DOCUMENT D'ARCHITECTURE

Introduction

Notre jeu est un jeu de plates-formes mettant en scène un aventurier qui en voulant explorer une vieille tour remplie de mythe et de légende, fini par réveiller un fantôme qui reposait au dernier étage.
Pour rester en vie notre héros doit redescendre tous les étages de cette tour remplis de pièges pour s'en évader, tout en évitant que le fantôme mette fin à ses jours !
Le code est composé de plusieurs parties.
En plus du « main », il y a les fichiers suivants : deplacement, fantome, menu, niveau ainsi que le header.
Ce document a pour but d'explorer ces différents fichiers et d'expliquer leurs contenus.

Création et affichage des niveaux :

Comme dit précédemment, le jeu se déroule dans une tour. Chaque étage de la tour représente un niveau représenté par la structure « block » dans le code.

Un block est composé d'un tableau de « forme », d'un tableau de « pique » ainsi que des coordonnées pour la sortie du niveau et de deux variables référant au nombre d'éléments dans les deux tableaux.

Dans le code du fichier « main.c » on utilise un tableau de « block » afin de pouvoir accéder et passer d'un niveau à l'autre grâce à une variable appelée « index_niveau ».

Une structure « block » est donc composée de plusieurs structures « forme ».

Celles-ci sont composées de variables (Xpt1, Ypt1, Xpt2, Ypt2) permettant de renseigner les coordonnées de deux points permettant de construire un rectangle ou un carré.

La structure contient aussi une image, qui permet de donner un sprite de texture à la forme, mais aussi de calculer les coordonnées du deuxième point pour l'initialiser. Une « forme » est donc initialisée de la manière suivante grâce à la fonction « initialise_forme » écrite dans « niveau.c ».

```
forme initialise_forme(float FacteurXpt1, float FacteurYpt1,  char *chemin_image){
        forme a;
        a.image_terrain = lisBMPRGB(chemin_image);
        a.Xpt1 = largeurFenetre()*FacteurXpt1;
        a.Ypt1 = hauteurFenetre()*FacteurYpt1;
        a.Xpt2 = a.Xpt1 + a.image_terrain->largeurImage;
        a.Ypt2 = a.Ypt1 + a.image_terrain->hauteurImage;

        return a;
}
```

La structure pique est composée de variable permettant d'initialiser trois points permettant de construire un triangle isocèle représentant un pique.

Le troisième point est calculé grâce aux coordonnées des deux autres dans la fonction « initialise_pique ».

La création de niveaux doit être hardcoder par des fonctions qui respectent un certain schéma.

Les fonctions sont appelées « initialise_block » suivie du numéro du niveaux qu'elles représentent.

Par exemple le niveau 3 est crée par une fonction qui s'appelle « initialise_block3 ».

Toutes les fonctions « initialise_block » sont écrites de la même manière :

- On commence par créer des n formes grâce à la fonction « initialise_forme ».
- On renseigne le nombre de formes que contient notre niveau
- On initialise les tableaux de forme avec les formes créées.
- On recommence la même chose avec les piques.
- On initialise la sortie du niveau.

Les fonctions « initialise_block » sont appelées dans le fichier « main.c » lors de la création du tableau de block.

L'affichage des niveaux :

Pour afficher les niveaux, on utilise deux boucles « for » qui boucle sur les tableaux de forme et de pique du niveau actuel :

```
for(int i=0; i<niveau[index_niveau].sizeMap; i++){
```

```
for(int i=0; i<niveau[index_niveau].nbPieges; i++){
```

Lorsque le joueur termine le niveau, index_niveau augmente, et du coup on boucle sur les tableaux du niveau suivant.

Dans ces boucles on va utiliser les fonctions « rectangle » et triangle afin d'afficher les formes permettant de dessiner notre niveau, et par-dessus les rectangles dessinés, on vient coller les images contenues dans la structure forme avec la fonction « ecrisImage ».

```
//affiche le terrain
couleurCourante(255, 0, 0);
for(int i=0; i<niveau[index_niveau].sizeMap; i++){
        rectangle(niveau[index_niveau].map[i].Xpt1, niveau[index_niveau].map[i].Ypt1, niveau[index_niveau].map[i].Xpt2, niveau[index_niveau].map[i].Ypt2);
        if(niveau[index_niveau].map[i].image_terrain != NULL) ecrisImage(niveau[index_niveau].map[i].Xpt1, niveau[index_niveau].map[i].Ypt1, niveau[index_niveau].map[i].image_terrai
}

//afiche les pieges
couleurCourante(128, 128, 128);
for(int i=0; i<niveau[index_niveau].nbPieges; i++){
        triangle(niveau[index_niveau].pieges[i].xCoin1, niveau[index_niveau].pieges[i].yCoin1, niveau[index_niveau].pieges[i].xCoin2, niveau[index_niveau].pieges[i].yCoin2, niveau[i
}
```

<u>Structure Joueur et déplacements :</u>

Ce jeu n'aurait aucun sens sans la possibilité d'incarner un personnage.

Pour cela on utilise une structure « joueur » possédant des variables permettant la construction de deux points, ce qui nous permettra d'afficher un joueur avec ces coordonnées.

La structure contient aussi deux sprites : joueur_idle et joueur_fall qui sont affiché quand le joueur est au sol ou lorsqu'il tome.

Les coordonnées du deuxième point sont calculées et initialisées grâce aux dimensions des sprites.

Bien évidemment il faut pouvoir déplacer le joueur, et pour cela on utilise les fonctions « moveLateral » et « moveHorizontal » qui comme leurs noms l'indique permettent de se déplacer verticalement et horizontalement.

Comment sont-elles utilisées ?

Dans le fichier « main.c » on tente de simuler les fonctions rattachéees à « Input » de l'API d'Unity : Unity - Scripting API: Input (unity3d.com)

Enfaite lorsque l'utilisateur appuie sur une des touches directionnelles (q, d, espace) leurs conditions respectives (condition_haut, condition_gauche, condition_droite) passent à 1, l'équivalent de l'état « True ».

Dans ce cas, dans le « case temporisation » on vient appeler les fonctions permettant de se déplacer.

Et si le joueur rencontre un obstacle ?

Si le joueur peut traverser à travers les murs ou le sol, le jeu n'a plus vraiment d'intérêt.

Donc dans le « case temporisation » avant d'utiliser les fonctions « moveVertical » et « moveHorizontal » on appelle les fonctions « bloque_direction_haut/bas/droite/gauche ».

Ces fonctions parcourent toutes les « forme » d'un niveau, et comparent leurs coordonnées à celles du joueur additionnées à une vitesse. Si les deux entités sont en contact alors on renvoie la valeur 1 et le joueur ne peut plus avancer.

Bien évidemment il n'y a pas forcément que des « forme » dans le niveau mais aussi des piques, et il faut donc pouvoir perdre contre ceux-ci. Pour cela on utilise la fonction « perdre_piege » qui fonction un peu comme les fonctions « bloque_direction ».

Si le joueur rentre en contact avec le piège, la fonction assigne 1 à la variable « condition_perdre » dans le fichier main.

Ainsi le joueur ne peut plus jouer et un menu s'affiche.

Le Fantôme :

Le jeu ne possède pas d'IA contre laquelle nous pouvons jouer, mais il y a quand même un ennemi.

La structure « fantome » sert à compliquer un peu le jeu, c'est un cercle dont les mouvements extrêmement sommaires sont calculés dans le fichier « fantome.c » par la fonction « mouvement_fantome ».

En début de partie le fantôme est initialisé pour apparaitre aléatoirement dans le niveau.

Bien évidemment on peut perdre si le joueur vient à toucher le fantôme, et cela grâce à la fonction « perdu » qui renvoie la valeur 1 si le joueur rentre en contact avec le fantôme.

Sauvegarde :

Pour éviter de tout recommencer à chaque fois qu'on lance le jeu, on a mis en place un système de sauvegarde.

Les fonctions de sauvegarde permettent donc de sauvegarder et charger une structure (parametre) contenant des informations sur le niveau dans lequel on se situe et sur le fantome.

La sauvegarde se fait automatiquement lorsque l'on quitte le jeu, et le joueur à le choix de la charger où non avant chaque début de partie.

Les menus :

Pour permettre de quitter, lancer une partie ou encore en charger une il nous fallait mettre en place des menus.

Ce sont des petites fonctions simples d'affichage. Lorsque le joueur arrive dans le jeu une variable « menu » est initiée à la valeur 0 et tant qu'elle est dans cet état, le jeu ne commence pas mais le menu principal est affiché.

Si dans le jeu, le joueur vient à appuyer sur « w » alors c'est la variable « condition_quitter » qui passe à 1 et dans ce cas un menu pour quitter s'affiche. Si le joueur change d'avis il peut réappuyer sur « w » pour sortir du menu.

Et enfin si le joueur vient à mourir, c'est la variable « condition_perdre » qui passe à la valeur 1. Ainsi le joueur a le choix entre quitter ou recommencer la partie.

L'utilisation des menus est possible grâce aux « case clavier » et « case boutonSouris ». Si la variable correspondante au menu est dans un état permettant d'afficher celui-ci et que le joueur clique au bon endroit, alors l'action écrite sur le menu sera réalisée.