# Programmation II

## Projet : Jumping Bananas

### ANALYSE DU PROBLÈME ET SPÉCIFICATION DE LA SOLUTION


**ALEXIS DEGRAEVE**

alexis.degraeve@student.unamur.be

*Faculté d'informatique*
*Université de Namur*
*Année Académique 2018-2019*

# Table des matières

# Introduction

Par la création du jeu Jumping Bananas, différentes notions seront abordées comme l'ouverture d'un fichier map. Le fichier map est un fichier texte qui contient la carte du jeu sous forme de caractère ASCII. Nous verrons également la gestion de structures, de tableaux, de pointeurs mais aussi comment découper le projet en différents fichiers. Il est évident qu'il faut découper le projet en plusieurs fichiers, ce qui évite d'avoir un seul fichier monolithique, qui contient tout le code. Le jeu Jumping Bananas est un jeu de plate-forme où le singe doit se déplacer à l'aide du clavier, en évitant les ennemis et en récoltant des points soit en prenant des bananes ou des pièces. Le singe a aussi des super-pouvoirs en lançant des bulles qui permettent de capturer l'ennemi (le serpent ou autre animaux). Dans ce rapport, ce qui sera abordé c'est toute la structure de programmation mais d'un point de vue haut niveau.

Je vous souhaite bonne lecture de ce rapport.

> ***"Le logiciel est une excellente combinaison entre l'art et l'ingénierie."***
>
> *Bill Gates*

# Jumping Bananas

## Choix de structures de données

Nous retrouverons différents types de structures dans le code :

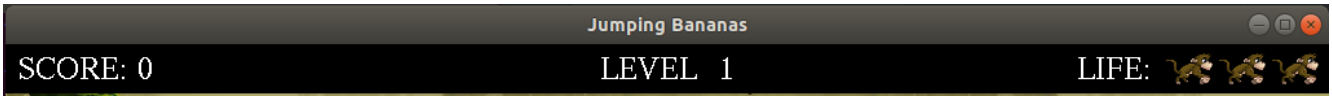
```
Struct Coord {
        GLfloat X ;
        GLfloat Y ;
}
```

Structure importante qui permet d'avoir la coordonnée d'une entité en GLfloat.

```
Struct Singe {
         struct Coord position;
         struct Coord posorigin;
         int vies;
         int points;
         bool right;
         bool jump;
         int keyanm;
}
```

La structure posorigin est une copie de la position au départ du chargement du niveau. En effet, cela permettra de restaurer la position du singe au cas où il se ferait tuer par un ennemi.

Dans cette structure on retrouve le singe l'acteur principal. Il a un nombre de vie trois au départ comme spécifié dans le règlement.




Le nombre de points est à 0 par défaut.

Il y a également l'orientation du singe, s'il se trouve vers la droite alors right est à true aussi non il sera à false.
La variable **jump** permet de savoir si le singe est entrain de sauter dès que l'on appuie sur la barre d'espace. La variable **keyanm**, de savoir l'image du sprite qui sera vue pendant l'animation (voir la section graphisme pour plus d'information sur l'animation).

Par défaut le nom de la variable du singe sera **cocoSinge**.

Comme on peut le voir, il a une position qui est très importante car elle se mettra à jour au fur et à mesure que le joueur déplace le singe.  Cette coordonnée servira pour les tests des collisions.

En prenant des items le singe peut collecter des points.
Le singe a aussi des bulles qu'il peut envoyer et qui fige l'ennemi pendant 16 secondes +/-. Si le singe ne saute pas sur la bulle alors l'ennemi se libère.

Pour l'ennemi on a ceci comme structure :

```
typedef struct Ennemi {
        struct Coord position;
        bool capture;
        bool left;
        bool jump;
        int keyanm;
        bool freeze;
        float currentspeed;
        float speed;
        bool banane;
        char type;
        struct Ennemi *nextEnnemi;
}Ennemi ;
```

Position donne la position de l'ennemi qui servira au test de collision. Il y aussi l'orientation de l'ennemi, s'il est vers la gauche alors left à est true sinon il est à false. Si l'ennemi est gelé, alors freeze est à 1. Speed permet de définir la vitesse de l'ennemi : plus la valeur est élevé plus l'ennemi se déplacera lentement. Pour le niveau 1, l'ennemi se déplace lentement sinon pour les autres niveaux l'ennemi se déplace plus rapidement. La variable banane permet de savoir s'il faut afficher une banane à la place de l'ennemi. Elle servira également au test de collision. Le type de l'ennemi est soit 'E' pour serpent et 'F' pour hérisson.
Il y a un test de collision entre l'ennemi et la bulle.

La fonction principale est void testCollideEnemy(bool *testcollide, int posx, int posy). Cette fonction regarde si la bulle est en intersection avec un ennemi et si le timer est en activité dans ce cas on fige l'ennemi. Si le timer est à zéro et que l'ennemi est prisonnier dans la bulle alors il est libéré. Si l'ennemi est dans une bulle et que le timer est enclenché et qu'il y a eu un saut du singe sur celle-ci alors la bulle éclate et on voit la banane. Je fais reculer le singe aussi pour qu'il n'aille pas directement sur la banane. Si la banane est récupérée, alors elle permet d'augmenter les points mais aussi l'ennemi est enlevé de la liste chaînée.

Un ennemi peut être un serpent ou un hérisson. Il n'a qu'une seule vie s'il est tué par une bulle. Une image gauche ou droite suivant l'état. Il a aussi une vitesse. Et un temps qu'il reste pour capturer l'ennemi. Il aura aussi des coordonnées qui serviront pour le test de collision. Pour l'ennemi j'ai utilisé une liste chaînée. On remarque aussi char type qui permet de définir si c'est un hérisson ou un serpent.

Le déplacement de l'ennemi se fait de gauche à droite. S'il y a du vide en-dessous il changera de sens grâce à la fonction **flipobject**.

Il y a des fonctions plus simples comme void **testMonkeyGetCoin**(int posx, int posy). Quand le singe a pris une pièce cela augmente son score.
Il y a également une fonction void **testMonkeyGetBanana**(int posx, int posy), quand le singe a pris une banane cela augmente son score.

**Test de physique**

Quand le singe saute en l'air on le fait simplement augmenter de 4 cases maximum. En revanche si le singe rencontre un obstacle comme une plate-forme dans ce cas le singe passe en mode chute avec la variable **fall** à true. Si la variable fall est à true le singe tombe vers le bas jusqu'à ce qu'il rencontre un sol ou une plateforme.
Pour gérer la gravité plusieurs fonctions ont été nécessaires :

- void testMonkeyHitThePlatform(int posx, int posy)
- void testMonkeyHitTheFloor(int posx, int posy)
- testMonkeyJumpAndFall(posx, posy);
- testMonkeyJumpAndNotFall(posx, posy);

On aura également une structure joueur :

```
struct Player {
        char *name;
        int currentlevel;
        int score;
        bool startagain;
};
```

On a la struct Player qui contient le nom par défaut c'est humain, ainsi que le niveau courant, le score. On remarque la variable **startagain** qui permet de dire si on redémarre le jeu ou pas (en tenant compte de l'augmentation de niveau si on a gagné).

Cette structure est la structure principale du programme.  Cette structure est composée d'un objet map qui est un tableau du niveau du jeu.

Le fichier map pourra contenir les caractères suivants :

| Caractère | Signification |
|---|---|
| **X** | Singe |
| **P** | Plateau |
| **E** | Serpent |
| **F** | Hérisson |
| **P** | Pièce (Coin) |
| **B** | Banane |
| **S** | SOL |
| **-** | (Rien) |

A l'initialisation de la map, on chargera le fichier texte et après il sera mis dans une variable char **platform. J'utilise un pointeur de pointeur de caractère. Pourquoi pas un tableau ? Comme cela le fichier de la map peut contenir autant de caractères qu'on veut, toujours le même nombre de caractères par ligne.
On fait un premier scan du fichier pour savoir le nombre de lignes et le nombre de colonnes. Après une lecture réelle, je charge la map dans un pointeur de pointeur de caractère (peut-être vu comme un tableau à 2D mais dont le nombre de lignes et de colonnes est variable).

La grille du jeu aura une taille maximale et minimale définit par deux constantes W et H tandis que MaxW et MaxH sont relatifs juste à l'écran. Le W et le H indique la taille totale de la grille de tous les écrans donc si par exemple on a 16 cases dans un écran sur 4 écrans on aura une grille de 64 en W et 16 en MaxW. MaxW représente la largeur maximale de la grille et MaxH la hauteur maximale de la grille par écran.

Pour l'instant il y a 3 niveaux. On voit que le niveau 1 se compose que d'un seul écran. En revanche, le niveau 2 se compose de plusieurs écrans ainsi que le niveau 3.

Pour le tableau des High Score, on affiche le score des joueurs par ordre décroissant des points gagnés.

Un objet tel que la banane ou une pièce rapporte des points une fois collecté. Dans la variable platform on aura les coordonnées de l'objet indiqué par la ligne et la colonne.

Il suffit de voir si le singe est en superposition avec cette coordonnée. Si c'est le cas l'objet est collecté si non ce n'est pas le cas. Voir les test de collision avec par exemple la fonction **testMonkeyGetCoin**.

**void initialize_platform(int level)** {

…

}

Cette fonction permettra d'Initialiser la plateforme du jeu. La fonction **read_platform** qui se trouvera dans le fichier **loadMap** permet de charger un fichier texte contenant les caractères vus précédemment. Et ensuite par les fonctions suivantes de substituer chaque caractère par l'élément graphique approprié.

- void displayMonkey(GLfloat posMonkeyX, GLfloat posMonkeyY, bool right, bool jump, int keyanm);
- displayEnnemy(char type, int posEnemyX, int posEnemyY, bool left, int keyanm, bool capture);
- void displayCoinSilver(int posCoinSilverX, int posCoinSilverY);
- void displayBananas(int posBananaX, int posBananaY);
- … (voir fichier controldisplay.h)

**displayBackground** () permet d'afficher un arrière-plan. Comme le viewport est juste sur une partie de l'écran on verra à l'écran qu'une partie de l'écran.

Exemple de scrolling sur 2 écrans :


Le scrolling peut se faire sur autant d'écrans que l'on veut, ici j'ai pris un exemple de deux écrans. Pour le scrolling j'utilise glTranslatef dans la fonction scrollingScreen . Quand le singe arrive à  l'extrémité de l'écran 1, alors le scrolling se met en route même chose pour les autres écrans. Le scrolling s'adapte correctement suivant que le singe se trouve vers la gauche ou la droite. Comme on peut le voir le Screen1 est juste une partie visible de l'image de l'arrière-plan qui est plus grande. Screen 1, 2 sont juxtaposés et contiennent l'image d'arrière-plan en entier mais on voit qu'une partie de l'image d'arrière-plan dans le viewport. Cela permettra d'aider pour le scrolling horizontal de l'arrière-plan. Pour faire un effet de scrolling on déplacera la coordonnée de tous les objets avec gltranslatef qui donnera l'illusion que cela bouge. Comme l'arrière-plan est plus grand quand on déplace la vue, on a l'impression que le décor bouge. On remarque aussi que l'en-tête ne bouge pas, c'est la seule partie ou on fait un remplacement de la matrice courante avec la matrice identité. C'est comme si gltranslatef était annulé et on restaure l'état précédent avec glPopMatrix.

Ainsi on retrouvera le code suivant :

```
glPushMatrix();
glLoadIdentity();
displayHeader(cocoSinge.points, cocoSinge.vies, humanplayer.currentlevel);
glPopMatrix();
glutSwapBuffers();
```

Suivant la position du singe en coordonnée X on sait quel emplacement de l'écran on doit montrer.

Pour gérer le scrolling il y a la méthode scrollingScreen() qui est appelée lors des tests de collisions. Si le singe est dans un seul écran pas besoin de scrolling par contre s'il y a plusieurs écrans, dès que le singe est à la limite de l'écran le scrolling s'active. Le scrolling ne s'activera pas si le singe se trouve dans le premier écran et s'il va en dehors des limites de l'écran. Le scrolling ne s'activera pas non plus si le singe va dans le dernier écran le plus à droite.

Il y a également des tests de limite de l'écran pour que le singe n'aille pas en dehors d'un écran qui n'existe pas. Avec la fonction **monkeyLeftMoveOutsideScreen** pour la gauche de l'écran et **monkeyRightMoveOutsideScreen** pour la droite de l'écran.

Par défaut lors de l'initialisation du jeu, le singe se trouve à l'endroit indiqué par M dans le fichier texte map.

On fera appel à des fonctions Glut pour la gestion de l'affichage de l'écran.
Une résolution de l'écran sera imposée ici 1024*768.

**Gestion du clavier**

Par l'intermédiaire de la fonction glutKeyboard :
glutKeyboardFunc(KeyboardJumpingBananas) ;

Pour la gestion du clavier, il y a plusieurs gestions du clavier une pour l'écran principal, l'autre pour l'écran de jeu, une pour l'écran des highscore.

On fera appel à la fonction keyboardGame () qui contiendra tous les déplacements nécessaires :

À savoir le déplacement du singe peut se faire avec les touches, Q pour la gauche et D pour la droite. Ainsi que la barre d'espace pour sauter. La touche A pour lancer des bulles. Il y a également la touche escape pour revenir au menu principal.

Quand on est dans le menu principal, la fonction keyboardMainMenu est appelée. Quand on est dans d'autre vue comme Highscore alors la fonction keyboardView est appelée. Le mapping du clavier est différent celui le contexte si c'est l'écran de menu ou l'écran de jeu ou l'écran de highscore ou des règles.

La méthode glutDisplayFunc(displayCurrentScreen) appellera la fonction **displayCurrentScreen ()** . Cette méthode permet de sélectionner l'écran approprié suivant ce qu'on sélectionne dans le menu principal. Par défaut on arrive sur le menu principal, si l'utilisateur presse 'n' il entre dans le jeu, si c'est 'r' il voit les règles du jeu, si c'est 'h' le tableau de highscore du jeu. Si c'est « ESC » il quitte le jeu. La fonction **selectGame**() afficher l'écran du jeu avec les différentes sous-fonction pour afficher l'arrière-plan et les spirites correspondant.

**void loading_level(bool refresh)** Méthode qui permet d'afficher l'écran courant du jeu. Si le paramètre de la fonction est à true alors on recharge complètement la map du niveau approprié suivant le currentlevel de la variable humanplayer.

**void testPassCollide()** Cette méthode pourra détecter la collision.

- Si la position du singe est la même que la position d'un ennemi ,alors le singe perd une vie sauf si l'ennemi a une valeur pour la variable capture qui est à true.

Voir la fonction **testCollideEnemy**() pour les différents tests de collision avec l'ennemi. Si le singe est tué alors il réapparait à la position de départ qu'il était dans le niveau de jeu. Grâce a la structure du singe on voit le nombre de vie qui décrémente de 1 si l'ennemi le touche et qu'il n'est pas dans une bulle. Si le singe perd toutes les vies on voit l'écran perdu.

- Si la bulle touche l'ennemi alors une valeur de 16 sec +/- d'emprisonnement est activé sur l'ennemi. Tant que le temps n'est pas écoulé et que le singe éclate la bulle, la variable capture est à tuer et le singe peut tuer l'ennemi. Le singe gagne des points en ramassant la banane et l'ennemi sort de la liste chainée ennemis.
- Si la position du singe est la même que la position d'une pièce ou d'une banane le singe gagne des points. Ensuite la pièce ou la banane disparait (retiré de la variable platform).
- Si la position du singe est sur du vide alors il peut aller à cet endroit
- Si la position du singe est la même que la plateforme il ne peut pas aller là sauf s'il saute par-dessus.

### HighScore

Le tableau des highscore est chargé à partir d'un fichier texte. Il sera mis dynamiquement dans une variable pointeur de structure. Lors du démarrage du jeu on précharge une fois le fichier. Ensuite les données de la structure seront modifiées au cours du jeu. Dans HighScore, on affiche les résultats pointés sur cette structure highscore. Quand on quitte le programme, on sauvegarde les données de la structure dans le fichier highscore.txt

### Fonction display win/loose

A chaque rafraîchissement d'écran la fonction show_win_loose vérifie si le nombre de vie est zéro alors oui, on a perdu. Cette fonction vérifie si on a toujours des vies et que toutes les bananes ont été mangées, dans ce cas là c'est gagné.

Une fois que l'utilisateur presse enter, l'écran gagné ou perdu disparaît et on passe au niveau suivant ou on reste au même niveau.

## Découpe en sous-problèmes

Au lieu de tout mettre dans le même fichier. On va séparer en différents fichiers pour une meilleure structure :

| Fichier | Description |
|---|---|
| **main.c** | Contient le main principale du programme. Celle fera appelle à l'initialisation du jeu. |
| **loadMap.c** | Chargement de la map du jeu |
| **loadMap.h** | En-tête des fonctions du fichier loadMap<br>LoadMap (charge le fichier texte de la map pour le mettre dans un tableau)<br>W (Taille maximale de la map en largeur) Il s'agit du cumul de plusieurs écran si on a 4 écran on devra faire 4 * la résolution de l'écran en largeur.<br>H (Taille maximale de la map en hauteur)<br>Gestion de l'animation de la bulle, de l'ennemi et du singe. |
| **Controlplayer.c** | Contrôle du déplacement du singe ainsi que les tests de collision. Collision entre le singe et d'autres éléments. Collision entre les ennemis et les bulles. Gestion du scrolling aussi. |
| **Controlplayer.h** | |
| **controlDisplay.c** | Dessiner l'écran de la map du Jeu<br>Contiendra plein de fonctions telles que DrawSinge(), DrawSerpent() … |
| **controlDisplay.h** | En-tête des fonctions du fichier drawMap<br>displayMonkey (…) // Dessine le Singe<br>displaySnakeWalk (…) // Dessine le serpent qui marche<br>displayHedgeHogWalk (…) // Dessine le hérisson qui marche<br>displayCoinGold (…) // Dessine la pièce<br>displayBananas (…) // Dessine la banane<br>displayFloor (…) // Dessine les plateformes du jeu<br>displayHeader (…) // dessine la barre de statuts (qui contient les vies)<br>displayBackground (…) // Dessine l'arrière-plan<br>… |

| | |
|---|---|
| **Maplevel1.txt** | Contient la map du jeu niveau 1 |
| **Maplevel2.txt** | Contient la map du jeu niveau 2 |
| **Maplevel3.txt** | Contient la map du jeu niveau 3 |
| **Menu.c** | Contient la gestion du menu |
| **Menu.h** | |
| **Personnages.h** | Description de la structure du singe, du serpent, des ennemis, de la bulle |
| **Readimage.c** | Fonction pour lire les images avec l'intermédiaire de la librairie SOIL |
| **Scores.txt** | Donnée pour le tableau des highscores |

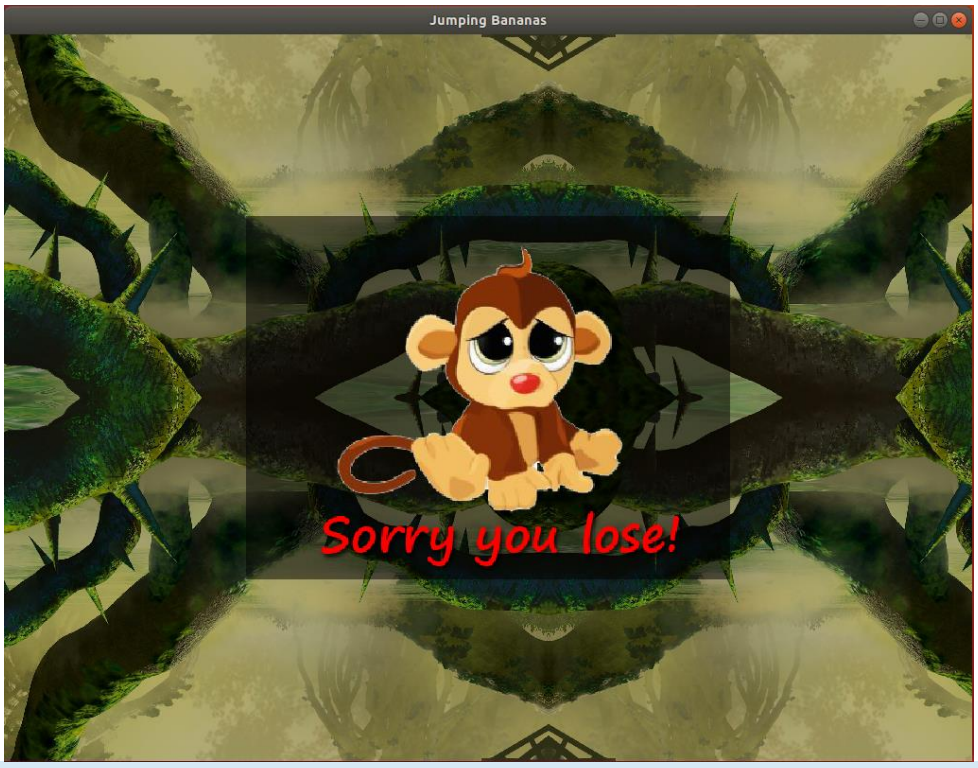
# Définition des écrans


Menu principal


Ecran du jeu


Ecran des Highscores


Ecran des règles

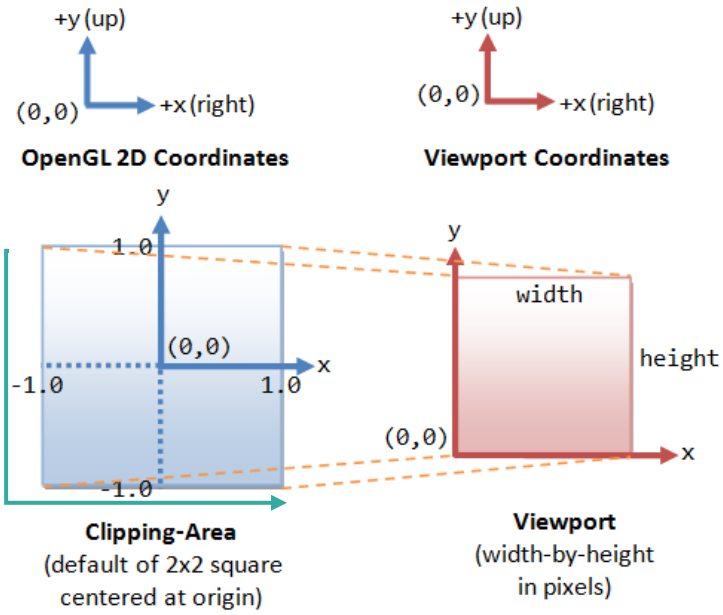



Ecran Perdu


Ecran Gagné

## Graphisme

On utilise le moteur GLUT qui est une bibliothèque utilitaire qui permet d'utiliser un moteur 3D reposant sur OpenGL. Cette librairie permet également de faire des interactions avec le système d'exploitation.




L'environnement se présente avec un axe x, y et z puisqu'il repose sur le moteur 3D OpenGL. La coordonnée d'un point est en (x,y,z). Pour dessiner les sprites, les images à l'écran on part du haut à gauche c'est-à-dire de la coordonnée (-1,1,0) ensuite (-1,-1,0) et puis (1,-1,0) et pour finir le tracé du polygone avec (1,1,0). Ainsi l'image sera orientée vers la droite.

Pour adapter l'écran et les sprites à la taille de l'écran, j'ai fait un rapport entre la taille de l'écran et le système -1 +1 utilisé par GLUT. Si on a un écran de 1024 et que le sprite fait 64 pixels. On divise 1024 par 2 pour avoir la taille de l'écran GLUT par 64 ce qui fait qu'on peut mettre 8 sprites sur une moitié d'écran en axe X et 8 sprites sur l'autre moitié de l'écran. Ce qui fait un total de 16 sprites en Longueur. On fait le même rapport aussi en vertical.

Pour accélérer l'affichage, on charge toutes les images avant et cela est mis dans un tableau. Lors de l'affichage, on rappelle l'image en question dans le tableau ave le glBindTexture.

Pour les sprites, l'animation du singe se compose de 6 images. J'ai dessiné l'animation dans Photoshop et exporté les images en .png

L'animation du singe qui saute (utilisation de l'animation quand on appuie sur la barre d'espace) :


monkey_jump_01.png


monkey_jump_02.png


monkey_jump_03.png


monkey_jump_04.png


monkey_jump_05.png


monkey_jump_06.png

L'animation du singe qui se ballade :


monkey_walk_01.png


monkey_walk_02.png


monkey_walk_03.png


monkey_walk_04.png


monkey_walk_05.png


monkey_walk_06.png

L'animation du hérisson de 4 images :


herisson_walk_01.png


herisson_walk_02.png


herisson_walk_03.png


herisson_walk_04.png

L'animation du serpent de 5 images :


snake_walk_01.png


snake_walk_02.png


snake_walk_03.png


snake_walk_04.png


snake_walk_05.png

Animation de la boule :

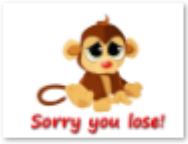

sprite_bubble.png


sprite_bubble_yellow.png

Comme la boule est semi-transparente, on peut observer l'ennemi emprisonné dedans.

Elément non animé :

Un écran pour gagner et pour perdre



screen_you_lose.png

screen_you_win.png

Un écran de fond


jungle_background.bmp

On remarque que l'écran de fond est répété de manière en miroir avec la fonction GLUT :

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_MIRRORED_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_MIRRORED_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
```

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);

Une banane

sprite_banana_small.png

Des pièces bananes

spritecoinbanana.png

spritecoinbananagray.png

Le sol et l'herbe

sprite_floor_small.png

sprite_grass_small.png

Un titre (pour pouvoir utiliser une police de caractère plus spécial)

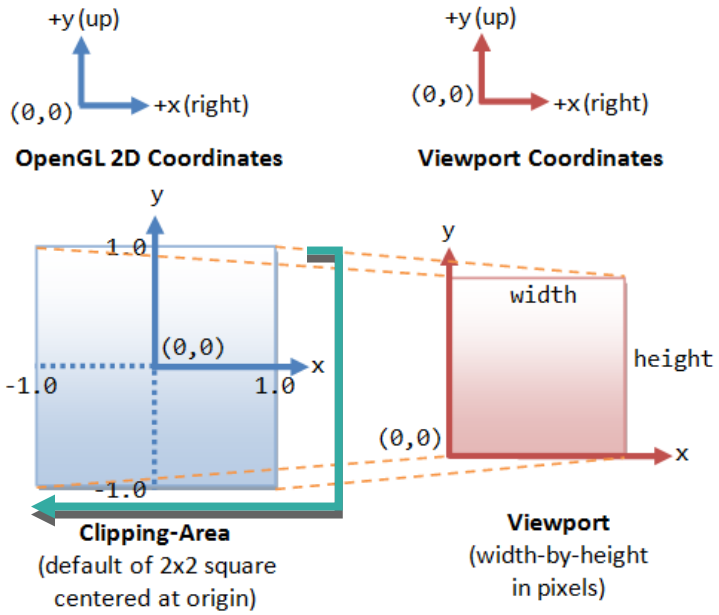

title.png

Les sprites ont une taille de 64*64.

Il y aussi une image de fond.

On voit que les images se construisent de la manière suivant coordonnée -1,-1 etc..

J’ai créé une méthode flip qui va dessiner l’image en suivant les coordonnées de manière suivante. On a vu précédemment que pour dessiner un sprite orienté vers la droite il faut commencer la première coordonnée des 4 points en haut à gauche. Et bien pour l’orienter vers la gauche on commencera la coordonnée toujours en (x,y,z) des 4 points en haut à droit.  Ce qui donne c’est-à-dire de (1,1,0) ensuite (1,-1,0) et puis (-1,-1,0) et pour finir le tracé du polygone avec (-1,1,0). Ainsi l’image sera orientée vers la gauche.




Pour afficher les sprites de manière transparente, j’utilise la fonction glut gl_blend, que je désactive après si je n’en ai pas besoin.
Par exemple pour l’affichage des scores au-dessus avec le nombre de vies restantes pour le fond noir le gl_blend n’est pas nécessaire, en revanche pour les singes en taille réduite qui permettent de voir le nombre de vies il faut utiliser le gl_blend.

Pour afficher du texte à l'écran, j'utilise la fonction Glut suivante : glutBitmapCharacter
Dans la fonction drawBitmapText(char *string,float x,float y,float z, bool bigsize) cela permet de dessiner du texte à une position x, y, z et de savoir si le texte sera grand ou pas.
On remarquera que pour afficher un texte numérique, il faut utiliser la fonction sprintf avec la chaîne de caractère convertie, le type du nombre à convertir, et la variable nombre en question.

Exemple pour le score :

```
char mscore[10];
sprintf(mscore, "%i", score);
drawBitmapText(mscore,-0.80f, 0.93f,0.0f, false);
```

## Librairie externe

Utilisation de SOIL librairie graphique qui permet entre autres d'afficher des png et d'utiliser la transparence. En combinant avec GLUT, on va pouvoir afficher une texture avec de la transparence sur un polygone.  Il y a un pré-chargement de toutes les textures avant l'affichage de jeu ce qui va accélérer l'affichage du jeu. En effet, à chaque rafraîchissement de l'écran, au lieu de recharger toutes les images il va restituer les images en mémoire et appliquer la texture sur le polygone qui aura éventuellement bougé.

Pour installer SOIL, il suffit de taper en ligne de commande LINUX :

```
sudo apt-get install libsoil-dev
```

Après, il faut recopier le fichier SOIL.h dans le projet et bien faire un include dans celui-ci. Ici, j'ai fait un include dans readimage.c. Il y a deux méthodes dans readimage.c **soilloadtextopaque**() qui est une méthode pour le chargement d'image opaque comme l'arrière-plan ou alors **soilloadtext**() qui permet le chargement d'une image avec de l'alpha comme un png.

# Conclusion

La phase d'analyse du projet est importante pour pouvoir mieux appréhender les difficultés de conception plus tard. Ici, on imagine une ossature qui sera adéquate pour le jeu. Il sera clair que des fonctionnalités se rajouteront mais le squelette de base est prêt et permettra de voir plus clair lors de l'implémentation du jeu. Je suis vraiment content de pouvoir apprendre la libraire graphique GLUT basé sur OPENGL ainsi que d'aborder les notions de programmation vues au cours (chargement de fichier, gestion de la mémoire, gestion de structure, gestion des tableaux, gestion des fonctions etc…) ainsi que la gestion du graphisme.

> ***" Les grandes réalisations sont toujours précédées par des grandes pensées"***
>
> *Steve Jobs*

# Bibliographie


- Site Web UNamur : https://www.unamur.be (Logo)
- Citation de Bill Gates : http://citation-celebre.leparisien.fr/auteur/bill-gates
- Citation de Steve Jobs : https://citations.ouest-france.fr/citations-steve-jobs-3618.html
- Certaines référence indirectement au cours sur Pacman : https://webcampus.unamur.be/pluginfile.php/153977/mod_resource/content/1/pacmanT pdf
- Certaines référence indirectement au cours sur OpenGL : https://webcampus.unamur.be/pluginfile.php/105581/mod_resource/content/3/openGL-presentation.pdf
- Référence sur OpenGL : http://www3.ntu.edu.sg/home/ehchua/programming/opengl/cg_introduction.html