Guillaume TREM
N° Étudiant : 17006514

Christophe LAFARGUE
N° Étudiant : 16006087

TRIFORCE

Rapport de stage

Fin de licence

Une approche de l'univers Zelda enrichi de jeux divers avec OpenGL

Maître de stage : Igor Stéphan

Année 2019/2020

L3 Informatique

Le projet

La licence d'informatique nous a permis d'aborder l'univers de la programmation au travers de multiples langages. La perspective de pouvoir créer une fonction avec des lignes de codes paraît magique et après ces trois années de licence, nous avions l'ambition de créer un véritable jeu.
La période de stage est toute trouvée pour pouvoir nous lancer dans un univers et imaginer à partir de jeux que nous connaissons bien, notre propre projet.
Nous avons donc choisi de piocher dans le jeu vidéo « The Legend of Zelda® ». Le joueur incarne donc le personnage de Link et peut se déplacer dans une zone géographique simple. Le choix initial se portait sur le passage du sanctuaire de Netzu'Yoma dans le jeu « The Legend of Zelda : Breath of the Wild » où Link doit esquiver des rochers qui descendent d'une cascade. Le projet avançant, nous avons décidé d'importer une multitude de jeux et de faire de notre application un regroupement de six mini-jeux.

Sanctuaire de NETZU'YOMA dans The Legend of Zelda : Breath of the Wild

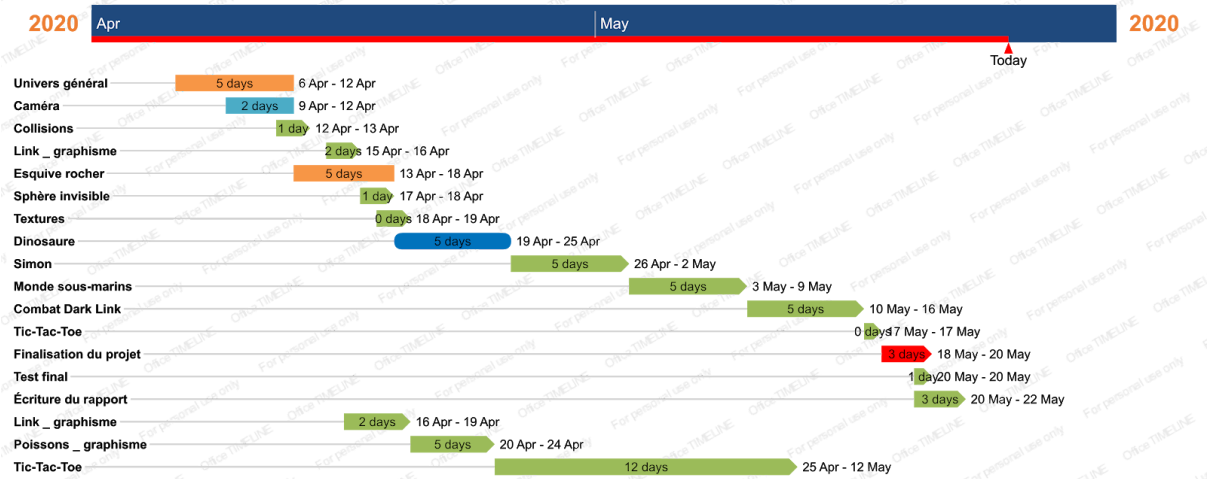
Table des matières

Temps de réalisation

Planning et répartition des tâches

Tout d'abord, Guillaume était seul lors des dix premiers jours du projet. Ainsi, l'objectif du projet était de modéliser un village et seulement le mini-jeu des rochers qui tombent de la falaise.
Ensuite, Christophe a rejoint le projet. Par conséquent, les objectifs et les capacités de programmation du projet furent grandement décuplés.
Étant donné que nous avions suivi tous les deux le cours de Synthèse d'Image, nous étions aptes à repousser nos objectifs et de pousser plus loin le projet initial. C'est ainsi que nous avions choisi de faire six différents mini-jeux en plus du village.

Tâches	Responsable
Modélisation Univers général	Guillaume
Caméra	Guillaume
Collisions	Guillaume
Modélisation Link	Christophe
Fonctionnement Link	Guillaume
Rochers qui tombent	Guillaume
Sphère entourant l'univers	Guillaume
Textures	Christophe
Dinosaures	Guillaume
Simon	Guillaume
Modélisation animaux marins	Christophe
Monde marin	Guillaume
Combat Dark Link	Guillaume
Tic-Tac-Toe	Christophe


2020
Apr
May
2020
Today
Univers général 5 days 6 Apr - 12 Apr
Caméra 2 days 9 Apr - 12 Apr
Collisions 1 day 12 Apr - 13 Apr
Link _ graphisme 2 days 15 Apr - 16 Apr
Esquive rocher 5 days 13 Apr - 18 Apr
Sphère invisible 1 day 17 Apr - 18 Apr
Textures 0 days 18 Apr - 19 Apr
Dinosaure 5 days 19 Apr - 25 Apr
Simon 5 days 26 Apr - 2 May
Monde sous-marins 5 days 3 May - 9 May
Combat Dark Link 5 days 10 May - 16 May
Tic-Tac-Toe 0 days 17 May - 17 May
Finalisation du projet 3 days 18 May - 20 May
Test final 1 day 20 May - 20 May
Écriture du rapport 3 days 20 May - 22 May
Link _ graphisme 2 days 16 Apr - 19 Apr
Poissons _ graphisme 5 days 20 Apr - 24 Apr
Tic-Tac-Toe 12 days 25 Apr - 12 May

Outils choisis

Nous avions tous deux suivi le cours d'introduction à la synthèse d'image qui nous a donné des bases conséquentes pour la réalisation de notre application.
Afin de réaliser ce projet nous avons en effet opté pour l'API OpenGL. Elle fournit beaucoup de fonctions permettant la mise en œuvre d'une application graphique et associée à la librairie GLU qui permet des interactions avec les entrées et sorties système grâce à certaines fonctions comme `glutKeyboardFunc` (pour les touches du clavier), `glutMouseFunc` (pour les boutons de la souris), `glutMotionFunc` (pour les mouvements de la souris avec un bouton pressé ou plus), `glutPassiveMotionFunc` (pour les mouvements de la souris sans aucun bouton pressé) que nous avons principalement utilisé, nous avions là deux outils qui suffiraient à notre projet.
Nous avons aussi eut besoin de la librairie GLU pour faciliter le développement d'objets, notamment pour l'utilisation de quadriques (cônes, cylindre, sphère).
En utilisant cette technologie, nous avons donc utilisé le langage C. Notre apprentissage du C++ durant la licence a suffit pour cela.

Architecture du logiciel

Nous avons choisi de reprendre l'architecture proposée par M. Igor Stéphan lors de nos TD/TP. Elle possède de nombreux fichiers qui nous ont été d'une grande aide pour commencer notre projet. Ces documents nous ont permis d'avoir une base déjà structurée pour démarrer avec un recueil fonctionnel et donc de commencer à coder notre monde visuel rapidement.
Voici la structure de l'application, décrite par notre professeur dans son cours[1].

main.c	>>	fichier principal contenant la majeure partie de la modélisation
*init.**	>>	initialisation d'OpenGL et de la fenêtre
*VM_init.**	>>	initialisation des matrices de projection et de modélisation
*switch_blend.**	>>	gestion de la transparence
*switch_light.**	>>	gestion de l'éclairage de la scène
*lumiere.**	>>	propriétés de la lumière
*axes.**	>>	tracé des axes selon l'état initial
touches.h	>>	déclaration des touches du clavier
*actions.**	>>	interaction avec les entrées clavier et souris
*opmath.**	>>	gestion des transformations géométriques
*ppm.**	>>	gestion de la lecture du format .ppm

[1] Lien vers le TP1 explicatif de la structure :
http://www.info.univ-angers.fr/pub/stephan/L3INFO/IMAGES_DE_SYNTHESE/TD_et_TP/TD_et_TP_1/td_et_tp_1.html

main.c

Le main.c est le fichier où se trouve toute la modélisation ainsi que le fonctionnement de notre projet.

Lorsque le joueur finit un mini-jeu, il est aléatoirement téléporté vers l'île du départ.

Plusieurs fonctions de création d'objets modélisés y sont décrites afin de faciliter et de factoriser le code. Se trouve par exemple les fonctions `void creer_fleur(float x, float y, float z)` ou bien `void creer_abre_partout()` qui sont bien évidemment des fonctions qui permettent de créer des fleurs à telles coordonnées ou bien de créer des arbres partout dans la scène.

```
void creer_fleur(float x, float y, float z)
{
        glPushMatrix(); //fleur intégrale
        {
                glTranslatef(x,y,z);

                glPushMatrix(); //fleur
                {
                        glEnable(GL_DEPTH_TEST);
                        glTranslatef(20,0.5,-20);
                        glRotatef(120,0,1,0);
                        glRotatef(-20,1,0,0);
                        glScalef(5,5,5);
                        glColor4f(1,0.71, 0.75, 0.4);
                        GLUquadric* cone = gluNewQuadric();

                    gluCylinder(cone,0.5,0,1.6,20,1);
                gluDeleteQuadric(cone);
                }
                glPopMatrix();

                glPushMatrix(); //tige
                {
                        glTranslatef(26,-9.5,-23.25);
                        glRotatef(90,0,0,1);
                        glColor3f(0.19,0.8,0.19);
                        glScalef(25,1,1);
                        glutSolidSphere(0.5,25,25);
                        glEnd();
                }
                glPopMatrix();

                glPushMatrix(); //feuille droite
                {
                        glTranslatef(24,-6,-23.25);
                        glRotatef(-135,0,0,1);
                        glColor3f(0,0.51,0);
                        glScalef(1,2,2);
                        glutSolidSphere(1, 25, 25);
                        glEnd();
                }
                glPopMatrix();

                glPushMatrix(); //feuille gauche
```

```
                {
                        glTranslatef(28,-12,-23.25);
                        glRotatef(135,0,0,1);
                        glColor3f(0,0.51,0);
                        glScalef(1,2,2);
                        glutSolidSphere(1, 25, 25);
                        glEnd();
                }
                glPopMatrix();

                glEnd();
        }
        glPopMatrix();
}
```

Enfin, une dernière fonction `bool Collision(struct AABB3D box1,struct AABB3D box2)` clôt cette partie du main.c. Cette fonction est l'une des plus importante du fait qu'elle permet grâce à des structures

```
struct AABB3D {
  float x,y,z;
  float w,h,d;
};
```

où l'on doit préciser les dimension de l'objet ainsi que ses positions, de vérifier la collision entre deux objets par leur "hitbox" (boite de collision qui entoure l'objet). Pour vérifier cela, elle compare les positions des sommets des cubes entourant les objets et vérifie si un des sommets d'un objet dépasse en terme de position l'autre sommet. Cette fonction doit être appelée sur tous les objets devant avoir une collision avec un objet en particulier.

```
bool Collision(struct AABB3D box1,struct AABB3D box2)
{
   if( (box2.x >= box1.x + box1.w)      // trop à droite
    || (box2.x + box2.w <= box1.x)      // trop à gauche
    || (box2.y >= box1.y + box1.h)      // trop en bas
    || (box2.y + box2.h <= box1.y)      // trop en haut
    || (box2.z >= box1.z + box1.d)      // trop derrière
    || (box2.z + box2.d <= box1.z))     // trop devant
          return false;
   else
          return true;
}
```

```
glPushMatrix(); //rocher milieux
                {

glTranslatef(position_x_rocher,position_y_rocher,position_z_rocher);
```

```
                        box_rocher_milieu.x = position_x_rocher;
                        box_rocher_milieu.y = position_y_rocher;
                        box_rocher_milieu.z = position_z_rocher;

                        box_rocher_milieu.w = 10;
                        box_rocher_milieu.h = 10;
                        box_rocher_milieu.d = 10;

                        if(Collision(box_rocher_milieu, box_personnage)
&& position_y <= 1420)
                        {
                              position_x = 0;
                              position_y = 2;
                              position_z = 20;
                        }
...
```

Ensuite, la fonction `GLvoid Modelisation()` définit toutes les modélisations du projet. Elles est appelée en boucle lors de l'exécution du projet. Elle dessine et définit le fonctionnement de chaque objet. Chaque objet est encadré des fonctions `GLPushMatrix()` et de `GLPopMatrix()` afin de faire entrer et sortir de la pile leur modélisation ainsi que toute les fonctions de translations, rotations, etc qui vont avec.

Link

Nous avons choisi de prendre Link comme sujet de notre projet.

Personnage de face

Personnage de côté

Lors de ses déplacements, ses jambes et ses bras sont animés d'un mouvement de marche. Il peut attaquer en donnant des coups d'épée ou bien se protéger en levant son bouclier.

Épée de côté

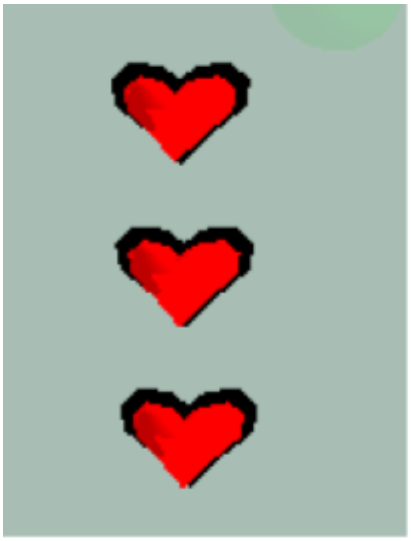
Épée d'en haut

Au-dessus de sa tête se trouvent six boules transparentes. Elles deviennent une à une opaque après chaque réussite d'un mini-jeu.

Boule au-dessus avant réussite d'un mini-jeu

Boule au-dessus après réussite d'un mini-jeu

Ensuite dans le mini-jeu de combat contre Dark Link, trois cœurs sont attribués au sujet. Ils sont inspirés des jeux "The Legend of Zelda".



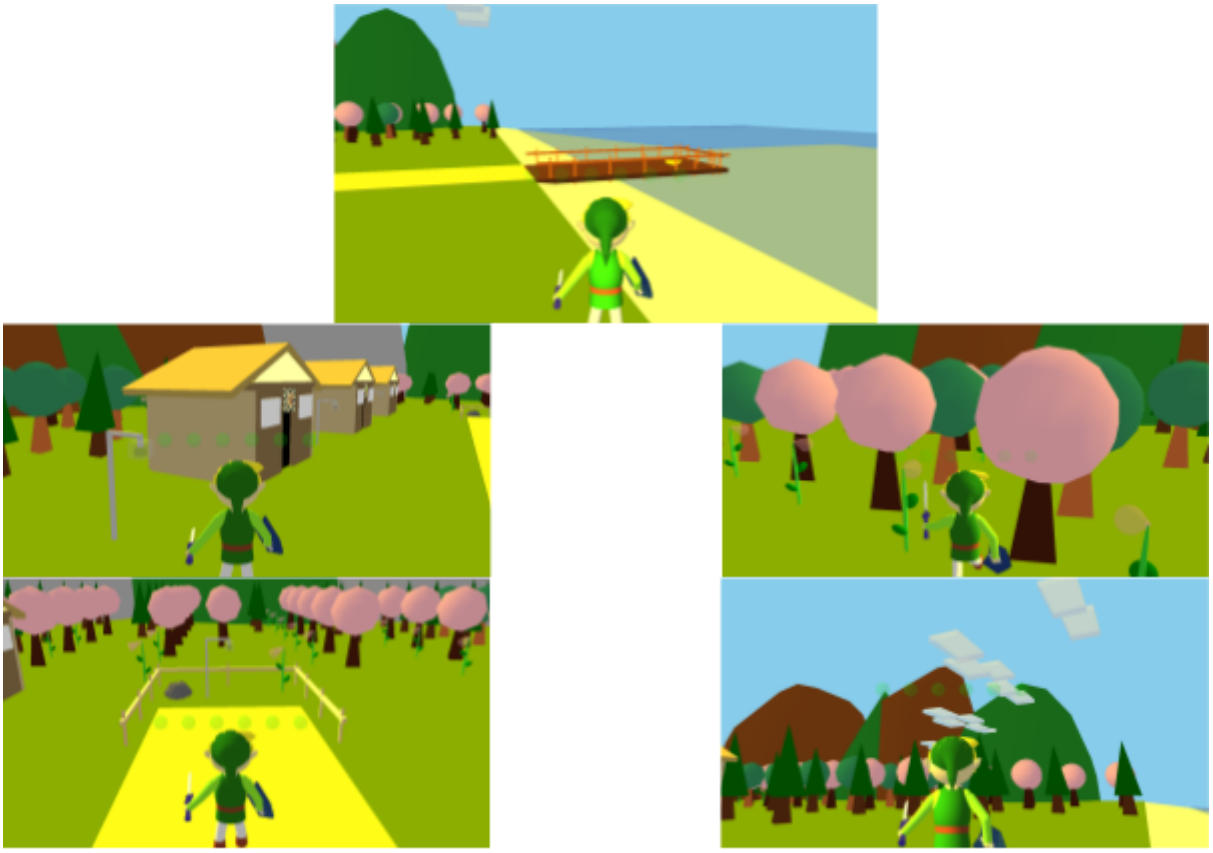
Coeur dans le projet

Coeur dans un jeu The Legend of Zelda

L'île du départ

S'ensuit la modélisation de l'île. Elle est composé de forêts de différents arbres et de fleurs, de six maisons regroupants les mini-jeux, d'une plage avec sa mer, d'un belvédère, de quelques barrières, de lampadaires, d'un petit rocher et de quelques collines entourant le village. Elle est surplombée de nuages se déplaçant aléatoirement.

Le joueur ne peut pas se déplacer en dehors de l'île, une sphère invisible l'entourant lui bloque le passage.



Éléments dans l'île

Enfin, la couleur du ciel change en fonction de l'astre (soleil ou lune).

Couleur des astres

Différentes lumières ont été implantées dans le projet. Premièrement, une lumière ambiante surplombe la scène. De ce fait, la scène se trouve éclairée que ce soit de nuit ou de jour comme si la lune ou le soleil éclairait la scène naturellement.

Deuxièmement, une lumière directionnelle provient du soleil vers la scène.

Lumière directionnelle provenant du soleil

Enfin troisièmement, plusieurs lumières positionnelles se trouvent dans le projet. Elles concernent les lampadaires se trouvant sur les différentes îles où se trouve les mini-jeux et sur l'île initiale.

Lumière positionnelle provenant des lampadaires

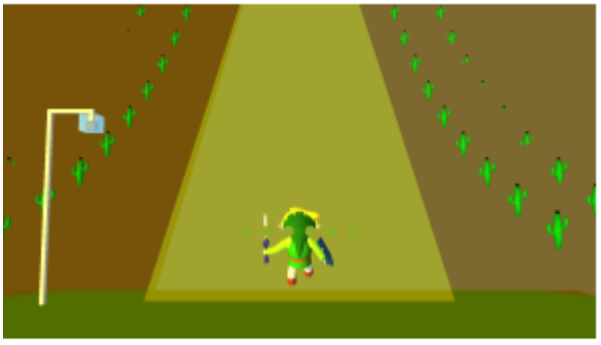
L'île des rochers

Ce mini-jeu se compose de deux îles : la première est le début de la falaise qui est l'endroit où l'on arrive une fois la porte de la maison du mini-jeu traversée, la seconde île se trouve à la fin de la pente et fait office d'arrivée. Cette dernière contient une maison où le personnage doit entrer afin de revenir sur l'île du départ et obtenir une boule.

Sur la falaise se trouve trois rochers qui déboulent.

Sur les côtés de la falaise se trouve deux pentes de terre où des cactus sont disposés.

Île d'arrivée des rochers



Île du départ des rochers

Rochers tombant de la falaise

L'île du T-rex game

Le T-rex game se compose d'une longue île où défilent un oiseau et deux cactus de différentes tailles, le dinosaure et une maison pour revenir sur l'île au besoin.

L'oiseau est aléatoirement positionnée dans la verticale tel le vrai jeu sur Google Chrome. L'oiseau bouge sa tête, ses ailes et sa queue aléatoirement.

Le dinosaure peut sauter en appuyant sur la touche espace.

Dinosaure au sol

Dinosaure en l'air

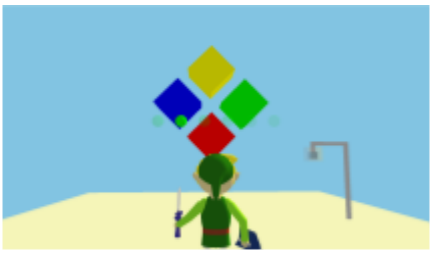
L'île du Simon

Ce mini-jeu se compose de quatre cubes, d'une maison et d'une île.

Ces quatres cubes sont par défaut transparents et deviennent opaques lorsqu'ils s'illuminent, c'est-à-dire, quand c'est à leur tour de briller. Il y a un court délais d'attente entre chaque illumination de cube afin de laisser le temps au joueur de mémoriser. L'utilisation de la fonction `void vMouse(int button, int state, int x, int y)` décrite plus tard dans action.h permet l'utilisation de la souris afin de cliquer sur les cubes et ainsi apporter une nouvelle dimension dans le gameplay en n'utilisant pas exclusivement le clavier pour jouer.

La mémorisation de l'ordre des cubes qui s'illuminent que ce soit au tour de l'IA ou de l'utilisateur se fait par deux tableaux qui enregistrent le cube illuminé par un `char ('b','r','j','v')`.

Afin de savoir où l'utilisateur clique, les positions de chaque sommet des cubes ont été mémorisées afin de créer des intervalles de positions.



Le jeu du Simon lorsque le joueur gagne

Le jeu du Simon quand un des cubes doivent être cliqué

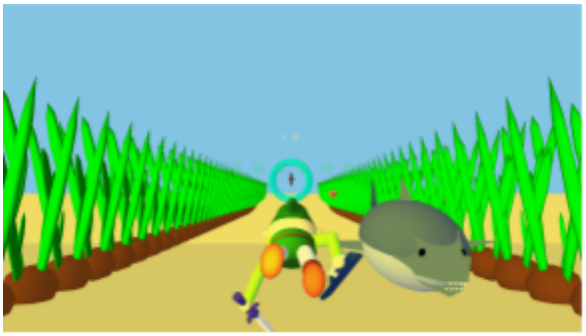
Le jeu du Simon quand un des cubes est cliqué

Le monde aquatique

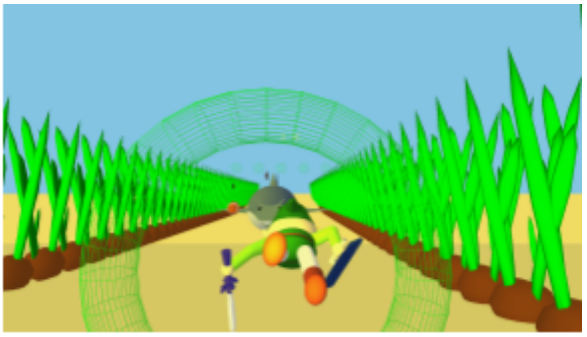
Ce mini-jeu se trouve sur l'île de départ mais dans l'océan.
Il se compose donc d'une mer, de sable, d'algues, d'une bouée et de quatre animaux marins qui sont : un requin, un turbot, un poisson clown et un zancle.

Lorsque le joueur passe dans la bouée, celle ci se colore différemment afin de comprendre que le passage dans la bouée est réussi. En effet, il faut que le joueur passe bien au centre de la bouée et pas seulement la toucher afin de valider le passage.

Un compteur de nombre de bouées validées en fonction du nombre de bouées à valider est affiché au-dessus de Link.



Monde aquatique lorsque le passage de la bouée n'est pas validé



Monde aquatique lorsque le passage de la bouée est validé

L'île du combat contre Dark Link

Ce mini-jeu se compose d'une grande île sous une fine couche d'eau. Cela fait référence aux combats contre Dark Link dans les différents opus de la saga du jeu "The Legend of Zelda". L'île se compose aussi d'une maison où le joueur peut entrer afin de revenir sur l'île du départ.

Île du combat contre Dark Link dans le projet

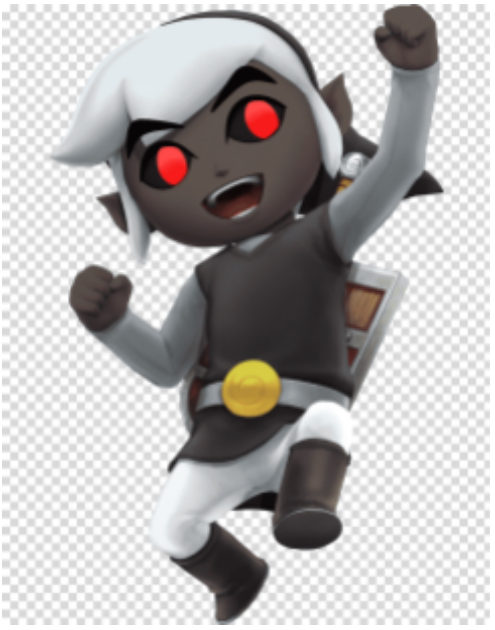
Salle de combat contre Dark Link dans "The Legend of Zelda : Ocarina of Time"

Dark Link se déplace aléatoirement de gauche à droite et ses jambes bougent ainsi de la même façon. Lorsque Dark Link ou le joueur active son bouclier, aucun dégât ne peut être subit.

Les couleurs de la barre de vie de Dark Link font référence aux différents jeux de combats que l'on trouve sur le marché.

Différentes barres de vie dans le projet ainsi que dans le jeu vidéo "Street Fighter V"

Les couleurs de Dark Link font référence au Dark Link de “The Legend of Zelda : Wind Waker”.



L'île du Tic-tac-toe

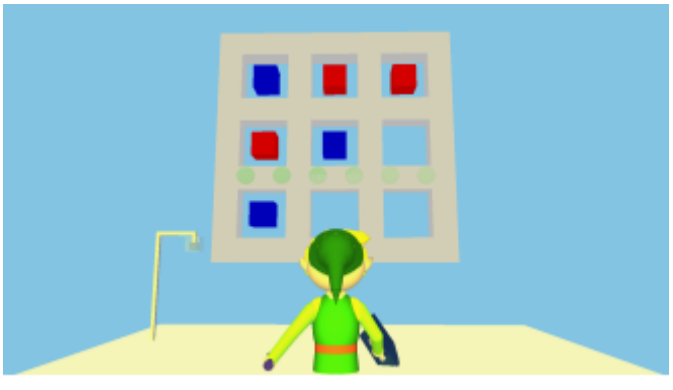
Ce mini-jeu est un morpion. Il se compose d'une île avec une maison, le quadrillage du morpion et des cubes du morpion.

L'IA choisit aléatoirement une case à remplir.

Le fonctionnement du morpion se fait entièrement par des conditions afin de vérifier la victoire ou non du joueur ou de l'IA. Ces conditions sont placées dans une fonction `int gagne()`.
Voici quelques tests :

```
//lignes
if((ttt_1 + ttt_2 + ttt_3 == 3) && (ttt_1_color == ttt_2_color) && (ttt_1_color ==
ttt_3_color)){
    if (ttt_tour%2 != 0){
        victoire = 1;
        ttt_1_color = ttt_2_color = ttt_3_color = 3;
    }
    if (ttt_tour%2 == 0){
        victoire = 2;
        ttt_1_color = ttt_2_color = ttt_3_color = 4;
    }
return victoire;
}
//diagonales
if((ttt_3 + ttt_5 + ttt_7 == 3) && (ttt_3_color == ttt_5_color) && (ttt_3_color ==
ttt_7_color)){
    if (ttt_tour%2 != 0){
        victoire = 1;
        ttt_3_color = ttt_5_color = ttt_7_color = 3;
    }
    if (ttt_tour%2 == 0){
        victoire = 2;
        ttt_3_color = ttt_5_color = ttt_7_color = 4;
    }
    return victoire;
}
```

La couleur des cubes du joueur est bleu et celle de l'IA est rouge.



Les couleurs du morpion

init.c

Ce fichier sert à initialiser la fenêtre qui apparaît, sa position, sa dimension et ses attributs de départ.

notre_init

La fonction principale du fichier `int notre_init(int argc, char** argv, void (*DrawGLScene)())` permet de faire toutes les initialisation. Dans l'ordre, elle initialise la librairie, sélectionne les buffers, créé la fenêtre, met en plein écran, enregistre la fonction de visualisation puis la fonction de modélisation puis la fonction de redimensionnement `GLvoid Redimensionne()`, puis celles pour récupérer les informations du clavier et de la souris en mode active, passive-active et passive.
Ensuite, elle permet la comparaison en profondeur des pixels.

```
{
	glutInit(&argc, argv); //initialisation de GLUT
	glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGBA | GLUT_DEPTH);
	glutCreateWindow("ZELDA PROJECT");
	glutFullScreen();
	glutDisplayFunc(*DrawGLScene);
	glutIdleFunc(*DrawGLScene);
	glutReshapeFunc(Redimensionne);
	glutKeyboardFunc(touche_pressee);
	glutMouseFunc(vMouse);
	glutMotionFunc(vMousemotion_middle);
	glutPassiveMotionFunc(vMousemotion);
	glClearColor(0,0,0,0);

	glEnable(GL_DEPTH_TEST);

	glutMainLoop();
	return 1;
}
```

redimensionne

Cette fonction s'occupe de la gestion de l'écran.
Elle gère la fenêtre, applique les opérations à la matrice de projection, charge la matrice identité, met en place la matrice de projection en perspective et applique les opérations à la matrice de la vue du modèle.

```
GLvoid Redimensionne(GLsizei Width, GLsizei Height)
{
	glViewport(0, 0, Width, Height);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	gluPerspective(45, (float) Width  /(float) Height, 0.1, 1500);
	glMatrixMode(GL_MODELVIEW);
```

Textures

Les textures ont besoin de lecteurs, qui sont ceux que notre professeur nous a proposé dans les TD/TP[2]. Les fichiers ppm.c et ppm.h permettent de lire un fichier au format ppm. Ici nous avons utilisé la texture Damier.ppm issue de ces mêmes sources. Nous n'avons pas réussi à utiliser une autre texture.

Ce code lit une texture, créé un tableau `texture[1]` génère un nom de texture et la lie avec une cible de texturation `GL_TEXTURE_2D`.
Ensuite il définit les paramètres de la texture avec `glTexParameteri` puis défini une image de texture à deux dimension avec `glTexImage2D`.
Finalement, effectue la texturation en 2D.

```
//Texture
glClearColor(0,0,0,0);
TEXTURE_STRUCT * txtstrct = readPpm("Damier.ppm");
GLuint texture[1];
glGenTextures(1,texture);
glBindTexture(GL_TEXTURE_2D,texture[0]);

glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_S,GL_CLAMP);
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);

glTexImage2D(GL_TEXTURE_2D,0,GL_RGB,
    txtstrct->width,txtstrct->height,
    0,GL_RGB,GL_UNSIGNED_BYTE,
    txtstrct->data);

glTexEnvf(GL_TEXTURE_ENV,GL_TEXTURE_ENV_MODE,GL_MODULATE);

glEnable(GL_TEXTURE_2D);
```

[2] http://www.info.univ-angers.fr/pub/stephan/L3INFO/IMAGES_DE_SYNTHESE/TD_et_TP/TD_et_TP_12/td_et_tp_12.html

VM_init.c

Le fichier VM_init.c gère le fonctionnement de la caméra et ainsi la position du personnage.
En effet, nous avons choisi que l'utilisateur pourra changer l'orientation de la caméra grâce au mouvement passif de la souris qui sera décrit plus tard dans le action.c et dans le init.c.
Étant donné que la caméra suit en permanence le personnage, sa position selon les axes x, y, z est plus ou moins la même. Par conséquent, un changement de position du personnage de Link entraînera un changement de position de la caméra.

Ainsi, l'utilisation du VM_init.c dans le projet aura pour but de gérer les changements des positions de Link lors des passages dans les portes des maisons principalement, en plus de gérer la position, la rotation et l'éloignement de la caméra par rapport au personnage.

La fonction `set_camera_3(float x, float y, float z)` est tout simplement un `GLTranslatef(-x, -y, -z)`.

Enfin, le VM_init.c gère le mode plan de vu dynamique qui correspond à l'autre point de vu de la caméra lorsqu'on appuie sur la touche de la molette de la souris et qui sera décrit plus tard dans le action.c.

```
void VM_init() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glMatrixMode( GL_MODELVIEW );
    glLoadIdentity();
    glTranslatef(0,0,-60);

    if(camera_middle == 1) {
        glRotatef(-yrot_middle, 0, 1, 1);
        glRotatef(xrot_middle, 0, 1, 1);
    }
    else {
        glRotatef(xrot, 1,0,0);
        glRotatef(yrot, 0, 1, 0);
    }

    if((position_z <= 55) && (position_z >= 42) && (position_x >= 100) && position_x <=
105) {
        position_x = 1000;
        position_z = 1039;
        position_y = 1020;
        xrot = -4.0;
        yrot = -180;
    }
    else if
    // [...] d'autres tests [...]
    else if (position_y > 1000 && position_y < 1425) {
        set_camera_3(position_x,position_y + 10,position_z -30);
    }
    else set_camera_3(position_x,position_y + 10,position_z);
}
```

action.c

Le fichier action.c décrit toute le fonctionnement des actions de l'utilisateur, que ce soit via le clavier ou la souris.

Plusieurs fonctions composent ce fichier, dont trois plus importantes :

void touche_pressee(unsigned char key, int x, int y)

Cette fonction décrit le travail des touches du clavier utiles au projet qui sont définies dans le touche.h. Cette fonction se compose d'un grand `switch case` qui vérifie quelle touche est pressée.

Nous ici un total de 18 touches du clavier qui sont décrites. Les plus importantes sont celle du mouvement du personnage (ZQSD) et ESCAPE (echap).

La gestion d'une sphère invisible entourant les îles bloquant les mouvements du personnage se fait grâce aux fonctions `void prediction_avancer(float x, float z, float sx, float sz)` et `void prediction_reculer(float x, float z, float sx, float sz)`.

La gestion du mouvement du personnage dans l'espace se fait par l'appel des fonctions `float avancer_x(float x)`, `float reculer_x(float x)`, `float avancer_z(float z)` et `float reculer_z(float z)`. Elles utilisent la trigonométrie afin de gérer le déplacement en fonction de l'orientation de la caméra.

void vMouse(int button, int state, int x, int y)

Cette fonction définit le travail de la souris lorsqu'une des touches de la souris est actionnée. Elle est exclusivement utile dans le jeu du Simon et pour les animations de Link.

Pour le jeu du Simon, la fonction enregistre avec l'appui du clic gauche de la souris la position de la souris selon x et y par rapport au coins supérieur gauche de l'écran ainsi que l'état du bouton (appuyé ou relâché). Ainsi, cela permet de savoir si l'utilisateur clique dans telle ou telle case du jeu du Simon.

Pour les animations de Link, cette fonction gère si l'utilisateur peut ou non attaquer ou se protéger grâce au clic gauche et droit respectivement, choses qui ne peut être fait simultanément.

Enfin, elle gère le choix du mode plan de vu dynamique en appuyant sur la molette et en bougeant la souris en faisant appel à la fonction `void vMousemotion_middle(int x, int y)`.

Différentes vue en caméra dynamique

void vMousemotion(int x, int y)

Cette fonction décrit le travail de la souris de façon passive. En effet, cette fonction s'appelle à chaque tour de boucle du programme car elle calcule la position de la souris sur l'écran en permanence.

Elle gère le mouvement de la caméra en fonction de la zone où elle se trouve et de son ancienne position.

Si elle se trouve sur l'un des bords de l'écran, la caméra fera une rotation vers ce bord de l'écran tant que la souris n'aura pas quitté ce dernier bord.

Si par exemple la souris se trouve à une position plus à gauche que sa position précédente, alors la caméra fera une rotation vers la gauche d'un pas. Et ainsi de suite pour les autres directions.

touche.h

Ce fichier définit tout utiles au bon fonctionnement du fichier action.c par conséquent, les touches du clavier.

Pour définir une touche, il suffit de "define" une variable avec le code ascii associé à la touche voulue.

```
#define TOUCHE_MIN_Z 122
#define TOUCHE_MAJ_Z 90

#define TOUCHE_MAJ_Q 81
#define TOUCHE_MIN_Q 113

#define TOUCHE_MAJ_S 83
#define TOUCHE_MIN_S 115

#define TOUCHE_MAJ_D 68
#define TOUCHE_MIN_D 100
```

Descriptif terminal

A l'issue des deux mois nous avons donc un jeu qui regroupe 6 jeux connus ou copies de jeux connus, que le joueur, en incarnant le personnage de Link, peut réaliser en entrant dans chaque maison.

Le monde se résume à une île qui fait penser au début du jeu de The Legend of Zelda : Wind Waker. Des collines entourent un village de six maisons situé en bord de mer. Une forêt protège le village. Lorsque le jeu commence, le joueur est situé au milieu de barrières franchissables à l'orée du bois. En appuyant sur les touches 'b' ou 'n' il peut respectivement passer du jour à la nuit ou allumer/éteindre les lampadaires.

Le village, de nuit.

Le village, de jour.

Au dessus du personnage, il y a constamment six boules transparentes. Le but du jeu est de réussir les six défis afin de récupérer les six boules dans le but de faire briller la triforce se trouvant sur le pont.

A gauche, de Link jusqu'à la plage, les maisons regroupent un jeu d'esquive de rochers  , un jeu aquatique de passage dans des cerceaux  et un tic-tac-toe  .

A droite, dans le même ordre, un jeu de sauts obstacles (T-rex game)  , un Simon  et un jeu de combat à l'épée contre Dark Link (combat contre soit même que l'on rencontre dans quasiment tous les opus de la saga de jeu The Legend of Zelda)  . Chaque maison possède au dessus de sa porte le logo du jeu qu'elle contient. Derrière le logo il y a un cadre texturé d'un damier.

Lorsque le joueur entre dans une maison, il est téléporté vers une île où le jeu démarre automatiquement. Il peut dès lors tenter de réussir le défi. S'il gagne, il est téléporté à son emplacement initial dans le village et l'une des boules est devenue opaque. Le choix de l'ordre des jeux n'a pas d'incidence. Une fois tous les jeux terminés, toutes les boules sont opaques et la triforce sur le pont possède une aura. En se déplaçant jusqu'à cette dernière, le joueur est téléporté sur une plateforme suspendue qui lui donne une vision d'ensemble sur le monde qui l'entoure où il peut apercevoir un message de félicitations.

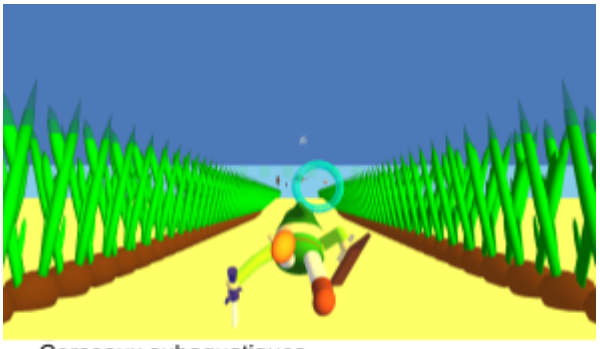
Esquive de rochers

Sur une pente raide, le joueur doit se déplacer pour monter en haut de la colline

avec “Z”. Il y a trois rampe imaginaire sur lesquelles sont susceptible de tomber des rochers. Le joueur doit gravir jusqu’au sommet sans se faire toucher par un rocher, pour les esquiver, il utilise les touches “Q” et “D”.
Une fois au sommet, il faut entrer dans la maison.

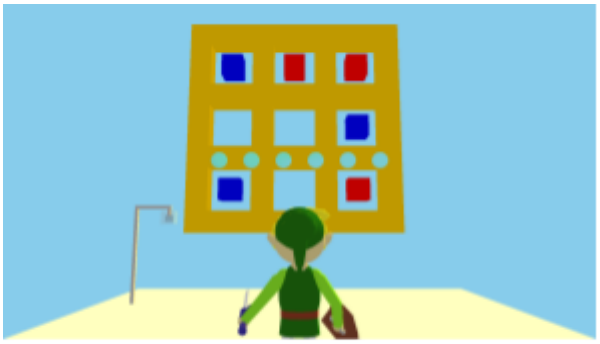
Cerceaux subaquatiques

Plongé sous l’eau, le joueur doit passer au milieu de six cerceaux qui apparaissent sur une large zone délimitée par des plantes marines. Il n’a à sa disposition que les touches “Q” et “D” pour se déplacer sur ses côtés. Si Link touche un poisson ou rate un cerceau, ses points sont décrémentés de un.



Cerceaux subaquatiques

Tic-tac-toe[3]



Tic-tac-toe

Le joueur peut jouer avec les chiffres du clavier numérique ou celles du clavier principal. L’ordre est le suivant :

7	8	9
4	5	6
1	2	3

Les règles sont inchangées, il faut aligner trois couleurs identiques sur une ligne, une colonne ou une diagonale.

[3] https://fr.wikipedia.org/wiki/Tic-tac-toe

T-Rex game[4]

Le fameux jeux que Google propose lors d'un problème de connexion[5]. Il faut faire sauter le T-Rex au dessus des obstacles. Si le T-Rex touche un cactus ou un oiseau, le joueur est téléporté à son emplacement initial avec le défi accompli. En effet nous n'avons pas réussi à créer une limite de sauts efficaces ou de distance parcourue.

T.-Rex game

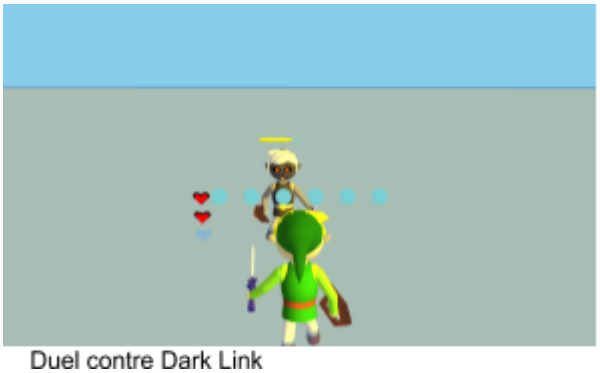
Simon[6]

Simon

Le but du jeu est de reproduire le schéma coloré que propose l'algorithme. Ce dernier allume l'un des cubes puis l'éteint. C'est alors le moment d'allumer la même lumière grâce à un clic long sur son choix. Puis l'algorithme reprend en ajoutant une lumière. Il faut attendre que la lumière soit éteinte pour essayer.
Le joueur gagne en répétant la suite jusqu'à cinq couleur correctement.

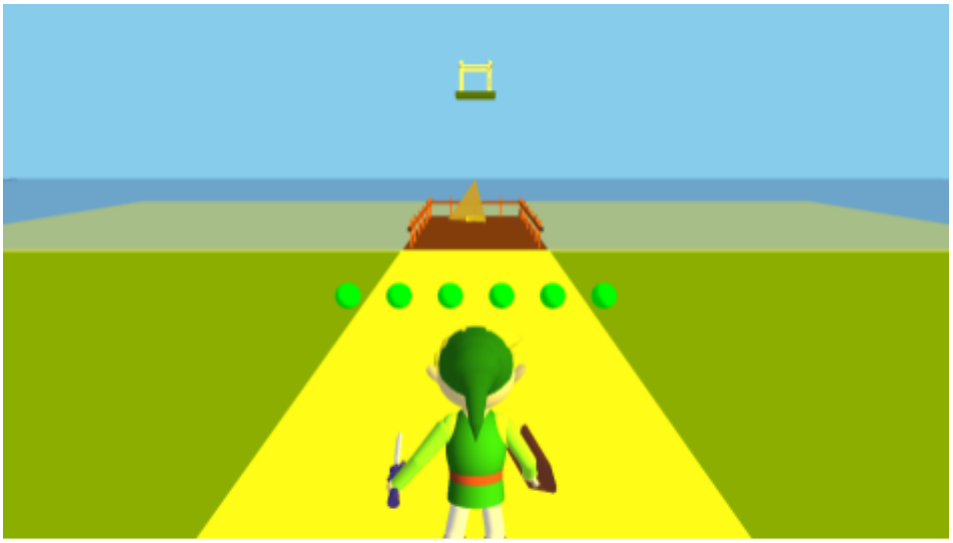
Dark Link

Le duel contre Dark Link est un combat avec épée et bouclier. Grâce au clic gauche de la souris, un coup d'épée est donnée. Le clic droit lève le bouclier pour se protéger.
Chaque coup donné ou reçu enlève un point de vie, représenté par des cœurs pour Link et par une barre de vie pour Dark Link.
S'il a toute sa vie, sa barre est verte, au ⅔ elle est orange à ⅓ elle devient rouge.
Lorsque le dernier coup est asséné, il meurt en s'élevant vers le ciel. Lorsque le joueur meurt, il revient sur l'île du départ afin de réessayer et Dark Link aura autant de vie que lorsque le joueur est mort.

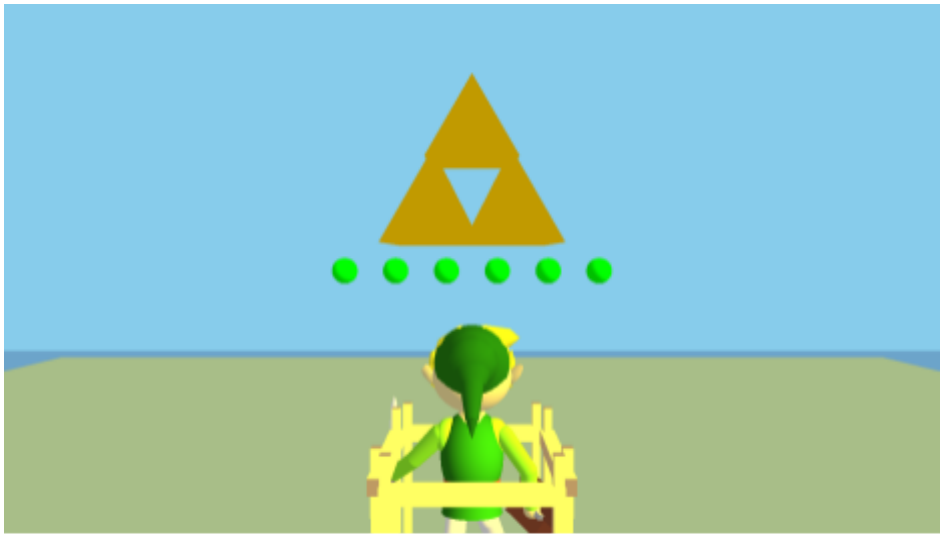


Duel contre Dark Link

[4] https://fr.wikipedia.org/wiki/T._Rex_Game
[5] Et qui est toujours disponible à cette adresse : chrome://dino/ !
[6] https://fr.wikipedia.org/wiki/Simon_(jeu)



L'aura de la triforce sur le pont



Une fois sur la plateforme, une grande triforce est visible en face

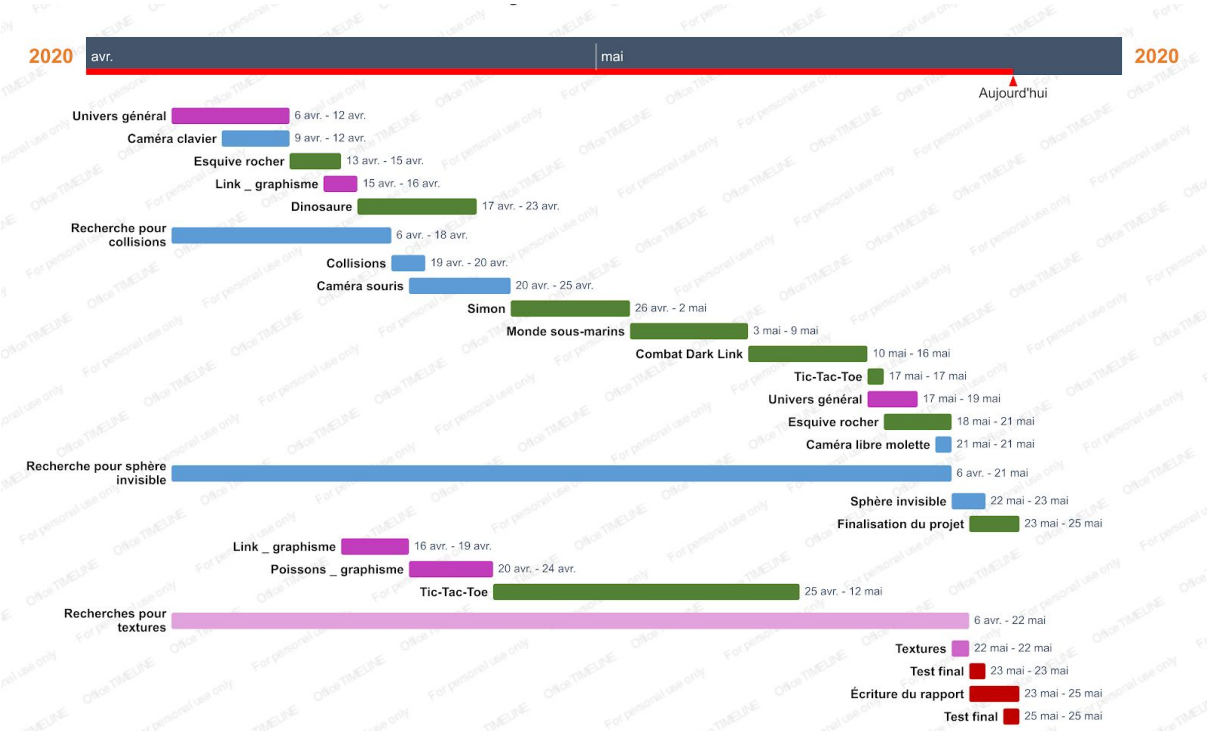
La vision globale du village, vu de la plateforme, avec les félicitations.

Problèmes et perspectives

Problèmes

Plusieurs difficultés se sont montrées face à nous durant la réalisation du projet.

Tout d'abord, le calendrier a plus ou moins été respecté. Nous considérons cela comme une réussite, car il n'a pas réellement été très impacté par des retards. Tous nos objectifs ont été plus ou moins accomplis. Bien sûr, il y a eu des tâches qui auraient pu être plus complètes, mais le manque de temps du projet ne nous a pas permis de les faire.



Tout d'abord, un premier bémol est de ne pas avoir réussi à texturer notre univers comme nous le désirions. Les outils et fonctions qui permettent de plaquer une texture sur un objet tel qu'une glutSolidSphere nous sont restés nébuleux. Malgré nos recherches, nous n'avons pas réussi à bien maîtriser cette partie d'OpenGL et de GLUT.

Ensuite, un deuxième problème a été celui des collisions. En effet, les collisions sont le coeur du projet et nous avions eu beaucoup de mal à les définir. Cela nous a pris plus de temps que prévu et nous a fait prendre un peu de retard, mais il n'a pas été si conséquent que cela.

Puis, une troisième difficulté s'est prononcée : la sphère entourant l'univers afin de bloquer les mouvements. En effet, nous essayons depuis le début du projet à développer cela. Cependant, n'y arrivant pas, nous avions choisi une solution alternative qui consistait à créer des intervalles de mouvement, mais cela empêchait Link de se déplacer de façon instinctive

dans l'univers, en plus d'être une méthode de développement plus longue et peu concise. Ainsi, grâce à l'aide de M. Stéphan, nous avons réussi à surmonter cette difficulté en faisant un calcul d'une distance par rapport à un point, solution trouvée seulement à la fin du projet.

Après, un problème au niveau du développement du mini-jeu des rochers qui tombent est survenu. Ne sachant pas faire les collisions au début du projet et n'ayant pas assez eu d'expérience dans la manipulation du personnage dans l'univers, il nous paraissait compliqué de coder entièrement le mini-jeu des rochers qui tombent dès le début du projet. C'est ainsi pourquoi la suite du développement du projet fut reporté à la fin du calendrier.

Puis, un autre problème, qui n'est pas en rapport avec le code est survenu peu avant les partiels à distance que nous avions eu le 28 et 29 avril 2020. En effet, Guillaume a perdu une journée entière de travail à cause d'un problème avec Sublime Text (éditeur de texte) le 26 avril 2020. Ce travail perdu concernait le développement quasiment entier du fonctionnement du monde aquatique. Après la réalisation de la perte de ces données, Guillaume perdu la motivation pour travailler sur le projet, ajouté à cela le travail de révision pour les partiels. Ainsi, il reprit le développement du projet seulement quatre jours plus tard, le 30 avril.

Améliorations et Perspectives

Avec un projet sur un temps plus long, nous aurions pu passer plus de temps sur certains détails pour peaufiner notre application. Nous aurions voulu parfaire les hitbox et les collisions pour que tous les objets soient réellement solides. Les jeux peuvent être bien améliorés :

- Le T.-Rex game devrait avoir un score à atteindre avant d'être réussi.
- Le Simon pourrait être plus rapide et nous pourrions ajouter des sons aux lumières avec OpenAL par exemple.
- Le Dark Link pourraient être plus poussé dans son combat, par exemple en détectant son ennemi et en visant.
- L'IA du tic-tac-toe peut facilement être améliorée en ajoutant des conditions qui lui permettraient de défendre une ligne prise par deux pions de la même couleur.
- Les poissons du jeu subaquatique pourraient posséder des mouvements de nage.
- Un monde aquatique plus fluide.
- Des déplacements animés dans le monde des rochers qui tombent
- L'ajout de roulades ou de sauts pour Link et Dark Link
- Plusieurs et différents coups d'épée lors d'attaques consécutives de Link et Dark Link

Conclusion

Ces deux mois nous ont permis de réaliser un beau projet en OpenGL. Le projet de base a été enrichi par de nouvelles idées car nous pensions cela réalisable dans ce laps de temps. Nous avons en effet terminé le jeu dans un état convenable pour ce que nous attendions. C'est-à-dire avoir un cheminement dans un univers ressemblant à celui de Zelda et pouvoir jouer à des jeux qui nous amusaient jusqu'à une réussite visible.
L'expérience que nous avons gagnée pendant ce temps de travail nous a permis de gagner en efficacité. La compréhension de la structure d'OpenGL s'est faite plus claire et l'utilisation de ses outils plus simple. Si l'utilisation de certains outils nous paraissaient flous ou de trop peu d'intérêt pendant les TP/TD comme par exemple l'utilisation des triangles pour Christophe, le besoin de travailler avec nous a permis de nous familiariser et de découvrir leurs avantages (cf gueule du requin).

Néanmoins, nous sommes satisfaits et heureux de ce que nous avons pu créer. Ce stage nous aura permis de nous rendre compte de la nature d'un projet de deux mois, des décisions à partager, du temps à réguler en essayant de ne pas trop se concentrer sur les détails excessivement minutieux et de la collaboration en groupe.

Annexes

Contrôles

[Général]

Touche ‘escape’ : *quitte l’application*
Touche ‘b’ ou ‘B’ : *place l’univers dans le jour ou la nuit*
Touche ‘n’ ou ‘N’ : *allume/éteint les lampadaires*
Touche ‘z’ ou ‘Z’ : *fait avancer Link*
Touche ‘s’ ou ‘S’ : *fait reculer Link*
Clic gauche : *donne un coup d’épée*
Clic droit : *lève/baisse le bouclier*

[Chute de rochers | Cerceaux subaquatiques]

Touche ‘q’ ou ‘Q’ : *déplacement sur la gauche*
Touche ‘d’ ou ‘D’ : *déplacement sur la droite*

[Tic-tac-toe]

Touche ‘0’ : *remet à zéro le morpion*
Touche ‘1’ à ‘9’ : *permet de jouer les cases*

Installations

Dans le dossier, le fichier *makefile* permet de faciliter la compilation et l’exécution du programme. Il suffit, dans un terminal, d’aller jusqu’à l’emplacement du dossier et de tapper la commande `make`. L’application se lance alors en plein écran.

Sitographie

Igor Stéphan. Introduction à la Synthèse d’Image. 2020
http://www.info.univ-angers.fr/pub/stephan/L3INFO/IMAGES_DE_SYNTHESE/Images_de_synthese.html