

BARREIRO Hugo
RIBEIRO Damien



Informatique Graphique pour la Science de Données :
Projet - des momies et des pyramides

Sommaire :

I) Répartition du travail (page 3)

II) Genèse et conception (page 4)

III) Explication de code (page 9)

IV) Erreurs, problèmes et difficultés (page 14)

V) Objectifs et apprentissage (page 17)

I) Répartition du travail

On a décidé de diviser le projet en 3 trois parties :

- Labyrinthe
- Désert + Pyramide
- Momie

Pour le labyrinthe :

Nous avons tous les deux participé à l'élaboration du labyrinthe.

Nous avons d'abord réalisé le TP sur le labyrinthe chacun de notre côté. Puis nous sommes partis de la version de Hugo (nous n'avons repris aucun élément de la correction donnée). Enfin, nous avons réglé/ajouté quelques détails.

Par exemple, Damien a rendu la taille de la map proportionnelle à la taille du labyrinthe (zoom/dézoom de la taille des blocs mais le contour reste de même taille). Hugo s'est occupé du changement de niveau. Et nous avons participé tous les deux à l'élaboration du téléporteur et des animations qui l'accompagnent.

D'autres détails ont été apportés par nous deux.

Pour le désert + pyramide (extérieur) :

C'est Hugo qui s'est entièrement chargé de la conception du désert et de la pyramide.

C'est-à-dire la mise en place du sable (texture, ondulation et dune), de la pyramide (texture, pavés, cube et triangle), de la lumière et des murs invisibles.

Pour la momie :

C'est Damien qui s'est entièrement chargé de la conception de la momie.

C'est-à-dire de la mise en place du design de la momie (forme, couleur, corps, etc...), et de ses mouvements (aléatoires).

II) Genèse et conception

Commandes :

Flèches directionnelles pour se déplacer (\uparrow pour avancer, \rightarrow pour tourner la caméra à droite, \leftarrow pour tourner la caméra à gauche). On peut appuyer à nouveau à chaque fin d'animation ou bien rester appuyé.

Contrôle (Ctrl) pour changer la vitesse de déplacement (appui simple). Mode fast par défaut.

Entrée (Enter) pour entrer dans le téléporteur (appui simple) ou pour rentrer dans le labyrinthe une fois sortie dans le désert (appui simple).

La souris permet de changer verticalement la direction du regard. (déplacer la souris de haut en bas, la caméra suit le mouvement).

Choix de conception :

Nous avons réalisé tout le contenu exigé : TP sur le labyrinthe, la pyramide (extérieur + sol + texture + couleur + changement d'atmosphère), la momie (forme + couleur variable + yeux + bras + mains + déplacements aléatoires), les textures (sur tous les éléments), les lumières (dehors, personnage et bouts de couloirs), la boussole et enfin le changement de niveau.

Néanmoins, nous n'avons pas superposé les différents niveaux de labyrinthes. En effet, nous avons décidé d'afficher uniquement le niveau courant. Cela permet d'obtenir de meilleures performances (il y a beaucoup moins d'éléments à afficher/charger). De plus, cela n'affecte pas l'expérience du joueur car il ne peut pas voir si les labyrinthes inférieurs/supérieurs sont affichés ou non, ainsi le gameplay n'est pas impacté négativement. Pour les mêmes arguments, nous n'affichons pas non plus la superposition des labyrinthes à l'extérieur de la pyramide.

Pour les téléporteurs, nous en avons placés deux : un à l'entrée et un à la sortie de chaque labyrinthe. Cela nous permet donc de remonter et de redescendre si besoin/envie (la progression de chaque labyrinthe est sauvegardée). La sortie du labyrinthe est placée de manière aléatoire du côté opposé de l'entrée lors de la création du labyrinthe. Dans le dernier labyrinthe, la sortie est toujours au même endroit tout en bas à droite du labyrinthe

La momie bloque le passage du joueur (et le joueur bloque le passage de la momie). Cela augmente la difficulté du jeu.

Il y a deux modes de déplacements : le mode fast et le mode slow (changement de mode avec Ctrl).

Notre jeu débute tout en haut de la pyramide (i.e. le plus petit labyrinthe en premier). Le but est donc de descendre et réussir à sortir de la pyramide (nous avons donc une difficulté croissante car les labyrinthes sont de plus en plus grands).

Explications supplémentaires :

On gère la sauvegarde des labyrinthes grâce à un tableau.

Le changement de niveau est géré lors du passage sur un téléporteur (+ appui sur Entrée). Selon le téléporteur, on augmente/diminue la taille courante du labyrinthe. Soit on affiche le labyrinthe s'il a déjà été créé, soit on en crée un nouveau et on le stocke.

Lorsque le joueur décide de se téléporter dans un nouveau labyrinthe alors une animation apparaît à l'écran. Le sens du téléporteur détermine la couleur de l'animation. (magenta lorsque l'on descend d'un étage et bleu cyan lorsque l'on monte d'un étage).

Image des animations lorsque le joueur se téléporte:



De plus, lorsque le joueur sort de la pyramide ou qu'il retourne à l'intérieur du labyrinthe, le joueur n'a plus le contrôle de la caméra qui se déplace automatiquement pendant quelques secondes.

Pour sortir du labyrinthe, le joueur doit simplement atteindre la sortie du dernier labyrinthe. Pour changer de labyrinthe ou pour retourner à l'intérieur de la pyramide, le joueur doit appuyer sur la touche entrer. Cela est indiqué à l'aide de texte que l'on affiche à l'écran et qui demande au joueur de "Press Enter to teleport" ou "Press Enter to go back inside". (Nous avons utilisé la police de base de Processing)

Image des différents messages de textes écrits:

Press Enter to teleport

Press Enter to teleport

Press Enter to go back inside

La pyramide est créée à l'aide d'un PShape pour la base et d'un PShape pour le toit. La base est composée de plusieurs pavés avec un cube au bout de chaque côté. Les textures sont appliquées différemment pour la surface haute de chaque cube pour que la texture soit continue avec les pavés pour les deux côtés du cube. Le toit est un PShape de triangles.

Le sable est une PShape constituée de pleins de petites plaques. On calcule la hauteur de chaque coin à chaque itération pour avoir des ondulations bien continues. Pour ne pas voir l'horizon, on entoure la pyramide de très grandes dunes (réalisée en ajustant la hauteur des dunes grâce à la fonction mathématique $x^2 + y^2$ centrée au centre du désert).

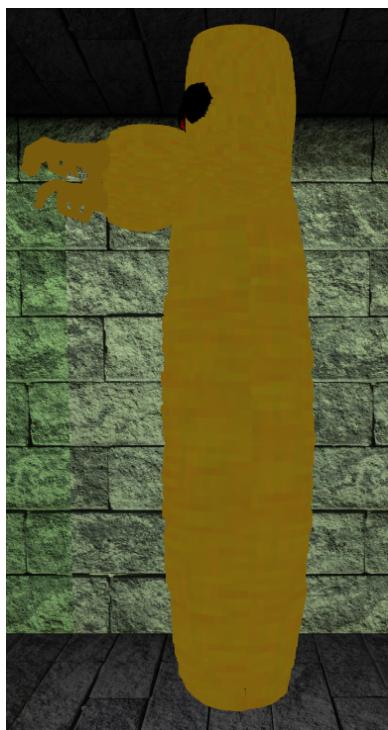
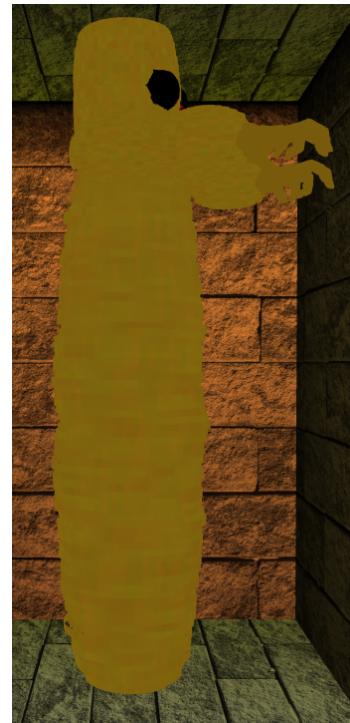
La lumière dans le désert part du coin inférieur droit et pointe sur la pyramide, ce qui permet de jouer avec les ombres sur les dunes (notamment du coin inférieur droit) et mettre en avant la pyramide qui est bien éclairée.

Pour ne pas rentrer dans la pyramide et les dunes, on a placé des "murs invisibles".

On peut re-reentrer dans le labyrinthe une fois sortie. Il y a une animation qui accompagne cette action.

La momie est une PShape constituée de plusieurs PShape. Le corps, la tête et les bras sont composées d'une multitude de PShape (en mode Quad) que l'on dessine autour d'un axe central tout en faisant varier le rayon du dessin avec de la fonction $\cos(\text{asin}())$ et avec de l'aléatoire (noise). Les yeux de la momie sont composés de deux PShape sphères. Enfin, les mains sont modélisées avec les modèles des mains qui sont fournis sur la page eCampus du cours.

Images de la momie:



Il n'y a qu'une momie par labyrinthe. On stocke les momies dans un tableau et on manipule seulement la momie du labyrinthe dans lequel se trouve le joueur. Cependant, afin d'éviter des bugs, lorsque le joueur change de labyrinthe on attend que la momie du labyrinthe précédent finisse son animation avant de commencer à déplacer la momie du labyrinthe actuel.

Les mouvements de la momie se déroulent de la manière suivante :

- On regarde si la momie peut aller tout droit, à gauche ou à droite dans le labyrinthe (une direction est valide s'il n'y a ni mur, ni joueur et que l'emplacement dans cette direction soit dans les limites du labyrinthes).
- Parmi, les déplacements possibles ont en choisi un de manière aléatoire.
(Si le déplacement choisi est d'aller tout droit alors la momie avance, si le déplacement choisi est d'aller à gauche et à droite, alors la momie se tourne dans le bon sens sans avancer)
- Si la momie se trouve dans un cul de sac alors elle se retourne.

Cette manière de faire les déplacements de la momie permet au joueur d'influencer les déplacements de la momie, il peut notamment bloquer un chemin pour forcer la momie à en prendre un autre. La momie apparaît sur la sortie du labyrinthe, ainsi, l'emplacement de la momie donne un indice au joueur sur la localisation de la sortie.

La map est toujours affichée sur la même partie de l'écran et la map ne change pas de taille, c'est la taille des murs des labyrinthes affichés à l'intérieur qui change. Nous avons aussi ajouté un curseur qui donne la position du joueur et dans quelle direction il regarde.

image du curseur:

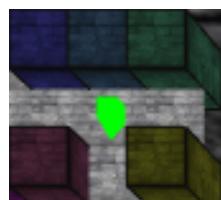
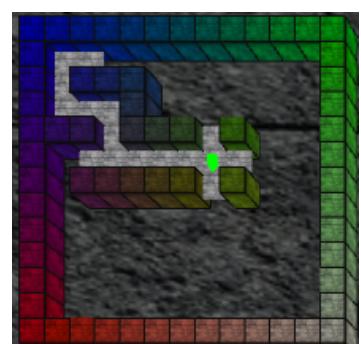
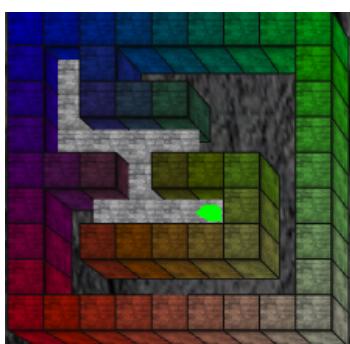


image de la map:



La boussole est une PShape qui rotate par rapport à l'axe Z en fonction de la rotation du personnage.

III) Explication de code

Création du désert :

Nous allons vous expliquer le code servant à créer le désert (sable + ondulation + dune).

Tout d'abord, le code en question :

```
void createDesert() {  
  
    // Variables  
  
    float n_i;  
    float n_j;  
    float vi;  
    float vj;  
    float vii;  
    float vjj;  
  
    float gs = grain_size;  
    float d = 125.;  
    float dh = gs / d;  
  
    float t = 3.5;  
    float min = cote * 0.25;  
    float hij;  
    float hii;  
    float hijj;  
    float hiijj;  
  
    float rd;  
  
    // Creation du desert  
  
    desert = createShape();  
  
    // Début dessin  
  
    desert.beginShape(QUADS);  
    desert.noStroke();  
    desert.texture(sand_img);  
  
    for (float i = 0; i < height; i += grain_size) {  
        for (float j = 0; j < width; j += grain_size) {  
  
            n_i = i / d;  
            n_j = j / d;  
  
            vi = abs(i - height / 2.);  
            vj = abs(j - width / 2.);  
            vii = abs((i + gs) - height / 2.);  
            vjj = abs((j + gs) - width / 2.);  
  
            hij = ( pow(vi, 2) / pow(height/2,2) + pow(vj,2) / pow(width/2,2) ) * (cote * t) + min;  
            hii = ( pow(vii, 2) / pow(height/2,2) + pow(vj,2) / pow(width/2,2) ) * (cote * t) + min;  
            hijj = ( pow(vi, 2) / pow(height/2,2) + pow(vjj,2) / pow(width/2,2) ) * (cote * t) + min;  
            hiijj = ( pow(vii, 2) / pow(height/2,2) + pow(vjj,2) / pow(width/2,2) ) * (cote * t) + min;  
  
            rd = random(100);  
        }  
    }  
}
```

```

        if (rd < 25) {
            desert.vertex(i, j, hij * noise(n_i, n_j), 0, 0);
            desert.vertex(i + gs, j, hii * noise(n_i + dh, n_j), s_u, 0);
            desert.vertex(i + gs, j + gs, hijj * noise(n_i + dh, n_j + dh), s_u, s_v);
            desert.vertex(i, j + gs, hij * noise(n_i, n_j + dh), 0, s_v);
        }
        else if (rd < 50) {
            desert.vertex(i, j, hij * noise(n_i, n_j), 0, s_v);
            desert.vertex(i + gs, j, hii * noise(n_i + dh, n_j), 0, 0);
            desert.vertex(i + gs, j + gs, hijj * noise(n_i + dh, n_j + dh), s_u, 0);
            desert.vertex(i, j + gs, hij * noise(n_i, n_j + dh), s_u, s_v);
        }
        else if (rd < 75) {
            desert.vertex(i, j, hij * noise(n_i, n_j), s_u, s_v);
            desert.vertex(i + gs, j, hii * noise(n_i + dh, n_j), 0, s_v);
            desert.vertex(i + gs, j + gs, hijj * noise(n_i + dh, n_j + dh), 0, 0);
            desert.vertex(i, j + gs, hij * noise(n_i, n_j + dh), s_u, 0);
        }
        else {
            desert.vertex(i, j, hij * noise(n_i, n_j), s_u, 0);
            desert.vertex(i + gs, j, hii * noise(n_i + dh, n_j), s_u, s_v);
            desert.vertex(i + gs, j + gs, hijj * noise(n_i + dh, n_j + dh), 0, s_v);
            desert.vertex(i, j + gs, hij * noise(n_i, n_j + dh), 0, 0);
        }
    }
}

// Fin de dessin

desert.endShape();

}

```

On va commencer par déclarer toutes les variables dont nous aurons besoin pour la création de ce désert.

Ensuite, on va commencer la création du désert avec “desert = createShape();”, puis on va rentrer en mode de construction “QUADS” qui va nous permettre de réaliser des grains de sable qui vont être des toutes petites plaques. On va également supprimer les bords et appliquer une texture créée préalablement grâce à Processing (image 100x100 avec chaque pixel de couleur aléatoire oscillant entre du jaune très très clair (presque blanc), du jaune/dorée et du jaune très très sombre (presque noir)).

On va à présent parcourir toute la taille de la fenêtre grâce à une double boucle for imbriquée (en augmentant de la taille d'un grain de sable à chaque fois).

On va “normaliser” i et j grâce à la variable d. Les n_j et les n_i vont servir à obtenir un bruit (noise) plus continu et moins homogène, ce qui va permettre d'obtenir un meilleur rendu (on calcule un noise différent pour chaque coin pour ne pas avoir de petit écart entre chaque plaque).

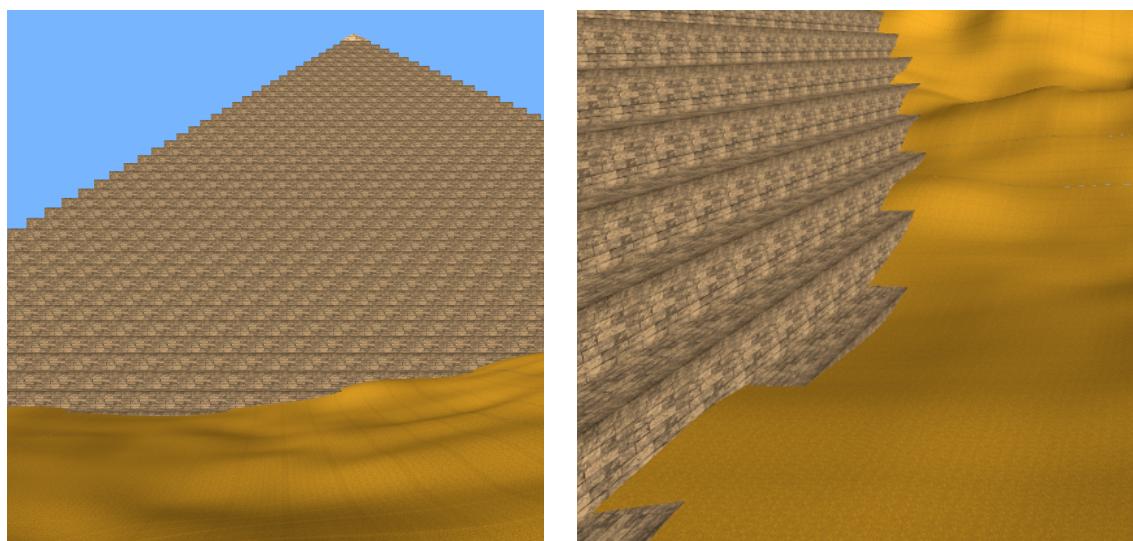
Ensuite, on va calculer la hauteur de chaque coin du grain de sable. Pour ce faire, on va servir de la fonction mathématique $X^2 + Y^2$. En effet, nous voulons obtenir de grandes dunes sur les côtes du désert afin de cacher l'horizon pour donner une illusion de désert infini et nous voulons de toutes petites oscillations au centre pour pouvoir se déplacer. $X^2 + Y^2$ possède ce comportement car son graphe dans R^2 correspond à une sorte de parabole en 2 dimensions centré en $(0, 0)$. Nous, nous voulons que cette parabole soit centrée en $(width / 2, height / 2)$. On va alors "normaliser" une nouvelle fois i et j mais cette fois-ci en les stockant dans vi , vj , vii , vjj . Puis, avec ces i et j normalisés, nous calculons la proportion de hauteur de chaque vi , vj , vii , vjj avec $vi^2 + vj^2$. Or, ici on va diviser chaque vi par $height^2$ et vj par $width^2$. Ceci va nous permettre d'obtenir un résultat entre 0 et 1. Ainsi, on aura plus qu'à multiplier par la hauteur maximale souhaitée pour obtenir un résultat continu et maîtrisé.

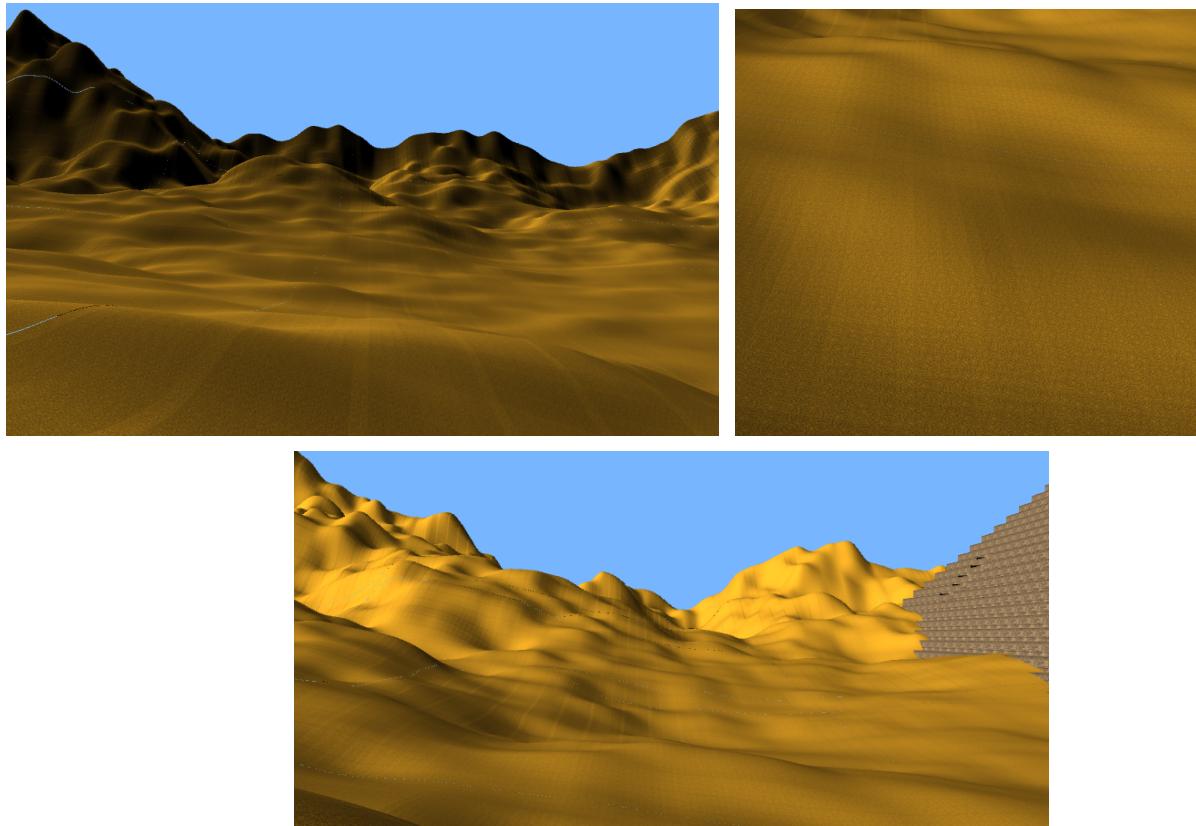
Nous devons calculer la hauteur de chaque côté séparément car sinon la succession de plaques ne serait pas continue et on obtiendrait des écarts entre chaque plaques. De plus, on ajoute également une hauteur minimale sinon au centre on aurait aucune ondulation (car les vi , vj valent 0 à cet endroit).

Enfin, pour finir on va tirer un nombre flottant aléatoire "rd" entre 0 et 99. Si $0 \leq rd < 25$ alors applique la texture dans un premier sens, sinon si $25 \leq rd < 50$ dans un autre sens, puis de même pour $50 \leq rd < 75$ et $75 \leq rd \leq 99$. Ainsi, comme la texture n'est pas appliquée dans le même sens pour chaque grain, on a l'impression d'avoir des petits grains disposés de manière aléatoire, ce qui n'est pas le cas si on applique la texture toujours dans le même sens (on observe un pattern).

Notre désert est tracé, on appelle donc "desert.endShape();" pour clôturer la création de notre désert.

Captures d'écrans du désert :





Fonctionnement du programme (Gamestate) :

Tout d'abord, le code en question :

```
state gameState;

public enum state {
    InLab, OutLab, InTpEntree, InTpExit, EnterPyr, ExitPyr, OnTpExit, OnTpEntree
}

void updateState(state s){
    /* met à jour l'état du jeu en fonction du paramètre passé en argument
     * s : state
     */
    gameState = s;
    if (gameState == state.InTpEntree || gameState == state.InTpExit){
        transition_timer = transition_sec * fps;
    }
}
```

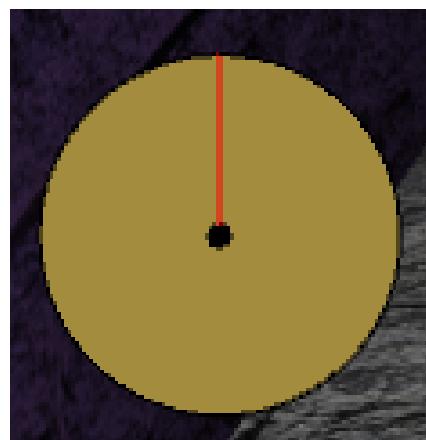
Au cours de l'exécution du programme, nous affichons différentes choses à l'écran et le nombre d'action à la disposition du joueur varie en fonction de son emplacement. Plutôt que d'utiliser une multitude de variables booléennes, nous avons créé un type énuméré que l'on appelle state. Et nous avons créé une variable globale gameState, la valeur de cette variable va déterminer ce que l'on affiche à l'écran et ce que peut faire le joueur.

Les différents états sont les suivants:

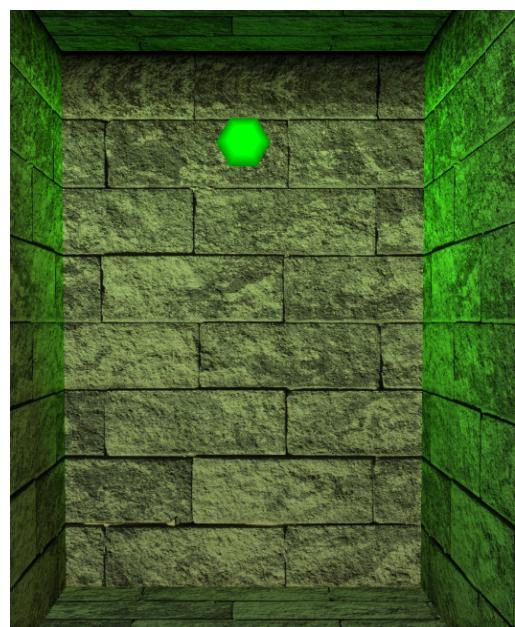
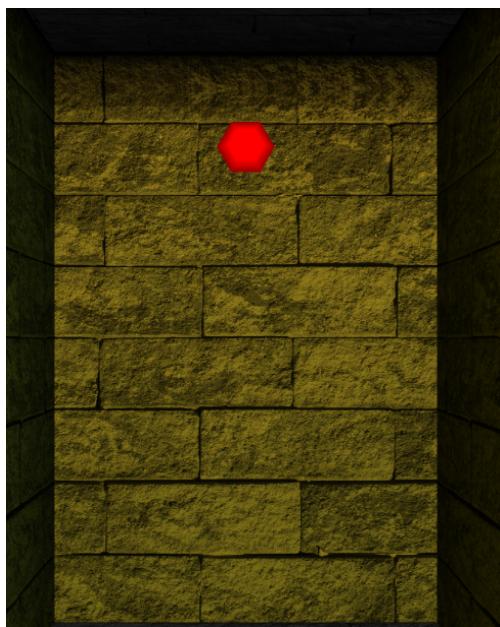
- InLab : le joueur se trouve dans le labyrinthe sur aucune case spéciale (ni sur une entrée, ni sur une sortie). On affiche alors le labyrinthe.
- OnTpEntree : Le joueur se trouve dans le labyrinthe, sur un téléporteur qui mène à l'étage d'au-dessus. On affiche alors le labyrinthe, un texte demandant à l'utilisateur d'appuyer sur ENTER, on réduit la taille de la sphère du téléporteur et si l'utilisateur appuie sur ENTER il est emmené à l'étage d'au-dessus.
- OnTpExit : Le joueur se trouve dans le labyrinthe, sur un téléporteur qui mène à l'étage d'en-dessous. On affiche alors le labyrinthe, un texte demandant à l'utilisateur d'appuyer sur ENTER, on réduit la taille de la sphère du téléporteur et si l'utilisateur appuie sur ENTER il est emmené à l'étage d'en-dessous.
- InTpEntree : Le joueur qui se trouvait sur un téléporteur d'entrée est amené au téléporteur de sortie de l'étage précédent. On affiche une animation pendant 1 seconde pour faire la transition entre les téléporteurs.
- InTpSortie : Le joueur qui se trouvait sur un téléporteur de sortie est amené au téléporteur d'entrée de l'étage précédent. On affiche une animation pendant 1 seconde pour faire la transition entre les téléporteurs.
- ExitPyr : Le joueur a atteint la sortie du dernier labyrinthe. Un déplacement automatique du joueur est réalisé, tout au long de celui-ci, on affiche le désert et la pyramide.
- OutLab : Le joueur se trouve à l'extérieur de la pyramide. On affiche alors le désert et la pyramide.
- EnterPyr : Le joueur a appuyé sur ENTER lorsqu'il se trouvait sur la case la plus proche de l'entrée de la pyramide et qu'il regardait en direction de la pyramide. Un déplacement automatique du joueur est réalisé, on affiche le désert et la pyramide. Une fois le déplacement terminé, le joueur est à nouveau dans le labyrinthe.

IV) Erreurs, problèmes et difficultés

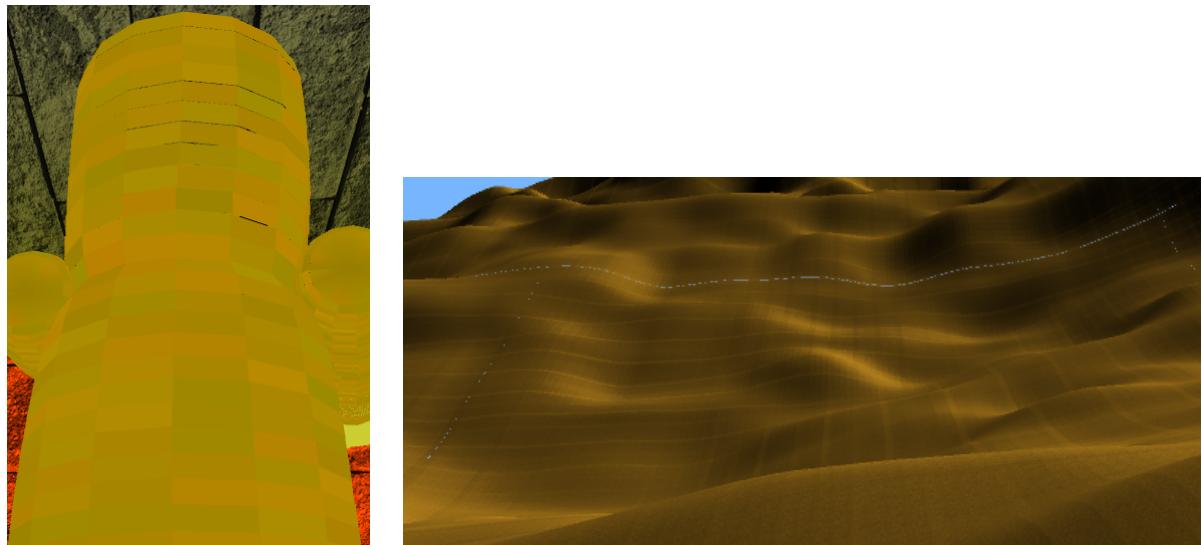
Nous avons voulu améliorer la boussole mais nous avons rencontré des problèmes. Nous ne pouvons pas appliquer de texture sur une ellipse (forme utilisée pour la boussole). Nous avons donc voulu utiliser une sphère (car on peut appliquer une texture dessus), néanmoins la texture s'appliquait mal dessus et la rotation de la sphère n'était pas concluante. Nous avons décidé de laisser notre boussole en forme d'ellipse avec une aiguille rouge qui pointe le Nord.



Nous avons rencontré un bug dès le début du projet : les lumières rouges dans les bouts de couloirs (c'est-à-dire les lumières sur les murs hauts) ne diffusent pas la lumière de la même manière que les autres (on ne voit pas de la lumière rouge sur les surfaces environnantes).

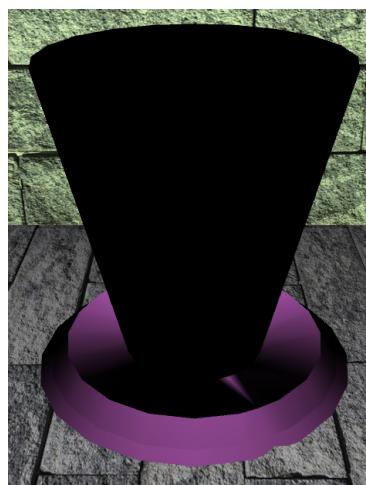


Ensuite, on a rencontré un problème sur l'affichage des bandelettes et des grains de sable. Parfois, on observe une espèce de ligne invisible alors que les objets sont censés être parfaitement collés. On se demande si c'est pas un bug d'affichage de Processing.



On a également beaucoup de mal à trouver des objets 3D sur internet. Comme Processing ne supporte que les .obj, les objets 3D d'un autre format ne fonctionnent pas. Or, sur internet on ne trouve que des formats .fbx, .USDZ, .glTF et .GLB. Lorsque l'on veut les convertir en .obj, le fichier .lib en généré automatiquement par défaut. Ainsi, le model 3D ne conserve que la forme principale mais perd ses détails et ses couleurs.

(Avec le fichier .lib généré par défaut, le socle et le vaisseau de lumière étaient entièrement blancs. On a donc modifier le fichier pour obtenir les couleurs qui nous intéressaient (on a perdu la transparence du vaisseau lumineux et les motifs sur le socle)).



Enfin, on a eu du mal à utiliser les PShape pour créer la pyramide et la momie notamment. En effet, lors des TPs on faisait souvent (presque systématiquement) recours à des translate et des rotates pour désigner. Or, lorsque l'on utilise les PShape, on ne peut pas translate le repère et rotate comme on le souhaite. Par exemple, pour la pyramide on a décidé de la tracer à partir d'un point de départ, puis de jouer manuellement avec des coordonnées courantes x et y. On s'est rendu compte à la fin du projet qu'on aurait pu (dû) utiliser plus de sous PShape afin de pouvoir les réutiliser et elles les rotate/translate. C'est ce que nous avons fait notamment avec la boussole réalisée à la fin du projet.

Nous n'avons réussi à charger qu'une seule fois les mains de la momie, en effet, lorsque l'on crée une momie on recharge les mains de la momie.

Il est aussi important de noter que si l'on modifie la taille maximale du labyrinthe dans la variable MAX_LAB_SIZE du fichier laby.pde alors il y peut y avoir des bugs avec notre désert, notre pyramide ainsi que les animations de sortie et d'entrée de la pyramide. Si MAX_LAB_SIZE est trop petit alors le joueur apparaîtra sous une dune, si MAX_LAB_SIZE est trop élevé alors le joueur se trouvera en dessous de la pyramide. La cause de cela est que la majorité des objets que l'on dessine à l'écran dépend d'une variable globale côté qui est initialisée dans la fonction setup. Cette variable varie en fonction de la valeur de MAX_LAB_SIZE. Pendant la majorité du projet nous avons travaillé avec la même valeur de MAX_LAB_SIZE, ainsi, nous nous sommes rendu compte trop tard de certains problèmes d'échelles.

V) Objectifs et apprentissage

Ce projet avait pour but l'élaboration d'un petit jeu en se servant de ce que l'on avait appris au cours du semestre. Créer des formes : grâce à des vertex, des modes de construction ou encore des PShape ; gérer différents angles et types de lumières ; également gérer des couleurs et des textures.

Nous avons également appris à se documenter et à gérer la conception d'un jeu (différentes étapes, état du jeu, etc..).

Tout cela était amusant, créatif et surtout très instructif. Nous avons revu des concepts mais nous en avons également appris de nouveaux. Cela nous a permis aussi de mettre en pratique la théorie vue le long du semestre.