Université Pierre et Marie Curie

Rapport Projet CPS
Street Fighter

Brit Mohamed et Ghersa Sofiane

Table des matières

I. Introduction

Street Fighter est une série de jeu vidéo de combat développée par Capcom, et dont le premier épisode est publié en 1987. Ce jeu de combat consiste simplement a faire combattre deux ou plusieurs combattant, Pour ce faire nous avons du tout au long de ce projet prendre en compte plusieurs contrainte comme a hitbox des joueurs ou encore les techniques et d'autres aspects.

II. Manuel d'utilisation de l'application

1 *Installation, compilation et exécution du projet*

Compile et exécuté la classe principal qui est : Login,java

2 Utilisation de l'application

*Compatible sur tout les systèmes d'exploitation Notre application est développer avec le langage JAVA et l'interface graphique avec JAVA*Swing .

III. *Architecture de l'application et choix effectués*

1. On a choisi de fixée les position des deux joueur, ça veux dire que le joueur de gauche reste toujours a gauche et vise ver ça .

2. le pointe de référence des HitBox et Engine est celle qui est tout en bas a gauche.

3. On a gardez que :

- Engine : qui présente la zone de combat.

- Character : qui présente le personnage.

- HitBox : qui présente le cadre qui contient le personnage.

- FightChar : c'est une extension Character qui présente le personnage avec quelque méthode en plus par rapporte au Character (traitement des techniques).

- Technique : qui présente les technique qui sont gérée par un personnage (On a pas crée les classe des technique on va les crée comme instance avant d'appelée la méthode qui va exécuté celle technique).

4. On a utiliser des threads pour les compteur de temps et quelque autre usage pour ne pas empêchée les gif de s'affiche .

5. les deux personnage son bloquée juste lorsque un joueur met un coup de point ou coup de pied

6. pour prendre compte au commande des deux joueur merci de mettre le curseur le la zone de texte qui est en haut a gauche (On a un petit problème pour Keypressed sur les Jpanel on a pas au assez de temps pour le résoudre).

IV. Fonctionnalité et caractéristiques de l'application

1. *Services et contrat de l'application*

HitboxService :

Cette interface représente notre la hitbox d'un personnage. **Pour la hitbox, nous avons considéré notre hitbox directement comme un rectangle.**

Notre hitbox contient les cardinalités (x et y) de son point de référence, mais aussi:

Un observateur belongsTo(int x,int y) qui permet de vérifier si un pixel fait bien parti de la hitbox.

Un observateur collidesWith(HitboxService hb) qui permet de tester si deux hitbox sont en collisions.

- Un observateur equalsTo(HitboxService hb) qui permet de tester l'égalité avec une autre hitbox passé en paramètre.

En plus de ses caractéristiques basique de notre hitbox, nous avons ajouté les observateurs pour gérer la taille (largeur et hauteur) de la hitbox afin qu'elle deviennent un rectangle.

Nous avons des invariant de base :

- On peut minimiser collideswith avec la définition de belongsTo, en effet si un point est commun aux deux hitbox on a une collision.
- On peut aussi minimiser equalsTo, en effet faire un equalsTo de 2 hitbox revient à vérifier que tous les points sont les mêmes pour les 2 ou de vérifier que les 4 coins de la hitbox sont

les mêmes que ceux de la deuxième. **C'est cette dernière solution que l'on a utiliser pour le contrat et la spécification.**

-

Nous avons rajouté un invariant pour RectangleHitbox :

EngineService :

Cette interface représente le moteur de notre jeu.

On a des observateurs qui décrive notre Engine notamment gameOver qui permet savoir si le jeu est finis ou encore la taille (largeur et hauteur) de l'arène .

Pour les invariant, on a un invariant logique qui permet de minimiser gameOver, en effet cela revient à dire qu'un des deux personnage n'a plus de vie.

L'opérateur Init permet d'initialiser les observateurs et possède comme précondition : la largeur et la hauteur et l'espace entre les deux joueurs doivent tous être supérieur à 0 .

L'opérateur Init possède aussi des postconditions : la hauteur doit correspondre au h passer en paramètre, nous avons respectivement la même vérification pour la largeur, les deux personnages passé en paramètre .

Nous avons aussi d'autre postCondition qui positionne les personnages au centre de l'arène et avec les bonne face et le bon espacement entre eux.

L'opérateur Step permet de faire bouger les deux personnages le premier personnage vers la direction indiquer par une commande c1 (respectivement pour le deuxième vers une commande c2)

Pour que cet opérateur puisse avoir lieu il faut que la partie ne soit pas terminé (gameOver) et on vérifie a la fin avec des postCondition que les personnage on bien bougé .

Dans le Contrat on capte les erreurs possible due au postconditions et précondition spécifier contractuellement dans la spécification.

CharacterService :

Pour ce service, nous avons des observateurs pour la position du personnage, sa hitbox, sa vie, sa vitesse etc .. .

Pour les invariants la position du x doit être compris entre 0 et la taille de l'arène (la largeur) (respectivement pour Y (la hauteur)). On peut minimiser la fonction isDead si le personnage arrive a 0 ou moins.

Init : comme pour les postconditions du init de engine on test si les éléments à la fin correspondent bien aux valeurs passées en paramètres.

Postcondition : getLife doit avoir pour valeur la vie passe en paramètres.

Pareil pour les autres.

- moveUp :

  • s'il y a collision on ne bouge pas.

  • si la vitesse (speed) est inférieur à getEngine().getHeight() – (positionY + getCharBox().getHeight()) et qu'il n'y a pas de collision ça veut dire qu'on peut bouger la hitbox aux coordonnées x précédent , y+speed .

  • si speed est supérieur ou égal à getEngine().getHeight() – (positionY + getCharBox().getHeight()) et qu'il n'y a pas de collision ça veut dire que la hitbox sort de l'arène et donc comme il ne doit pas sortir de l'arène on le remet aux coordonnées (ancienX, positionY+speed).

  • la face ne change pas la vie du personnage n'est pas impacté et la position du x non plus.

- moveDown :

  • s'il y a collision on ne bouge pas.

  • si la position de Y est inférieur a speed et qu'il n'y a pas de collision ça veut dire que la hitbox sort de l'arène et donc comme il ne doit pas sortir de l'arène on le remet aux coordonnées (ancienX, 0).

  • si speed est inférieur à la position de Y et qu'il n'y a pas de collision alors on peut déplacer le personnage grâce à speed (ancienX, positionY+speed).

  • la face ne change pas la vie du personnage n'est pas impacté et la position du y non plus.

- ***moveLeft :***

  • s'il y a collision on ne bouge pas.

• si la position de X est inférieur a speed et qu'il n'y a pas de collision ça veut dire que la hitbox sort de l'arène et donc comme il ne doit pas sortir de l'arène on le remet aux coordonnées (0, 0).

• si speed est inférieur à la position de X et qu'il n'y a pas de collision alors on peut déplacer le personnage grâce à speed a x-speed, y précédent .

• la face ne change pas la vie du personnage n'est pas impacté et la position du y non plus.

- moveRight :

  • s'il y a collision on ne bouge pas.

  • si la vitesse (speed) est inférieur à getEngine().getWidth() – (positionX + getCharBox().getWidth()) et qu'il n'y a pas de collision ça veut dire qu'on peut bouger la hitbox aux coordonnées x+speed, y précédent .

  • si speed est supérieur ou égal à getEngine().getWidth() – (positionX + getCharBox().getWidth())et qu'il n'y a pas de collision ça veut dire que la hitbox sort de l'arène et donc comme il ne doit pas sortir de l'arène on le remet aux coordonnées (getEngine().getWidth() - getCharBox().getWidth(), 0).

  • la face ne change pas la vie du personnage n'est pas impacté et la position du y non plus.

Pour les autres mouvements, nous avons aussi fait les spécification et contrat qui n'est qu'un mixe des spécification précédentes mais ou il faut tester la face du personnage et s'il ont bien changé de position.

Pour L'opérateur **switchSide** on à deux postconditions :

- le fait de changer de face veut dire ne modifie pas la position du X
- Logiquement si on effectue un switchSide la face n'est plus la même.

FightCharService :

Ce service est un raffinement de CharacterService il permet de gérer la partie technique des personnages.

Dans ce service on a les éléments de base ainsi que les preConditions de base.

Mais on a aussi deux postConditions de plus pour l'opérateur startTech.

En effet, en postCondition on vérifie que la vie de l'autre personnage est qui se prends les coups de la technique a son compteur de vie qui a baissé que s'il n'est pas en position de blocage.

La deuxième postcondition est qu'on teste que la vie du personnage n'a pas changé s'il est en position de blocage de technique .

2. *Fonctionnalités de l'application*

- *Dans cette section on va explique l'implémentation des classes et les méthode qui sont importent.*

  - ✔ *Login.java : classe principal dans la quelle on a mets le code de l'interface graphique*

  - ✔ *Audio.java : contient la méthode de lecteur de son.*

  - ✔ *Character.java : représente le personnage avec tout ces caractéristique (point de reference , life , speed , isfaceright , isdead , hitbox).*

  On associé a chaque personnage une hitbox.

  - ✔ *Engine.java : représente le terrain de combat.*

  On a choisi d'associé a notre Engine deux hitbox a la quelle il appartiens chaque personnage parce que dans le sujet y a pas mal d'erreur on trouve que Engine elle prend deux personnage en paramètre en même temps un personnage il prend un Engine.

  - ✔ *FightChar : c'est une extension de Character a la quelle ajoute les attribut pour savoir si notre personnage il est pas bloque , si il est en train de faire une technique , une méthode pour exécuté une technique.*

    - *StartTech : avant de faire le traitement il faux que les deux personnage ne sont pas bloque et aussi il sont toujours en vie.*

  - ✔ *Hitbox.java : elle représente la zone qui contient notre personnage.*

    - *CollidesWith : pour teste si deux hitbox en collision on regarde les pointe de référence des deux il faut en moins un X de l'une soit entre le X et X+Width de l'autre.*

    - *EqualsTo : pour que deux hitbox soit égaux il faut que le X , Width et Height des deux soit égaux.*

    - *MoveLeft : pour déplacé une hitbox vers la gauche c'est ça position de X moins ça vitesse avant on va voire si il est a cote d'autre jouer on le mets juste a cote de ce dernière sinon si il est a la fine de la zone de combat on le mets aussi a cote de fine zone sinon il peux ce déplacer normalement , et ça sera le même traitement pour les autre commande de déplacement.*

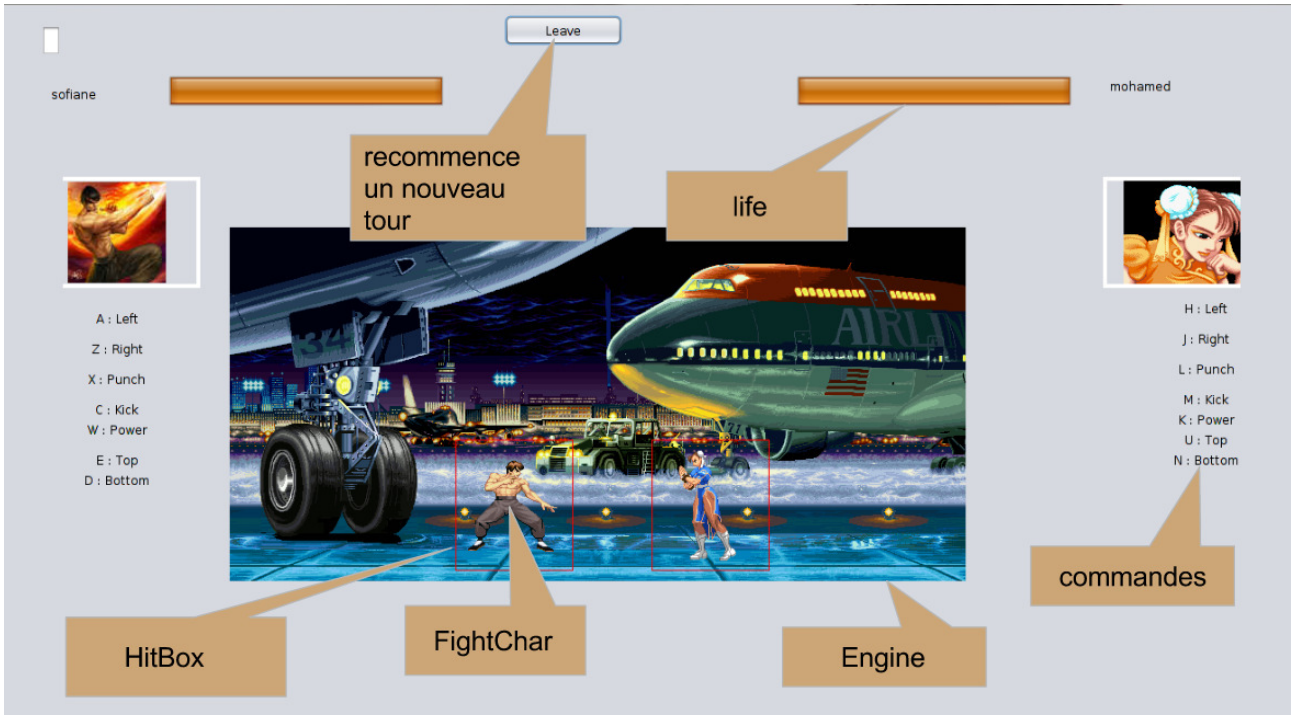
✔ *Technique.java : elle représente les caractéristique de chaque technique.*

La on a choisi que le temps de blocage soit le même pour chaque phase est pour les deux personnage.

◆ *Capture d'écran de notre interface graphiques avec quelque commentaires.*

Fenêtre de connexion :

Fenêtre de jeux :



Conclusion :

Ce projet nous a permis d'avoir un aperçu de la programmation par contrat

et du pattern Require-Provide qui se sont avérés très pratiques pour le débuggage

et notamment pour la mise en place d'une interface utilisateur où il a seulement

fallut lier les différentes fonctions des services aux éléments graphiques. Malgré

tout, une grosse faiblesse de ce modèle se trouve dans l'ajout de nouvelles fonctionnalités qui nécessite de modifier dans tous les éléments du pattern ce qui peut rendre la tache fastidieuse et longue.