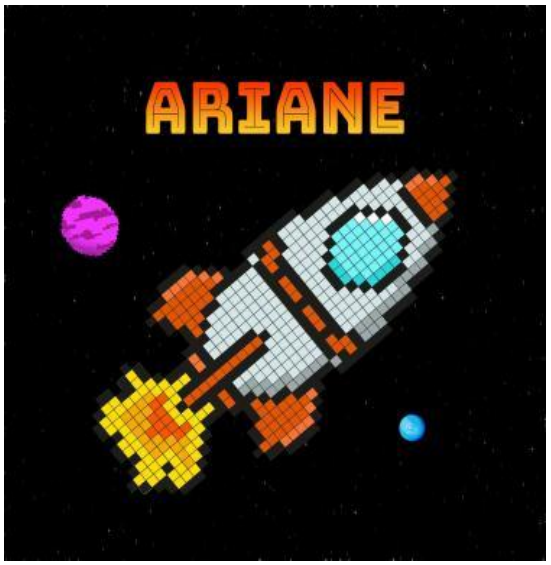
ARIANE

Deuxième rapport de soutenance, Avril 2021



Groupe CLTN:

-MANONGO Claude Cédric (chef du projet *Ariane*)

- CARION-VIGNAUD Titouan

- HASS Léo

- NDE Daniel Nolan Nathanael

Table des matières:

1-INTRODUCTION:

En vue de réaliser notre projet de Semestre 2, nous avons créé notre groupe pour mettre en commun nos différentes compétences, nos différentes idées, nos différentes connaissances acquises en cours et pendant les Travaux Pratiques.

Nous nous sommes très vite mis d'accord sur l'idée de faire un jeu vidéo de type platformer 2D. Le jeu se nomme *Ariane* comme le nom du projet en référence au domaine aérospatial et la mythologie grecque.

De plus nous avons trouvé original de faire un jeu où le personnage principal est une femme car pour la plupart des platformer 2D ou même des jeux en général, le personnage principal est un homme ou un garçon.

Notre rapport de projet est composé de deux parties :

La première partie montrera les différents avancements de chaque étape composant notre projet tout en les comparants avec les différents objectifs que nous nous étions fixés pour cette deuxième soutenance.
Ensuite, nous enchainerons sur les prévisions futures pour notre projet. Nous aborderons les choses développées pour atteindre nos objectifs prévus pour la soutenance finale.
Enfin, nous terminerons ce rapport par une conclusion sur l'avancement général de ce projet, sur les potentielles choses à améliorer et sur l'état d'esprit général du groupe.

2-AVANCEMENT DU PROJET, PLANNING ET PRÉVISIONS FUTURES :

Tout d'abord voici notre planning et les tâches assignées aux différents membres du groupe :

RESPONSABLE	SUPPLEANT	TACHES
NATHANAEL	Léo	Site internet
CLAUDE	Titouan	Editeur 2D/graphiste
TITOUAN	Léo	Animateur
TITOUAN	Nathanael	Ingé son
CLAUDE	Léo	Gestionnaire physique
TITOUAN	Nathanael	Scénariste
CLAUDE	Nathanael	Gestion Perso principal
LEO	Titouan	Gestion Perso mob
CLAUDE	Léo	Gestion Perso Boss
CLAUDE	Nathanael	Gestion Obstacle
TITOUAN	Claude	Interface
LEO	Nathanael	Réseau

	Première soutenance	Deuxième soutenance	Troisième soutenance
Site internet	50%	100%	100%
Editeur 2D/graphiste	20%	50%	100%
Animateur	20%	50%	100%
Ingé son	0%	10%	100%
Gestion perso	40%	60%	100%
Interface	40%	60%	100%
Réseau	0%	20%	100%

Editeur 2D/Graphisme

Construction des niveaux dans Unity :

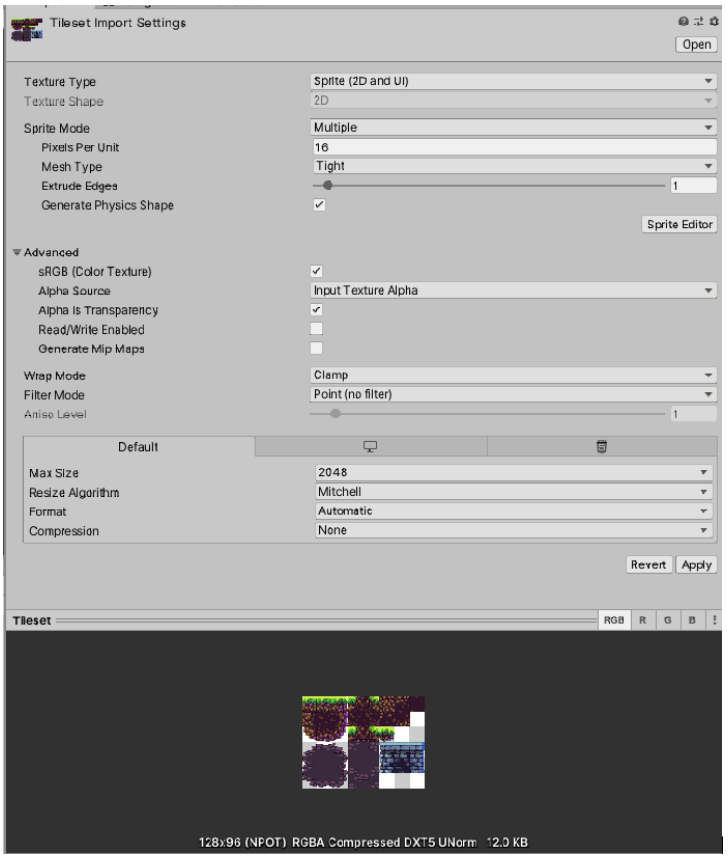
Suite à la conception des tilemaps (une fiche avec tous les graphismes nécessaires pour la conception du niveau) nous devons désormais les implémenter dans Unity pour concevoir nos niveaux.

Pour se faire, on va devoir créer des scènes dans Unity, chaque scène correspond à un niveau.

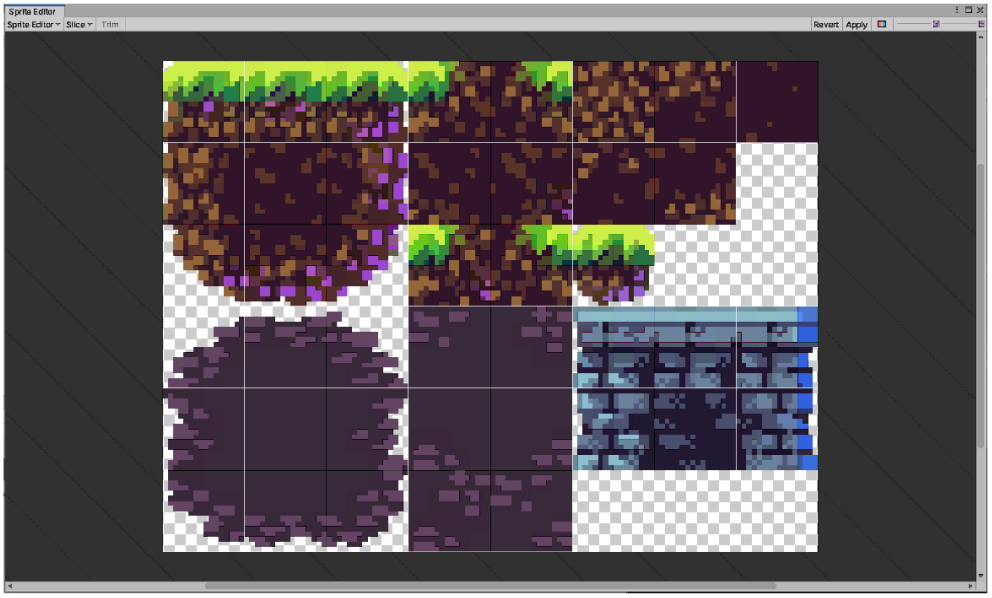
Pour rappel le premier niveau a pour thème l'espace, le deuxième se passe un dans un univers futuriste et le troisième se passe sur terre.

On commence donc par créer une scène et lui rajouter ses textures dans la section assets

On importe les graphismes au sein du logiciel et on va paramétrer l'image de façon à pouvoir la scinder avec un quadrillage plus tard :

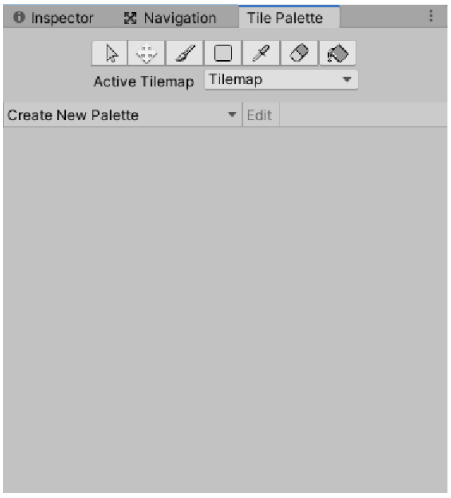


Une fois le paramétrage terminé on ouvre un éditeur en cliquant sur le bouton "Sprite Editor". Une petite fenêtre s'ouvre avec à l'écran nos textures, tout en haut l'un des boutons permet de scinder ces textures les divisant en petits carreaux : c'est le bouton "Slice". Nous avons paramétré 16 pixels par cellules plus tôt donc nous allons devoir scinder la fiche de texture en petits carreaux de 16 pixels par 16 pixels.

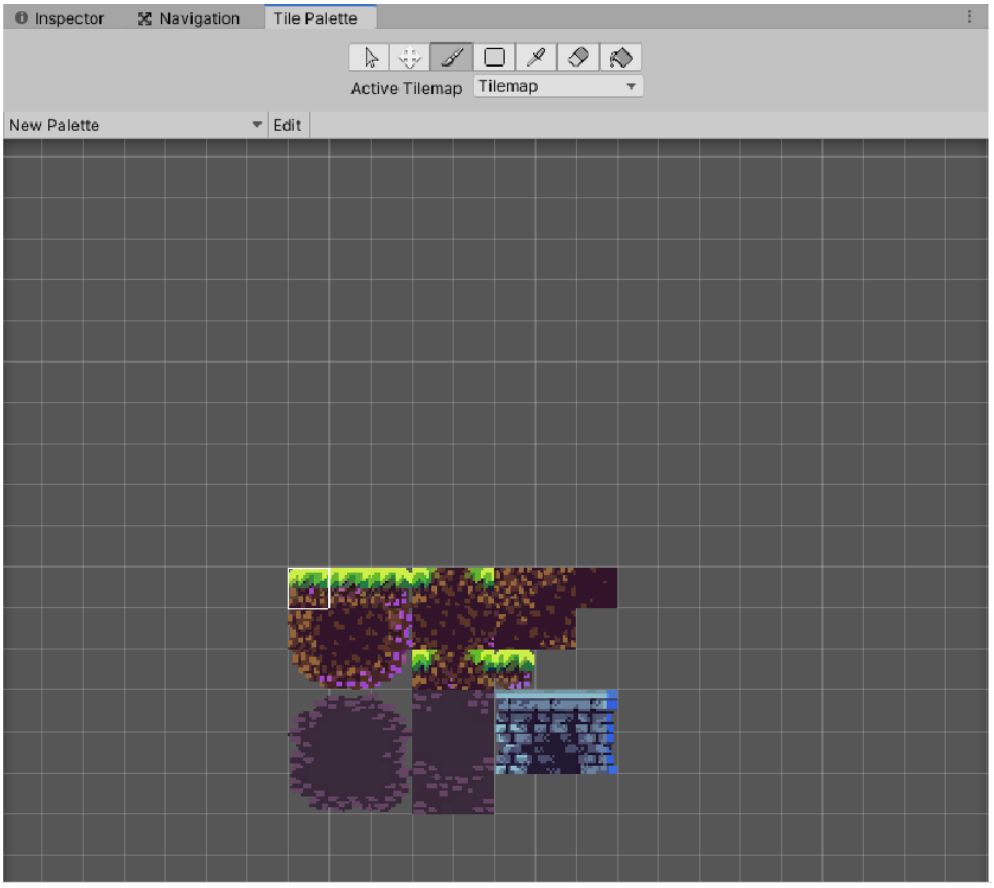


Ensuite nous allons devoir recréer cette "tilemap" ou fiche de texture au sein du logiciel.
Nous nous plaçons dans la fenêtre "Hierarchy", il faudra alors faire un clic droit, choisir l'option "2D Object" puis "Tilemap", on se retrouve alors avec cette interface où chaque carreau sera parfait pour contenir les textures que l'on a scindé auparavant.

On va alors devoir concevoir une "Tile Palette" qui nous permettra directement d'accéder aux blocs et les utiliser comme nous le souhaitons. Pour se faire il faut aller dans la section "Window" sélectionner "2D" puis "Tile Palette".

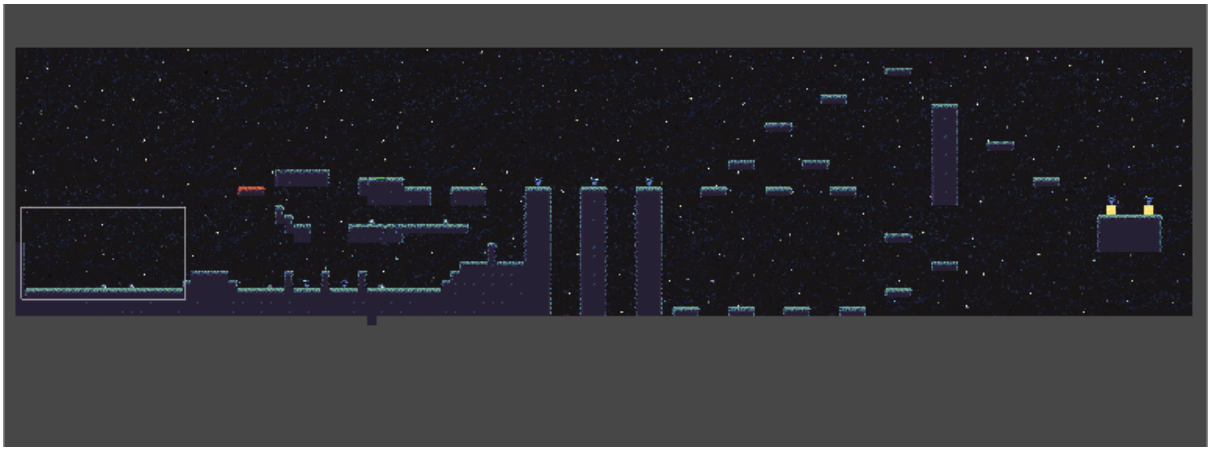


Nous avons ici tous les outils nécessaires à la conception d'un niveau en deux dimensions. Cliquez sur "Create Palette", puis il va falloir cliquer sur le dossier des textures importées sur Unity auparavant et le glisser dans l'espace gris vide comme vous pouvez le voir ci-dessus. On obtient alors ce résultat :

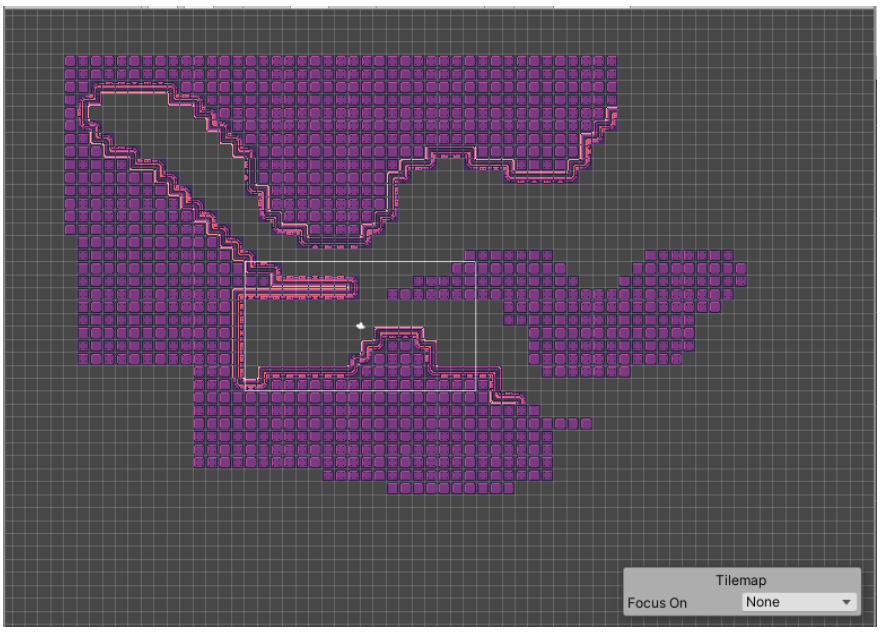


Notons que la surbrillance en haut à gauche signifie que ce carré est sélectionné et qu'en se rendant sur la scène principale et en sélectionnant l'outil pinceau nous pouvons poser des blocs comme nous le souhaitons.

En suivant ce procédé nous avons donc pu concevoir le niveau 1, il est court et va surtout servir d'introduction au mouvement du personnage de façon à ce que le joueur puisse s'habituer au moteur de jeu.

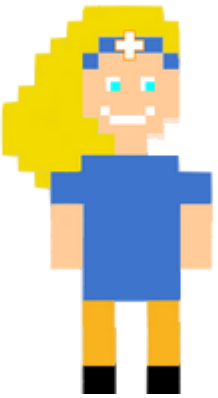


Voici également le niveau 2 en cours de production.



Sprites d'Ariane :

Pour Ariane nous avons repris notre illustration de départ que nous pouvons retrouver en haut de notre site web, et nous l'avons pixélisé afin d'être plus cohérent avec le jeu. Pour la rendre plus vivante, nous avons créé deux sprites pour lorsqu'elle se déplace ainsi que deux sprites lorsqu'elle est au repos où nous pouvons voir sa respiration.



Interface :

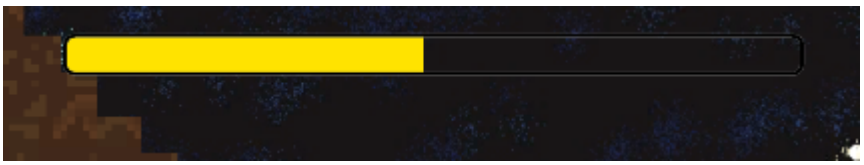
Après avoir finalisé notre premier niveau, nous avons inclus un menu “Pause” en appuyant sur la touche “Echap” du clavier, suite à cela un menu s’ouvre avec trois boutons, un bouton “Reprendre” pour revenir au jeu, un bouton “menu” pour revenir au menu principal ainsi qu’un bouton “Quitter” pour quitter le jeu.

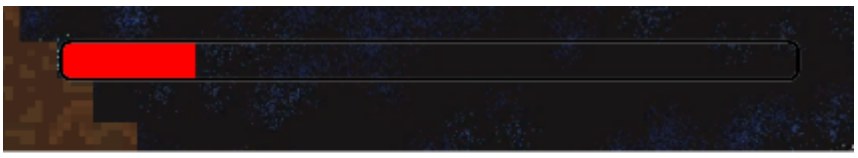
Code du menu pause :

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class PauseMenu : MonoBehaviour
{
   public static bool GameIsPause = false;
   public GameObject pauseMenuUI;

   // Update is called once per frame
   void Update()
   {
      if (Input.GetKeyDown(KeyCode.Escape))
      {
         if (GameIsPause)
         {
            Resume();
         }
         else
         {
            Pause();
         }
```

```
        }
    }

    public void Resume()
    {
        pauseMenuUI.SetActive(false);
        Time.timeScale = 1f;
        GameIsPause = false;
    }

    void Pause()
    {
        pauseMenuUI.SetActive(true);
        Time.timeScale = 0f;
        GameIsPause = true;
    }
    public void menu()
    {
        Time.timeScale = 1f;
        SceneManager.LoadScene(0);
    }
    public void loadquit()
    {
        Debug.Log("quitting game....");
        Application.Quit();
    }
}
```

Dans le jeu nous avons aussi implémenter une barre de vie mais elle n'est toujours pas fonctionnelle, en effet nous n'avons pas encore codé le lien entre la vie du joueur et la barre.





Gestion du Personnage et monstre :

Correction de bug et modification au niveau du personnage principal :

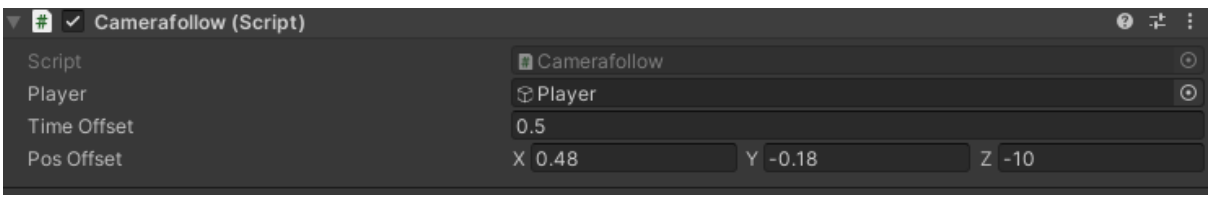
Après notre première soutenance et en manipulant le personnage, nous avons vite remarqué que celui-ci possédait quelques bugs.
En effet le personnage ne sautait pas systématiquement lorsqu'on appuyait sur la barre espace et il arrivait parfois qu'il se bloque contre des textures graphiques.
Nous avons également ajouté à notre jeu une caméra qui suit le personnage et également une rotation de 180° du personnage lorsqu'il va de la gauche vers la droite ou de la droite vers la gauche.

Nous avons commencé par ajouter la caméra suivant le personnage.
Pour cela nous avons utilisé un *SmoothDamp* pour déplacer la caméra d'un endroit à un autre.

```
public class Camerafollow : MonoBehaviour
{
    public GameObject player;
    public float timeOffset;
    public Vector3 posOffset;

    private Vector3 velocity;
    Message Unity | 0 références
    void Update()
    {
        //déplacer la caméra vers la position du joueur dans le temps
        transform.position = Vector3.SmoothDamp(transform.position, player.transform.position + posOffset, ref velocity, timeOffset);
        //transform correspond à l'objet caméra
        //Smoothdamp permet de déplacer un objet d'un endroit à un autre
        //posOffset sert a donner du recul a notre caméra
        //timeOffset permet d'ajouter un décalage dans le temps entre le moment ou le joueur bouge et entre le moment ou la caméra bouge
        //velocity est obligatoire pour utiliser un SmoothDamp
    }
}
```

Code pour le déplacement de caméra

Ce code permet à la caméra de se déplacer de sa position actuelle (*transform.position*) à la position du personnage (*player.transform.position*) tout cela en donnant du recul à notre caméra (*posOffset*) et en ajoutant un léger délai entre le moment où le joueur bouge et entre le moment où la caméra bouge (*timeOffset*). Ces deux derniers paramètres sont modifiables à notre guise.



Pour ce qui est de la rotation de notre personnage en fonction de sa direction, cela n'a pas été très difficile.

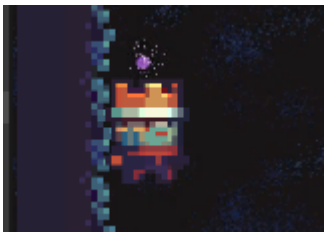
```
void flip(float _velocity) //pour se tourner quand on va a gauche ou a droite
{
    if(_velocity > 0.1f)
    {
        spriterenderer.flipX = false;
    }
    else if(_velocity < -0.1f)
    {
        spriterenderer.flipX = true;
    }
}
```

Comme vous pouvez le voir dans le code ci-dessus nous avons seulement regardé sur l'axe horizontal(x) la vélocité du personnage. Si celle-ci est négative, cela veut dire que notre personnage va vers la gauche et inversement si elle est positive cela veut dire que notre personnage va vers la droite et pendant ces passages de la droite vers la gauche ou inversement c'est la où l'on va tourner notre personnage à 180° à l'aide de *flipX* (rotation sur l'axe x).

(personnage tournée vers la droite)

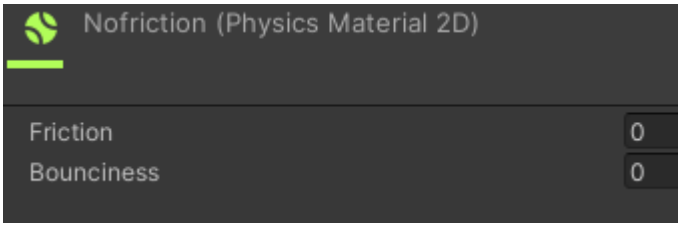
(personnage tournée vers la gauche)

Nous avons ensuite corrigé le problème de blocage.(Voir photo ci-dessous)



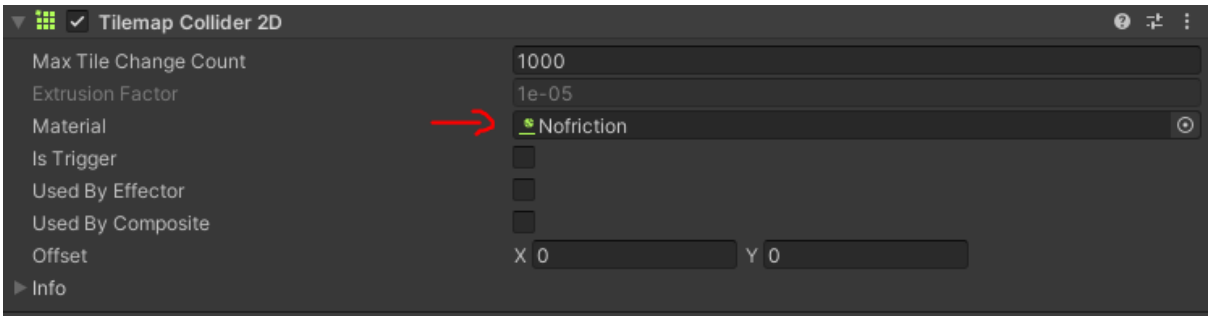
Ici le personnage est accroché au mur alors qu'il devrait chuter. Cela est dû à la friction du personnage avec les différents graphiques de notre monde. La boîte de collision des textures s'accrochait à celle du personnage.

Pour pallier à ce problème nous avons créé un Physics Material 2D, celui-ci va permettre de donner différentes propriétés à des textures voulues.
Ce Physics possède deux paramètres: la friction et le rebondissement, nous avons mis ces deux paramètres à zéro.



(Bounciness signifie rebondissement)
En mettant la friction à zéro, on évite le phénomène de friction entre le personnage et la texture. Nous avons laissé le paramètre “Rebondissement” à zéro car nous ne souhaitons pas que notre personnage soit repoussé au contact d’une texture.

Enfin, nous avons ajouté ce Physics dans la boîte de collision des textures du niveau, pour rendre celui-ci fonctionnel.

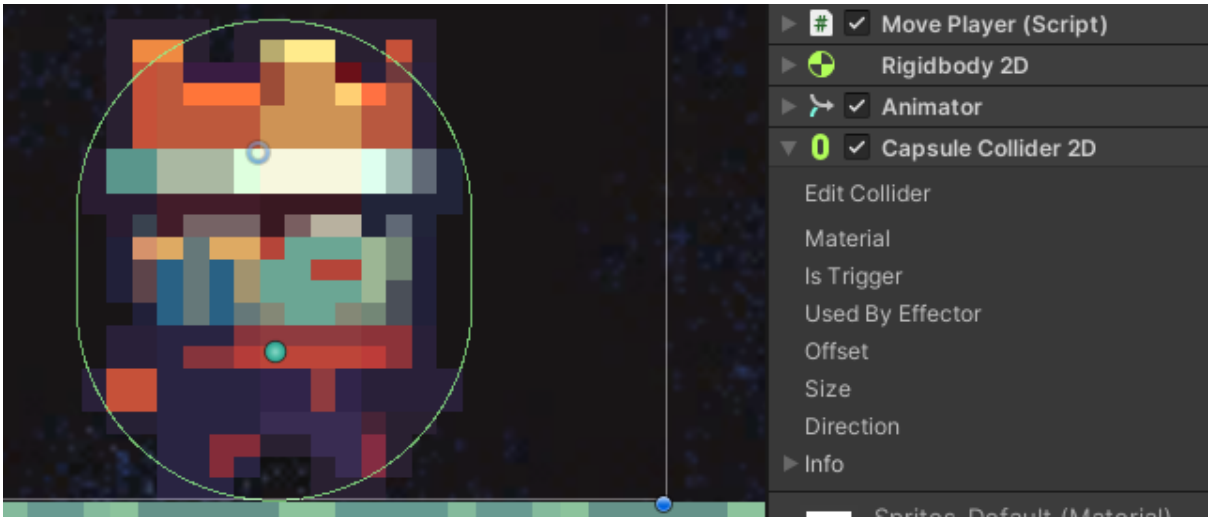


Ainsi lorsque notre personnage entre en collision avec les textures de la carte, il ne s’accroche plus et quand il doit tomber, il tombe.

Nous nous sommes ensuite penchés sur le problème de saut.
Pour le corriger, nous avons d’abord commencé par modifier notre code, en effet nous avons remarqué que nous avions mélangé des notions Physique avec d’autres notions comme par exemple les notions d’inputs. (Dans notre cas Inputs sert à savoir quand le joueur va appuyer sur la barre espace)

Pour cela nous avons créé une autre fonction update pour pouvoir séparer le physique du reste.
On a ensuite revu totalement les boîtes de collisions de notre personnage.

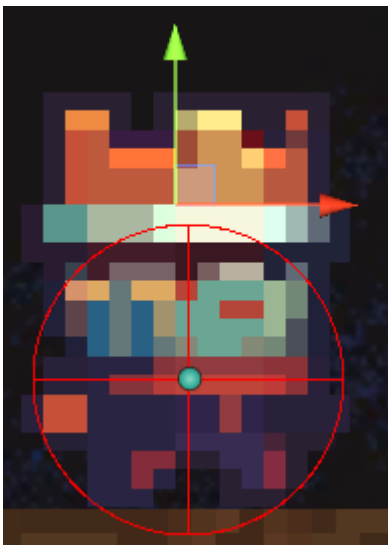
Nous avons précédemment utilisé deux boîtes de collisions, une boîte carrée pour les contacts horizontaux et ceux au niveau de la tête et une boîte en forme de cercle pour les contacts au sol.
Nous avons remarqué qu'il existait une boîte de collision regroupant la forme de nos deux boîtes de collisions précédentes: le capsule collader.
Comme son nom l'indique, il s'agit d'une boite de collision en forme de capsule, ainsi elle peut recouvrir tous les types de collision. (Voir photo ci-dessous)



Nous avions créé pour la soutenance 1, 2 zones pour savoir si notre joueur était au sol.
Grâce à cette nouvelle boite de collision, nous en avons créé seulement une seule.

(Avant), ici on a deux zones (les deux points verts)

Nous avons ensuite créé une sphère autour de cette zone qui nous permettra de savoir si le personnage touche le sol.
Pour créer la sphère nous nous sommes servis de la méthode *OverlapCircle*. Nous nous sommes également servis de *layer*, un *layer* est un tag (étiquette). (Voir images ci-dessous)

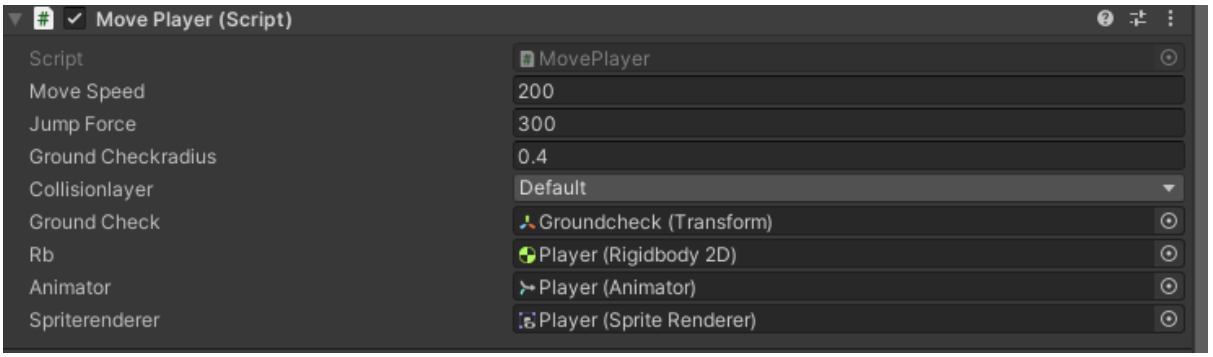


(Après), le point bleu est notre nouvelle zone, autour de ce point bleu nous avons la sphère en rouge.

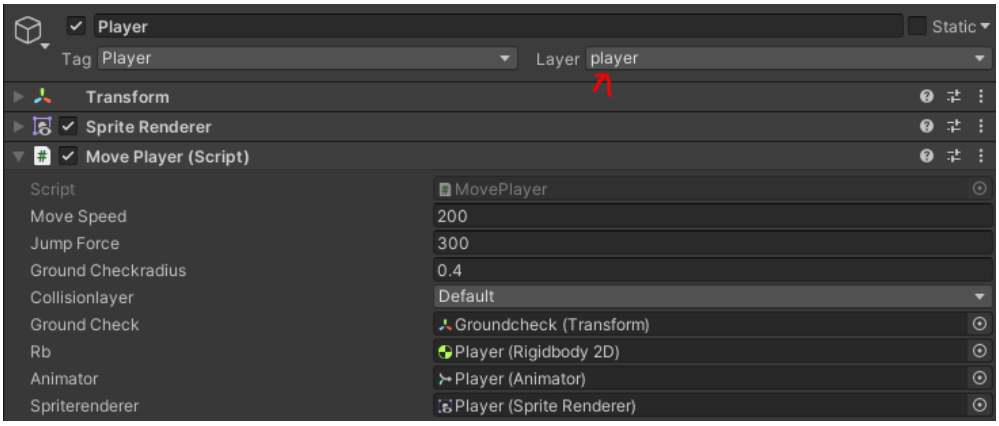
```
isGrounded = Physics2D.OverlapCircle(GroundCheck.position, groundCheckradius, collisionlayer);
```

- *GroundChack.position* est la position du point bleu.
- *groundCheckradius* est le rayon de la sphère.
- *collisionlayer* fait référence aux tags avec lesquels le personnage pourra rentrer ou non en collision.

Nous avons mis le *collisionlayer* à *Default* ce qui fait que le joueur ne pourra sauter que s'il rencontre des éléments possédant le layer *Default*. C'est-à-dire toutes les textures de notre monde. (Voir photo ci-dessous)



Nous avons également créé un layer *Player pour* éviter que le personnage ne se détecte lui-même dans le système de collision. Si on laissait le layer du personnage à *Default*, celui-ci pourrait sauter à l'infini. (Voir photo ci-dessous)



Voilà notre personnage saute sans problème et il ne peut faire qu'un seul saut.
Nous vous avons mis ci-dessous tout le code permettant le saut.

```
public float moveSpeed; //vitesse de déplacement
public float jumpForce; //force de saut
private float horizontalMouvement;
public float groundCheckradius;
public LayerMask collisionlayer;


private bool isJumping;
private bool isGrounded;


public Transform GroundCheck;


public Rigidbody2D rb; //ref au rb du perso
private Vector3 velocity = Vector3.zero;
public Animator animator;
public SpriteRenderer spriterenderer;


Message Unity | 0 références
void Update() //uptade réservé à tout ce qui n'est pas physique
{

    if (Input.GetButtonDown("Jump") && isGrounded) //pour savoir si le jouer veut sauter et si il est au sol
    {
        isJumping = true;
    }
```

```
    flip(rb.velocity.x);
    float characterVelocity = Mathf.Abs(rb.velocity.x); //renvoyer une vitesse positive
    animator.SetFloat("Speed", characterVelocity);
}

Message Unity | 0 références
void FixedUpdate() //update réservé à physique
{
    horizontalMouvement = Input.GetAxis("Horizontal") * moveSpeed * Time.deltaTime; //mouvement horizontal au fil du temps
    isGrounded = Physics2D.OverlapCircle(GroundCheck.position, groundCheckradius, collisionlayer); // pour checker si le joueur est au sol
    Movejoueur(horizontalMouvement); //déplacement du joueur

}


1 référence
void Movejoueur(float _horizontalMouvement)
{
    Vector3 targetVelocity = new Vector2(_horizontalMouvement, rb.velocity.y); //calcul de la vélocité du perso
    rb.velocity = Vector3.SmoothDamp(rb.velocity, targetVelocity, ref velocity,.05f);

    if(isJumping == true)
    {
        rb.AddForce(new Vector2(0f, jumpForce));
        isJumping = false;
    }
}
```

Ajout d'un premier monstre:

Pour l'ajout du premier monstre nous avons utilisé un graphisme d'Internet que nous changerons plus tard avec notre propre monstre.

Le graphisme en question

Nous avons voulu faire un monstre un peu dans le style de Mario Bros. C'est-à-dire que celui-ci va se déplacer d'un point A à un point B que nous choisissons nous-même dans le niveau.

On a d'abord commencé par lui ajouter une boite de collision pour qu'il puisse entrer en collision avec les différentes textures et avec le personnage.

Ensuite nous avons commencé à programmer son déplacement d'un point A à un point B.

Pour cela, nous avons choisi de faire un système de waypoints, un waypoint est en réalité simplement un point, le monstre va se déplacer à l'infini d'un waypoint à un autre. Nous avons donc créé deux waypoints pour notre monstre.

```
public class Enemypatrouille : MonoBehaviour
{
    public float speed; //vitesse de l'ennemie
    public Transform[] waypoints; //on mettra à l'intérieur tout les points vers lequel l'enemie doit se deplacer
    private Transform target; //cible vers laquelle l'ennemie va se déplacer
    private int destpoint = 0; //cela va etre le point de destination

    public SpriteRenderer graphics;


    Message Unity | 0 références
    void Start()
    {
        target = waypoints[0];
    }
```

```
void Update()
{
    Vector3 dir = target.position - transform.position;
    //position cible - position acutuelle de l'ennemie pour savoir dans quelle
    //direction il faut se déplacer pour aller au prochain waypoint
    transform.Translate(dir.normalized * speed * Time.deltaTime, Space.World);
    //deplacement par rapport au parent

    if(Vector3.Distance(transform.position, target.position) < 0.3f)
        //si la distance entre object actuel et le waypoint qui se dirige est inférieur a 0.3 on va vers un autre waypoint,
        //0.3 est une valeur de sécurité
    {
        destpoint = (destpoint + 1) % waypoints.Length;
        //destpoint correspond à l'indexe du waypoint vers lequel on se dirige, le modulo permet a l'ennemie une fois qu'il
        //a parcouru tout les waypoints de recommencer  à 0

        target = waypoints[destpoint];
        graphics.flipX = !graphics.flipX;
    }
}
```

Nous avons donc créé une liste de waypoints, dans notre cas elle en possède deux. Nous parcourons cette liste à l'infinie avec l'index *destpoint* et donc le *waypoints[destpoint]* nous le stockons dans *target* qui est la cible vers laquelle l'ennemi va se déplacer. (Voir code ci-dessus)

Voilà l'ennemi se déplace d'un point A à un point B.

Nous avons atteint les objectifs pour cette deuxième soutenance, notre personnage se déplace parfaitement et nous avons notre premier ennemi fonctionnel.
Pour la prochaine soutenance nous devons ajouter d'autres monstres et ainsi implémenter de nouveaux mouvements pour ceux-ci.

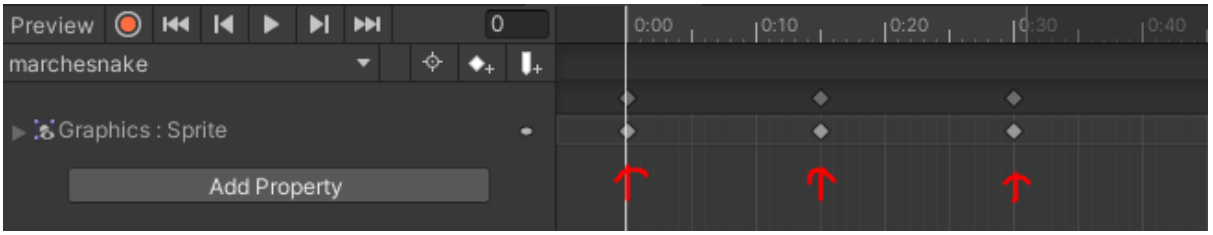
Animateur :

Animation du premier monstre et sa destruction:

On a continué avec les animations du monstre et la destruction de celui-ci. Pour la destruction, elle s'effectue lorsque le personnage écrasera la tête du monstre.

Pour la première animation du monstre nous avons implémenté la rotation à 180° lorsque le serpent va de droite à gauche et de gauche à droite. Nous n'allons pas vous réexpliquer le procédé puisqu'il est semblable à celui utilisé pour le personnage.(Voir page X)

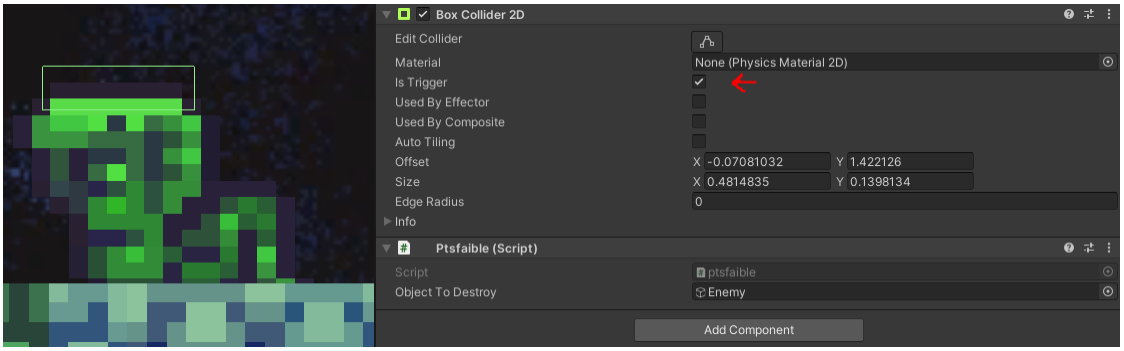
Pour la deuxième animation nous avons implanté l'animation de marche, nous nous sommes servis de l'outil Animation qui nous a permis de superposer différentes images du monstre avec quelques secondes de décalage et ainsi obtenir l'animation voulu.



Nous avons laissé 15 secondes entre chaque image pour éviter un effet de marche trop rapide. (Voir photo ci-dessus)

Pour la destruction du monstre nous avons ajouté à celui-ci un point faible au niveau de sa tête.
Le point faible est en réalité une boite de collision.
Lorsque le personnage entrera en collision avec cette boite de collision, le monstre se verra détruit.

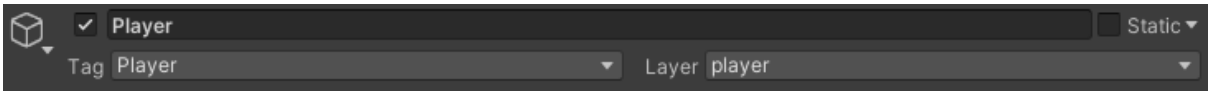
Nous avons coché le paramètre is Trigger car celui-ci va permettre à la boîte de collision de se faire traverser.
(Voir image ci-dessous)



On a ensuite programmé sa destruction. (Voir photo ci-dessous)

```
public class ptsfaible : MonoBehaviour
{
    public GameObject objectToDestroy;
    Message Unity | 0 références
    public void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.CompareTag("Player"))//vérifier si le player rentre en collision
        {
            Destroy(objectToDestroy);//détruire de monstre
        }
    }
}
```

Premièrement on a assigné le tag “Player” a notre personnage, ainsi avec *CompareTag* on va détecter si le joueur entre dans le point faible du monstre et s’il entre on va détruire le monstre avec *Destroy(objectTodestroy)*.

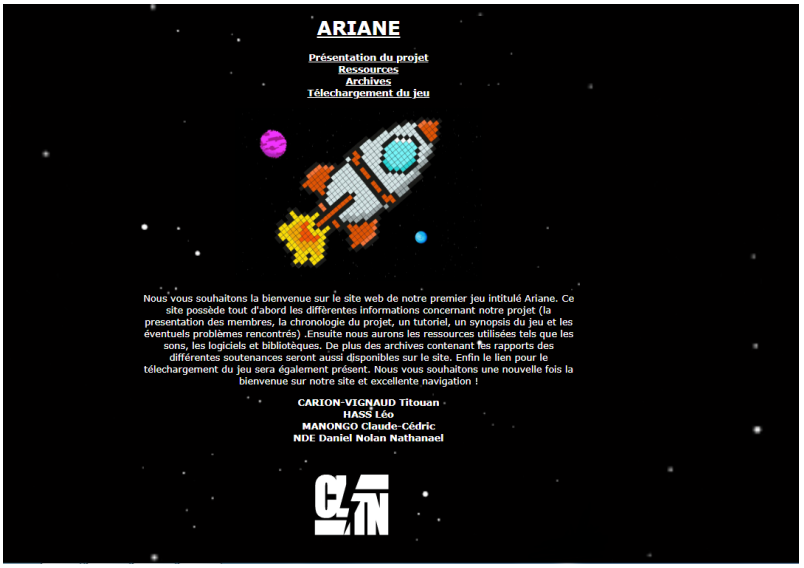


(Image montrant le tag “Player”)

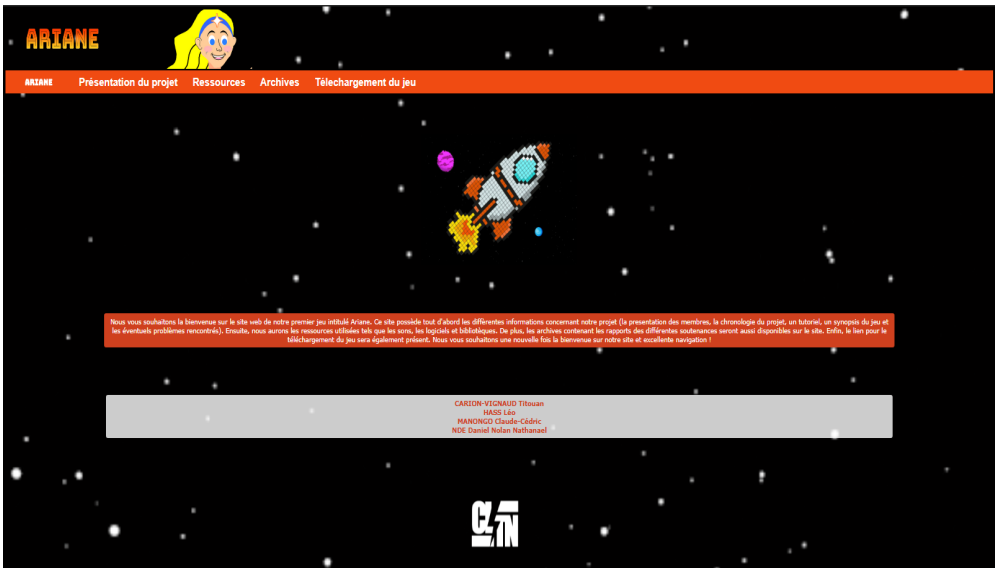
Voila notre serpent se tourne en fonction de sa direction et le joueur peut l'éliminer s’il en a envie.
Concernant les animations nous avons atteint nos objectifs pour cette soutenance, il faut que pour la prochaine soutenance nous implémentons des animations de combat et de mort.

Site Web:

Pour la première soutenance nous avons pu faire les différentes pages du site web grâce au HTML et au CSS. De plus, la problématique de l'hébergement avait été traité également à l'aide de la plateforme Github qui proposait un hébergement gratuit du site.
Entre la première et la deuxième soutenance, nous avons décidé de customiser notre site Internet pour le rendre plus esthétique, ceci notamment grâce au fait que maintenant nous sommes plus à l'aise avec le CSS et le HTML.



Notre page d'accueil à la première soutenance



Notre page d'accueil après modification pour cette deuxième soutenance

Nous avons fait nos différentes modifications directement sur la plateforme Github où nous avions à disposition notre code: cela nous permettait de constater nos changements beaucoup facilement et rapidement que si nous devions modifier le code sur Sublime Text pour devoir le réactualiser sur la plateforme Github.

```
text-align : left;
background: #CF401E;
border-radius: 6px;
margin: 5% 0;
padding: 10px 10px;
```

Code pour le fond des blocs de textes

```
<ul id="nav"><!--
        --><li><a href="Ariane.html"><img width="180" height="20" border="0" src="https://github.com/Nthnl47/Nthnl47.github.io/blob/master/Images/Sans%20titre%
        --><li><a href="Presentation du projet.html"> Présentation du projet </a></li><!--
        --><li><a href="Ressources.html"> Ressources </a></li><!--
        --><li><a href="Archives.html"> Archives </a></li><!--
        --><li><a href="Télechargement du jeu.html"> Télechargement du jeu </a></li>
</ul>
<div id="block">
```

Code pour le menu

Nous avons ajouté plus de couleurs sur la page avec de l'orange au niveau du fond des blocs de texte.

(par exemple le synopsis et le but du jeu)

De plus, la barre d'affichage du menu a été refait pour le rendre plus accessible que l'ancien qui n'est pas très bien présenté et nous a ajouté l'égérie du jeu Ariane pour rendre la barre de menu plus décoratif.

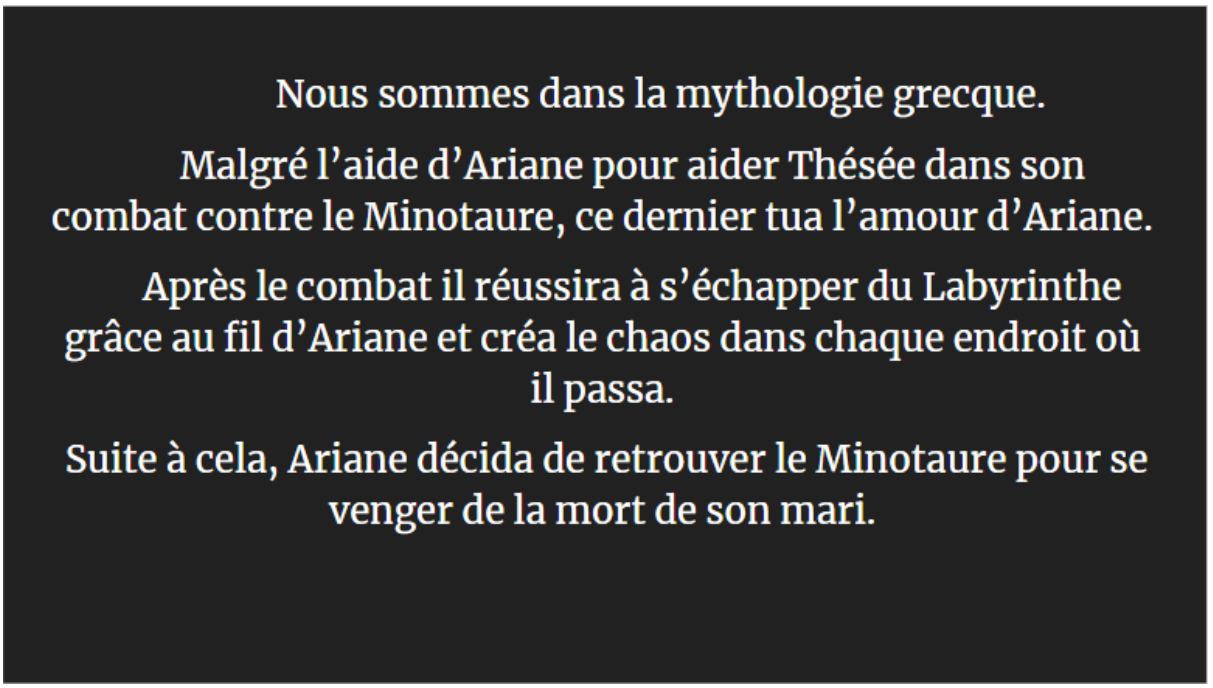
Le menu du site

En outre nous avons rajouté du contenu puisqu'avant nos pages étaient vierges. Ainsi nous avons pu aborder la présentation du projet plus précisément du but du jeu et du synopsis pour permettre aux joueurs de mieux comprendre le concept et l'intérêt du jeu et encore de la présentation des membres du groupe avec nos photos. En plus de cela nous avons mis à disposition dans la rubrique *Archives* notre premier rapport de soutenance afin qu'il puisse être consulté à tout moment.
En somme, notre site est bien développé avec les différents onglets qui sont remplis en fonction de l'avancement du projet avec une présentation du site agréable à regarder.

Pour la dernière soutenance en ce qui concerne le site web, nous devrons ajouter ce deuxième rapport de soutenance ainsi que le dernier. De plus nous compléterons la chronologie du projet avec les différentes étapes de l'avancement de celui-ci en fonction des objectifs prévus initialement. Nous devrons mettre à jour nos ressources utilisées tel que les différentes bandes sons qui animeront les actions du joueur dans sa partie; les logiciels utilisés et tout ce qui nous aura permis de mener à bien notre projet. Enfin le lien pour télécharger notre jeu devra être disponible dans l'onglet *Téléchargement du jeu*.

Scénario:

Pour rendre notre jeu plus complet, nous avons décidé d'intégrer à notre jeu de slides dynamiques avec de la narration pour permettre au joueur de mieux comprendre le contexte et le décor dans lequel il est. Nous aurons au début le synopsis pour expliquer au joueur le contexte et pour lui permettre de comprendre le concept et le but du jeu. Ainsi entre chaque niveau nous aurons une slide qui va indiquer le décor dans lequel est le joueur.



Le synopsis du jeu qui sera affiché au tout début du jeu

Pour l'instant on laissera les slides en fond noir. Pour la dernière soutenance nous ferons des modifications sur le fond des slides avec le décor de chaque niveau pour le rendre plus esthétique et plus décoratif.

Ingénieur son et réseaux:

Concernant le son et le réseau nous avons commencé à bien nous pencher sur le sujet mais nous ne les avons pas encore implantés dans notre jeu. Nous nous sommes mis d'accord sur ce que nous voudrions faire pour ces deux notions tout de même importantes dans notre projet.

Pour le son nous comptons mettre une musique dans le menu Pause et dans le menu d'accueil. Nous ajouterons également une musique en jeu et des bruitages de mort de mouvement et de monstres.
Tous ces bruitages ou musiques seront en accord avec le thème de notre projet qui est l'espace.
Nous avons commencé à télécharger des bruitages en accord avec nos idées ainsi que des musiques potentielles.

Concernant le réseau nous avons choisi de faire du local, des joueurs se retrouveront sur un même écran et ils joueront ensemble sur un écran scindé un peu comme *Mario Kart*. (voir image ci-dessous).

Le but pour chacun des joueurs sera d'arriver en premier et nous avons pensé qu'il serait intéressant de faire perdre de la vie aux joueur(s) qui seraient trop loin du joueur en tête.
En somme nous avons atteint les objectifs puisque nous n'avions mis que 10 % d'avancement pour le son et 20 % d'avancement pour le réseau. Nous avons une banque de son et nous savons ce que nous voulons faire pour le réseau.
Pour la prochaine soutenance il faudra donc rajouter tous les différents sons et implanter le réseau sur notre projet.

3-Conclusion:

Les objectifs fixés pour cette soutenance intermédiaire ont été globalement atteints. Pour cette soutenance nous avons privilégié terminer le site et corriger tous les problèmes que nous avions rencontrés après la première soutenance.

Il a été difficile de réunir tous les membres de groupe compte tenu de la crise sanitaire et des emplois du temps de chacun.
Malgré cela le groupe s'entend bien, la cohésion est bonne grâce notamment aux différentes plateformes qui nous permettent de nous appeler à distance.

Nous envisageons pour la future soutenance de remplir les objectifs fixés car nous serions très heureux de voir le résultat final de notre première création informatique en groupe et nous espérons que celle-ci marchera comme nous l'aurions voulu.