

Institut TELECOM  
TELECOM SudParis  
Année scolaire 2007/2008



Projet Informatique 1<sup>ère</sup> Année  
CSC 3502

## **Développement d'un jeu de combat 2D type arcade**

**Valérie BRANCHE**  
**Jacqueline TAN**  
**Tian Sheng TAN**  
**Muqiu ZHENG**

Enseignant responsable : **Sébastien LERICHE** du département Informatique

# Sommaire

<b>1. Introduction</b> .....	<b>3</b>
<b>2. Cahier des charges</b> .....	<b>3</b>
<b>2.1 Fonctions à remplir par le logiciel</b> .....	<b>3</b>
<b>2.2 Performances requises</b> .....	<b>3</b>
<b>2.3 Contraintes de réalisation</b> .....	<b>4</b>
<b>2.4 Exigences de qualités requises</b> .....	<b>4</b>
<b>2.5 Interfaces du logiciel avec son environnement humain, matériel, logiciel</b> .....	<b>5</b>
<b>3. Développement</b> .....	<b>5</b>
<b>3.1 Analyse du problème et spécification fonctionnelle</b> .....	<b>5</b>
<b>3.2 Conception préliminaire</b> .....	<b>5</b>
3.2.1 Définition des modules.....	5
3.2.2 Définition des structures de données .....	6
3.2.3 Plan des tests d'intégrations.....	7
<b>3.3 Conception détaillée</b> .....	<b>7</b>
3.3.1 Règles d'animation : diagrammes d'état/transition des mouvements animés.....	7
3.3.1.1 Coups de poings & coups de pieds .....	7
3.3.1.2 Sauts.....	7
3.3.1.3 Marche .....	8
3.3.1.4 Accroupissement.....	8
3.3.1.5 Blocage .....	8
3.3.2 Règles de mouvement.....	8
3.3.2.1 Les sauts.....	8
3.3.2.2 La gestion des collisions .....	9
3.3.3 Règles des coups portés .....	9
3.3.3.1 Zones de coups possibles .....	9
3.3.3.2 Pertes des points de vie .....	10
<b>3.4 Codage</b> .....	<b>10</b>
3.4.1 Description de la SDL .....	10
3.4.2 Commentaires des codes .....	12
3.4.4.1 Lancement du jeu : accueil .....	12
3.4.4.2 Algorithme principal du jeu .....	12
3.4.4.3 Mise à jour de l'écran après tous les calculs d'états et de positions.....	14
<b>3.5 Tests</b> .....	<b>14</b>
<b>3.6 Intégration</b> .....	<b>16</b>
<b>4. Conclusion</b> .....	<b>16</b>
<b>5. Bibliographie</b> .....	<b>17</b>
<b>6. Annexes</b> .....	<b>17</b>
<b>6.1 Annexes : Gestion de projet</b> .....	<b>17</b>
6.1.1 Plan de charge prévisionnel et suivi d'activités .....	17
6.1.2 Planning Prévisionnel.....	18
<b>6.2 Annexes : Codes sources</b> .....	<b>18</b>
<b>6.3 Annexes : Site web</b> .....	<b>18</b>
6.3.1 Contenu du site web .....	18
6.3.2 Document type de nos comptes-rendus .....	19

# 1. Introduction

Nous sommes un groupe de quatre étudiants en formation ingénieur à TELECOM SudParis. Dans le cadre de notre première année, nous avons un projet informatique à réaliser. Nous avons le choix entre cinquante sujets et au final nous nous sommes mis d'accord pour un projet portant sur les jeux vidéos et l'animation 2D. En effet, certains d'entre nous aiment jouer aux jeux vidéo et de manière générale, il était intéressant de voir ce qui se « cache » derrière l'aspect purement ludique et de manipuler une interface graphique. D'un point de vue technique, ce projet nous permet de découvrir une nouvelle bibliothèque, la Simple DirectMedia Layer (SDL), tout en nous appuyant sur ce que nous avons appris en programmation.

## 2. Cahier des charges

### 2.1 Fonctions à remplir par le logiciel

Le projet consiste à concevoir un jeu de combat 2D de type arcade, inspiré de la série Street Fighter. Le principe consiste à diriger un personnage à l'écran, capable de réagir par des mouvements différents suivant les touches clavier utilisées. Un second personnage symétrique sert d'adversaire. Suivant les positions des personnages, les mouvements peuvent être des coups, qui font perdre des points de vie à l'adversaire. Le premier à ne plus avoir de points de vie a perdu.

### 2.2 Performances requises

Le bon fonctionnement de ce logiciel sur un ordinateur requiert la plate-forme minimale suivante :

#### **Affichage**

- Résolution min. de l'écran **640x480**
- Couleurs d'affichage prises en charge **32 bits**

#### **Audio**

- Type de sortie audio **Carte son**

#### **Caractéristiques techniques**

- Mode de saisie **Clavier**

#### **Disque dur**

- Espace **10 Mo**

#### **Mémoire**

- RAM **256 Mo**

#### **Processeur**

- Vitesse du processeur	<b>1.86 GHz</b>
<b>Système d'exploitation</b>	
- Plate-forme	<b>PC</b>
- Système d'exploitation	<b>Tous</b>

## 2.3 Contraintes de réalisation

- ❖ Le jeu doit être portable, c'est-à-dire pouvoir être exploité sous différents systèmes tels que Linux et Windows mais nous n'avons pas réussi à répondre à ce critère.
- ❖ Le codage des programmes doit être fait en langage C.
- ❖ Pour simplifier le travail et nous consacrer à des aspects de programmation, nous utiliserons d'une part la librairie SDL (gestion des images, des entrées/sorties...) et d'autre part les graphiques disponibles sur différents sites internet (cf. références ci-dessous). Toujours dans cet esprit, nous concentrerons le travail sur un unique personnage.

Ressources graphiques à utiliser : <http://spriters-resource.com/capcom/fighting/street>

- ❖ L'ensemble du code sera sous licence [GNU/GPL](#).

## 2.4 Exigences de qualités requises

- ❖ Exigence dans la construction de la documentation : rédaction d'un plan de charges prévisionnel, d'un planning prévisionnel.

Le plan de charge contient un tableau (éventuellement réalisé sous un tableur) avec la liste des tâches et, pour chaque tâche, la charge estimée et les personnes affectées (avec une indication du volume de travail estimé pour chaque personne pour cette tâche).

Le planning contient la liste des périodes prévisionnelles (date, heure de début et durée) affectées au projet et pour chaque période la liste des tâches envisagées sur le projet ainsi que les participants à ces tâches.

- ❖ Exigence dans la réalisation du logiciel : recours au cycle en V

- ❖ Exigences dans le suivi du projet :

Au fur et à mesure de l'avancement du projet, il est conseillé à chaque participant au projet de noter (sur son agenda) quand il commence une tâche et quand il la finit. Cela lui permet de compléter son suivi d'activités en notant la tâche et la durée passée sur celle-ci.

Des réunions avec l'enseignant encadrant sont organisées aussi souvent que cela est nécessaire pour le bon avancement du projet. (une réunion par semaine par exemple)

Le site web est un moyen de suivi du travail du projet pour l'ensemble des participants. Il est géré par un des étudiants du groupe projet mais son contenu est sous la responsabilité de l'ensemble du groupe.

## 2.5 Interfaces du logiciel avec son environnement humain, matériel, logiciel

- ❖ Interface logiciel / homme : écran d'ordinateur, clavier.

## 3. Développement

### 3.1 Analyse du problème et spécification fonctionnelle

Notre logiciel met à disposition un jeu de combat faisant interagir deux joueurs. Chacun des joueurs contrôle un personnage par l'intermédiaire d'un clavier (entrée) et d'un écran pour l'affichage graphique (sortie).

Les utilisateurs doivent avoir la possibilité de jouer une partie respectant des règles d'animations, de mouvements et d'arbitrage définies dans la partie conception détaillée.

### 3.2 Conception préliminaire

#### 3.2.1 Définition des modules

- ❖ Modules pour l'affichage

`apply_surface.h` : pour pouvoir appliquer des surfaces  
`apply_surface.c, main.c, restart.c, welcome.c`

`constant.h` : définition des variables globales uniquement utilisées par le main  
`main.c`

`define.h` : définition des constantes  
`displayScreen.c, inverse.c, keycontrol.c, main.c, result.c`

`extern.h` : définition des variables globales utilisées par les autres fichiers que le main  
`displayScreen.c, keycontrol.c, restart.c, stateChange.c, welcome.c`

`load_image.h` : pour pouvoir charger les images  
`load_image.c, displayScreen.c, main.c, restart.c, welcome.c`

`restart.h` : pour pouvoir recommencer une partie de combat  
`restart.c, keycontrol.c`

`welcome.h` : pour la page d'accueil du jeu  
`welcome.c, main.c`

- ❖ Module pour la gestion des personnages

`inverse.h` : pour inverser notre personnage et obtenir un personnage adversaire  
`inverse.c, displayScreen.c`

- ❖ Module pour les structures de données

`datastructure.h` : définition de nos structures de données  
`keycontrol.c, lifeBar.c, main.c`

❖ Module pour les règles d'animation

`keycontrol.h` : pour voir l'état du clavier et donc vérifier l'état et l'action des personnages  
`keycontrol.c, main.c`

`displayScreen.h` : étude des différents cas possibles afin de gérer entre autre l'animation  
`displayScreen.c, keycontrol.c`

`stateChange.h` : pour le calcul de l'état et de la position du personnage après une action  
`stateChange.c, displayScreen.c, keycontrol.c`

`result.h` : pour afficher le résultat après les calculs des états et positions  
`result.c, keycontrol.c`

❖ Module pour les règles de mouvement

`displayScreen.h` : étude des différents cas afin de gérer entre autres les mouvements  
`displayScreen.c, keycontrol.c`

`keycontrol.h` : pour voir l'état du clavier et donc vérifier l'état et l'action des personnages  
`keycontrol.c`

❖ Module pour l'arbitrage

`lifeBar.h` : pour l'arbitrage notamment avec la gestion de barre de vie  
`lifeBar.c, displayScreen.c`

### 3.2.2 Définition des structures de données

```
typedef struct
{
    int playerNumber;           //player's number
    char name[10];             //player's name
    float life;                 //player's life
    int x;                      //player's x-coordinate
    int y;                      //players's y-coorindate
    int sprite_counter;        //player's sprite counter
    int jumpingCounter;        //player's jump counter
    int previousJumpCounter;   //player's previous jump counter
    Booleen resumeBackToJump; //whether the player returned to the
                               //jumping state after executing a punch/kick
    int isAttacked;           //whether the player is being attacked or not
    SDL_Surface *surface;
    states state;
}Person;
```

### 3.2.3 Plan des tests d'intégrations

Le logiciel est réalisé à partir d'un personnage. Les tests consistent à vérifier que le jeu fonctionne en opposant deux personnages :

- ❖ Test sur la non superposition des personnages
- ❖ Test sur la réaction d'un personnage suite à un coup
- ❖ Tests sur la synchronicité coups/arbitrage
- ❖ Test sur la fin d'une partie de jeu

## 3.3 Conception détaillée

### 3.3.1 Règles d'animation : diagrammes d'état/transition des mouvements animés

#### 3.3.1.1 Coups de poings & coups de pieds

3 niveaux de coups de poing : haut (= high punch), milieu (= mid punch), bas (= low punch)

3 niveaux de coups de pieds : haut (= high kick), milieu (= mid kick), bas (= low kick)

#### Propriétés :

- ❖ Tout mouvement ou enchaînement de mouvements doit être réalisé intégralement avant de pouvoir en exécuter un nouveau, donc impossibilité d'interrompre un début de mouvement.
- ❖ Après qu'une combinaison soit intégralement exécutée, le personnage retourne à son état initial de repos. Ex. : accroupissement → saut → repos
- ❖ A la base nous voulions que lorsqu'une touche reste enfoncée, le personnage n'exécute qu'une seule action. Mais le fait que SDL\_enableKeyRepeat ne marche pas sous Windows fait qu'en pratique, quand on laisse une touche enfoncée, le personnage peut continuer indéfiniment à exécuter l'action.

#### 3.3.1.2 Sauts

2 types de saut : saut vertical, saut en avant.

#### Propriétés :

- ❖ Pendant un saut, une manipulation des touches directionnelles n'aura aucune influence sur le mouvement du personnage. Seuls les coups de poings et de pieds seront exécutés pendant un saut.

### 3.3.1.3 Marche

Les coups de poings et coups de pieds peuvent interrompre la marche d'un personnage qui se met alors à exécuter l'attaque demandée. Si une touche directionnelle est enfoncée au moment où l'attaque se termine, le personnage va marcher dans la direction demandée, en d'autres termes, le personnage ne retourne pas à l'état de repos après l'attaque.

### 3.3.1.4 Accroupissement

Pendant un accroupissement :

- ❖ Possibilité de commencer un mouvement d'attaque.
- ❖ Possibilité de rester dans cette position autant qu'on veut en laissant les touches enfoncées.
- ❖ Possibilité de commencer un saut.
- ❖ Impossibilité de bouger d'avant en arrière.

### 3.3.1.5 Blocage

La touche de « blocage » permet au personnage de se défendre et de ne recevoir aucun dégât pendant que l'adversaire l'attaque. Cette touche ne s'active que lorsque les deux personnages sont assez proches et que l'adversaire tente d'infliger un coup à notre personnage.

#### Propriété :

- ❖ Quelque soit la position, debout ou accroupie, une fois en mode « blocage », le personnage est invulnérable à toute attaque.

## 3.3.2 Règles de mouvement

### 3.3.2.1 Les sauts

Pendant plusieurs semaines, nous avons tenté d'implémenter les sauts en utilisant la notion de pesanteur afin d'apporter un aspect réaliste aux sauts. Voyant que nous n'arrivions pas à utiliser la pesanteur, nous avons opté pour un saut purement linéaire.

Pour faire sauter notre personnage, nous définirons sa position par son abscisse  $x$  et son ordonnée  $y$ .

Pour la première moitié des images qui composent l'ensemble du mouvement, on diminue  $y$  de 30 avant chaque blittage d'image. Ceci correspond à la montée du saut. Et pour la seconde moitié des images, on augmente  $y$  de 30 avant chaque blittage d'image, ce qui correspond cette fois-ci à la retombée du saut.

Si en même temps, on veut déplacer latéralement notre personnage, on fera en sorte que la combinaison des touches de saut et de direction gauche/droite engendre un déplacement dans les airs. Ainsi, en plus de ce qui a été décrit dans le paragraphe précédent, x sera augmenté de 10 si on veut aller à droite et diminué de 10 si on veut aller à gauche.

C'est une méthode de coder les sauts sans avoir à prendre en compte l'effet de pesanteur tout en obtenant au final un résultat visuel convaincant.

### 3.3.2.2 La gestion des collisions

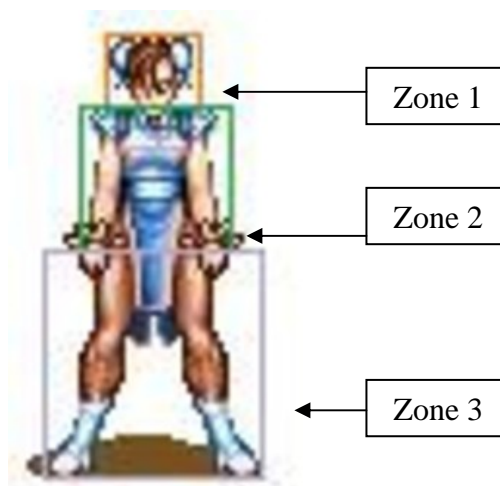
Dans cette partie, nous avons également envisagé le problème des rebonds, toujours dans l'idée d'apporter un aspect visuel agréable. En effet, il s'agit ici de gérer la projection au sol d'un personnage lorsqu'il reçoit un coup sans s'être protégé.

Finalement, nous nous sommes contentés des différents sprites à notre disposition, représentant la situation où un personnage est projeté au sol.

### 3.3.3 Règles des coups portés

#### 3.3.3.1 Zones de coups possibles

- ❖ Zone de la tête et du cou (Zone 1)
- ❖ Zone allant des épaules jusqu'au bas ventre (Zone 2)
- ❖ Zone des jambes (Zone 3)



Afin de gérer la réaction d'un personnage face aux différents coups qu'il peut recevoir, nous étions partis sur l'idée de définir géométriquement trois surfaces liées à ces trois zones de coups. Ensuite, il s'agissait de vérifier avec des inégalités que le personnage adverse donnait un coup dans telle ou telle zone. Par exemple, si le pied de l'adversaire se trouve dans la zone 1 de la tête, cela signifie que notre personnage a reçu un coup au visage et doit réagir de la bonne façon.

Finalement, ce concept s'est avéré trop difficile à mettre en place. Nous avons uniquement défini des états pour les personnages pour savoir s'ils sont au repos, en train de donner un coup de pied à la tête, ou un coup de poing ou niveau de ventre... En fonction de ces états, nous avons définis les réactions adéquates. Concrètement, c'est le coup donné par l'adversaire qui détermine la perte de vie infligée au personnage, quelque soit la partie de son corps ayant reçu le coup.

### 3.3.3.2 Pertes des points de vie

Représentation de la vie du personnage par une barre horizontale colorée.

Barre pleine = 100%

- ❖ Coup de poing supérieur : 9 %
- ❖ Coup de poing central : 6 %
- ❖ Coup de poing inférieur : 3 %
- ❖ Coup de pied supérieur : 11 %
- ❖ Coup de pied central : 8 %
- ❖ Coup de pied inférieur: 6 %

Ces valeurs sont valables que le coup soit porté durant un saut ou non.

L'arbitrage n'a pas été mis au point de manière dynamique. Les points de vie sont représentés par des surfaces rectangulaires d'une taille initiale définie. Les rectangles diminuent en taille dès que le personnage associé reçoit un coup.

## 3.4 Codage

### 3.4.1 Description de la SDL

La SDL est une bibliothèque très utilisée dans le monde de la création d'applications multimédias en deux dimensions comme les jeux vidéo.

Elle est écrite en C (mais peut parfaitement être utilisée par d'autres langages de programmation) et elle gère en outre :

- l'affichage vidéo,
- l'audio numérique,
- la gestion des périphériques communs comme la clavier, la souris mais aussi le joystick,
- les événements,
- l'utilisation de timers,
- ...

Il existe des modules spécifiques à la SDL tels que :

- SDL\_image : pour la gestion d'un large type d'images (PNG, BMP, JPEG...),
- SDL\_ttf : pour la gestion de certaines polices,
- fmod : pour la gestion audio
- SDL\_net : pour la gestion réseau,
- ...

Dans le cadre de notre projet, nous avons exploité le module `SDL_image` afin de pouvoir manipuler des images en format PNG (Portable Network Graphics). Nous souhaitons travailler en PNG pour plusieurs raisons :

- la compression sans perte,
- la gestion de la transparence,
- il est conçu selon standard ISO,
- c'est un format ouvert.

Afin d'améliorer la présentation de notre logiciel, nous avons défini une page d'accueil où les joueurs pourront mettre leurs noms, pour cela nous avons utilisé le module `SDL_ttf`. Toujours dans cet esprit d'esthétisme, le module `fmod` nous a servi vers la fin du projet afin d'ajouter un fond sonore au jeu.

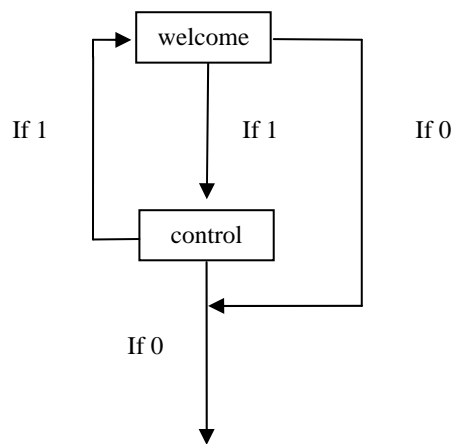
L'intérêt de cette bibliothèque est qu'elle offre des fonctions préexistantes essentielles quant au chargement d'images, la gestion du temps...

Voici certaines fonctions que nous avons utilisées :

<code>SDL_PollEvent</code> :	permet de sonder et voir les événements en attente.
<code>SDL_Flip</code> :	permet de dire à la carte graphique qu'un code travaillant sur un tampon est fini et qu'elle peut à présent afficher le résultat (utilisé pour les double-buffers video)
<code>SDL_BlitSurface</code> :	permet de copier une surface à l'écran
<code>SDL_FreeSurface</code> :	permet de libérer une surface
<code>SDL_FillRect</code> :	permet de remplir une surface rectangulaire avec une couleur
<code>SDL_Delay</code> :	permet d'attendre un certain nombre de millisecondes avant de faire un retour
<code>SDL_GetTicks</code> :	donne le nombre de millisecondes écoulées depuis l'initialisation de la librairie SDL
<code>SDL_GetKeyState</code> :	donne une aperçu de l'état courant du clavier
<code>SDL_CreateRGBSurface</code> :	permet de créer une <code>SDL_Surface</code> vide (une <code>SDL_Surface</code> est une structure de donnée définissant une surface graphique)
<code>SDL_SetColorKey</code> :	permet d'ajouter une couleur de transparence, afin que les surfaces ne soient pas toutes des "gros carrés".
<code>SDL_MapRGB</code> :	permet d'obtenir le code d'une couleur en fonction du format d'une surface.
<code>IMG_Load</code> :	permet de lire correctement et de charger une image selon son format.0

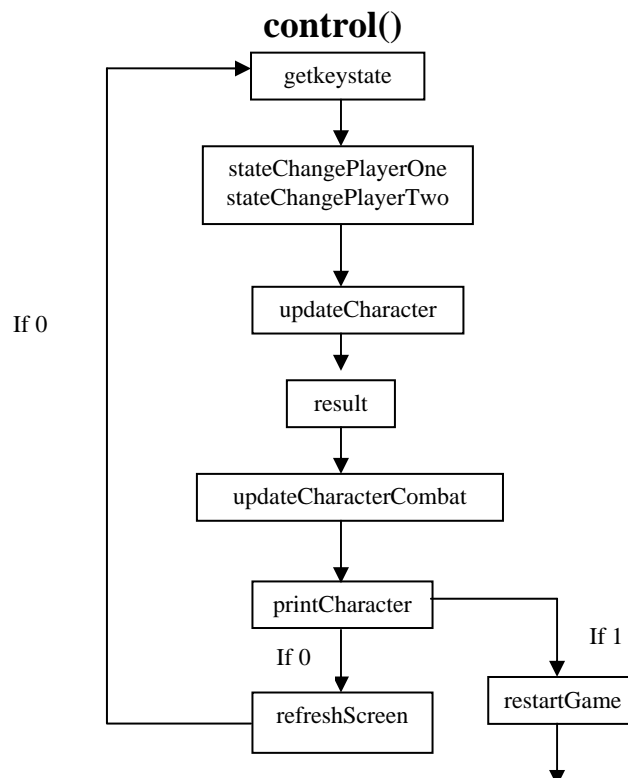
### 3.4.2 Commentaires des codes

#### 3.4.4.1 Lancement du jeu : accueil



**int welcomePage ()** – Cette fonction initialise les structures de données des deux personnages. Elle demande ensuite au joueur de rentrer les valeurs 1 pour commencer une partie ou 2 pour quitter le jeu. S’il décide de commencer une partie, il doit alors rentrer les noms des deux personnages avant de pouvoir débiter le jeu.

#### 3.4.4.2 Algorithme principal du jeu



Nous devons vérifier l'état du clavier pour savoir quelle touche a été appuyée, grâce à la fonction **getkeystate**. Il s'agit ensuite de calculer les nouveaux états et les nouvelles positions des personnages.

**states stateChangePlayerOne (Person \*characterOne, Person \*characterTwo, Uint8 \*keystates)** – Retourne l'état du personnage en concordance avec la touche qui aura été appuyée et l'état actuel du personnage. Cette fonction ainsi que **stateChangePlayerTwo** définissent les combinaisons de touches pour les deux personnages.

**void updateCharacter (Person \*character, states newState )** – S'il y a un changement dans l'état du personnage, cette fonction met son état à jour et remet le compteur de sprites à zéro, ou alors elle décrémente le compteur de sprites.

La mise à jour des personnages permet alors l'application de nos règles d'arbitrage.

**States result (Person \*attacker, Person \*defender)** – Cette fonction détermine si les deux personnages sont en train de combattre. Si c'est le cas, elle déterminera aussi lequel attaque et lequel est en défense. Elle détermine et applique les dommages infligés par l'attaquant en concordance avec la nature des coups, seulement si l'autre personnage ne se défend pas.

La fonction suivante s'applique pour des cas particuliers :

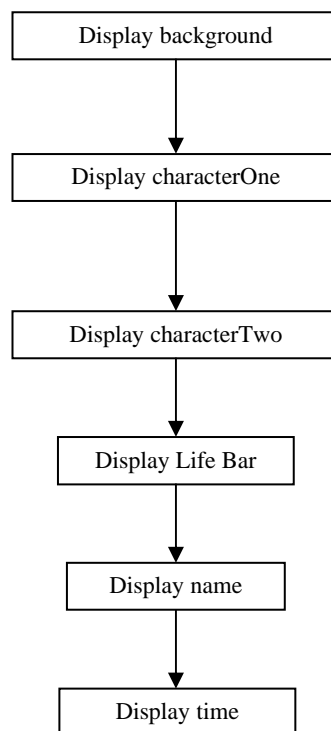
**void updateCharacterCombat (Person \*character, states newState )** – Cette fonction met à jour l'état des personnages dans les cas où ils reçoivent un coup (au visage, dans la partie torse/ventre et en position accroupie) ou dans les cas d'une victoire et d'un KO.

**int printCharacter (Person \*characterOne, Person \*characterTwo)** – Cette fonction permet de charger l'image sur la surface attribuée. Elle retourne 1 si l'état du personnage est KO ou VICTORY, 0 sinon.

**int restartGame()** – Cette fonction demande au joueur de choisir entre relancer une partie ou quitter le jeu.

### 3.4.4.3 Mise à jour de l'écran après tous les calculs d'états et de positions

**refreshScreen(Person \*characterOne, Person \*characterTwo)**



**void refreshScreen (Person \*characterOne, Person \*characterTwo)** – Cette fonction appelle les autres fonctions de la SDL **apply\_surface**, **SDL\_Flip**, and **SDL\_Delay** pour mettre à jour l'écran tout en vérifiant réalisant l'inversion du personnage afin d'obtenir un deuxième personnage. Il affiche ensuite les personnages, leurs barres de vie, leurs noms et le temps écoulé depuis le début d'un combat.

## 3.5 Tests

### ❖ Test sur la non superposition des personnages

#### 1) Modules nécessaires :

Affichage, gestion des personnages, règle de mouvement, règle d'animations

#### 2) Fonctionnement du test :

On met en contact deux personnages soit en les rapprochant par la marche ou par des coups.

#### 3) Oracle :

Si un personnage se rapproche de son adversaire en marchant, son image ne doit pas « traverser » celle de son adversaire même si le joueur laisse la touche marche enfoncée. Dans ce cas, on doit voir le personnage continuer à marcher sur place en restant collé à son adversaire sans le traverser.

De même, quand un personnage donne un coup, ce dernier ne doit pas traverser le corps de l'adversaire.

### ❖ **Test sur la réaction d'un personnage suite à un coup**

#### 1) Modules nécessaires :

Affichage, gestion des personnages, règles de mouvements, règles d'animations

#### 2) Fonctionnement du test :

On prend un personnage et on lui fait exécuter les différents coups définis sur un personnage adverse afin d'observer comment ce dernier réagit selon qu'il est protégé ou pas et en fonction du type de coup donné et de la zone où il a été touché.

#### 3) Oracle :

Quand le personnage adverse reçoit un coup, s'il s'est protégé, il reste sur place en position de défense.

Sinon quelque soit la zone touchée, s'il reçoit un coup normal alors il tombe a terre ; s'il reçoit un coup fort (heavy hit), il est projeté au sol.

### ❖ **Tests sur la synchronicité coups/arbitrage**

#### 1) Modules nécessaires :

Tous

#### 2) Fonctionnement du test :

On affronte deux personnages qui se donnent des coups simultanément ou pas et on observe comment évoluent leurs barres de vie en fonctions des coups donnés par les deux personnages.

#### 3) Oracle :

Le décompte des points de vie doit se faire pour les deux personnages et non pas un seul ni aucun des deux personnages.

Le décompte des points de vie doit être le même pour chacun des deux personnages donc s'assurer qu'il n'y en ait pas un qui perde plus de vie que l'autre pour un même coup donné.

Dès qu'un coup est reçu par un personnage, on doit observer immédiatement après (du moins avant qu'un autre coup ne soit reçu) une perte de points dans la barre de vie associée à ce personnage. Le test échoue si la perte de point ne s'effectue pas dans la durée autorisée ou si elle se produit dans la mauvaise barre de vie.

### ❖ **Test sur la fin d'une partie de jeu**

#### 1) Modules nécessaires :

Tous

#### 2) Fonctionnement du test :

On fait affronter deux personnages jusqu'à ce qu'un des deux perde tous ses points de vie.

### 3) Oracle :

Dès que l'une des deux barres de vie est complètement vide, la partie doit s'arrêter. Le personnage vaincu doit tomber à terre et rester dans cette position tandis que le vainqueur saute de joie.

## **3.6 Intégration**

Le test d'intégration consiste à exécuter le programme en vérifiant que tous les différents tests définis ci-dessus sont validés ensemble. Dans le cadre de notre projet, nous ne pouvons pas vraiment parler de tests d'intégration, car les modules ne sont pas indépendants les uns des autres. Il n'est pas possible de tester un module séparément des autres car ils ont tous besoin les uns des autres.

Suivant l'évolution suivant du développement de notre projet :

- codage des mouvements (excepté les sauts) d'un seul personnage,
- puis codage pour le second personnage symétrique,
- ensuite implémentation des sauts,
- finalement, codage de la défense et de l'arbitrage,

les tests d'intégrations ont essentiellement consisté à :

- jouer avec les deux personnages opposés en vérifiant que le personnage symétrique ne pose pas de problème,
- vérifier la compatibilité entre les coups de pied/poing et les sauts ainsi que les combinaisons sauts+coups

## **4. Conclusion**

Nous avons passé plus de trois mois à réaliser ce jeu de combat 2D de type arcade que nous intitulé *FIGHT SPIRIT*.

Il nous a fallu installer et apprendre à utiliser la bibliothèque SDL ce qui nous a pris un certain temps. Les fonctions nous étant toutes données, l'important était de savoir manipuler les sprites. Ainsi, tout ce qui a précédé la partie codage était essentiel pour pouvoir avancer dans notre projet, à savoir la définition des diagrammes d'états, des règles d'animations et de mouvements. Finalement, pour des raisons pratiques de codage, nous avons dû changer certaines de nos règles initiales (par ex : pour les règles de sauts, abandon de la pesanteur pour un saut purement linéaire...)

Nous avons passé beaucoup de temps à implémenter les différents coups, puis les sauts et enfin la gestion des collisions ainsi que l'arbitrage. Nous nous sommes permis la dernière semaine du projet, d'ajouter des éléments supplémentaires pour le côté « fun » du jeu, avec une page d'accueil et un fond sonore.

Ce programme appelle évidemment à des améliorations telles que :

- le rendre exploitable sur tout système, en effet nous n'avons pas pu répondre à cette contrainte de réalisation,
- l'ajout de personnages supplémentaires afin de laisser le choix du personnage aux joueurs,
- définir des règles de jeu plus évoluées comme des parties en plusieurs tours,
- définir des « coups spéciaux » pour donner un aspect encore plus fun au jeu,
- une conception dynamique des règles d'arbitrage,
- ...

Pour finir, quant à l'intérêt du travail réalisé par notre équipe, nous en sommes relativement satisfaits. Les tâches ont été bien définies et réparties entre nous dès le début du projet (site web, gestion de projet, codage). Nous considérons avoir travaillé assez régulièrement, ce qui a permis au projet d'avancer correctement sans retard pénalisant.

Nous avons néanmoins connu un problème au niveau de la communication au sein de l'équipe : en effet, certains d'entre nous n'étant pas sur la MAISEL, nous communiquions principalement via MSN ou par mails. Cette communication indirecte n'a pas été propice notamment quant à l'envoi des nouveaux documents et des nouveaux codes, ou même pour prévenir les autres de ce que chacun avait pu faire sur le projet au cours de la semaine. Mais au final, nous avons réussi à nous adapter à cette façon de communiquer.

## 5. Bibliographie

Installation et tutoriels SDL

[http://www.siteduzero.com/tuto-3-5252-1-installation-de-la-sdl.html#ss\\_part\\_3](http://www.siteduzero.com/tuto-3-5252-1-installation-de-la-sdl.html#ss_part_3)

[http://lazyfoo.net/SDL\\_tutorials/index.php](http://lazyfoo.net/SDL_tutorials/index.php)

## 6. Annexes

### 6.1 Annexes : Gestion de projet

#### 6.1.1 Plan de charge prévisionnel et suivi d'activités

Description de l'activité	PLAN DE CHARGES PREVISIONNEL						SUMI.D'ACTIVITES					
	Charge en %	Charge en H	Charge en H par participant				Charge en %	Charge en H	Charge en H / participant			
			Branche	Tan J.	Tan T.	Zheng			Branche	Tan J.	Tan T.	Zheng
Total	100%	286	76	76	71	71	100%	256	30	61	82	83
<b>Gestion de projets</b>	<b>21,0</b>	<b>60</b>	<b>19</b>	<b>19</b>	<b>11</b>	<b>11</b>	<b>23,8</b>	<b>61</b>	<b>18</b>	<b>24</b>	<b>9</b>	<b>10</b>
Réunion de lancement	2,8	8	2	2	2	2	3,1	8	2	2	2	2
Planning prévisionnel et Suivi d'activités	1,4	4	1	3	0	0	2,0	5	0	5	0	0
Réunions de suivi	9,8	28	7	7	7	7	12,1	31	8	8	7	8
Rédaction	4,9	14	5	5	2	2	4,3	11	2	9	0	0
Site Web	2,1	6	4	2	0	0	2,3	6	6	0	0	0
<b>Découverte de la bibliothèque SDL</b>	<b>16,8</b>	<b>48</b>	<b>12</b>	<b>12</b>	<b>12</b>	<b>12</b>	<b>15,6</b>	<b>40</b>	<b>4</b>	<b>12</b>	<b>12</b>	<b>12</b>
Tutoriaux	11,2	32	8	8	8	8	12,9	33	3	10	10	10
Codes sources	5,6	16	4	4	4	4	2,7	7	1	2	2	2
<b>Spécification</b>	<b>2,8</b>	<b>8</b>	<b>2</b>	<b>2</b>	<b>2</b>	<b>2</b>	<b>3,1</b>	<b>8</b>	<b>2</b>	<b>2</b>	<b>2</b>	<b>2</b>
Définition des fonctionnalités	2,8	8	2	2	2	2	3,1	8	2	2	2	2
<b>Conception préliminaire</b>	<b>9,8</b>	<b>28</b>	<b>7</b>	<b>7</b>	<b>7</b>	<b>7</b>	<b>5,9</b>	<b>15</b>	<b>1</b>	<b>4</b>	<b>5</b>	<b>5</b>
Définitions des fonctionnalités	7,0	20	5	5	5	5	4,7	12	1	3	4	4
Définitions des différents modules (.h)	2,8	8	2	2	2	2	1,2	3	0	1	1	1
<b>Conception Détaillée</b>	<b>25,1</b>	<b>72</b>	<b>17</b>	<b>17</b>	<b>19</b>	<b>19</b>	<b>25,4</b>	<b>65</b>	<b>1</b>	<b>8</b>	<b>28</b>	<b>28</b>
Définition des structures de données	2,8	8	2	2	2	2	2,7	7	0	1	3	3
Définition des règles d'animation : diagrammes état/transition des mouvements physiques	3,1	9	1	1	6	1	3,5	9	0	0	8	1
Définition des règles de mouvement : les sauts	2,8	8	1	5	1	1	1,6	4	1	1	1	1
Définition des règles de mouvement : les rebonds	2,8	8	5	1	1	1	0,0	0	0	0	0	0
Définition des règles de coups portés	3,1	9	1	1	1	6	3,9	10	0	1	1	8
Définitions des fonctions	5,6	16	4	4	4	4	7,4	19	0	3	8	8
Définition des tests unitaires	4,9	14	3	3	4	4	6,3	16	0	2	7	7
<b>Codage</b>	<b>14,7</b>	<b>42</b>	<b>10</b>	<b>10</b>	<b>11</b>	<b>11</b>	<b>15,2</b>	<b>39</b>	<b>0</b>	<b>7</b>	<b>16</b>	<b>16</b>
Ecriture des interfaces (.h)	1,4	4	1	1	1	1	1,2	3	0	1	1	1
Ecriture des différentes fonctions	7,7	22	5	5	6	6	9,7	25	0	5	10	10
Test unitaires	4,2	12	3	3	3	3	4,3	11	0	1	5	5
<b>Intégration</b>	<b>4,2</b>	<b>12</b>	<b>3</b>	<b>3</b>	<b>3</b>	<b>3</b>	<b>4,7</b>	<b>12</b>	<b>0</b>	<b>0</b>	<b>6</b>	<b>6</b>
Intégration des différents modules	2,8	8	2	2	2	2	3,9	10	0	0	5	5
Tests d'intégration	1,4	4	1	1	1	1	0,8	2	0	0	1	1
<b>Soutenance</b>	<b>5,6</b>	<b>16</b>	<b>4</b>	<b>4</b>	<b>4</b>	<b>4</b>	<b>6,3</b>	<b>16</b>	<b>4</b>	<b>4</b>	<b>4</b>	<b>4</b>
Préparation de la soutenance	4,2	12	3	3	3	3	4,7	12	3	3	3	3
Soutenance	1,4	4	1	1	1	1	1,6	4	1	1	1	1

## 6.1.2 Planning Prévisionnel

Pour le	Planning Prévisionnel (mis à jour semaine après semaine)	Temps passé sur l'ensemble du projet (en H)			
		Branche	Tan J.	Tan T.	Zheng
15/02/2008	Réunion de lancement de projet	1	1	1	1
	Installation et apprentissage de la SDL : faire les tutoriaux	3	10	10	10
22/02/2008	obtenir une fenêtre en plein écran, résolution 800x600, affichage 16bits, faire bouger un carré sur l'écran et le faire rebondir sur les bords de l'écran	3	3	3	3
	rédaction du planning prévisionnel et du plan de charge prévisionnel et faire le suivi d'activité	0	2	0	0
	Réfléchir au site web et à son contenu (sujet, présentation du groupe, codes source, documents...)	6	0	0	0
	Charger et découper les sprites	0	5	0	0
14/03/2008	Définir les structures de données associées aux mouvements	1	1	1	1
	Commencer à réaliser des animations	0	2	5	5
	Définir les diagrammes d'états/transition et les règles d'animations et de mouvements (pesanteur, rebonds)	0	0	8	1
	Définir les coups portés et les zones de coups	0	1	1	8
	Réfléchir sur l'architecture matérielle minimale supportant notre programme	0	1	0	0
21/03/2008	Prendre en compte la gestion du temps	0	0	0	0
	Continuer à coder	0	...	...	...
	Finir les règles d'animations.....	0	0	...	...
	Terminer la conception détaillée avec les structures de données et les modules nécessaires	0	1	0	0
04/04/2008	Définir nos combinaisons de touches	0	...	...	...
	Rédiger le pré-rapport (conception préliminaire, conception détaillée, cahier des charges, planning prévisionnel)	0	2	...	...
	Comme d'habitude, continuer le codage	0	...	...	...
	Concernant le pré-rapport : bien définir les tests (modules nécessaires, description, oracle)	0	2	0	0
	Donner plus de détails sur les règles de mouvements	0	0	0	0
	<b>Absolument définir les structures de données notamment tout ce qui est lié au personnage</b>	0	0	1	1
	<b>Se booster pour en finir avec les sauts et les rebonds !!</b>	0	0	...	...
18/04/2008 Annulé donc 05/05/2008	Commencer à coder tous les différents coups pour pouvoir faire des combinaisons de coups	0	0	3	3
	<b>!!!!!!! Remettre en forme notre codage !!!!!!!!</b>	0	1	4	4
	Régler le problème des scintillements avec l'idée de l'off-screen	0	0	0	0
23/05/2008	<b>!!!! Absolument mettre en place l'arbitrage !!!!</b>	0	2	2	2
	>>>>> Faire fonctionner globalement le jeu <<<<<<	0	...	...	...
30/05/2008	Régler les derniers bugs d'animation pour le personnage inversé	0	1	5	5
	<b>!!!!!! Rédiger le rapport final, les transparents pour la soutenance et le manuel d'utilisateur !!!!!</b>	1	4	0	0
...	Signifie que l'on passé pas mal de temps en discontinu donc difficile d'évaluer				

## 6.2 Annexes : Codes sources

Les codes sources sont consultables sur notre site web à l'adresse suivante :

<http://site.voila.fr/piarcade/Maitre.htm>

Cliquez ensuite sur l'onglet « La programmation » du menu défilant.

## 6.3 Annexes : Site web

### 6.3.1 Contenu du site web

Notre site web a servi comme il a été précisé dans les exigences de suivi du projet, de plate-forme permettant aux membres de l'équipe et à notre encadrant, d'y retrouver :

- les comptes-rendus de nos réunions
- divers documents de gestion de projet :
- le planning prévisionnel,
- le cahier des charges,
- le pré-rapport,
- le rapport final.

Pour les consulter, allez sur <http://site.voila.fr/piarcade/Maitre.htm> et cliquez sur l'onglet désiré du menu défilant.

## 6.3.2 Document type de nos comptes-rendus

# Compte-rendu de la réunion du 29 Février 2008

### 1) Résumé du travail effectué :

→ Une première version du planning a été réalisée.

Les remarques et explications à propos du planning sont :

- La définition de structure permet de travailler sur le jeu, il s'agit par exemple de définir et de programmer les actions possibles pour les personnages.
- L'intégration est en fait le rajout de bout de codes qui permettent au jeu de fonctionner (corrections, ajouts, ...).
- La spécification et la conception désigne en fait l'analyse des besoins et inclus donc les règles de jeu par exemple ainsi que la conception détaillée.

→ Les tutoriaux de la bibliothèque SDL ont été faits par une partie du groupe, et quelques explications supplémentaires ont été apportées.

Quelques explications sur la SDL :

- SDL travaille avec la mémoire vidéo, ce qui explique la grande rapidité de fonctionnement.
- Pour les sprites, il faut charger l'image (composée en fait de plusieurs images) et la découper pour avoir les images qui forment les séquences-mouvement. Pour produire l'animation, il est possible d'utiliser un compteur pour passer d'une image à l'autre (utilisation de pointeurs)
- Il faudra définir une structure pour chaque mouvement.
- Il faudra également définir les zones images.

### 2) Entrée dans le vif du sujet :

→ Le travail à réaliser en priorité est:

- Charger les sprites et les découper
- Définir les structures de données correspondants au mouvement
- Réaliser les animations mouvement.

→ A faire rapidement également : définir des règles d'animation et les coups.

- Un diagramme est souhaitable : diagramme à deux niveaux avec les mouvements importants (ie ceux à coder absolument) et les autres, définissant les liens entre mouvements et positions : par exemple quels mouvements sont possibles à partir du repos (état de départ) ou à partir d'un déplacement vers la droite. Ce diagramme présentera les mouvements et états, les commandes pour passer de l'un à l'autre.
- Les lois physiques doivent être prises en compte dans la définition des mouvements, c'est-à-dire la pesanteur, le poids, le principe d'inertie, etc
- Deux mouvements (parmi d'autres) sont à envisager : la trajectoire de chute (saut) et le rebond (contre le sol, la paroi ou un autre personnage)
- Il faut également définir le nombre d'image par seconde (25 parait un bon chiffre, ce qui donne un temps de 40ms par image, l'image est fluide est le temps de calcul pour

l'ordinateur est plus important que pour 50images par seconde). Pour le programme ne passe pas à une image avant d'avoir finis la précédente, on peut utiliser les « ticks » et mettre une condition dans la programmation sur le nombre de ticks.

→ Dans les règles du jeu il faudra au moins :

- Coups portés à définir : sur quelles parties de l'adversaire (définissables avec des zones rectangulaires par exemple).
- Les pertes d'énergie (en fonction de la zone et éventuellement de la vitesse de frappe).

### **3) Les questions durant la réunion :**

**Q° :** Doit-on aussi définir des barres de vie et ce genre de choses ?

**Rep. :** Oui

**Q° :** Concernant l'utilisation des spriters, faut-il les faire nous-même si on veut en inventer

**Rep. :** Attention au copyright de celles fournies.