Projet de Développement Logiciel et Base de Données

Logiciel Avancé – B2A

Projet 1

Serveur de Matchmaking

Projet réalisé par le groupe 17 :

GONZALEZ Marie

ALARD Adrien

Suivi par :

LIMA Alex

Sommaire

GONZALEZ Marie
ALARD Adrien

I. Introduction

1. Choix du sujet

Pour ce projet de Développement Logiciel, nous nous sommes décidés sur le premier sujet, soit le sujet "Serveur de matchmaking".

Nous avons pris cette décision car le deuxième sujet demandait d'avoir une Raspberry Pi que nous ne possédions pas, et nous ne voulions pas prendre le risque de se rendre au Campus avec la pandémique actuelle. Aussi, nous n'avions jamais eu l'occasion de créer un logiciel connectant deux machines et nous étions curieux de nous y essayer.

En effet, nous nous intéressons tous les deux aux jeux-vidéos et avons une affection pour les jeux en ligne, ce projet nous offrait donc la possibilité de comprendre leur fonctionnement et même d'en créer un nous-même.

2. Choix des outils

Nous avons décidé de réaliser ce projet sur le logiciel **Unity**, et plus particulièrement la version **2019.4.12f1**.

Nous avons tous les deux eu l'occasion d'utiliser ce logiciel à travers nos Ydays respectifs et nous pensions qu'utiliser un logiciel que nous connaissions bien nous permettrait d'explorer la création d'un jeu vidéo en ligne plus facilement, créant ainsi un équilibre entre la difficulté et la découverte du sujet d'un côté, et la confiance et l'aise que nous offre le logiciel de l'autre.

Afin de faire la connexion entre les différentes machines qui utiliseraient notre jeu, nous avons utilisé un plug-in de l'Asset Store de **Unity** nommé **Photon PUN 2**.

Nous avons utilisé **XAMMP** pour accéder au serveur **Apache** pour stocker les script **PHP** et **MySQL** pour se servir d'une base de données afin de pouvoir stocker le pseudo et la date, heure, minute ,seconde d'arrivée.

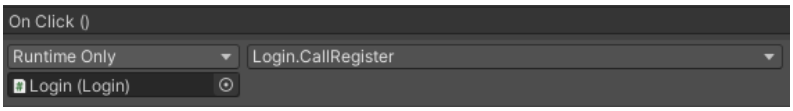
GONZALEZ Marie
ALARD Adrien

II. Ecran de login

1. Etape graphique

Dans la scène de Login, il y a un message d'accueil, un espace où rentrer son pseudo et un bouton qui va servir à envoyer les informations dans la base de données, dans une table qui enregistre les joueurs et leur pseudo.

Une fois le pseudo renseigné, on clique sur le bouton valider et celui-ci lance la fonction **CallRegister()** du script **login.cs** décrite ci-dessous.



GONZALEZ Marie
ALARD Adrien

3. Etapes d'enregistrement

La fonction **CallRegister()** appelle la coroutine **Register.**

```
public void CallRegister()
{
    StartCoroutine(Register());
}

1 référence
IEnumerator Register()
{
    WWWForm form = new WWWForm();
    form.AddField("name", nameField.text);
    form.AddField("hours", System.DateTime.Now.Hour);
    form.AddField("minutes", System.DateTime.Now.Minute);
    form.AddField("secondes", System.DateTime.Now.Second);

    WWW www = new WWW("http://localhost/php/sqlconnect/register.php", form);
    yield return www;
    if (www.text == "0")
    {
        Debug.Log("User created successfully.");
        DBManager.username = nameField.text;
        //UnityEngine.SceneManagement.SceneManager.LoadScene(1);
        UserLogIn();
    }
    else
    {
        Debug.Log("User creation failed. Error#" + www.text);
    }
}
```

On stock le nom, les heures, minutes et secondes dans la base de données, on récupère le nom du **namefield**, c'est-à-dire l'input dans lequel on a écrit notre pseudo, et on rentre le temps d'arrivée en prenant les variables système grâce à "`System.DateTime.Now.`".

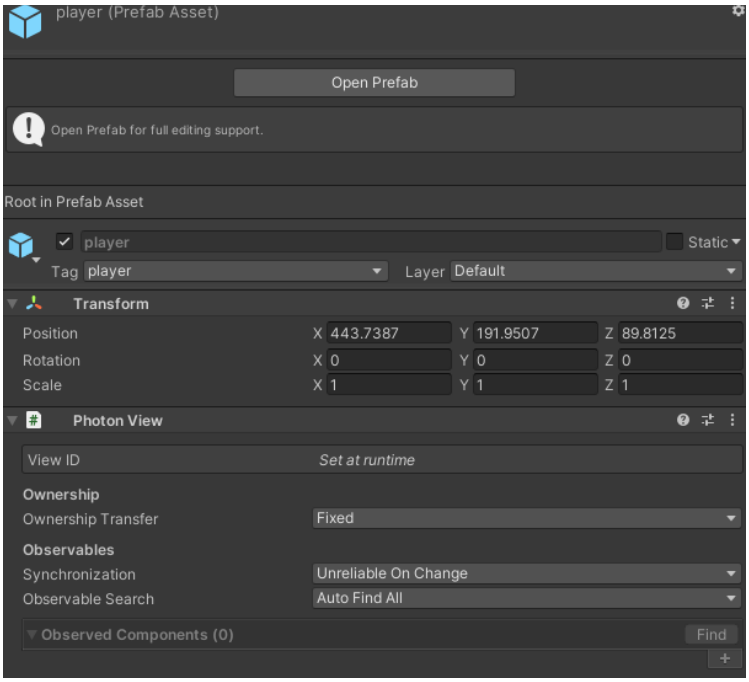
4. La salle d'attente

Une fois login on appelle la fonction **UserLogIn**, ce qui nous amène à l'écran de matchmaking où l'on attend un adversaire.

```
1 référence
public void UserLogIn(){
    LoginCanvas.SetActive(false);
    WaitCanvas.SetActive(true);
    PhotonNetwork.Instantiate("player", Vector3.zero, Quaternion.identity);
}
```

Dans cette fonction, on change de canvas pour afficher celui d'attente et cacher celui du login, puis on créé un **EmptyGameObject** appelé **player** dans le dossier **Resources** avec le composant **PhotonView** et un Tag **player** afin de lister les joueurs facilement.

GONZALEZ Marie
ALARD Adrien



Quand une deuxième personne s'inscrira dans le jeu et rejoindra l'écran d'attente, un nouveau clone de **player** rejoindra la salle et le jeu pourra commencer grâce au script **WaitForPlayer.cs** qui vérifie s'il y a deux **player** dans la salle en les repérant grâce à leur tag **player**.

```
public GameObject[] players;

Message Unity | 0 références
void Update()
{
    players = GameObject.FindGameObjectsWithTag("player");
    if (players.Length >= 2)
    {
        UnityEngine.SceneManagement.SceneManager.LoadScene(1);
    }
}
```

On stock tout les joueur dans un array de GameObject appelés "players". Si la taille de la liste est supérieure ou égale à 2, on lance la scène suivante, c'est-à-dire le jeu avec la ligne de code **`UnityEngine.SceneManagement.SceneManager.LoadScene(1);`**.

GONZALEZ Marie
ALARD Adrien

III. Base de données

1. Création de la base de données

Dans la base de données que nous avons appelée **unityaccess**, il y a une table nommée **players** qui stock les informations suivantes : ID (auto-incrémente), nom, heure, minute, seconde.

	#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra	Action		
☐	1	id	int(10)			Non	*Aucun(e)*		AUTO_INCREMENT	Modifier	Supprimer	Plus
☐	2	username	varchar(16)	utf8mb4_general_ci		Non	*Aucun(e)*			Modifier	Supprimer	Plus
☐	3	heure	int(2)			Non	*Aucun(e)*			Modifier	Supprimer	Plus
☐	4	minutes	int(2)			Non	*Aucun(e)*			Modifier	Supprimer	Plus
☐	5	secondes	int(2)			Non	*Aucun(e)*			Modifier	Supprimer	Plus

Quand on clique sur le bouton pour s'enregistrer, le script créé un formulaire avec les informations nomées ci-dessus grâce à la méthode **WWWForm** et envoie ce formulaire au script **register.php** logé sur le serveur Apache avec la méthode **WWW**.

```
WWWForm form = new WWWForm();
form.AddField("name", nameField.text);
form.AddField("hours", System.DateTime.Now.Hour);
form.AddField("minutes", System.DateTime.Now.Minute);
form.AddField("secondes", System.DateTime.Now.Second);

WWW www = new WWW("http://localhost/php/sqlconnect/register.php", form);
```

Le script **Register.php** permet la connexion à la base de données avec la fonction **mysqli_connect()** ».

```
$con = mysqli_connect('localhost', 'admin', 'admin' , 'unityaccess');
```

On récupère ensuite les informations envoyées par le formulaire avec la méthode **Post**.

```
$username = $_POST ["name"];
$hours = $_POST ["hours"];
$minutes = $_POST ["minutes"];
$secondes = $_POST ["secondes"];
```

Puis, on regarde ensuite si le nom du joueur existe déjà pour ne pas que deux joueurs aient le même nom.

```
$namecheckquery = "SELECT username FROM players WHERE username='" . $username . "';";

$namecheck = mysqli_query($con,$namecheckquery) or die("2 : name check query failed"); //code d'erreur #2 = name check query failed

if (mysqli_num_rows($namecheck) > 0)
{
    echo "3 : name already exists";
    exit();
}
```

GONZALEZ Marie
ALARD Adrien

Si le nom est bien valide, on peut enregistrer les informations dans la base de données.

```
$insertuserquery = "INSERT INTO players (username , heure , minutes , secondes) VALUES ('" . $username ."','" . $hours . "','" . $minutes . "','" . $secondes . "');";
mysqli_query($con, $insertuserquery) or die("4 : insert player query failed");

echo("0");
```

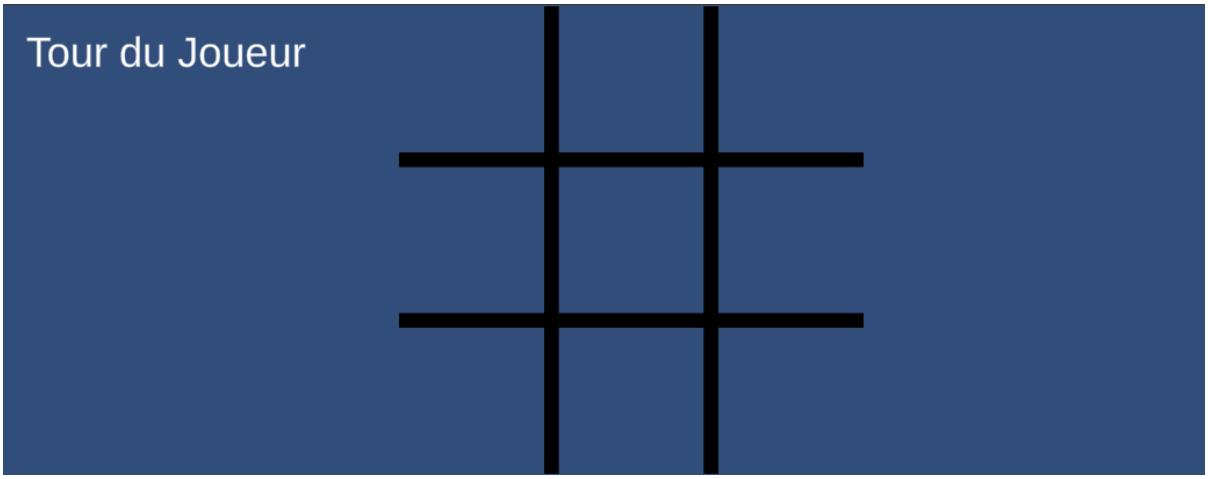
La ligne **echo("0");** est là pour indiquer que tout s'est bien passé et pour ensuite pouvoir continuer grâce à la condition "if" suivante.

```
if (www.text == "0")
{
    Debug.Log("User created successfully.");
    DBManager.username = nameField.text;
    //UnityEngine.SceneManagement.SceneManager.LoadScene(1);
    UserLogIn();
}
else
{
    Debug.Log("User creation failed. Error#" + www.text);
}
```

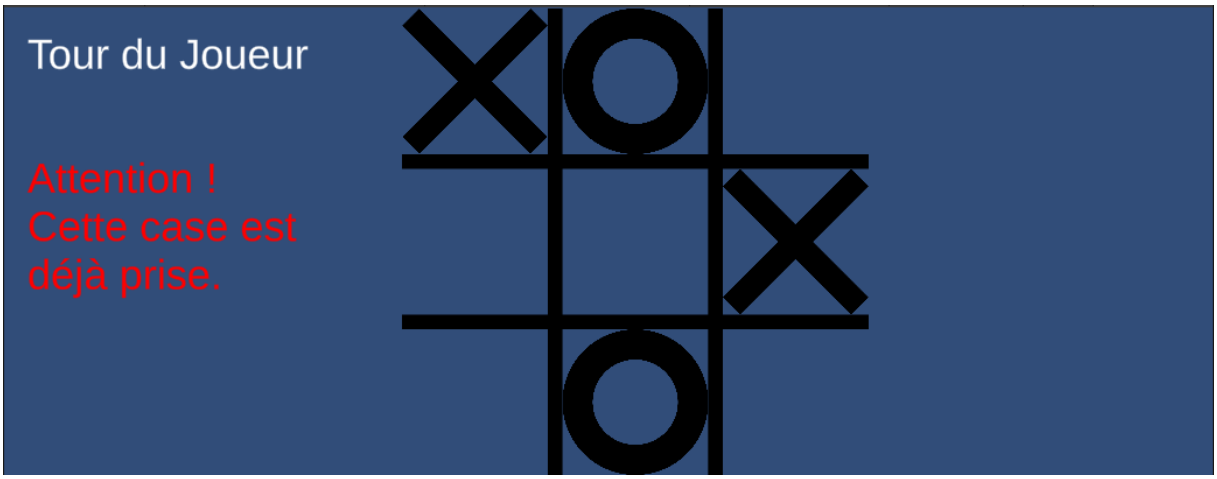
GONZALEZ Marie
ALARD Adrien

IV. Jeu du morpion

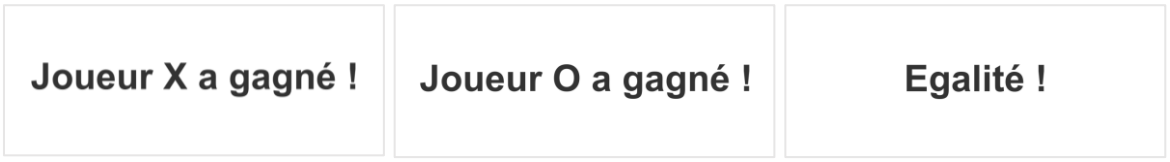
1. L'affichage



L'écran du jeu est très simple et se contente d'afficher le plateau du jeu à peu près au centre de l'écran et un texte à gauche informe l'utilisateur s'il s'agit du "Tour du Joueur" ou du "Tour de l'Adversaire". Les joueurs pourront ensuite remplir le plateau de jeu de cercles et de croix en fonction de leur symbole attitré, comme on peut le voir ici :

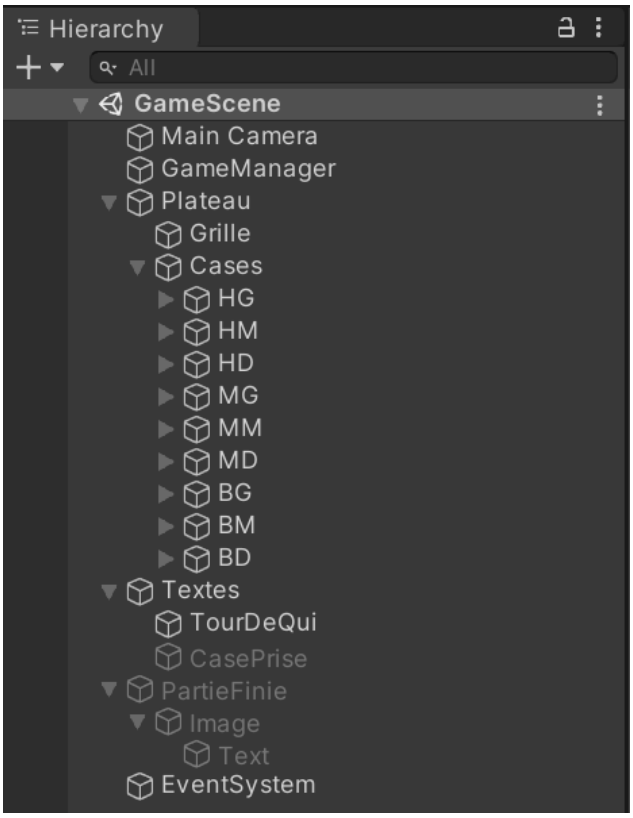


On peut aussi voir sur la capture d'écran ci-dessus que, lorsqu'un joueur clique sur une case du plateau qui est déjà prise, un message s'affiche en rouge pour l'en informer. Ce message disparaitra quelques secondes plus tard. Lorsqu'une partie est finie, on peut voir l'un des trois écrans de fin suivants :
(Dans l'ordre, l'écran de fin quand le joueur avec les croix a gagné, l'écran de fin quand le joueur avec les cercles a gagné, l'écran de fin quand les joueurs sont arrivés à égalité).



GONZALEZ Marie
ALARD Adrien

2. L'architecture des objets



L'objet **Main Camera** est, tout simplement, la caméra qui nous permet de décider exactement la zone de la scène que le joueur verra. Il est ajouté par défaut lors de la création d'une scène.

L'objet **GameManager**, souvent présent dans les jeux-vidéos, est invisible qui contient les scripts gérant le déroulement d'une partie ou les paramètres décidés par le joueur par exemple. Notre **GameManager** contient le script **TourManager.cs** que nous expliciteront plus loin dans cette documentation. Il contient aussi le script NetworkManager qui permet la connexion entre machines grâce au plugin PUN.

L'objet **Plateau** sert à regrouper les objets qui lui sont enfants et qui forment le plateau du jeu. Il contient l'objet **Grille**, une simple image du quadrillage du plateau, et l'objet **Cases**, contenant lui-même des boutons qui correspondent à chaque case du plateau.

Ces boutons sont tous nommés par deux lettres qui correspondent à leur emplacement sur le plateau. Par exemple, l'objet **HG** est un bouton dont le nom correspond à "Haut Gauche", il s'agit donc de la case en haut à gauche du plateau. Chaque objet contenu dans Cases contient le script **ClicCase.cs** que nous expliciteront plus loin dans cette documentation.

L'objet **Textes** est un objet de type "Canvas" qui contient un objet **TourDeQui** qui contient le texte informant l'utilisateur qui doit jouer son tour, et un objet **CasePrise**, désactivée par défaut, qui s'active quand le joueur clique sur une case déjà prise et qui affiche le texte l'en informant.

L'objet **PartieFinie** est un autre objet de type "Canvas" qui contient une image de fond et un texte affichant quel joueur a gagné ou que la partie s'est finie en égalité. Evidemment, cet objet est désactivé par défaut et ne s'active que lorsque la partie est finie.

Enfin, l'objet **EventSystem** est un objet invisible ajouté automatiquement lorsqu'on a un objet de type "Canvas" pour qu'il soit interactif.

GONZALEZ Marie
ALARD Adrien

4. Les scripts – TourManager.cs

Les scripts du jeu de morpion ont été commentés pour plus de précision et d'informations si nécessaire.

a. PlaceImage(string nomBouton)

Dans un premier temps, elle récupère l'objet correspondant à la case ciblée à partir de son nom (pour rappel, le nom des cases est les initiales de sa position. Ex: "HG" correspond à la case en Haut à Gauche). Ensuite, une condition "if" vérifie qui doit jouer grâce à la variable booléenne **AQuiLeTour** qui est "true" pour le joueur O et qui est "false" pour le joueur X.
Selon le résultat, la case ciblée affiche désormais le symbole correspondant au joueur dont c'était le tour grâce à la fonction **ChangeImage(Sprite symbole)** du script **ClicCase.cs** expliqué plus loin dans cette documentation, le texte informant aux utilisateurs quel joueur doit jouer change, la variable **AQuiLeTour** aussi et enfin, le mot "cercle" est renvoyé s'il s'agissait du tour du joueur O, ou le mot "croix" s'il s'agissait du tour du joueur X. Cette valeur sera récupérée dans la fonction suivante.

```
string PlaceImage(string nomBouton){
    GameObject buttonClicked = GameObject.Find(nomBouton);
    if(AQuiLeTour){
        buttonClicked.GetComponent<ClicCase>().ChangeImage(cercle);
        TourDeQui.SetText("Tour du Joueur X");  //Affiche qu'il s'agit maintenant du tour du Joueur X
        AQuiLeTour = !AQuiLeTour;   //Change de tour
        return "cercle";    //renvoie la valeur à enregistrer dans la variable is[NomDeCase]taken
    }else{
        buttonClicked.GetComponent<ClicCase>().ChangeImage(croix);
        TourDeQui.SetText("Tour du Joueur O");  //Affiche qu'il s'agit maintenant du tour du Joueur O
        AQuiLeTour = !AQuiLeTour;
        return "croix";
    }
}
```

b. CheckCase(string nomBouton)

Une condition "switch" vérifie quel est la valeur de **nomBouton** qui correspond au nom de la case ciblée. Si la case ciblée n'est pas prise (vérifié avec les variables dont le nom est au format **is[NomDeLaCase]Taken**), alors la fonction précédente, **PlaceImage(string nomBouton)** est lancée. On enregistre la valeur qu'elle renvoi dans la variable **is[NomDeLaCase]Taken** correspondante à la case ciblée.

```
[PunRPC]
public void CheckCase(string nomBouton){
    if(!finTour){   //Si c'est le tour du joueur
        switch(nomBouton){ //Selon le nom de la case (qui correspond à sa position)
            case "HG":  //Haut Gauche
                if(isHGtaken == null){ //Si la case n'est pas prise
                    isHGtaken = PlaceImage(nomBouton);   //Place le symbole à l'endroit sélectionné et indique
                }else{          //Si la case est prise
                    StartCoroutine(ShowCasePriseText());    //L'indique au joueur avec un message
                }
                break;
            case "HM":  //Haut Milieu
                if(isHMtaken == null){
                    isHMtaken = PlaceImage(nomBouton);
                }else{
                    StartCoroutine(ShowCasePriseText());;
                }
                break;
```

GONZALEZ Marie
ALARD Adrien

c. CheckWin(string symbole)

Cette fonction vérifie simplement, grâce à une condition "if", si le symbole donné en argument possède des cases qui constituent une formation gagnante. Si c'est le cas, la fonction renvoie "true", sinon elle renvoie "false".

```
bool CheckWin(string symbole){
    //Condition if qui vérifie toutes les combinaisons gagnantes possibles pour le joueur donné
    if((isHGtaken == symbole && isHMtaken == symbole && isHDtaken == symbole) ||    //ligne du haut
    (isMGtaken == symbole && isMMtaken == symbole && isMDtaken == symbole) ||       //ligne du milieu
    (isBGtaken == symbole && isBMtaken == symbole && isBDtaken == symbole) ||       //ligne du bas
    (isHGtaken == symbole && isMGtaken == symbole && isBGtaken == symbole) ||       //colonne de gauche
    (isHMtaken == symbole && isMMtaken == symbole && isBMtaken == symbole) ||       //colonne du milieu
    (isHDtaken == symbole && isMDtaken == symbole && isBDtaken == symbole) ||       //colonne de droite
    (isHGtaken == symbole && isMMtaken == symbole && isBDtaken == symbole) ||       //diagonale \
    (isHDtaken == symbole && isMMtaken == symbole && isBGtaken == symbole)){        //diagonale /
        return true;
    }
    return false;
}
```

d. Update()

La fonction **CheckWin(string symbole)** est utilisée dans la fonction **Update()**, une fonction qui est appelée à chaque image. Une variable booléenne **finJeu** permet de vérifier si la partie est finie ("true" si la partie est finie, "false" si elle est toujours en cours). Si la partie n'est pas finie, alors la fonction **CheckWin(string symbole)** est lancée avec, en argument, "croix". Si elle renvoie, "true", c'est que le joueur X a gagné. La partie est finie, l'écran de fin s'affiche et le texte informe les joueurs du gagnant. Si elle renvoie "false", elle est lancée à nouveau mais, cette fois, avec "cercle" comme argument et les mêmes vérifications sont faites. Si elle renvoie "false" avec "cercle" aussi, alors le jeu vérifie si toutes les cases du plateau ont été prises. Si tel est le cas, alors il n'y a plus de coups possibles, il n'y a pas de gagnant, et il y a donc égalité. L'écran de fin s'affiche et informe les utilisateurs de l'issue de la partie.

```
void Update()
{
    if(!finJeu){    //tant que la partie n'est pas finie...
        if(CheckWin("croix")){  //...vérifie si le joueur X a gagné...
            finJeu = true;      //...si oui, la partie est finie,...
            PartieFinie.SetActive(true);    //...un nouvel écran s'affiche...
            gagnant.text = "Joueur X a gagné !";    //...et affiche que le joueur X a gagné
        }
        else if(CheckWin("cercle")){    //...vérifie si le joueur O a gagné et fait la même chose qu'avec X
            finJeu = true;
            PartieFinie.SetActive(true);
            gagnant.text = "Joueur O a gagné !";
        //Si personne n'a gagné mais que toutes les cases sont prises, alors il y a égalité
        }else if(isHGtaken != null && isHMtaken != null && isHDtaken != null && isMGtaken != null &&
        isMMtaken != null && isMDtaken != null && isBGtaken != null && isBMtaken != null && isBDtaken != null){
            finJeu = true;
            PartieFinie.SetActive(true);
            gagnant.text = "Egalité !";
        }
    }

}
```

e. IEnumerator ShowCasePriseText()

Il s'agit tout simplement de la fonction qui affiche le message informant le joueur qu'il ne peut pas prendre la case ciblée car elle est déjà prise, et qui retire le message au bout de deux secondes.

```
IEnumerator ShowCasePriseText(){
    CasePriseText.SetActive(true);  //Affiche le texte
    yield return new WaitForSeconds(2); //Attend 2 secondes
    CasePriseText.SetActive(false); //Désactive le texte
}
```

5. Les scripts - ClicCase.cs

Pour rappel le script **ClicCase.cs** se trouve sur chaque bouton des cases du plateau.

Il contient notamment la fonction **ChoixCase()** qui permet tout simplement de lancer la fonction CheckCase(string nomBouton) du script TourManager.cs mais à partir de l'objet sur lequel l'utilisateur a cliqué, ce qui permet de récupérer le nom de la case. Il est aussi lancé via le plugin PUN, que nous expliqueront dans la partie suivante de cette documentation.

```
public void ChoixCase(){
    tourManager.photonView.RPC("CheckCase", RpcTarget.All, gameObject.name);
}
```

La fonction **PlaceImage(string nomBouton)** décrite plus tôt dans cette documentation utilise la fonction suivante, **ChangeImage(Sprite symbole)**. Cette fonction change l'image affichée dans la case sur laquelle se trouve le script en fonction du "Sprite" mis en argument et l'active pour qu'elle soit visible à l'utilisateur.

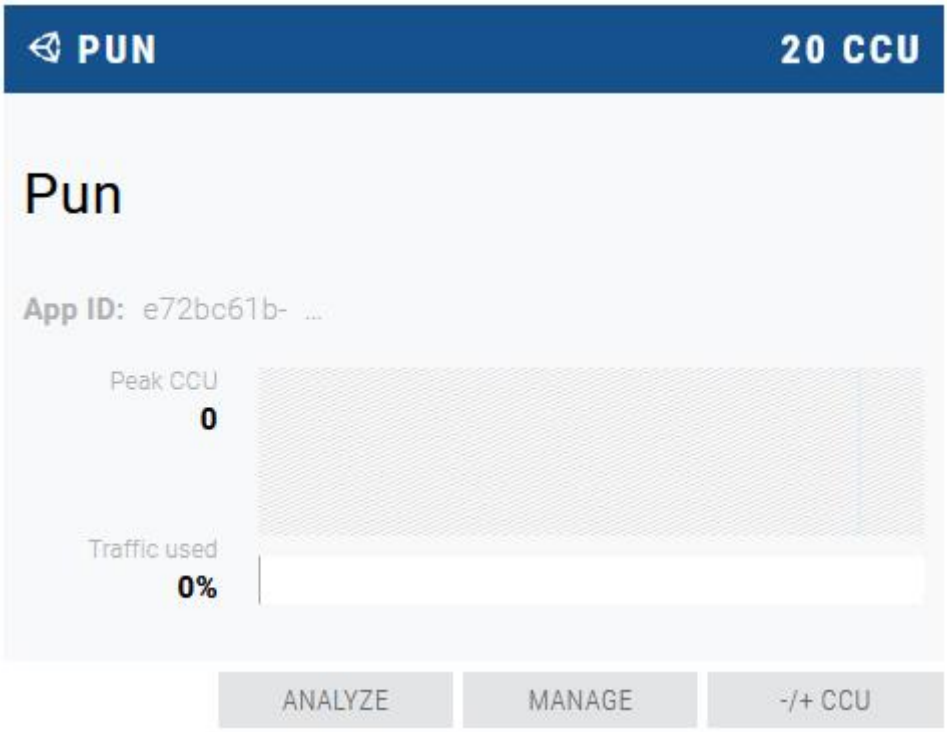
```
public void ChangeImage(Sprite symbole){
    image.enabled = true;   //active l'image
    image.sprite = symbole; //met l'image du symbole donné (X ou O)
}
```

GONZALEZ Marie
ALARD Adrien

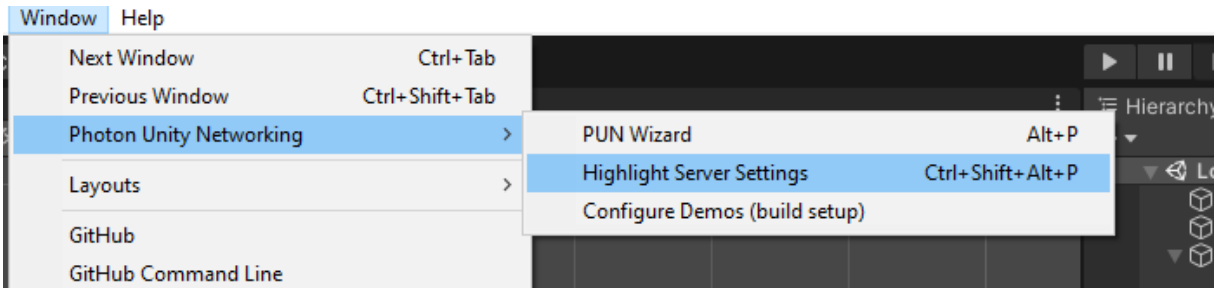
V. Connexion des machines

1. Setup de Photon

Photon étant présent dans l'asset store de Unity, nous avons pris la version gratuite de **Photon Pun 2**. une fois les packages installés, il faut se créer un compte sur Photon Engine et créer ensuite le serveur. Après avoir cliqué sur le bouton « Create new app », on choisira l'application PUN. L'application de notre projet ressemble donc à ça :



Ce qui nous intéresse dans l'application est l'AppID qu'il faudra renseigner sur Unity pour que celui-ci puisse se connecter à un serveur en se rendant sur **Windows** -> **Photon Unity Networking** -> **Highlight Server Settings**.



GONZALEZ Marie
ALARD Adrien

2. Connexion avec les scripts

Nous avons réalisé la connexion entre les différentes machines grâce au plugin PUN de Photon.

```
void Start()
{
    ConnectToServer();
}

void ConnectToServer()
{
    PhotonNetwork.ConnectUsingSettings();
}

public override void OnConnectedToMaster()
{
    base.OnConnectedToMaster();
    RoomOptions roomOption = new RoomOptions();
    roomOption.MaxPlayers = 10;
    roomOption.IsVisible = true;
    roomOption.IsOpen = true;
    PhotonNetwork.JoinOrCreateRoom("Room 1", roomOption, TypedLobby.Default);
}

public override void OnJoinedRoom()
{
    base.OnJoinedRoom();
}

public override void OnPlayerEnteredRoom(Player newPlayer)
{
    base.OnPlayerEnteredRoom(newPlayer);
}
```

D'abord, nous devions placer le script **NetworkManager.cs** sur le "GameObject" qui possédait aussi le script qui initialisera la variable nécessaire au plugin. Pour le jeu du morpion, il s'agit du **GameManager** qui contient le script **TourManager.cs** qui initialise la variable **PhotonView photonView**.

Le script **NetworkManager.cs** contient des fonctions permettant de renseigner les paramètres par défaut et de se connecter au serveur.

Pour utiliser le plugin PUN dans nos scripts, il faut d'abord importer la librairie **Photon.Pun** et initialiser un objet de type **PhotonView**.

```
using Photon.Pun;

public class TourManager : MonoBehaviourPun
{
    public PhotonView photonView;  //instancie photon
```

Pour qu'une fonction soit prise en compte par le plugin comme étant une fonction à réaliser sur d'autres machines connectée, il faut la précéder de **[PunRPC]**. C'est le cas notamment de la fonction **CheckCase(string nomBouton)**.

```
[PunRPC]
public void CheckCase(string nomBouton){
```

Ensuite, pour la lancer, il faut utiliser la fonction **RPC** de l'objet de type **PhotonView** qui prend en argument le nom de la fonction à lancer (ici, **CheckCase**), les machines connectées sur lesquelles lancer la fonction (ici, toutes les machines), puis les arguments de la fonction (ici, **string nomBouton**).

```
tourManager.photonView.RPC("CheckCase", RpcTarget.All, gameObject.name);
```