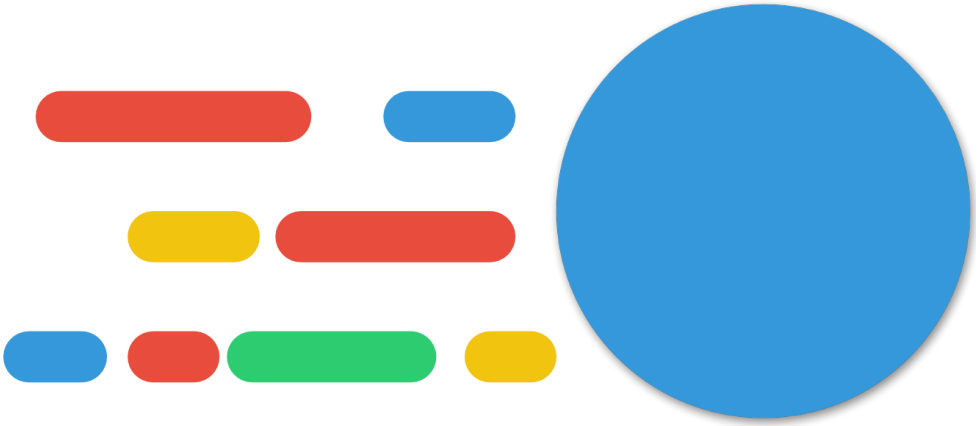
Rapport de soutenance



Colors

Rémi VERNAY

P1D

Alexandre BERMUDEZ

Jean-Baptiste DESPUJOL

Romain GREGOIRE

Alex POIRON

Sommaire

1. Introduction

Comme dans notre précédent rapport, nous allons commencer par faire un point sur l'avancement des différents membres du groupe ainsi que sur leur avis concernant l'ambiance au sein du groupe et l'évolution du projet. L'ambiance dans le groupe est toujours aussi bonne, la coordination s'améliore de plus en plus, chaque membre arrive à trouver les informations et les méthodes sur le travail qui lui est demandé. Cela a permis de bien avancer le projet et même de commencer à réfléchir sur les derniers éléments de jeu ainsi que sur certains aspects de la **jouabilité**. Ainsi toutes les échéances ont été respectées, ainsi un nombre important d'axes du projet ont été plus avancés que ce qui était attendu .

Compte tenu du fait que notre jeu est un side scroller et que la manière dont nous avons réparti notre travail nous permet d'avancer séparément sans jamais retarder un autre membre du groupe nous avons pu faire évoluer notre jeu à un point, qui ressemblerait bien plus à ce à quoi notre jeu ressemblera à son état final. Nous avons donc terminé la base du code pour le personnages en général, les ennemis ainsi que sur leur déplacements. Nous avons même pu commencer à imaginer les écrans de game over et les animations de rebond et de mort.

Pour ce qui est du site internet, son développement, s'est déroulé sans accroc. Les ajouts se sont faits naturellement et le site semble pratiquement terminé avec les informations adéquates.

Les interfaces sont pratiquement toutes implémentées dans Unity et connectées à la base de données, c'est une très grande avancé pour cette partie puisque tout fonctionne comme voulu.

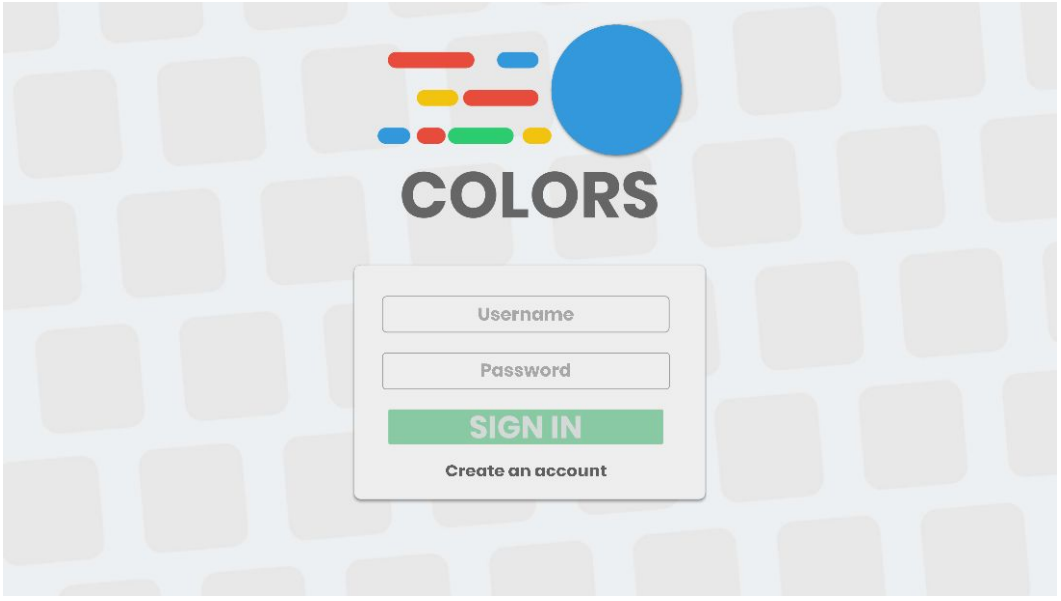
2. Réalisation du projet

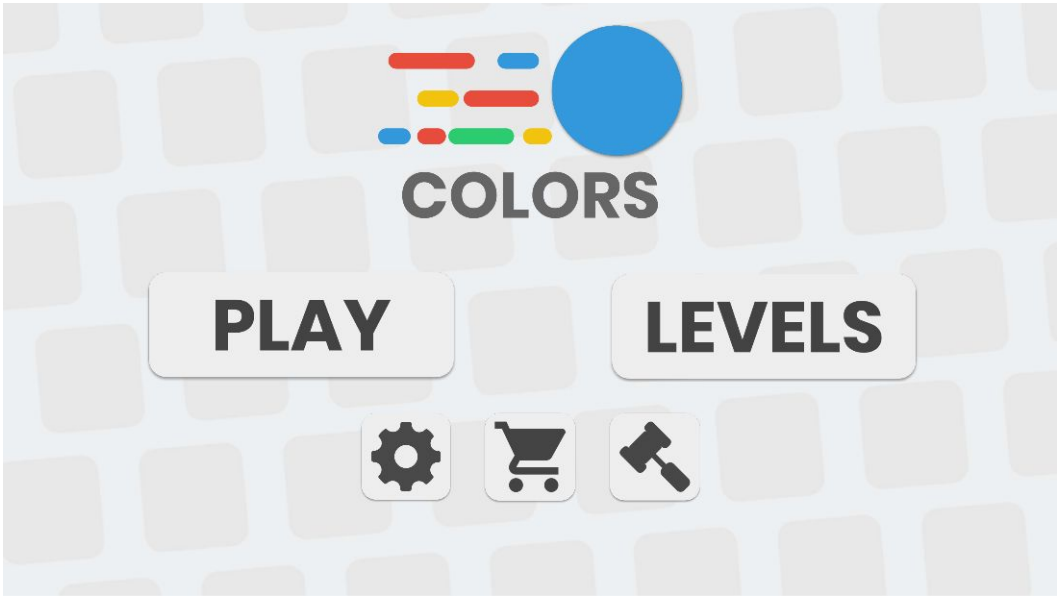
2.1 Graphisme/Design (Alexandre, Alex)

Nous avons déjà présenté lors de la dernière soutenance une version très avancée de notre design que nous avons réalisé sur des logiciels de design comme Adobe Illustrator et Adobe Design. Lors de l'implémentation dans Unity nous avons dû baisser nos espérances suite au fait que Unity n'intègre pas des outils aussi poussé que ceux de la suite Adobe en terme de design.

Nous avons fait notre possible pour être le plus fidèle à notre schéma. Seulement, les interactions comme la sélection n'ont pas pu être respectées à cause des limites de notre expérience sur Unity.

Voici des images de l'implémentation :



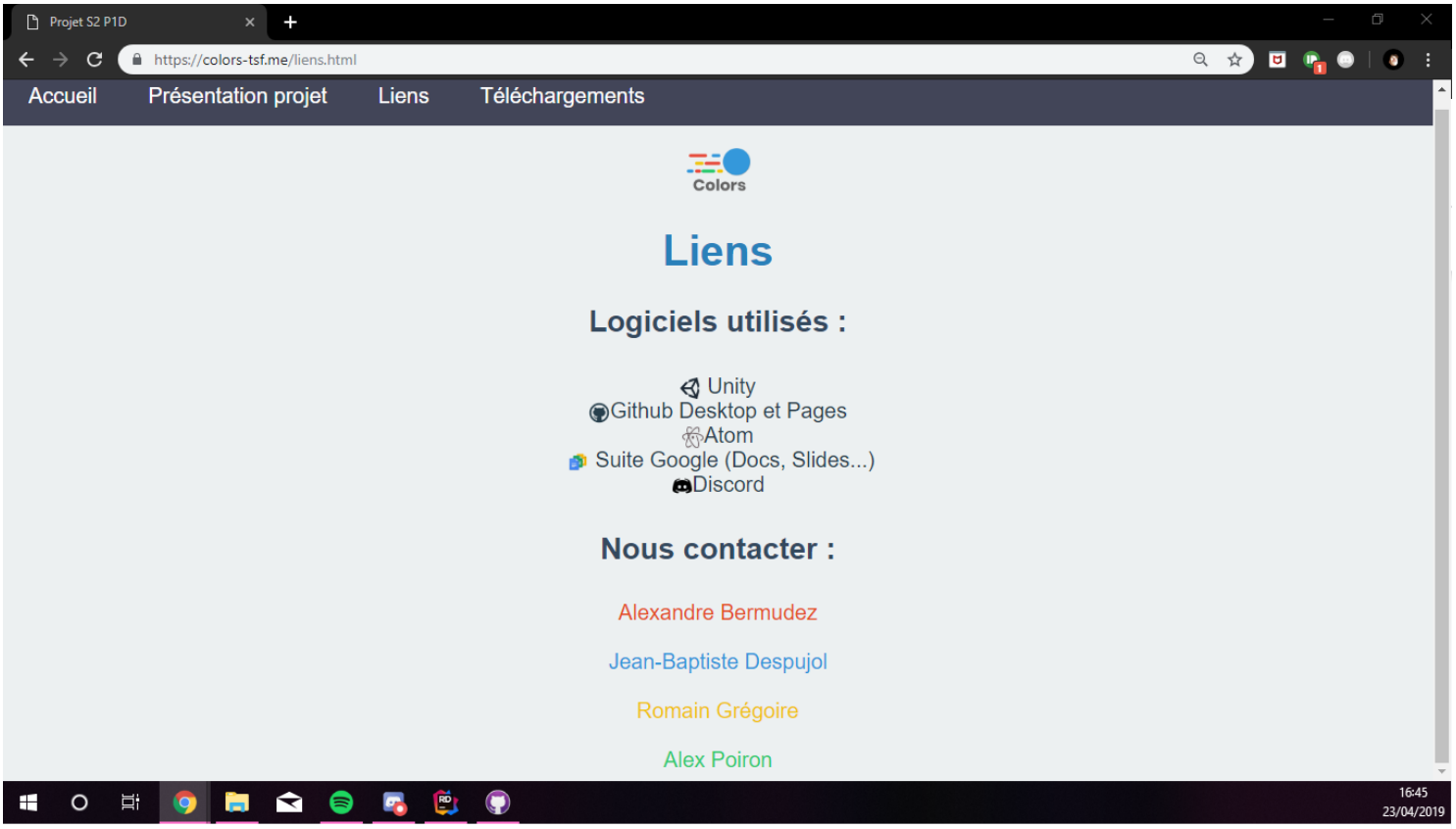


2.2 Site Internet (Jean-Baptiste, Alexandre)

Pour la soutenance précédente nous avions indiqué que notre but pour la soutenance 2 était d'ajouter du contenu au site internet. En effet, le design de ce dernier était d'ores et déjà au point, il fallait donc le remplir.

Tout d'abord dans l'onglet Liens, nous avons ajouté les logiciels qui nous ont permis de réaliser le projet. Leurs différents noms sont des liens cliquables menant à leur site internet respectif. Nous avons retiré la décoration du texte afin qu'ils ne ressemblent pas à des liens classiques. Sans cette manipulation, le texte se serait coloré en violet une fois le lien visité. Nous avons organisé ces différents liens sous forme d'une liste verticale. Nous avons centré tous les éléments afin de coller à l'esthétique générale du site internet.

Nous avons rajouté les logos des différents logiciels. Il nous a fallu trouver des images au format PNG afin de ne pas avoir de fond contrastant avec la couleur du fond du site. Les logos sont donc très bien intégrés.

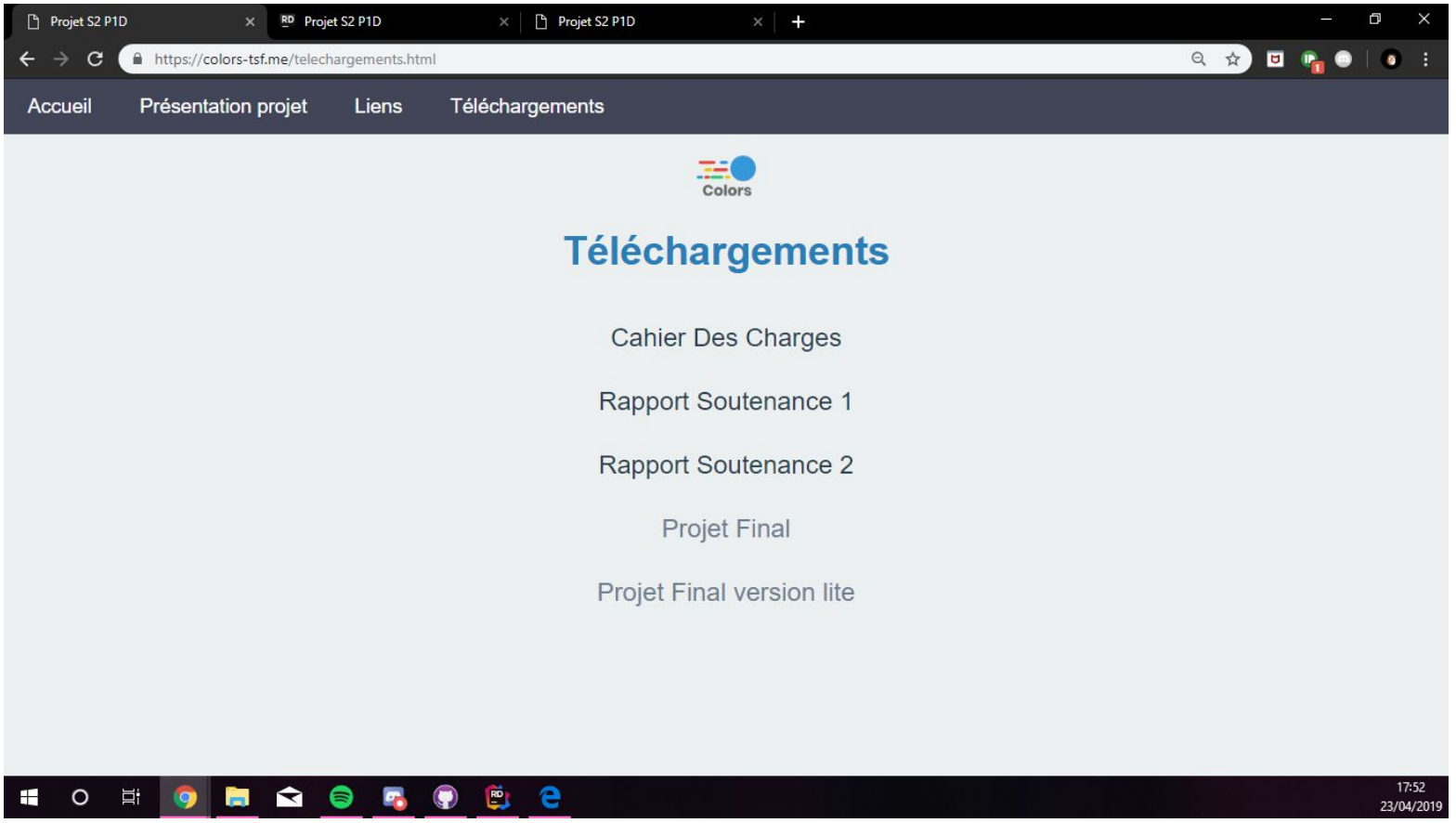


Pour ce qui est de la partie “Nous contacter” nous avons simplement lié le texte avec une méthode “mailto”. Cette dernière permet d’ouvrir le logiciel d’email par défaut de l’ordinateur qui créera un nouveau mail avec le champ du destinataire déjà rempli par l’adresse d’un des membres du groupe.

Pour ce qui est de l’onglet présentation nous avons mis les membres du groupe et leur photo. Nous avons d’abord créé ces vignettes à l’aide de Google Slide, que nous avons ensuite exportées ces photos au format PNG encore une fois pour qu’il n’y ait pas d’information ou de fond superflux. Nous nous sommes également servis de descriptions d’images afin que les noms de chaque membre soient alignés avec la photo correspondante.

Ensuite nous avons ajouté 4 paragraphes détaillant un problème majeur que chaque membre a rencontré et comment nous l’avons réglé. Il nous permet de nous rendre compte des grandes difficultés auxquelles nous avons fait face.

Et finalement nous avons créé un historique de réalisation qui permet de suivre nos avancements majeurs dans le projet.



Pour finir, pour ce qui est du dernier onglet, nous avons stockés les documents téléchargeables sur le Git de notre groupe. Nous avons fait en sorte que lorsqu'un utilisateur clique sur les liens, un nouvel onglet s'ouvre dans le navigateur pour que le site et le document soient ouverts en même temps.

Les derniers liens ne sont pas encore disponibles et sont donc grisés, ils renvoient vers la page de téléchargement. Pour l'utilisateur, cela aura seulement l'air d'un lien qui ne mène nulle-part.

Pour la dernière soutenance, il nous restera quelques petits détails à améliorer comme faciliter la navigation sur le site. Mais nous sommes très satisfaits de l'allure générale et des informations qu'il contient.

2.3 Personnage (Romain, Jean-Baptiste)

Depuis notre dernier rapport de soutenance de nombreux éléments concernant le personnage ont été ajoutés et une importante modification a été faite sur le comportement du personnage vis à vis de ses déplacements. Nous allons donc dans un premier temps expliquer la nature et la raison de la modification puis nous nous attarderons sur les ajouts qui ont été faits.

Tout d'abord, il faut savoir qu'une refonte complète de l'organisation du **fichier UNITY** été faite. En effet nous avons fait le choix de tout reprendre dans un fichier commun de manière à ce que l'échange des données se fasse de manière plus simple et que nous puissions d'ores et déjà avoir une idée de la forme finale de notre dossier UNITY.

Ainsi lors de cette refonte nous avons choisi de modifier légèrement la méchanique de **side scroll**. A la base nous souhaitions que notre personnage se déplace verticalement pour esquiver les ennemis, néanmoins il est apparu que ce style de déplacements était en complète opposition avec la mécanique centrale du jeu qui est le changements de couleur du personnage . En effet lors des premiers test du jeu il est apparu que si nous laissions le joueur se déplacer librement deux problèmes principaux se heurtaient à nous :

-Le premier : La sortie d'écran. Pour Colors, nous avons fait le choix de mettre une caméra qui traque les mouvements de notre personnage sur l'axe des x via un script que nous avons implémenté. Malheureusement si nous laissions la possibilité à notre personnage de se déplacer avec la première idée que nous avions eu, celui-ci se retrouvait beaucoup trop souvent hors du cadre imposé par la caméra entraînant ainsi l'échec du niveau .

-Le second : La trop grande facilité d'évitement. Comme nous l'avons expliqué plus haut la mécanique principale de notre jeu est le changement de couleurs. Or nous ne voulons pas que le joueur choisisse tout le temps d'éviter les obstacles (nous pensons que cela enlève un certains côté technique nécessaire, pour accroître la durée de vie d'un jeu). Ainsi lors de nos test nous nous sommes rendu compte qu'au vu des déplacements des ennemis, que nous évoquerons plus bas dans notre rapport, il était beaucoup plus simple et instinctif d'esquiver l'ennemi que d'essayer de passer à travers celui-ci.

Et donc , après concertation avec le groupe, nous nous sommes mis d'accord pour remplacer les déplacements verticaux par un saut beaucoup plus restrictif quant à l'esquive et qui force l'utilisation du changement de couleur. Grâce à cela nous pouvons garder une certaine forme de verticalité dans notre jeu pour que celui-ci ne paraisse pas redondant.

Nous allons donc maintenant pouvoir aborder les ajouts que nous avons fait.

Le premier ajout que nous avons réalisé est celui du **changement de couleur** pour cela nous avons utilisé les propriétés de base de UNITY que nous avons légèrement modifié pour récupérer **les entrées clavier basique z,q,s,d** .

Par la suite nous avons assignés à chacune de ses touches une couleur particulières pour cela nous avons simplement modifié **la couleur de rendu de l'objet**(z pour le jaune, s pour le vert, q pour le rouge et d pour le bleu). Nous avons donc utilisé un switch pour séparer les 4 cas possibles et la méthode **Input.GetButtonDown("String")** pour récupérer ces cas.

Le second ajout que nous avons réalisé est la mécanique de saut. Pour cela nous récupérons l'entrée clavier espace pour faire sauter notre personnage . En effet, contrairement à ce qui à déjà été fait dans des jeux comme Geometry Dash nous avons choisi de modifier la valeur en y du vecteur vitesse d'origine pour que notre personnage soit plus contrôlable et que les mouvements paraissent plus naturelles .

Nous avons également fait le choix de **modifier légèrement le système de vitesse et d'accélération**. Pour donner à notre jeu une certaine logique de vitesse nous avons créé 3 vecteurs vitesse différents auxquels vient s'ajouter notre ancien système d'accélération, ceux-ci sont sélectionné au début du niveau. En des termes plus claires nous avons édité un script qui récupère le numéro du niveau et ajuste la vitesse en fonction de cela pour régler la difficulté .

Enfin le dernier ajout concernant le personnage est **l'ajout des tags**. Après avoir testé le jeu et à la suite de certaines de nos recherches, il est apparu que le système de tags était très efficace et très simple d'utilisation. Mais ce dernier nous exposait également à de **nombreuses erreurs lors de collisions avec les ennemis ou encore les pièces** sur lesquels nous reviendront plus tard.

Pour remédier à cela nous avons séparé et ajouter un tag particulier à notre personnage qui lui permet de toujours être **"Indépendant"** vis à vis des interactions entre les objets.

Donc pour ce qui est de cette partie du projet, nous avons terminé toute la partie basique il ne nous reste donc plus qu'à implémenter les différentes animations de notre personnage pour rendre le jeu plus vivant et plus plaisant à regarder.

2.4 Obstacles (Romain, Alex)

Pour ce qui est des obstacles, comme pour le personnage nous avons grandement avancé quant au design ainsi que sur leurs interactions. Lors de la dernière soutenance nous n'avions imaginé qu'un seul personnage qui ne possédait qu'une seule couleur. De plus, une chose simple mais qui allait totalement à l'encontre de notre jeu était que nous laissions le choix au joueur d'esquiver les ennemis au lieu de changer de couleur. Il est donc apparu que ce système était fort désagréable à regarder et rendait le jeu pauvre.

.

Pour contrecarrer cette effet, nous avons totalement changé notre manière de voir les obstacles. Désormais le joueur aura beaucoup plus de mal à simplement les éviter et dans certains cas ne pourra pas du tout le faire. Nous avons également fait en sorte que tous nos obstacles soient constitués de chacune des couleurs de notre jeu.

Pour l'instant leur apparence est donc, soit sous forme de carré vide ou de cercle tout deux découpés en quatre, soit sous forme de bandes qui traversent tout l'écran en se déroulant. Nous avons donc ainsi dupliquer des ennemis afin de créer des enchaînements intéressant et compliqué à éviter. Pour ce qui est des ennemis carrées et circulaires, nous avons éditer un script de manière à ce qu'ils tournent sur eux-même. Pour cela nous avons utilisé la fonction transform.Rotate() selon l'axe z.

Néanmoins malgrés le fait que nous ayons bien avancé l'aspect général de nos ennemis, nous nous sommes heurtés sur deux grosses difficultés que nous avons eu du mal à résoudre :

-La première: **les rotations**. Cela peut paraître simple lorsqu'on y pense mais faire une rotation dans un plan qui n'existe pas dans notre jeu présentait un certains nombre de problèmes. En effet nous avons remarqué que nos obstacles tournaient autour d'un seul et même axe, comme si nous devions créer une orbite. A cause de cela nous avons quelque difficultés à créer de nombreux ennemis en rotation constante.

-la seconde: **les boîtes de collisions**. Pour nos obstacles circulaires nous avons importé 4 **sprites** que nous avons assemblé pour en créer un seul et unique . Mais dans UNITY, il n'existe aucune boîte de collision pour arc de cercle, et aucun des différents colliders de UNITY nous permettait de correspondre exactement à la forme de nos arcs de cercle. Mais après des recherches, il est apparu que le polygon collider 2D correspondait à nos attentes en termes de hitbox.

Pour conclure sur cette partie, nous pouvons dire que nous sommes proches du rendu final et qu'il ne nous reste plus qu'à imaginer de nouveaux **motifs** pour leurs déplacements

2.5 Shop/Comptes (Jean-Baptiste, Alexandre)

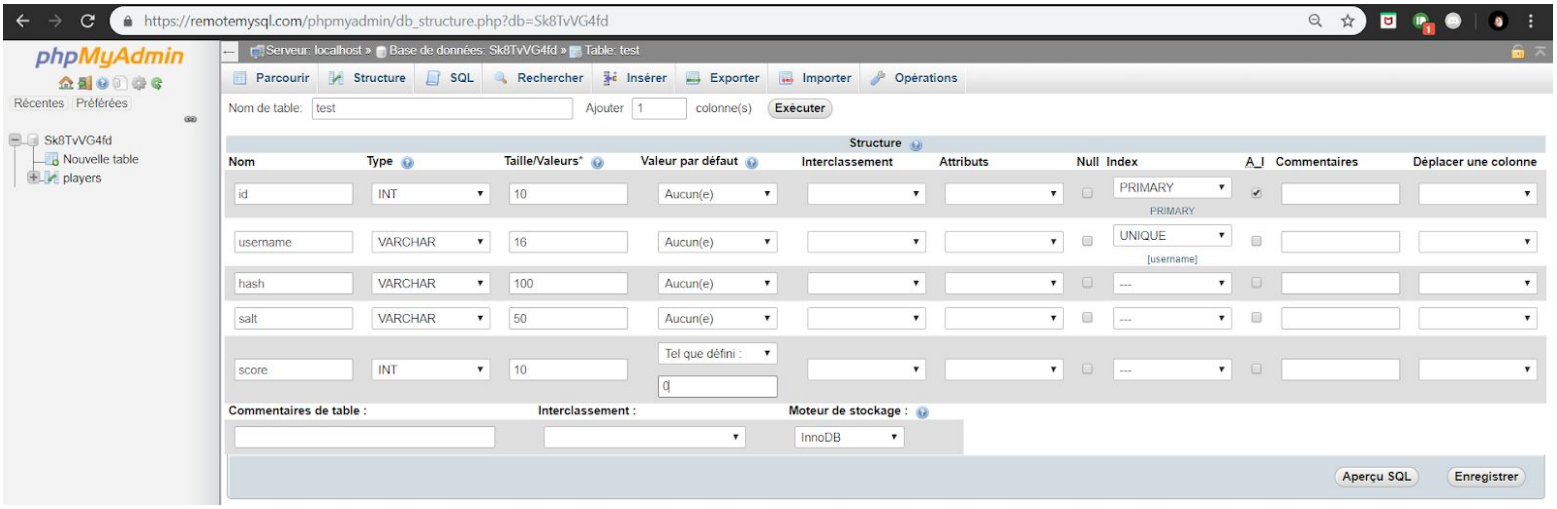
Durant une longue période, nous avons concentré nos recherches vers l'utilisation d'un Google Sheet comme support pour une base de données (database) pour les joueurs. Nous avions compris dès le départ que cette implémentation allait être assez complexe. Nous nous sommes tout de même obstinés du fait qu'Alexandre avait déjà utilisé ce système pour un jeu en Python l'an passé. Il lui avait été très facile d'utiliser et de comprendre une bibliothèque existante dans son code de jeu dans ce langage.

Mais après avoir écumé la plupart des informations disponibles à ce sujet sur internet, et après avoir réalisé de nombreux tests infructueux, nous avons compris qu'il était préférable d'explorer une autre voie pour le stockage de nos données en lignes. Nous avons donc recherché d'autres moyens de réaliser une base de données pouvant être liée à notre projet UNITY.

Nous avons découvert le système de gestion de données MySQL. C'est un logiciel libre, open source qui permet de créer la base de données dont nous avions besoin. Nous avons ensuite trouvé un tutoriel vidéo très détaillé qui montrait comment mettre en place une base de données localement sur nos ordinateurs et comment la lier à notre jeu. Cette méthode utilisait des fichiers PHP. Ce nouveau langage de prime abord fait peur puisque nous ne l'avions jamais utilisé auparavant. Mais, il est en fait assez intuitif et les explications données dans le tutoriels sont assez claires.

Nous avons bien compris après quelques recherches que ce langage permet de réaliser des actions sur des serveurs web. Il nous est donc paru étrange d'utiliser un tel langage pour des informations stockées localement. Mais après un peu de réflexion, nous avons compris que notre stockage local se comportait comme un serveur à la seule différence qu'il n'était pas connecté à internet.

Dans le processus, nous avons utilisé le logiciel Mamp. Ce dernier nous permet de stocker facilement la database sur nos appareils. Après avoir choisi notre dossier de dépôt, le logiciel nous redirige vers une page internet MySQL. Sur cette dernière, nous pouvons créer facilement une base de données ainsi que les tables qui la constituent. Les tables sont très facilement configurables, on choisit le type de données à stocker dans chaque ligne, le nombre maximal de caractères ou même la valeur par défaut du paramètre.



Ensuite, nous avons commencé à écrire notre fichier php qui allait communiquer et modifier notre base de données pour créer de nouveaux utilisateurs du jeu.

```php
<?php

    $con = mysqli_connect('localhost', 'root', 'root', 'unityaccess');
//premier paramètre va être remplacé par l'url de l'hote de la base de
données

    //vérification que la connexion existe
    if(mysqli_connect_errno())
    {
        echo "1: conection failed"; //code d'erreur #1 = connexion ratée
        exit();
    }

    $username = $_POST["name"];
    $password = $_POST["password"];

    //vérification que le nom existe
    $namecheckquery = "SELECT username FROM players WHERE username='" .
$username . "';";

    $namecheck = mysqli_query($con, $namecheckquery) or die("2: Name
check query failed"); //code d'erreur #2 - requete de verification du
nom ratee

    if(mysqli_num_rows($namecheck) > 0)
    {
        echo "3: Name already exists"; //code d'erreur #3 - le nom
```

```
existe, impossible de s'enregistrer
        exit();
    }

    //ajout de l'utilisateur a notre base de donnée
    $salt = "\$5\$rounds=5000\$" . "culasse" . $username . "\$";
    $hash = crypt($password, $salt);
    $insertuserquery = "INSERT INTO players (username, hash, salt)
VALUES ('" . $username . "','" . $hash . "', '" . $salt . "');";
    mysqli_query($con, $insertuserquery) or die("4: Insert player query
failed"); //code d'erreur #4 - requete d'insertion ratee

    echo("0");


?>
```

Nous avons recopié ce fichier qui était présent dans le tutoriel vidéo. Mais nous avons compris l'intégralité des manipulations effectuées lors de son exécution. La première fonction "*mysqli_connect*" crée un lien avec la base de données, le premier paramètre de cette fonction prend l'URL de l'hôte, le deuxième paramètre prend l'utilisateur de l'hôte, le troisième prend le mot de passe, et finalement le dernier prend le nom de la base de données. Comme à ce stade, l'hôte est local, il n'y a pas d'URL classique dans les paramètres.

"*mysqli_connect_errno()*" renvoie true si la connexion à la base de données n'est pas possible. La fonction echo en PHP est l'équivalent d'un print que nous connaissons en C#. Chaque sortie comporte un chiffre de référence pour comprendre de quelle erreur il s'agit. Ces sorties renvoyées par le fichier PHP seront captées ensuite par le script de UNITY. Si aucune erreur n'a été rencontrée durant l'exécution, le fichier renvoie 0.

Ensuite, nous avons créé des variables "*username*" et "*password*" qui nous seront utiles pour la suite.

On initialise "*namecheckquery*" comme une requête pour récupérer les données du nom d'utilisateur dans la base. Nous utilisons ensuite "*namecheckquery*" dans la méthode "*mysqli_query*". Cette dernière permet simplement d'appliquer une requête à notre base de données. Si cette action n'est pas réalisable, la méthode "*die*" est appelée et écrit un message et quitte l'exécution du fichier.

Une fois la requête précédente réussie, nous vérifions que le nom d'utilisateur donné par le joueur n'est pas déjà présent dans la base. C'est la

méthode “*mysql_num_rows*” qui va nous aider à le faire car elle renvoie le nombre de lignes d’un élément spécifique (le nom d’utilisateur fourni par le joueur ici).

Si tout s’est bien passé jusqu’ici, nous allons passer à l’ajout de nos données utilisateur à notre base.

Pour respecter un minimum les données et garantir un certain niveau de sécurité, nous avons utilisé le hachage. En effet, nous hachons le mot de passe de l’utilisateur lorsqu’il crée son compte.

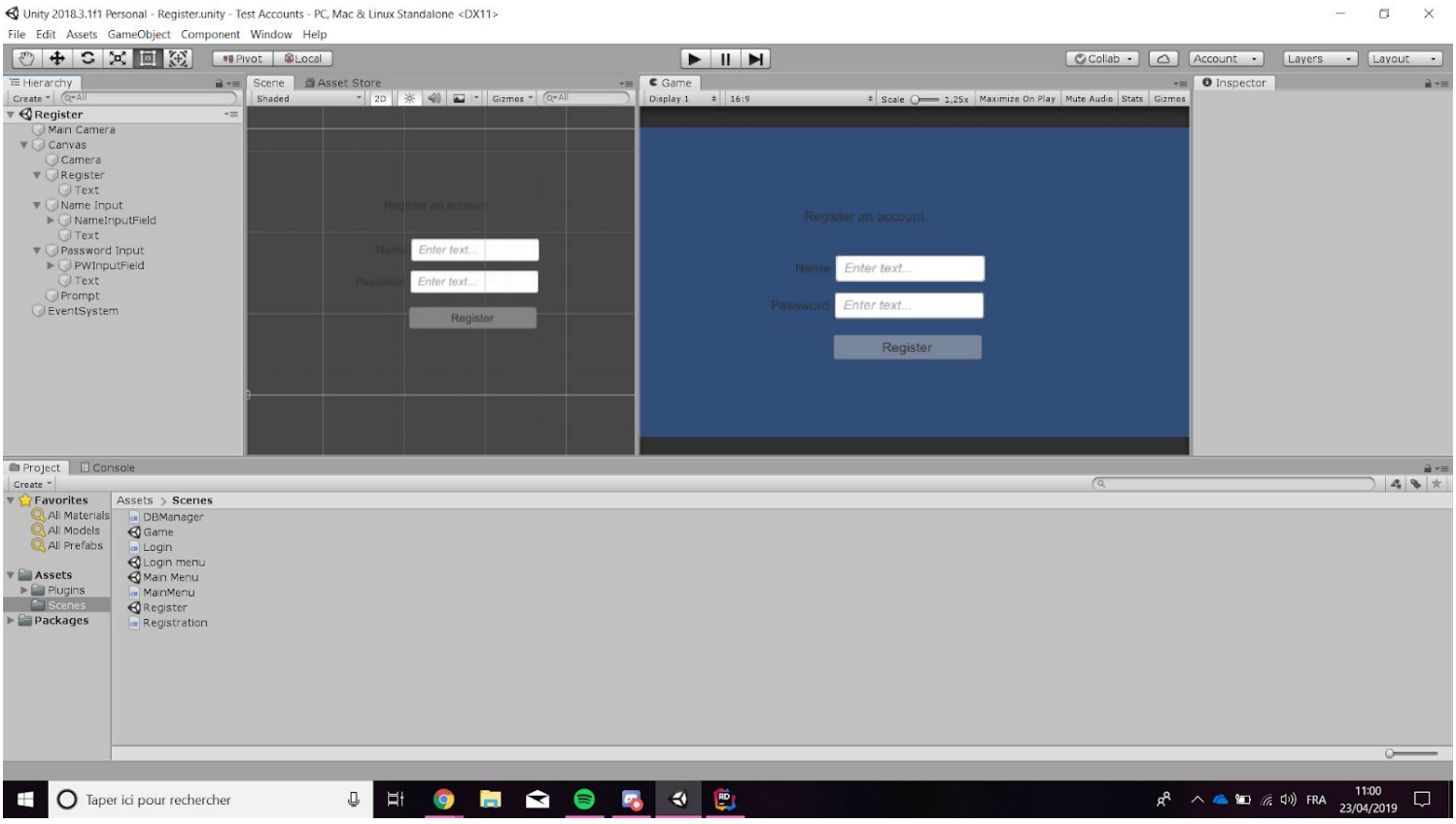
Nous créons d’abord un grain de sel personnel afin de sécuriser l’encryption. Le grain de sel personnel est une petite donnée additionnelle qui renforce significativement la puissance du hachage. Nous en créons un afin éviter d’utiliser un grain de sel préfait facilement trouvable à l’aide de dictionnaires disponible sur internet. Cela compromettrait la sécurité des informations.

La méthode “*crypt*” applique notre grain de sel sur le mot de passe et renvoie sa version encryptée.

Nous pouvons maintenant ajouter le nom d’utilisateur, le grain de sel, et le mot de passe haché à notre base de données. La sécurité est vérifiée car même si une personne avait accès à la base de données, il serait impossible pour cette dernière de récupérer le mot de passe puisque le hachage est une encryption à sens unique.

Si toutes les étapes requises par le fichier PHP ont été réalisées, ce dernier renvoie le code 0. Cette information permettra au script Unity de savoir que l’utilisateur a été créé correctement.

Maintenant, pour ce qui est de Unity au niveau du test, nous avons créé une scène assez basique pour vérifier la bonne communication avec la base de données.



Pour cette scène d'enregistrement nous avons besoin d'un script pour paramétrer les boutons et les champs d'entrée.

```
IEnumerator Register()
{
    WWWForm form = new WWWForm();
    form.AddField("name", nameField.text);
    form.AddField("password", passwordField.text);
    WWW www = new WWW("http://localhost/sqlconnect/register.php", form);
    yield return www;
    if (www.text == "0")
    {
        Debug.Log("User created successfully.");
        UnityEngine.SceneManagement.SceneManager.LoadScene(0);
    }
    else
    {
        Debug.Log("User creation failed. Error #" + www.text);
    }
}
```

D'abord, nous créons un objet WWWform qui va contenir les entrées utilisateur dans les menus du jeu. Ensuite, une fois les données récupérées par le jeu, le script envoie une requête au fichier PHP vu précédemment avec l'objet WWW.

C'est ici, que ce que renvoie le fichier PHP est important, s'il renvoie 0, le script envoie un message au Debug d'Unity pour donner l'information que tout s'est bien passé. Sinon, le script renvoie un message d'erreur.

Maintenant que nos comptes peuvent être enregistrés sur notre base de données, nous devons nous occuper de la partie de connexion.

Nous reprenons une scène Unity assez équivalente à celle d'enregistrement. C'est en majeure partie le script et le fichier PHP qui va changer.

```php
$namecheckquery = "SELECT username, salt, hash, score FROM players WHERE username='" . $username . "';";

$namecheck = mysqli_query($con, $namecheckquery) or die("2: Name check query failed"); //code d'erreur #2 - requete de verification du nom ratee
if(mysqli_num_rows($namecheck)!=1)
{
    echo "5: Either no user with name or more than one"; // code d'erreur #5 - nombre de names != 1
    exit();
}

//recuperation des infos de login
$existinginfo = mysqli_fetch_assoc($namecheck);
$salt = $existinginfo["salt"];
$hash = $existinginfo["hash"]

$loginhash = crypt($password, $salt);
if($hash != $loginhash)
{
    echo "6: Incorrect password"; //code d'erreur #6 - le password qui a ete hash ne correspond pas à celui de la table de la base de données
    exit();
}
```

Il nous faut récupérer toutes les informations de notre utilisateurs (le nom, le grain de sable, le mot de passe haché, et le nombre de pièces).

Nous vérifions ensuite qu'il existe exactement un utilisateur avec le même nom pour qu'il n'y ait pas de problème au niveau des pièces et mots de passe.

Ensuite on hache le mot de passe donné par l'utilisateur lors de sa connexion pour pouvoir le comparer au mot de passe haché présent sur la base de données (c'est le seul moyen de vérifier la validité du mot de passe car le hachage est à sens unique).

Pour finir, nous voulions pousser le principe jusqu'au bout nous avons décidé de mettre en ligne cette base de données, mais nous avons rencontré un soucis : notre site **colors-tsf.me** n'accepte pas les pages dynamique comme le PHP. De ce fait nous avons pris un autre domaine : **colors-tsf.000webhostapp.com**. Il nous permet d'exécuter nos fichier PHP. L'hébergeur nous permet de créer directement une base de données MySQL très facilement. Nous avons donc simplement recopié la base données locale pour recopier les identifiants données par le site dans nos fichiers PHP.

De ce fait nous pouvons nous connecter depuis n'importe quel ordinateur partout dans le monde.

Des informations supplémentaires quant à l'implémentation de cette base de données dans les menus du jeu seront détaillées dans la partie Interface.

2.6 Interface (Alex , Jean-Baptiste, Alexandre)

Après avoir une base de données il nous faut une interface pour interagir avec cette dernière et le jeu.

Nous avons plutôt bien avancé depuis la soutenance dernière. Nous sommes passé d'un simple menu mal optimisé à plusieurs interfaces synchronisées avec la base de données contrôlable à la manette.

En effet nous utilisons la base de données pour garder en mémoire les objets déverrouillés par les joueurs, mais aussi leur avancée dans les niveaux (niveau terminé, terminé à 100%, non fait ou indisponible).

Nous utilisons cette même base pour l'achat des objets et l'affichage du pseudo et du nombre de pièces que le joueur possède.

Pour effectuer cette liaison entre l'interface et la base de données nous devons en premier temps les stocker dans la base. J'ai donc créé 30 colonnes supplémentaires : 15 pour les objets et 15 pour l'état des niveaux. Elles contiennent un seul chiffre dont voici la traduction :

Pour le magasin:	**Pour les niveaux :**
0 : Objet non acheté	-1 : Niveau indisponible
1 : Objet acheté	0 : Niveau non réalisé
2: Objet équipé	1 : Niveau réalisé
	2 : Niveau réalisé à 100%

Une fois la structure de la base de données réalisée nous devons récupérer ces données en fonction de chaque joueur.

Nous avons donc déclarer les variables dans notre fichier PHP ($namecheckquery) que nous allons utiliser pour réaliser cette interface :

```
$namecheckquery = "SELECT username, salt, hash, score, lvl11, lvl12,
lvl13, lvl14, lvl15, lvl21, lvl22, lvl23, lvl24, lvl25, lvl31, lvl32,
lvl33, lvl34, lvl35, skin1, skin2
FROM players WHERE username='" . $username . "';";
```

Cela va nous permettre de savoir si la connexion peut être établie et que les identifiants sont corrects. Tout cela pour ensuite retourner les informations correspondantes au joueurs.

Pour réaliser ces vérifications, il faut que nous séparions les éléments avec un caractère spécial pour faciliter le “Parsing” que nous effectuerons sur UNITY.
De ce fait nous séparons tous les éléments par le caractère ‘\t’ comme montré ci dessous.

```
echo "0\t" . $existinginfo["score"] . "\t" . $existinginfo["lvl11"] .
"\t" . $existinginfo["lvl12"] . "\t" . $existinginfo["lvl13"] . "\t" .
$existinginfo["lvl14"] . "\t" . $existinginfo["lvl15"]
    . "\t" . $existinginfo["lvl21"] . "\t" . $existinginfo["lvl22"] .
"\t" . $existinginfo["lvl23"] . "\t" . $existinginfo["lvl24"] . "\t" .
$existinginfo["lvl25"] . "\t" . $existinginfo["lvl31"]
    . "\t" . $existinginfo["lvl32"] . "\t" . $existinginfo["lvl33"] .
"\t" . $existinginfo["lvl34"] . "\t" . $existinginfo["lvl35"] . "\t" .
$existinginfo["skin1"] . "\t" . $existinginfo["skin2"];
```

Au final les informations sont contenue dans une grande chaine de caractère :

www.text

Nous utilisons donc la fonction Split pour avec chaque informations séparément avec le paramètre : ‘\t’
Nous pouvons donc appliquer la diffusion des données à nos button en changeant leur couleurs en fonction. Pour les données suivantes voici l’interface correspondante :

0	0	1	0	0	2	0	0	1	0	0	2	0	0	-1	-1	-1	0	0

2.7 Editeur de niveaux (Alexandre, Romain)

Lorsque nous avons entamé notre projet nous avions eu comme idée de créer un système d'éditeur de niveau, cet aspect du projet est toujours à l'état d'idée et reste pour nous optionnel.

Au vu du travail nécessaire dans la réalisation du projet dans sa globalité et de l'investissement d'ores et déjà requis, il apparaît que l'éditeur de niveau risque d'être abandonné si sa réalisation nécessite trop de temps et de ressources.

2.8 Scènes UNITY (Alex, Romain)

Depuis la fin de la dernière soutenance nous avions quelques idées avec Romain afin d'améliorer et de faire évoluer nos niveaux. Dans cette partie nous présenterons les deux nouveautés implémentées dans les scènes UNITY de Colors.

Dans un premier temps, nous avons ajouté un fond pour nos niveaux toujours sous la maîtrise du **Flat Design** sur toutes nos créations. Il a fallu inclure un script en C# pour que le fond ne sorte pas de l'écran de jeu puisque notre personnage avance horizontalement.

Par la suite, nous avons ajouté des pièces que le joueur peut récupérer, pour ensuite les échanger dans la boutique de notre jeu. Pour cela nous avons créer un nouveau **symbole graphique** (**Sprite** en anglais). A ce sprite nous avons donc donné l'image d'une pièce de monnaie venant de Flat Icons pour que le rendu soit propre. Pour que le joueur puisse collecter ces pièces il faut donc ajouter une fonction au personnage, qui va détecter quand celui va rentrer en collision avec une pièce et seulement une pièce. Pour cela nous avons utilisé une fonction appelée **OnTriggerEnter2D** présentée ci-dessous :

```
void OnTriggerEnter2D(Collider2D other)
{
   if (other.tag != currentcolor)
   {
       UnityEngine.Debug.Log("Game Over");
   }

   if (other.IsTouching(CollisionPlayer) &&
other.CompareTag("MoneyCoin"))
   {
       Destroy(MoneyCoin.gameObject);
       CoinCounter++;
   }
}
```

Dans toute la première ligne du **If**, on vérifie en premier si le joueur est en collision avec n'importe quel objet. On spécifie si cet objet possède comme **tag**, donc comme nom "**MoneyCoin**", nom que nous avons associé à nos pièces. Si cette condition est vérifiée, alors on utilise la méthode **Destroy** qui est déjà implémentée dans UNITY qui permet tout simplement de faire disparaître un élément de la scène.

Enfin nous avons décidé d'implémenter un compteur de pièce qui apparaît en haut à gauche de l'écran. Pour cela il fallait ajouter un texte de type **UI (User Interface)**. Pour pouvoir actualiser notre compteur de pièce, nous avons simplement ajouté des variables dans notre script du personnage.

```
public Text CurrentCoin;
public int CoinCounter;
```

- La variable **currentcoin** de type **Text** est celle qui est utilisée dans le nouvel élément ajouté à la scène de type **Text**. C'est avec cette variable que l'on peut afficher le compteur sur la scène.

- La variable **coinlvl** qui est donc de type **int** va nous permettre d'actualiser notre compteur à chaque collision avec la fonction vue plus haut.

Ensuite dans la fonction **Update** de UNITY, il nous suffit simplement d'écrire l'égalité suivante qui va donc actualiser le compteur à chaque collision entre le joueur et une pièce.

```
CurrentCoin.text = CoinCounter.ToString();
```

En conclusion de cette partie nous avons bien avancé le niveau où il ne reste plus qu'à ajouter quelques obstacles en plus. Il nous faut également lier avec les scènes représentant les différents menu du jeu.

4. Conclusion

Pour conclure, nous pouvons dire que contrairement à la précédente soutenance, nous avons bien mieux su gérer les problèmes de documentation en allant directement chercher les informations à la source sur le site de UNITY.

Pour le site internet cela c'est aussi fait de manière plus naturelle puisque les plus grandes difficultés résultaient généralement de l'apprentissage des nouveaux langages. Aujourd'hui nous sommes plus familiers à tous ses outils, il ne nous reste à présent plus qu'à apporter des éléments plus de l'ordre du détail.

Néanmoins nous savons que nous ne devons pas nous conforter dans cette état d'esprit et que nous devons encore faire des efforts pour peaufiner et finaliser notre projet.