Projet Unity 3D

G. Chierchia L. Buzer

L'objectif de ce document est de vous guider dans la création d'un simple jeu de bowling. On vous fournit un projet « coquille vide » avec toutes les ressources nécessaires pour créer une version basique du jeu. Le travail est organisé en plusieurs étapes qui touchent progressivement tous les aspects du jeu.

1. Démarrage

La création du jeu vidéo a lieu dans la scène `Game` (située dans le dossier `Assets/Scenes`), qui contient déjà une piste de bowling, une caméra, une lumière, et des éléments GUI. Notez la présence de l'entité `Bowling`, qui est tout simplement un « dossier » recueillant toutes les entités du jeu (CONSEIL : mettez ici toutes les entités que vous créerez dans la suite).

Pour le moment, la seule entité du jeu présent dans la scène est `Floor`. Cette entité a été créée à partir de la primitive `Cube` de Unity. Par default, un cube est de taille 1m x 1m x 1m. Etant donné que l'attribut « scale » de l'entité `Floor` est x = 1.06, y = 0.1 et z = 19.5, on peut déduire que la largeur de la piste est 1.06 m, alors que la longueur est 19.5 m. Cela correspond à la taille standard d'une piste de bowling. Egalement, la piste a été décalée de 19.5 / 2 = 9.75 m sur l'axe Z et -0.1 / 2 = -0.05 sur l'axe Y, afin de centrer le début de la piste dans la position (0, 0, 0). Notez également que nous avons associé cette entité au *tag* « Floor » (créé par nous-mêmes) et au *layer* « Ignore Raycast » (déjà existant dans Unity).

L'entité `Floor` contient deux « Box Colliders » qui permettent de soutenir les quilles et de faire glisser la boule à l'aide du moteur physique. Les deux colliders couvrent respectivement le 60% et 40% de la piste. La différence entre les deux est dans les matériels physiques (« oil » et « dry »), qui spécifient le fait que la piste est huilée sur les premiers 12-13 m. Cette différence de rugosité est à l'origine de l'effet courbe de la boule vers la fin de la piste (la boule va tourner sur elle-même sur la surface huilée, puis « accrocher » sur le sec pour se mettre à rouler, entraînant un changement de trajectoire).

Remarquez aussi la présence de l'entité `Grid`. Elle est un carré de taille 1.06 m x 1.06 m qui marque la position des quilles sur la piste.

1.1 Création de la boule de bowling

D'abord, il faut créer un `GameObject` vide nommé `Ball`, faire un reset du composant « Transform », ajouter le *tag* « Ball » (qu'il faut créer auparavant en cliquant sur « *add tag…* »), spécifier le layer « Ignore Raycast », et l'inclure dans l'entité-dossier `Bowling`. Ensuite, il faut ajouter les composants suivants :

- « Mesh Filter », pour indiquer le maillage « Ball » ;
- « Mesh Renderer », pour indiquer le matériel « Ball » et le shader « specular » ;

- « Sphere Collider », pour indiquer le matériel physique « HardBall » ;
- « Rigidbody », où il faut spécifier Mass = 7, Drag = 0.05, Angular Drag = 0.05, Use Gravity = true, Is Kinematic = true, Interpolate = None, Collision Detection = Continuous Dynamic ;
- « Audio Source », pour indiquer l'audio clip « Bowling_Ball_Rolling » ;
- Script « BallSound ».

En comparant la boule avec la primitive `Sphere`, on peut s'apercevoir qu'elle a un diamètre de 1 m. Donc, il suffit de changer l'attribut « scale » avec x = 0.22, y = 0.22 et z = 0.22 pour avoir une boule réglementaire de 22 cm. Egalement, changez l'attribut « rotation » avec x = 180 pour tourner les trous de la boule vers la camera, et changez l'attribut « position » avec y = 0.15 pour soulever légèrement la boule de la piste.

Pour vérifier que tout marche bien, on peut créer un simple script de test. Ajoutez un nouveau script à la boule, et nommez-le `TestBall`. Dans la classe homonyme, éditez la fonction `Update()` comme suive :

```
void Update ()
{
	Rigidbody body = GetComponent<Rigidbody>();
	if (Input.GetMouseButtonDown (0))
	{
		body.isKinematic = false;
		body.AddForce( 50*Vector3.forward, ForceMode.Impulse );
	}
}
```

Démarrez le jeu, et dès que vous cliquez sur le bouton de la souris, la boule sera lancée sur la piste. On remarque tout de suite un petit ennui : la boule se plonge dans la piste de manière pas réaliste. Pour fixer cela, fermez le jeu, allez dans **Edit > Project Setting > Physics**, et diminuez le paramètre « Min penetration for penalty » (par exemple, mettez 0.0001). Lancez le jeu, et cette fois tout devrait marcher.

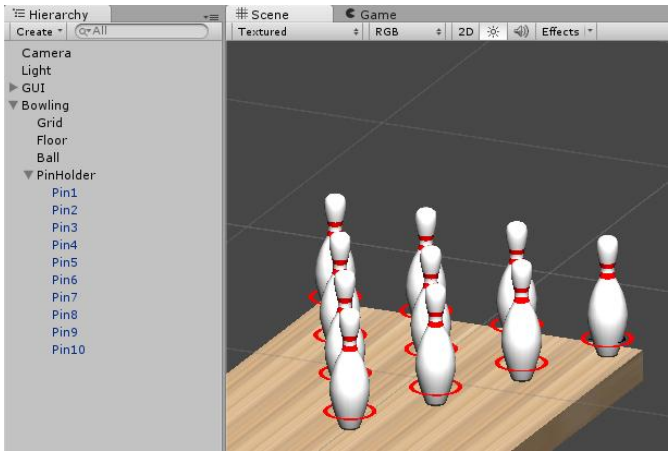
1.2 Création du prefab « quille »

La procédure pour créer les quilles est similaire. Commencez par créer un `GameObject` vide nommé `Pin`, faire un reset du composants « Transform », associer le *tag* « Pin » (qu'il faut créer auparavant en cliquant sur « *add tag…* »), spécifier le layer « Ignore Raycast », et ajouter les composants suivants :

- « Mesh Filter », pour indiquer le maillage « Pin » ;
- « Mesh Renderer », pour indiquer trois matériels : « Pin_Gum », « Pin » et « Pin_Red » ;
- « Mesh Collider », où il faut cocher la case « Convex » et indique le matériel « BouncyPin » ;
- « Rigidbody », où il faut spécifier Mass = 1.5, Drag = 0.1, Angular Drag = 0.2, Use Gravity = true, Is Kinematic = false, Interpolate = None, Collision Detection = Continuous ;
- « Audio Source », pour indiquer l'audio clip « Bowling_Pin_Hit » ;
- Script « PinSound ».

Ajuster la rotation et l'échelle de la quille pour atteindre sa taille réglementaire de 38 cm en hauteur et 12 cm en largeur (aidez-vous avec un cube de taille 0.12 x 0.38 x 0.12). Normalement, les bons paramètres de l'attribut « scale » de l'entité `Pin` sont x = 0.065, y = 0.065 et z = 0.06. Faire glissez l'entité dans le dossier **Assets > Prefabs** pour créer le prefab, et ensuite supprimer l'entité de la scène.

Maintenant, il ne vous reste que placer les 10 quilles aux bons endroits. Commencez par créer une entité vide nommé `PinHolder`, faire un reset du composant « Transform », associer le *tag* « PinHolder » (qu'il faut créer auparavant en cliquant sur « *add tag…* »), spécifier le layer « Ignore Raycast », et l'ajouter dans l'entité-dossier `Bowling`. Ensuite, ajoutez 10 quilles dans l'entité qu'on vient de créer et placez-les sur les positions marquées sur la piste. Voici comment la scène se présent après l'ajout des quilles.



Pour vérifier que tout marche bien, démarrez le jeu et lancez la boule. Le moteur physique de Unity fera le reste. Une petite remarque sur la gestion des collisions. Essayez d'augmenter la force appliquée à la boule, par exemple :

```
body.AddForce( 300*Vector3.forward, ForceMode.Impulse );
```

Est-ce que la boule passe à travers les quilles sans les abattre ??? Si cela est le cas, il faut diminuer le paramètre « Fixed Timestep » placé dans **Edit > Project Setting > Time** (la valeur 0.001 devrait suffire).

2. Mécanique du jeu

Après la mise en place des entités principales du jeu, on peut passer à l'implémentation des mécaniques basiques du bowling. En particulier, les fonctions permettant de contrôler la boule et les quilles sont placées dans les scripts `BallController` et `PinsController` fournis avec le projet. L'objectif de cette étape est d'implémenter ces scripts en suivant les indications reportées dans les sections suivantes.

Avant de procéder à la phase de codage, c'est conseillé de rédiger un script de test. Plus précisément, l'idée est de faire du développement « piloté par les tests », qui comporte les phases suivantes :

1. écrire un premier test ;
2. vérifier qu'il échoue (car le code à tester n'existe pas) ;
3. écrire le code suffisant pour passer le test ;
4. vérifier que le test passe ;
5. puis, éventuellement, améliorer le code tout en gardant les mêmes fonctionnalités.

Dans notre cas, cela se traduit dans les actions suivantes :

- ajoutez le script `BallController` à l'entité `Ball` (et supprimez l'ancien script `TestBall`), ainsi que le script `PinsController` à l'entité `PinHolder`.
- ajoutez un script nommé `TestController` à l'entité `Bowling`, et éditez-le comme suive :

```
public class TestController : MonoBehaviour {
    BallController ball;
    PinsController pins;

    GUIText score;
    GUIText help;

    bool aim_mode;

    void Start () {
        ball  = GameObject.Find("Ball")      .GetComponent<BallController>();
        pins  = GameObject.Find("PinHolder").GetComponent<PinsController>();
        score = GameObject.Find("Score")     .GetComponent<GUIText>();
        help  = GameObject.Find("Help" )     .GetComponent<GUIText>();

        aim_mode = true;
        help.text = "Click to proceed.";
    }

    void Update ()
    {
        score.text = "Knocked-out pins: " + pins.KnockedOut();

        if( aim_mode )
        {
            ball.FollowMouse();

            if (Input.GetMouseButtonUp (0))
            {
                  aim_mode = false;
                  ball.Shoot(Vector3.forward, 50, 0);
            }
        }
        else
        {
            if( ball.HasDone() && pins.HasDone()&& Input.GetMouseButtonUp(0) )
            {
                  aim_mode = true;
                  ball.Reset();
                  pins.RemoveKnockedOut();

                  if( pins.AllDown() )
                     pins.Reset();
            }
        }
    }
}
```

Analysez attentivement la fonction `Update()`. Au départ du jeu, on est dans l'état « mire », et donc la boule suit la souris. Avec un clic de souris, on lance la boule et on passe à l'état « tir ». Dans cet état, on attend que le tir soit terminé et, lors d'un clic de souris, on remet la boule et les quilles à leur place, et on revient à l'état « mire ».

2.1 Contrôle de la boule

Les fonctions permettant de contrôler la boule sont les suivantes :

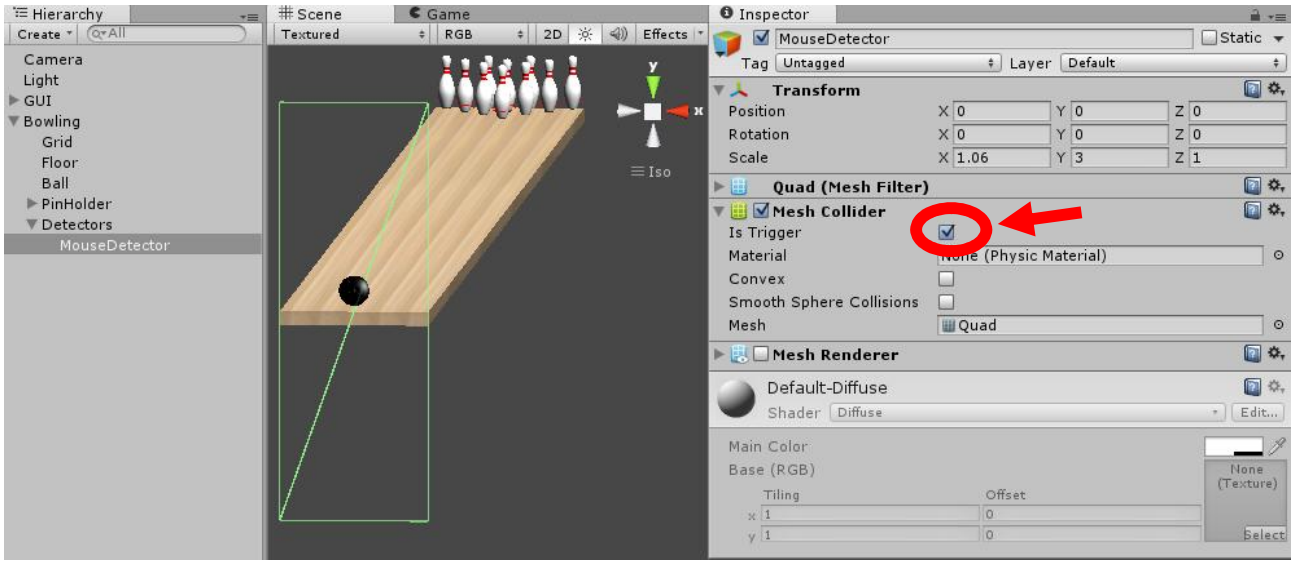
```
public class BallController : MonoBehaviour
{
  public void Reset() { // remettre la boule à son état initial }

  public void FollowMouse() { // modifier la position de la boule }

  public void Shoot(Vector3 direction, float strength, float torque) {
      // lancer la boule vers les quilles }

  public bool HasDone() { // vérifier si la boule a quitté la piste }
}
```

Voici quelques conseils pour l'implémentation.

- Pour récupérer la position d'une entité lors de la création de la scène, vous pouvez vous servir du script `ResetPosition` fourni avec le projet. Attachez-le à toutes les entités dont vous souhaitez mémoriser leurs positions initiales. Pour remplacer les positions courantes avec celles d'origine, il suffit d'appeler la fonction `Reset()` du script `ResetPosition` de la manière suivante :

  ```
  GetComponent<ResetPosition> ().Reset ();
  ```

- Pour retrouver la position de la souris, vous pouvez vous servir du script `FollowMouse`, lequel fait partir un rayon de la position où le pointeur de la souris est situé. Ce script nécessite d'un trigger placé devant la caméra afin d'intercepter le rayon. Voici un exemple :



Pour déplacer la boule, il ne suffit qu'utiliser la position horizontale de la souris.

- Pour lancer la boule, vous pouvez reprendre le code vu dans le script `TestBall`, sans oublier de remplacer le vecteur `Vector3.forward` par la direction de tire en entrée à la fonction. Egalement, vous pouvez imprimer à la boule un effet rotatif au travers de la fonction `addTorque`. Dans tous les cas, n'oubliez pas de mettre à faux l'attribut booléen `isKinematic` (et de le remettre à vrai lors d'un reset), sinon le moteur physique n'agira pas sur la boule.

- Pour vérifier que la boule a quitté la piste, vous pouvez vous servir de l'attribut booléen `render.isVisible`, qui est vrai lorsque la boule est hors de vue.

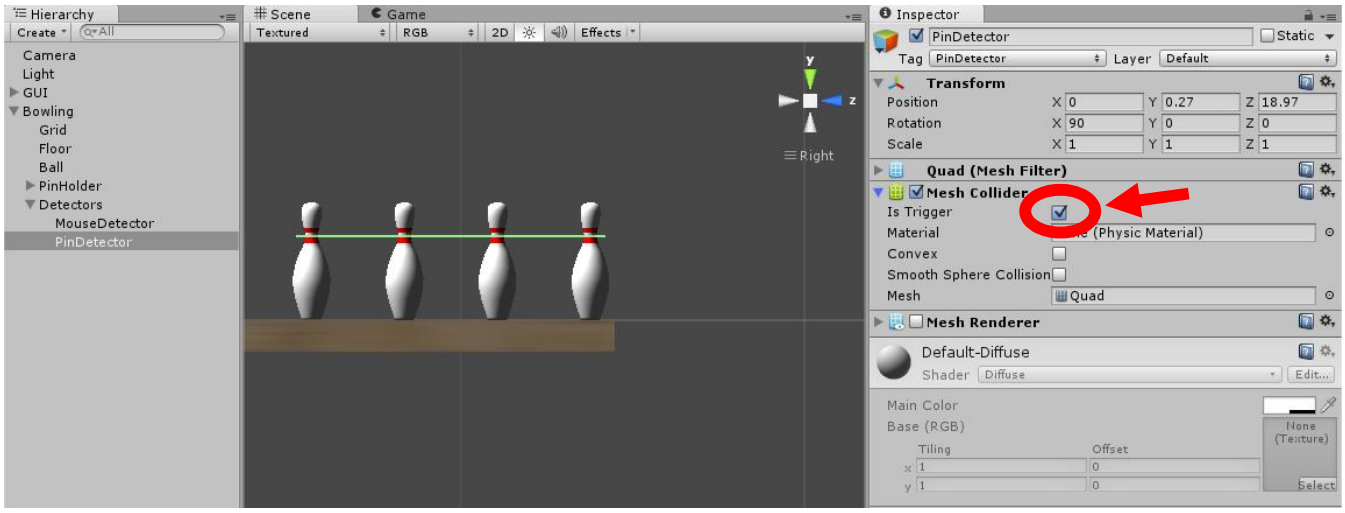
2.2 Contrôle des quilles

Les fonctions permettant de contrôler les quilles sont les suivantes :

```
public class PinsController : MonoBehaviour
{
  public void Reset() { // remettre les quilles à leur état initial }

  public int KnockedOut () { // compter le nombre de quilles abattues }

  public bool AllDown () { // vérifier que toutes les quilles sont abattues }

  public bool HasDone() { // vérifier que les quilles ne bougent pas }

  public bool RemoveKnockedOut () { // faire disparaitre les quilles abattues }
}
```

Voici quelques conseils pour implémenter les fonctionnalités des quilles.

- Pour détecter les quilles abattues, on peut placer un trigger sur leur tête :



De cette manière, on peut vérifier qu'une quille est encore début avec l'instruction suivante :

```
GameObject pin      = ...;
GameObject detector = ...;
if( pin.collider.bounds.Intersects(detector.collider.bounds) ) {...}
```

- Etant donné que chaque fonction du script agit sur les quilles et le détecteur, l'approche la plus simple est de créer des champs d'instance initialisés dans la fonction `Start()` :

```
GameObject[] pins;
GameObject detector;
void Start () {
      pins     = GameObject.FindGameObjectsWithTag("Pin");
      detector = GameObject.FindGameObjectWithTag("PinDetector");
}
```

- Pour retirer une quille abattue, on peut désactiver l'entité correspondante :

```
pin.gameObject.SetActive(false);
```

et pour vérifier si l'entité est active, on peut tester l'attribut associé :

```
if( pin.gameObject.activeSelf ) { ... }
```

- Dans la fonction `KnockedOut()`, il faut faire attention à que le comptage des quilles abattues ne tient pas en compte de celles déjà désactivées.

3. Dynamique du jeu

Après l'implémentation des mécaniques basiques, on peut passer à la gestion des règles du bowling. Une partie de bowling compte 10 *frames*. Chaque joueur lance deux boules à chaque frame (sauf en cas de strike). Les comptage des points est basé sur quatre règles :

- en cas de strike (toutes les quilles abattues au premier tir), le nombre de points attribué au joueur est 10 plus le nombre de quilles abattues avec les deux tirs suivants ;
- en cas de spare (toutes les quilles abattues avec les deux tirs consécutifs du carreau), le nombre de points attribué est 10 plus le nombre de quilles abattues avec le tir suivant ;
- sinon, le nombre de points est égale au nombre de quilles abattues.
- le dixième carreau est particulier : le strike donne le droit à deux tirs supplémentaires, alors que le spare donne le droit à un tir supplémentaire.

Le comptage des points est déjà implémenté dans le script `Scoreboard` fourni avec le projet. Voici quelques fonctionnalités offertes par ce script :

```
public class Frame {
   public int ball1; // quilles abattues au premier tir
   public int ball2; // quilles abattues au deuxième tir
   public int ball3; // quilles abattues au troisième tir (pour le carreau 10)
   public int total; // score total (il peut inclure les tirs futurs)

   public void Clear();    // reset des variables
   public bool IsStrike(); // vrai si ball1 = 10
   public bool IsSpare();  // vrai si ball1 + ball2 = 10

}
```

```
public class Scoreboard
{
   public Frame[] scores;           // les 10 carreaux d'un match

   public void Clear();             // reset des carreaux

   public bool IsEnded (int frame); // vrai s'il n'y a pas d'autres tirs

   public void SetScore(int frame, int rolls, int count);
   // calcule le score en fonction du nombre de quilles abattues
   // INPUTS
   //   frame = indice du carreau courant (entier entre 0 e 9)
   //   rolls = indice du tir courant (1=ball one, 2=ball two, 3=ball three)
   //   count = nombre de quilles abattues avec le tir (entier entre 0 e 10)
}
```

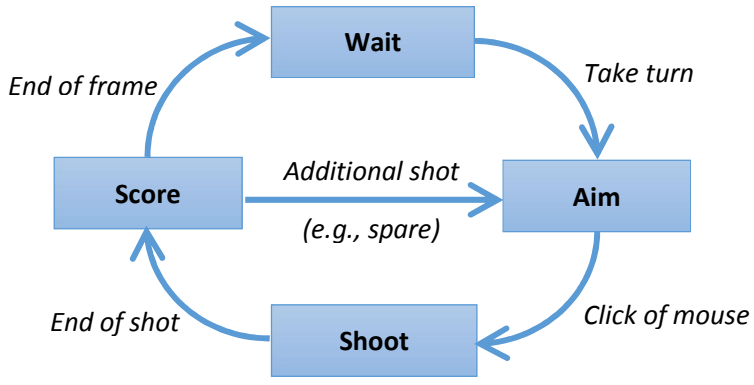
L'objectif de cette étape est d'arriver à gérer correctement le déroulement d'un match avec un seul joueur. Avant de continuer, n'oubliez pas de supprimer le script `TestController` dans l'entité `Bowling`.

3.1 Gestion d'un joueur

Le comportement d'un joueur de bowling peut être représenté par un automate fini à 4 états :

- **Wait** : le joueur attend son tour ;
- **Aim** : le jouer prendre la mire ;
- **Shoot** : la boule est lancée vers les quilles ;
- **Score** : les points sont calculés en fonction des quilles abattues.

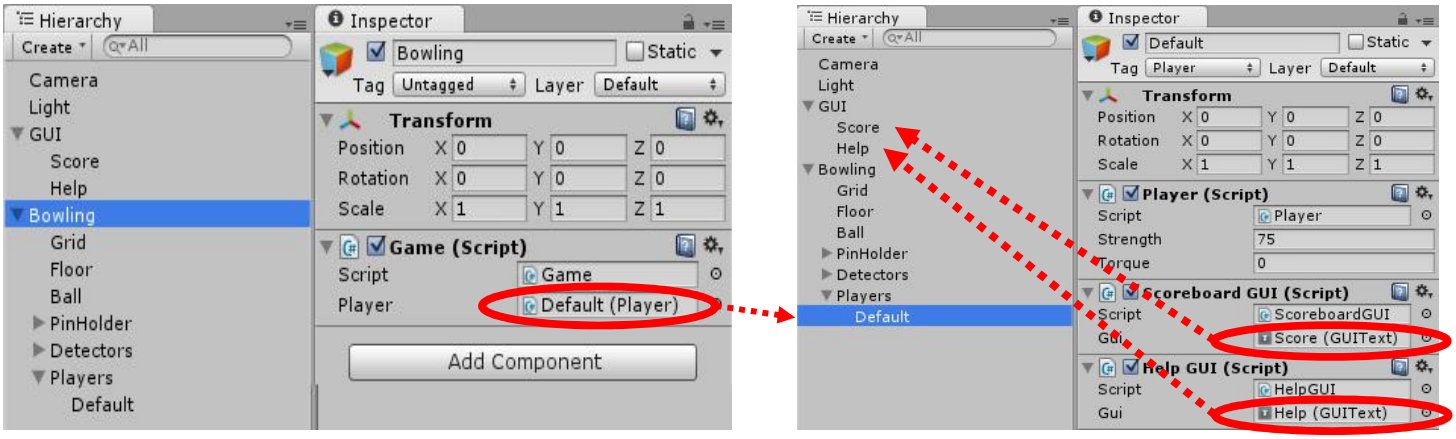
Voici le diagramme des transitions associé à un joueur :



Complétez le script `Player` (fournit dans le projet) avec l'implémentation de la machine à états ci-dessus.

3.2 Testing

Pour vous aider dans l'implémentation du script `Player`, le projet contient déjà le script `Game` pour gérer un match à un seul joueur. Dans la scène, créez une entité vide nommée « DefaultPlayer », à laquelle vous ajouterez les scripts `Player`, `ScoreboardGUI`, et `HelpGUI`. Ensuite, sélectionnez l'entité « Bowling », ajoutez le script `Game` et initialisez l'attribut `Player` avec l'entité « DefaultPlayer ». Voici un exemple :



Si le script `Player` est bien implémenté, vous serais capable de jouer un match complet de bowling, avec le score qui s'affiche en haut de l'écran, et des petits messages d'aide qui apparaissent sur la gauche.

4. Perfectionnement du jeu

Les trois premières étapes couvrent les aspects basilaires du bowling. A partir d'ici, vous pouvez procéder dans plusieurs directions, afin d'améliorer le jeu. Voici quelques idées.

- On peut améliorer la scène en rajoutant les gouttières sur les deux côtés de la piste, la zone d'approche devant la piste, et une sorte de protection au fond de la piste pour éviter que les quilles et la boule tombent dans le vide (le projet contient des modelés 3D qui peuvent être utilisés).

- On peut ajouter une deuxième camera pour avoir une vue du haut des quilles.

- On peut créer des effets d'animation avec la camera, comme suivre la boule sur la piste.

- On peut ajouter un bouton qui permet de visualiser un « replay » du dernier tir effectué.

- On peut modifier le script `Game` afin d'introduire l'aspect multi-joueurs. Ensuite, on peut implémenter un simple script d'intelligence artificielle pour jouer des matchs contre l'ordinateur.

- On peut introduire une interface « homme-machine » pour mieux contrôler le lancer de la boule, y compris la direction, la puissance, et l'effet rotatif de la boule.