TD3 : Ray casting et interactions entre objets

Rémy Frenoy

Yann Soullard

Florian Jeanne

Retour sur les TDs 1 et 2

Lors des deux premières séances de TD, nous avons appris à créer différents types d'objets dans Unity, à les déplacer de différentes manières (*AddForce*, *Translate*). Nous en connaissons un petit peu plus sur la gestion de la physique (*rigidbody, colliders, triggers*). Nous connaissons plusieurs fonctions de mise à jour (Update, FixedUpdate, LateUpdate). Nous avons également travaillé sur le positionnement de la caméra (fixe, héritage d'un objet, suivi des translations via un script dédié).

Standard Assets

Unity dispose de packages "standards" correspondant à des éléments de base utilisés dans la majorité des projets. Ces éléments sont disponibles depuis **Assets > Import packages**. Si ces packages sont très utiles, ils doivent être utilisés à bon escient, et il est important de connaître le fonctionnement global des éléments que l'on importe dans un projet. Nous allons aujourd'hui utiliser le package "Character". Après la création d'un nouveau projet "TD3", importez le package : **Assets > Import packages > Characters**. Inutile d'importer tout le contenu du package, l'ensemble des éléments contenus dans les dossiers *Editor, FirstPersonCharacter, CrossPlatformInput* et *Utility* sont suffisants pour aujourd'hui.

Une fois importés, les standards assets sont présents dans la fenêtre *Project*, dans l'arborescence *Assets/Standard Assets*. Vous pouvez à présent glisser-déposer n'importe quel élément importé dans votre scène.

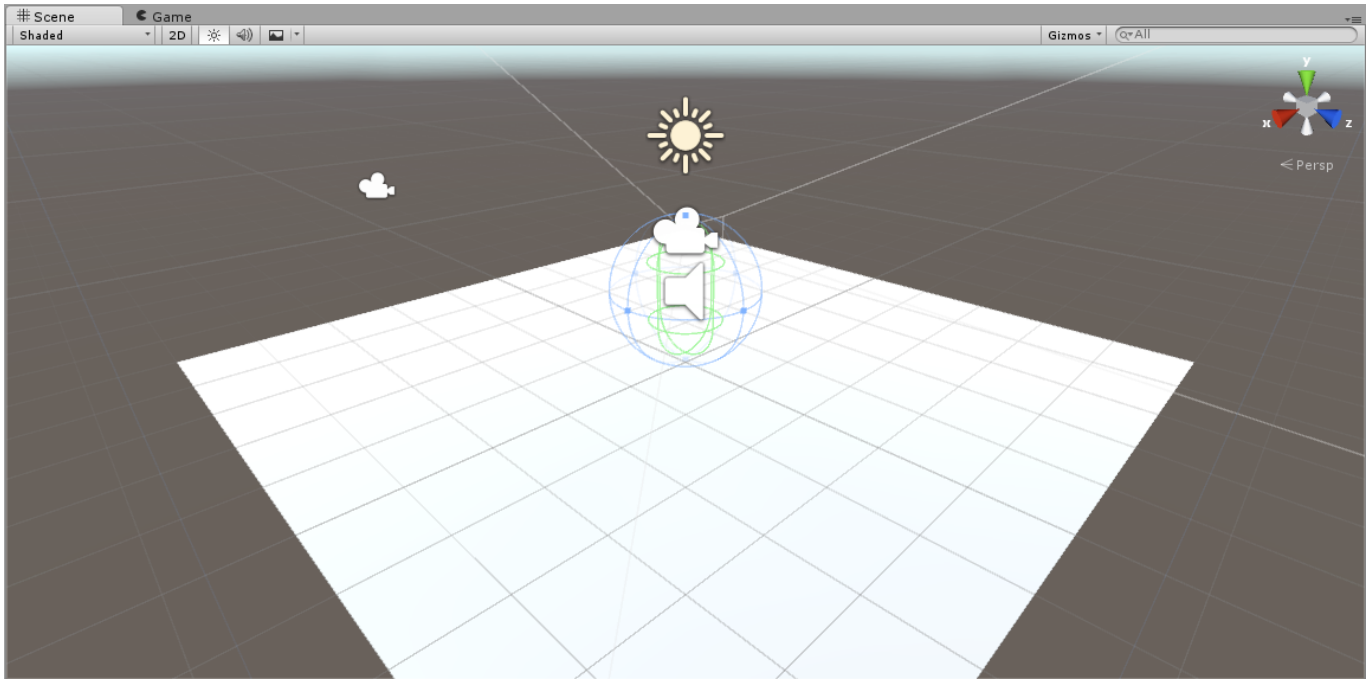
First Person Character

Nous allons utiliser l'asset *FirstPersonCharacter*, qui correspond à un personnage à la première personne (quel choix de nom pertinent !). Dans le dossier FirstPersonCharacter, vous trouverez un fichier guideline qui vous indiquera comment utiliser l'asset (une bonne habitude à prendre pour les projets…).

Vous avez deux prefabs à votre disposition, avec des propriétés bien différentes :

- **RigidBodyFPSController** : Possède un rigidbody et un collider (vous pouvez voir ces éléments dans la fenêtre *Inspector*). Si vous regardez le script associé à ce prefab (toujours très instructif), vous verrez que l'on déplace ce controller en utilisant des AddForce.
- **FPSController** : Il n'est **pas controllé grâce à de la physique**. Une propriété très importante du rigidbody est le booléen isKinematic, et lorsqu'il est activé (ce qui est le cas ici), les forces ne s'appliquent pas. Ce controller utilise un CharacterController et la fonction Move pour se déplacer. La fonction Move ne gère pas nativement la gravité, mais le script *FirstPersonController* s'en occupe.

Ajoutez un plan dans votre scène, puis un FPSController (assurez-vous que le FPSController est bien au dessus du plan). Enfin, lancez l'application.



Le FirstPersonCharacter est sur le plan.

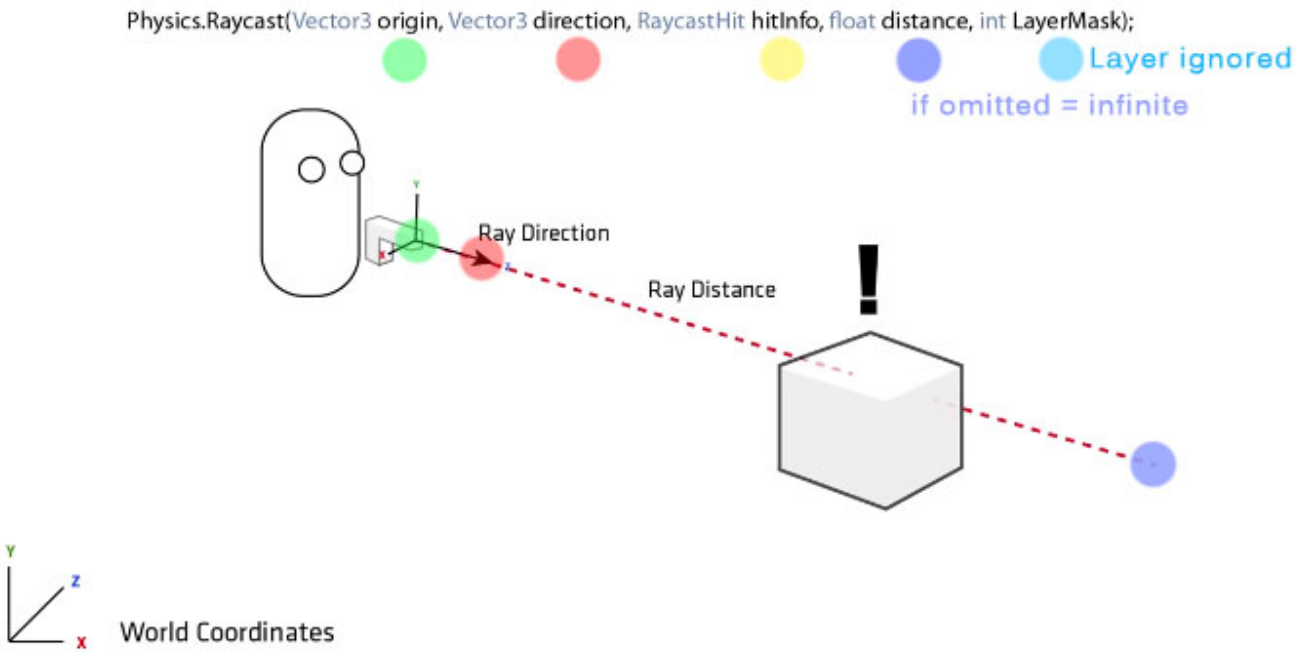
Vous pouvez vous déplacer sur le plan grâce aux flèches directionnelles, ainsi que bouger la tête grâce à la souris (comme un personnage à la première personne en fait…). Le code correspondant au déplacement de la caméra grâce à la souris est contenu dans le script *MouseLook*.

Pour info : Dans le script MouseLook, il ne s'agit pas seulement de déplacer la caméra lorsque la souris bouge, il faut également que les déplacements de la caméra ne soient pas trop rapides pour une impression visuelle plus agréable (vomir en jouant, c'est pas cool !). Des interpolations spheriques linéaires sont faites grâce à la fonction Slerp de la structure Quaternion (l'ami Wikipedia en connait également un rayon sur les quaternions…).

Le Ray Casting

On souhaite désormais pouvoir saisir des objets dans l'environnement virtuel. On va pour cela utiliser le ray casting. Le principe du ray casting est de créer un rayon, depuis une position d'origine et dans une direction particulière. Ce rayon va nous indiquer s'il est entré en collision avec un objet ou non. Pour davantage d'information sur le ray casting, c'est ici.

Le ray casting a de nombreuses application, cette video vous montre par exemple son utilisation pour l'ouverture d'un parachute en environnement virtuel.



Le principe du ray casting, la fonction renvoie true si le rayon a détecté un objet, false sinon. Les informations concernant l'objet détecté (s'il y en a un) se retrouvent dans la variable hitInfo.

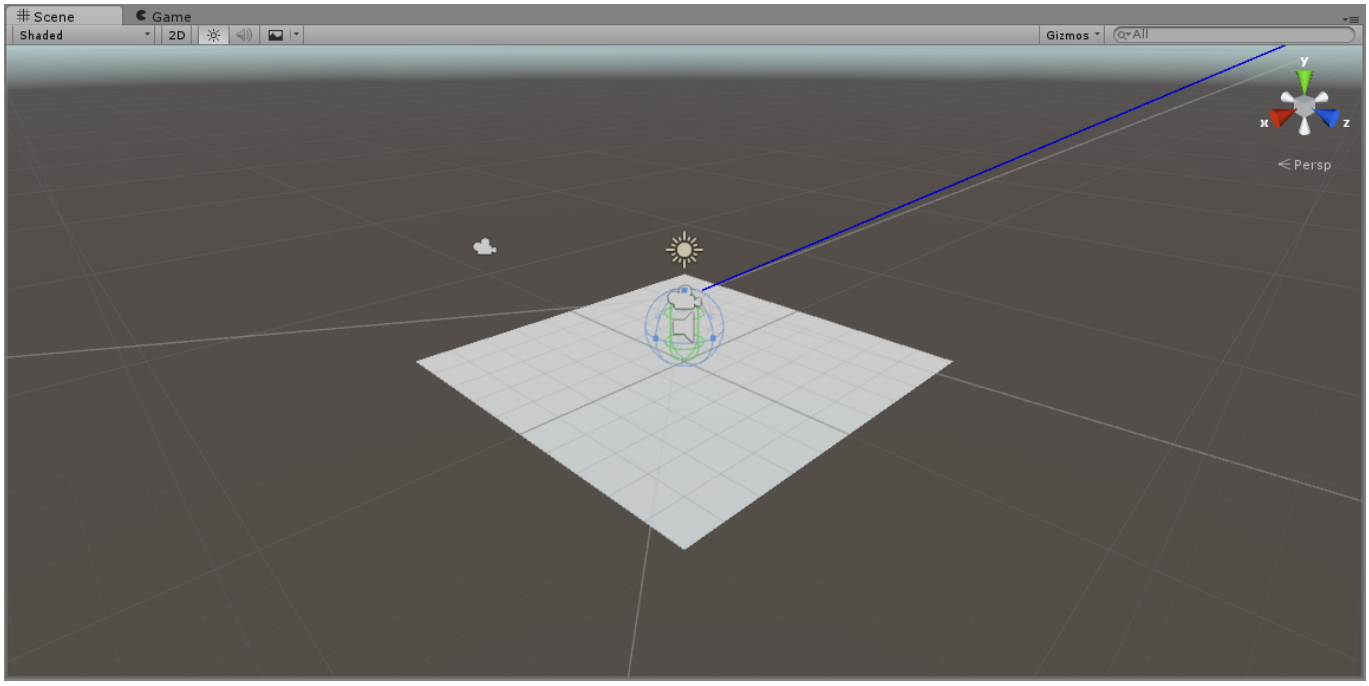
Nous allons utiliser le ray casting pour cibler les objets que l'on pourra déplacer. Pour cela ajoutez un script "RayCasting".

On souhaite que le rayon parte de la caméra, dans la direction de la position du curseur dans l'environnement. Vous trouverez des informations utiles pour générer ce type particulier de ray casting ici.

Pour débugger et voir où se trouve votre rayon, vous pouvez utiliser la fonction Debug.DrawLine.

Attention, cette fonction n'affiche le rayon que dans la fenêtre *Scene*, pas dans la fenêtre

Game. Vous pouvez changer le layout Unity vers *2 by 3* pour voir les deux fenêtres simultanément.



Debug du rayon, en bleu.

Le rayon suivant désormais correctement les mouvements de la souris, attaquons-nous à la saisie d'objets. Nous vous proposons l'algorithme suivant lors de la mise à jour:

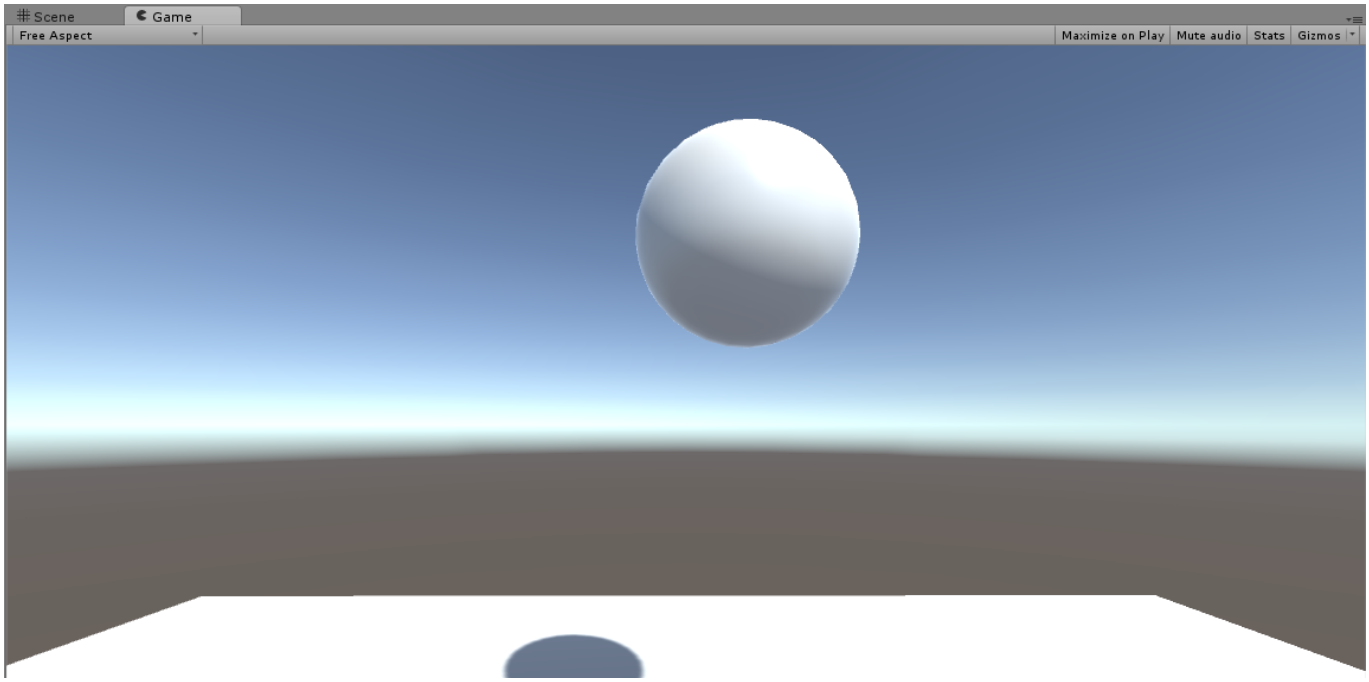
```
Si l'utilisateur clique alors
    Si le raycast indique une collision avec un objet alors
        Saisir l'objet
Sinon si l'utilisateur lache le click et qu'un objet est saisi alors
    Relacher l'objet
Si le click est maintenu et qu'un objet est saisi alors
    Déplacer l'objet à la position du rayon
```

Une solution possible pour saisir un objet est de récupérer son rigidbody. Pour le déplacer lorsque le click est maintenu, il suffit de le positionner (via la fonction MovePosition par exemple) aux coordonnées suivantes :

```
ray.origin + (ray.direction * offset)
```

offset représentant la distance initiale entre le personnage et l'objet saisi.

Note : Cette solution ne permet pas de rapprocher ou d'éloigner l'objet une fois saisi. La distance entre le personnage est l'objet est constante, et l'objet se déplace donc dans une sphère de rayon offset dans le référentiel du personnage.



J'ai saisi la balle !

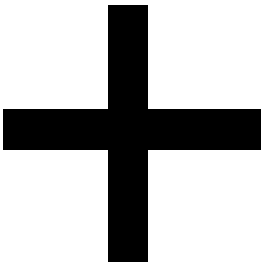
Problème 1 : La gravité sur l'objet saisi

Vous pouvez constater un premier problème. La sphère ayant un rigidbody, les forces continuent de s'y appliquer (et notamment la gravité) lorsque je la saisis. Il est possible de contourner ce problème en jouant sur la variable *isKinematic* mentionnée plus tôt dans ce sujet. En mettant le booléen *isKinematic* à *true* lorsque l'on saisit la sphère, la gravité ne s'y

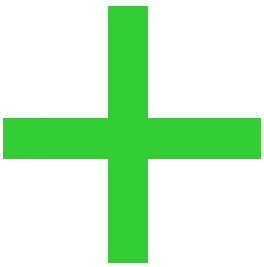
appliquera plus. Attention toutefois à remettre le booléen à *false* lorsque la sphere est dessaisie.

Problème 2 : Comment savoir si un objet est saisissable ou non, saisi ou non

Une des grandes problématiques lorsque l'on travaille dans les environnements virtuels est de le rendre facilement compréhensible pour l'utilisateur. On imagine bien que dans un environnement plus riche que le notre, certains objets seront manipulables, d'autres non. De la même manière, il est intéressant d'avoir des indicateurs nous permettant de savoir si l'on est en train de saisir un objet ou non. Nous allons ici utiliser de très simples **métaphores visuelles** en jouant sur les curseurs. Nous vous proposons trois types de curseurs :

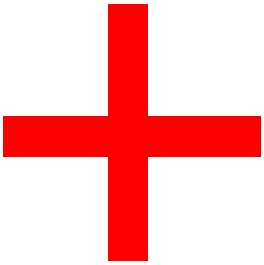


Aucun objet saisissable détécté.



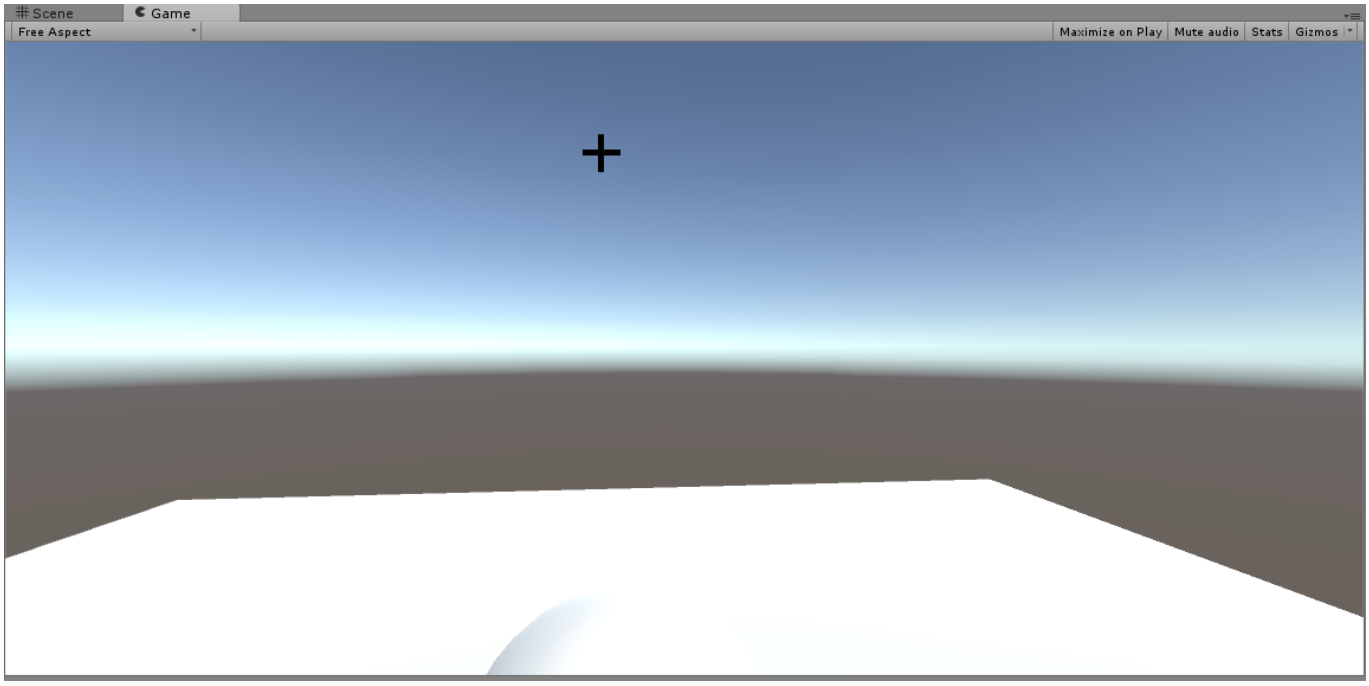
Objet saisissable (mais non-saisi

à l'instant t).

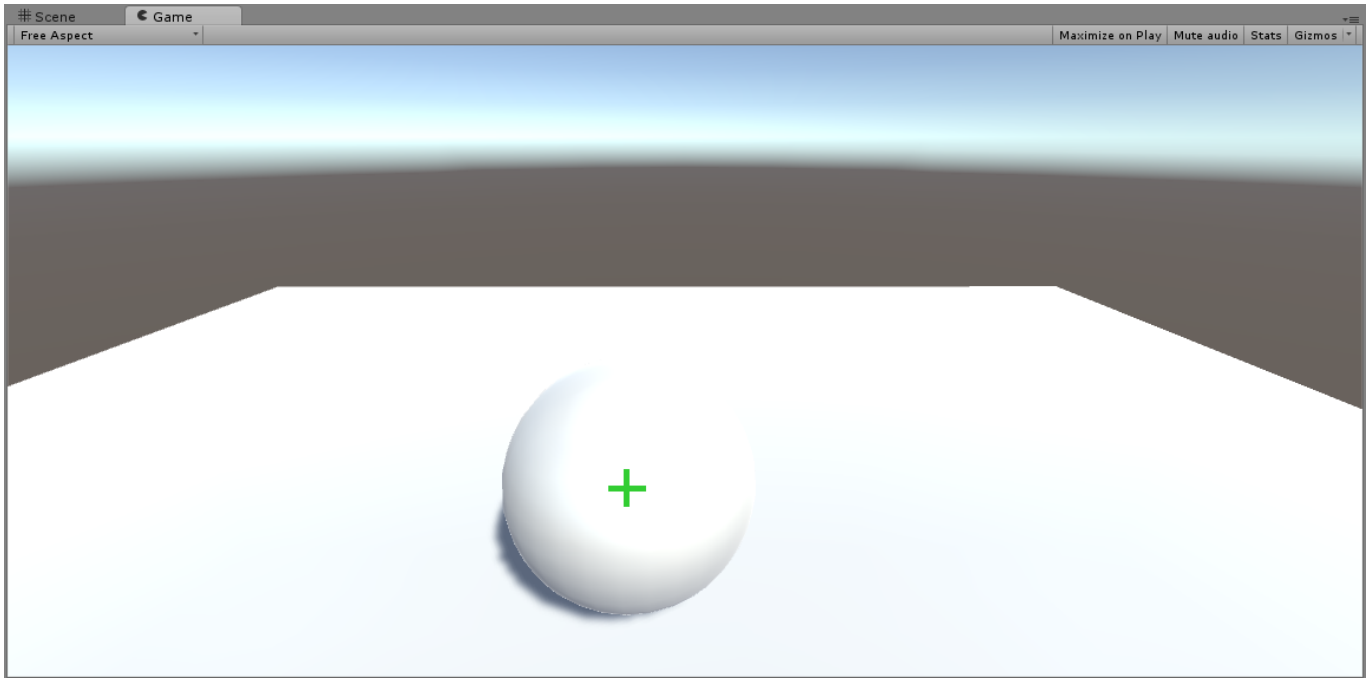


Objet saisi.

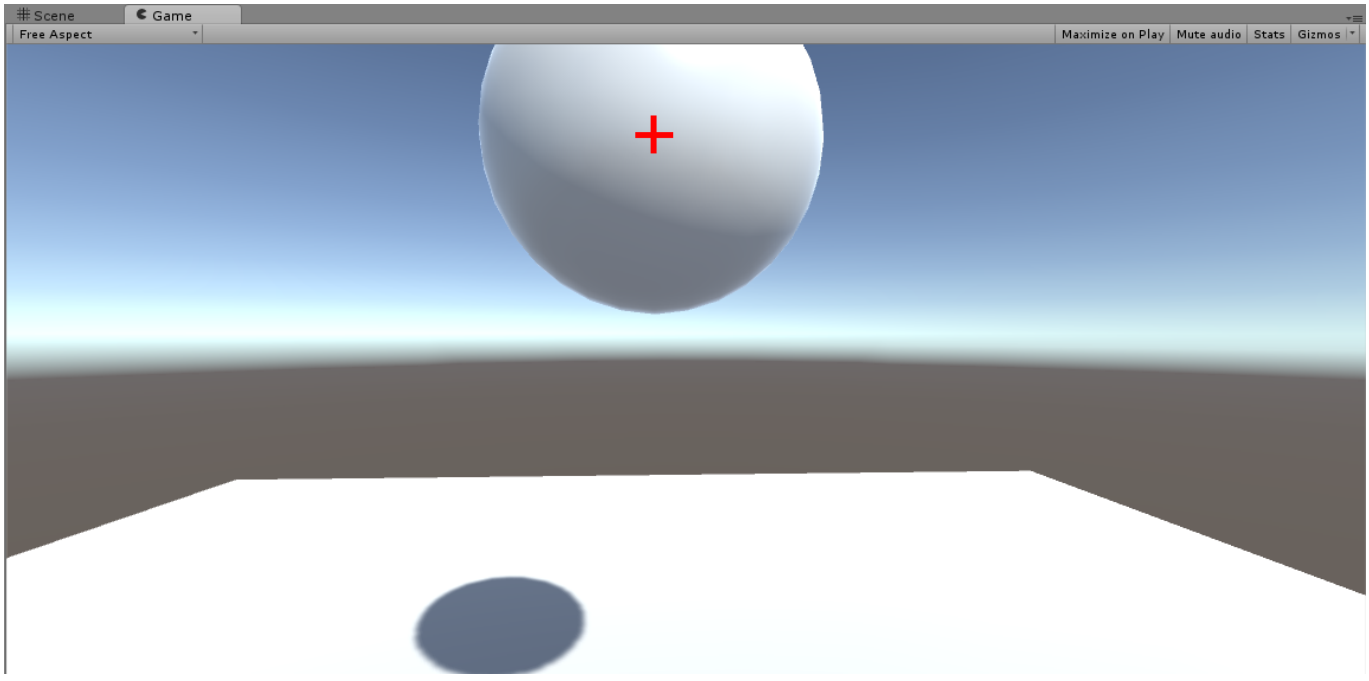
Pour pouvoir utiliser un curseur, il faut que l'image correspondante fasse partie des assets. Dans l'inspector, modifier la propriété *Texture Type* pour lui donner la valeur *Cursor*. Vous pouvez désormais modifier dans votre script *Raycasting* le curseur lorsque le rayon détecte ou pas un objet, ou lorsqu'un objet est saisi. Vous devriez arriver à quelque chose comme ça :



L'espace n'est pas saisissable.



La sphère est saisissable.



La sphère est saisissable.

Problème 3 : Le plan est un objet, mon programme le considère donc comme saisissable

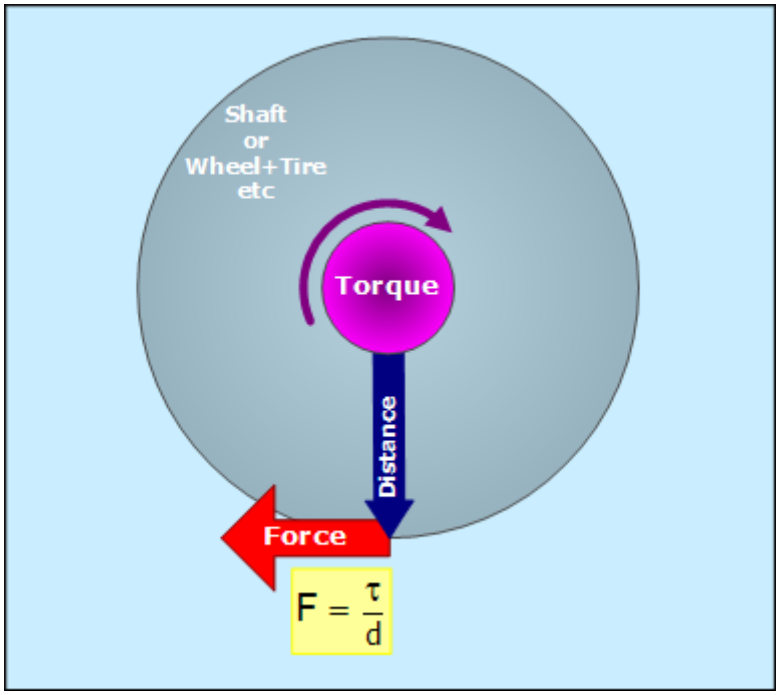
Et oui, pour l'instant nous détectons tous les objets, et le plan en fait partie. Vous devriez néanmoins avoir une idée concernant la solution : utiliser un tag ! En créant un nouveau tag "Saisissable", et en ajoutant un test sur ce tag dans votre script, vous devriez être capable de différencier la sphère (qui sera tagée "Saisissable") du plan (qui lui ne l'aura pas). L'utilisation de ce tag a un autre avantage (et non des moindres) : vous pouvez désormais ajouter autant d'objets que vous le souhaitez sur voter scène, et définir lesquels seront saisissables en les taggant tout simplement. A titre d'exercice, vous pouvez placer des cubes et des cylindres sur votre scène. Appliquez le tag "Saisissable" aux cubes, mais pas aux cylindres. Tout devrait fonctionner sans problème.

Note : Une autre (et meilleure dans ce cas) façon d'arriver au même résultat est d'utiliser l'argument layerMask de la fonction Physics.Raycast afin qu'elle ne renvoie true que lorsqu'elle rencontre un objet présent sur le masque.

Pour aller plus loin

- AddTorque

Il est intéressant de regarder la façon dont sont codés les différents assets standard. En examinant l'asset RollerBall, vous découvrez notamment une méthode que nous n'avons pas utilisé pour déplacer la balle : le AddTorque.



Crédits : http://craig.backfire.ca/pages/autos/horsepower

- Terrain

Depuis le début, nous utilisons de simples plans comme support de nos scènes. Vous pouvez un outil beaucoup plus évolué en utilisant un terrain plutôt qu'un plan **GameObject > 3D Object > Terrain**. Dans l'inspector, vous aurez accès à de nombreuses propriétés vous permettant d'ajouter du relief, des arbres, et bien plus encore. Pour en savoir davantage, c'est ici que ça se passe. Vous pouvez également modifier le "ciel" en jouant sur la *skybox*.