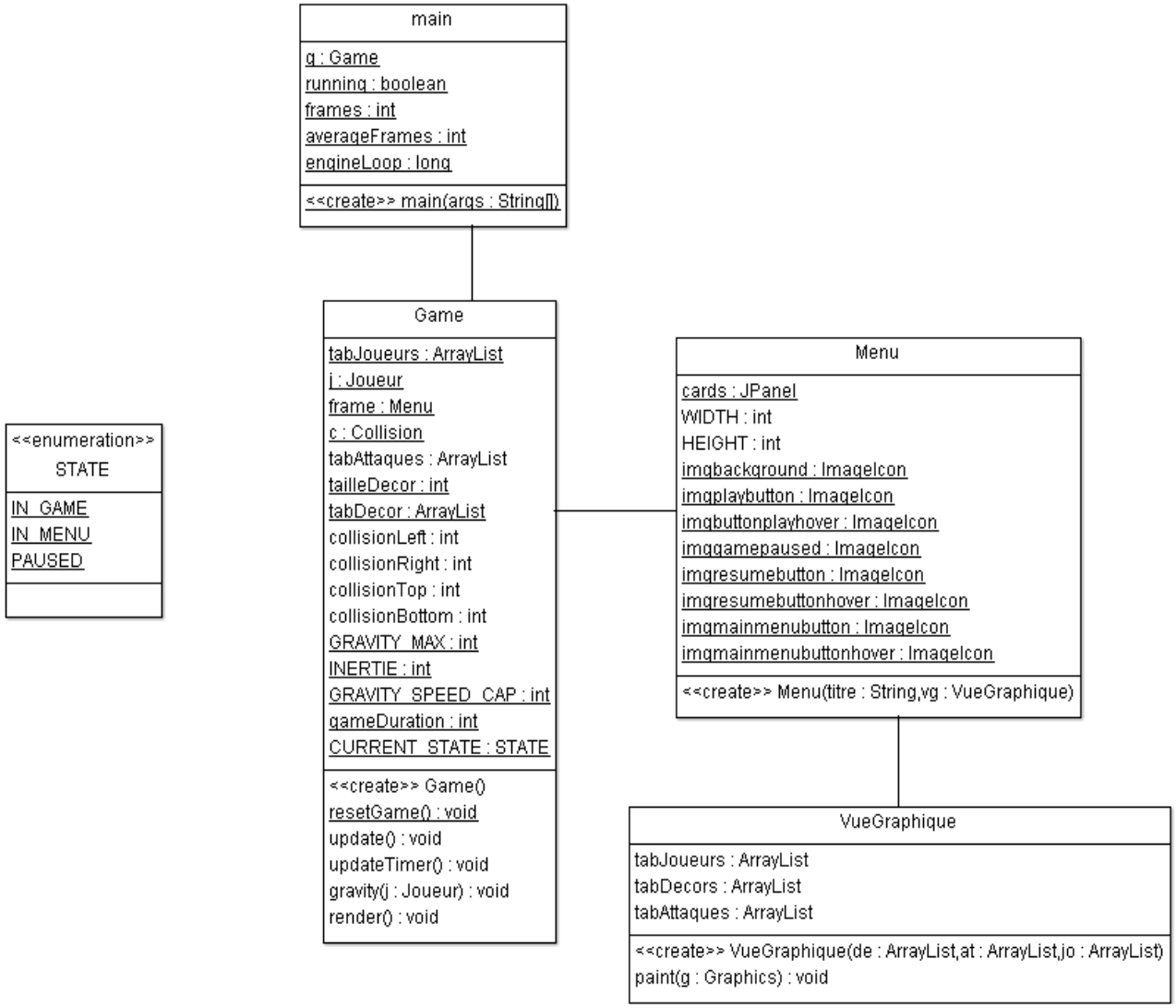
Nous allons créer un nouvelle classe Menu qui héritera de JFrame, cette classe aura un attribut static JPanel muni du CardLayout, nous passerons notre Vue dans le constructeur de cette classe pour l'ajouter au CardLayout. Le constructeur de cette classe construira également les deux autres JPanel que nous allons utiliser, c'est-à-dire un qui sera le menu principal avec 2 boutons (Play et Exit) et un autre qui sera l'écran de pause avec également 2 boutons (Resume et Main Menu).

La classe Game aura juste à créer un objet Menu et lui passer la Vue .

Nous devons cependant gérer un problème : Le jeu ne doit pas « tourner » en fond, c'est-à-dire que le timer ne doit pas s'écouler et les calculs de collisions ne doivent pas s'effectuer. Pour ceci, nous allons créer une énumération d'état du jeu(IN_GAME, IN_MENU, PAUSED).

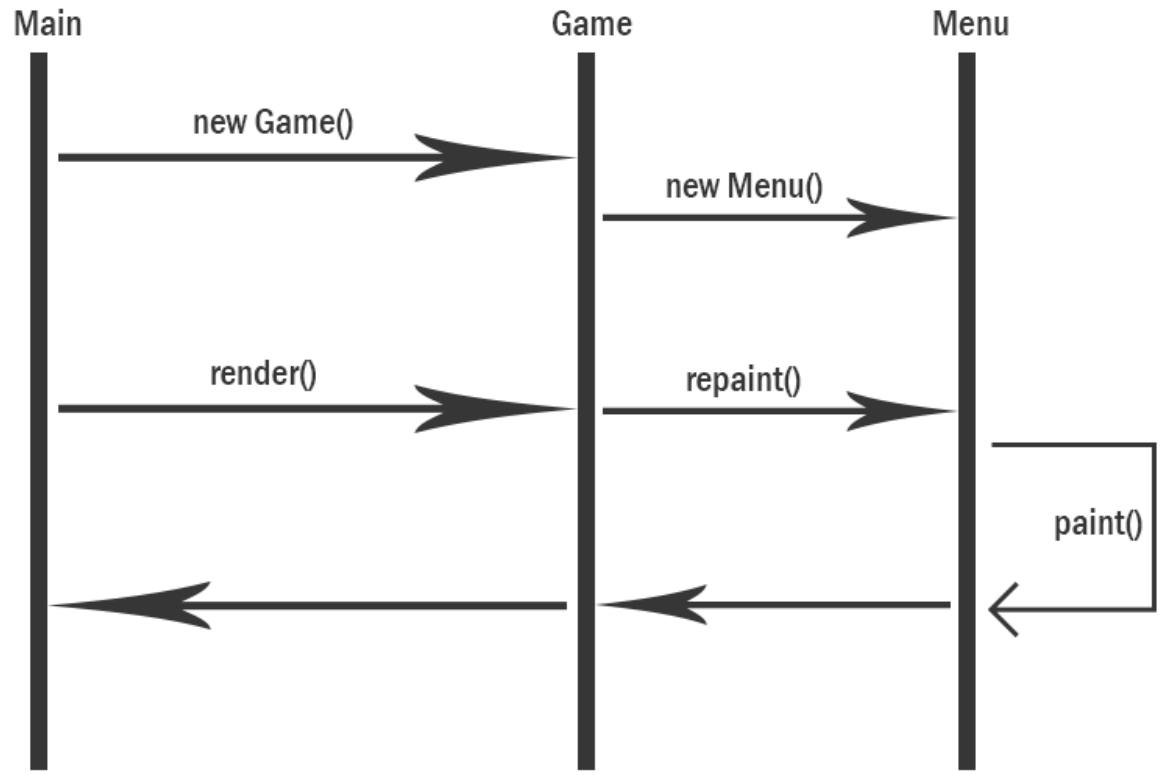
Il suffira donc de changer les états en même que la vue du CardLayout pour que le jeu ne tourne pas en fond. Les calculs de collision ne se feront que lorsque le jeu sera à l'état « IN_GAME », de même pour le timer.

Voici le nouveau diagramme de classe avec les modification et l'ajout de la classe Menu :

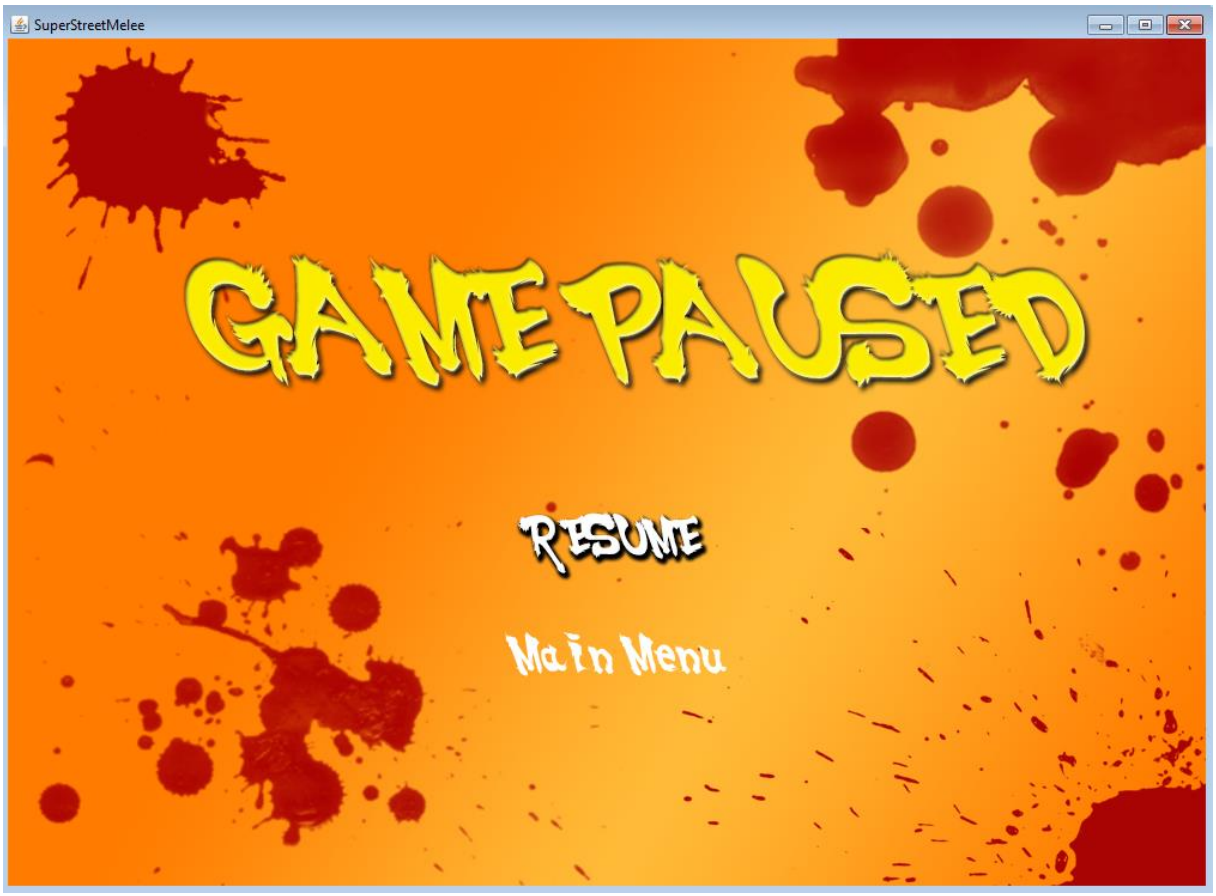


Menu remplace donc l'ancienne JFrame de la classe Game.

Le diagramme de séquence :



Voici les écrans du menu principal et de la pause :



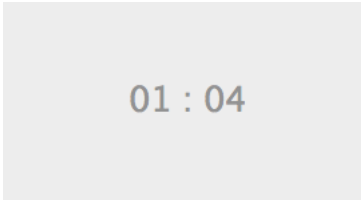
(Les boutons « PLAY » et « Resume » sont survolés avec le curseur sur les images)

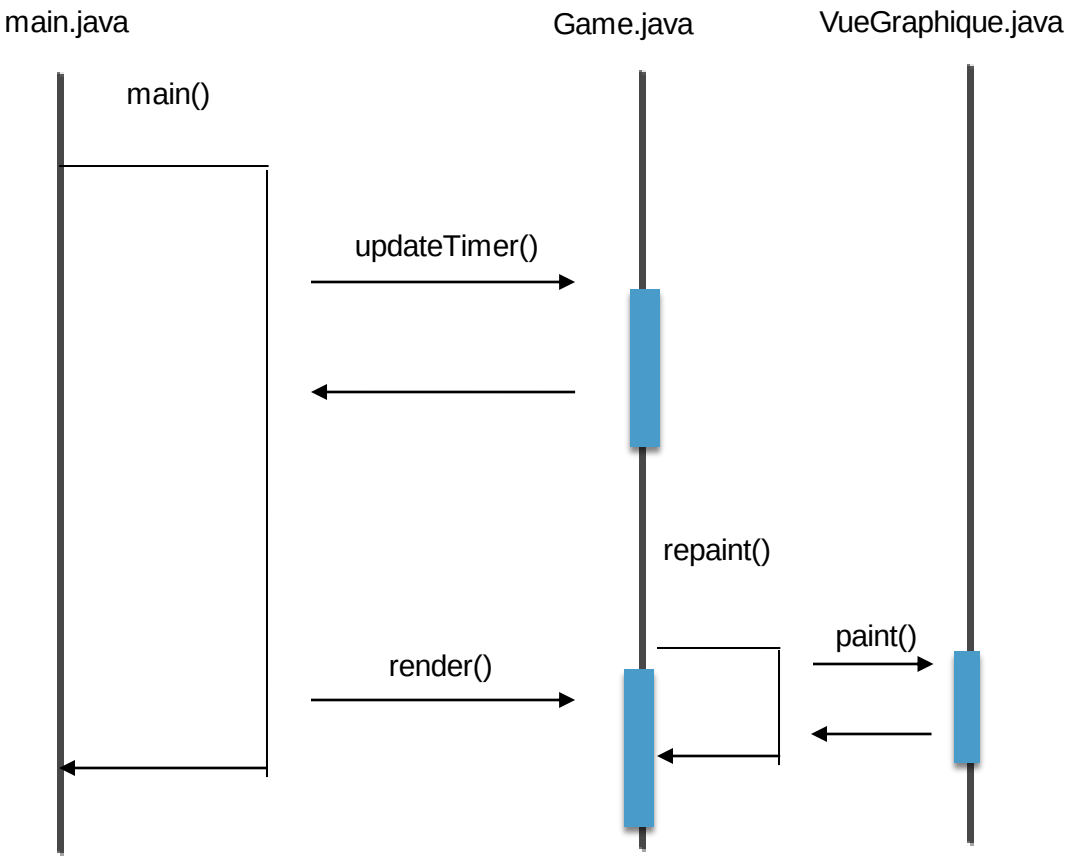
BESSON Léonard
CORNAT Jacques
LUC Aymeric
RAULOT Adrien

15/01/2014

TIMER

Tout d'abord, le timer (en secondes) est initialisé lors de la création d'une partie, et donc concrètement lors de la création d'un objet de la classe Game(). Puis, à chaque seconde passée en jeu, il est mis à jour grâce à la méthode updateTimer() présente dans la classe Game. Celle-ci retire tout simplement 1 au timer, s'il est supérieur à 0. Sinon, le timer est mis à 0, et la partie est terminée. Au niveau de l'affichage du timer en jeu, il se fait bien sûr dans la classe VueGraphique() dans la méthode paint(), en accédant au timer présent dans la classe Game().





BESSON Léonard
CORNAT Jacques
LUC Aymeric
RAULOT Adrien

15/01/2014

MODIFICATION DE GAMEPLAY

Nous allons détailler les modifications qui ont été faites sur les classes Game.java et Joueur.java.

Nous avons inversé le système de Game Play en modifiant la vie des joueurs. Au lieu d'avoir une santé qui décroit, nous avons une santé qui croit de manière infinie. La vie actuelle du joueur se répercutera sur la force des coups reçus.

Nous avons donc modifié la méthode receiveHit() en calculant des puissances proportionnelles à la santé du joueur frappé.

Lors de la réalisation, nous avons dû modifier de manière instinctive la puissance appliquée au joueur à l'aide de coefficients et nous avons pu constater que si la puissance était inférieure à 10 l'effet n'était pas assez saisissant donc nous juste rajouté une condition.

REPARTITION DES TACHES

Voici un schéma des tâches à faire :

	Sprites attaques basiques	Sprites perso	Ajout de limites au terrain	Son minimal	Caméra qui suit les joueurs	Affichage du nombre de victoires ou de vies restantes	Combos joueur	Modifier le moteur pour améliorer le gameplay	Musique de fond	S'accrocher a des plates-formes
Critères de succès	2 attaques basiques avec sprite	Simple sprite non animé	Un personnage qui passe hors des limites du terrain se voit ramené avec toute sa vie sur le terrain	1 effet quand fait une attaque, 1 quand touché	Joueurs toujours à l'écran, Effet de zoom arrière limité	Nombre de vies restantes	1 attaque spéciale avec un combo particulier (exemple, droite+A appuyé pendant 2 secondes)	Rebondir sur les murs		
Importance	100	100	80	70	60	50	40	30	20	10
Difficulté (en jours	5	5	2	5	20	1	5	3	2	4
Phases supplémentaires	Sprites perso animés									

Pour les 2 prochaines, les différentes tâches seront réparties comme ceci :

RAULOT Adrien & CORNAT Jacques : Sprites perso + Sprites attaques basiques

BESSON Léonard : Ajout de limites au terrain

LUC Aymeric : Son minimal