4

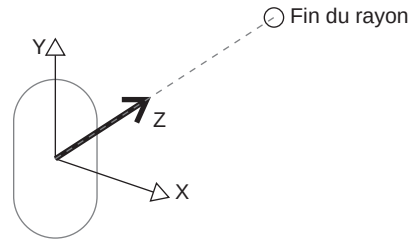
Interactions

Nous allons voir ici, plus en détail, d'autres interactions et nous plonger dans deux éléments essentiels au développement d'un jeu : la *détection de collision* et le *raycasting* (tracé de rayon).

Pour détecter les interactions physiques entre les objets de jeu, la méthode la plus courante consiste à utiliser un collider – un filet invisible tout autour d'un objet chargé de détecter les collisions avec d'autres objets. La détection de collision regroupe à la fois la détection et la récupération des informations provenant de ces collisions.

Il est non seulement possible de détecter que deux colliders (composants de collision) interagissent, mais également d'anticiper une collision et d'effectuer de nombreuses autres tâches à l'aide d'une technique appelée raycasting, qui dessine un rayon – une ligne vectorielle invisible (non rendue) tracée entre deux points dans l'espace 3D – qui peut aussi être utilisé pour détecter une intersection avec le collider d'un objet de jeu. Le raycasting permet également de récupérer de nombreuses autres informations utiles comme la longueur du rayon (et donc la distance entre deux objets) ou le point d'impact de la fin de la ligne.

Figure 4.1



Dans cet exemple, un rayon pointant vers l'avant part de notre personnage. On peut lui donner une direction mais également une longueur spécifique, ou le projeter jusqu'à ce qu'il trouve un objet.

Au cours de ce chapitre, vous travaillerez avec le modèle d'avant-poste que vous avez importé au Chapitre 2, "Environnements". Nous avons créé pour vous l'animation d'ouverture et de fermeture de la porte et vous allez voir comment les déclencher une fois le modèle intégré dans la scène en implémentant la détection de collision et le raycasting.

Commençons par examiner la détection de collision et voir quand la remplacer ou la compléter par le raycasting.

Les collisions

Lorsque des objets entrent en collision dans n'importe quel moteur de jeu, des informations sur cet événement deviennent alors disponibles. En enregistrant différentes informations au moment de l'impact, le moteur de jeu peut ensuite réagir de manière réaliste. Dans un jeu tenant compte des lois physiques par exemple, si un objet tombe au sol d'une certaine hauteur, le moteur a besoin de savoir quelle partie de cet objet touche le sol en premier pour contrôler de façon correcte et réaliste la réaction de l'objet à cet impact.

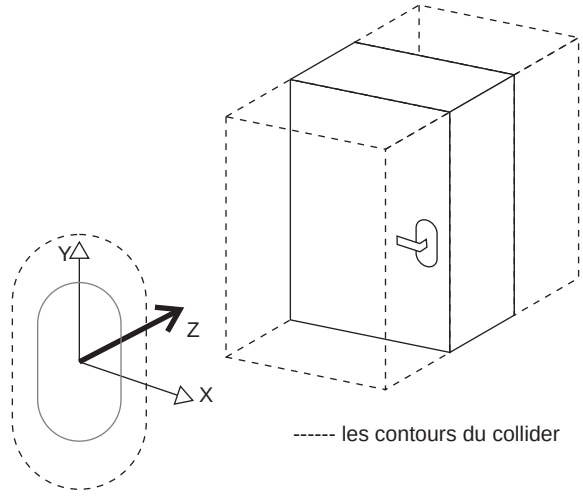
Bien entendu, Unity gère ce type de collisions et stocke ces informations pour vous, si bien que vous avez seulement à les récupérer afin de définir la réaction qui doit se produire.

Dans le cas de l'ouverture d'une porte, vous avez besoin de détecter les collisions entre le collider du personnage du joueur et celui situé sur – ou près de – la porte. Cela n'aurait guère de sens de détecter les collisions ailleurs, puisque l'animation de la porte doit se déclencher lorsque le joueur est assez proche pour espérer qu'elle s'ouvre ou pour franchir le seuil.

Par conséquent, vous allez vérifier les collisions entre le collider du personnage et celui de la porte. Vous devez toutefois étendre la profondeur du collider de la porte afin que celui du personnage du joueur n'ait pas besoin de s'appuyer contre la porte pour que la collision se

déclenche (voir Figure 4.2). Cependant en étendant la profondeur du collider, son interaction avec le jeu devient irréaliste.

Figure 4.2



Dans notre exemple, si le collider de la porte dépassait de la surface visuelle de la porte, le personnage entrerait en collision avec une surface invisible qui l'arrêterait net. En outre, bien que vous puissiez utiliser cette collision pour déclencher l'animation d'ouverture de la porte, l'impact initial dans le collider semblerait peu naturel pour le joueur, ce qui nuirait à son immersion dans le jeu. Si la détection de collision fonctionne très bien entre le collider du personnage et celui de la porte d'un point de vue technique, cette approche présente donc trop d'inconvénients pour qu'un développeur de jeux créatif ne recherche pas une approche plus intuitive. C'est là qu'intervient le raycasting.

Le raycasting

Bien qu'il soit possible de détecter les collisions entre le collider du personnage et un collider qui s'adapte à l'objet porte, il semble plus judicieux que la porte s'ouvre lorsque le joueur lui fait face et à une certaine distance, ce qui peut être réalisé en projetant un rayon d'une longueur limitée depuis l'avant du personnage. Ainsi, le joueur n'a pas besoin de s'avancer jusqu'à la porte ou d'entrer en collision avec un collider étendu pour que le rayon soit détecté. Cette méthode garantit également que le joueur ne peut pas ouvrir la porte s'il

lui tourne le dos ; avec le raycasting, il doit faire face à l'objet pour l'utiliser, ce qui est plus logique.

Cette méthode est couramment utilisée lorsque la détection de collision se montre trop imprécise pour obtenir une réaction correcte, par exemple lorsque les réactions doivent se produire à un niveau de détail très précis (image par image) car elles se produisent alors trop rapidement. Dans notre exemple, vous avez besoin de détecter à l'avance si la collision est susceptible de se produire plutôt que d'y réagir. Prenons un exemple pratique de ce problème.

L'image manquante

Dans le cas d'un jeu de tir en 3D, le raycasting s'utilise pour prédire l'impact d'une balle tirée. En raison de la vitesse d'une balle réelle, il est très difficile d'en représenter visuellement le trajet vers la cible de façon satisfaisante pour le joueur, car cette vitesse est largement supérieure à la cadence des images rendues dans les jeux.

Lors d'un tir au pistolet dans le monde réel, la balle met si peu de temps pour atteindre sa cible que l'événement semble instantané à l'œil nu. On peut donc admettre que la balle devrait atteindre son objectif en quelques images seulement dans le jeu, même avec un rendu supérieur à 25 images par seconde.

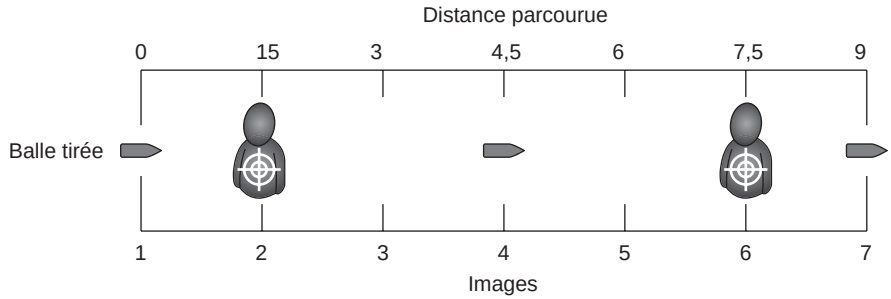


Figure 4.3

À la Figure 4.3, une balle est tirée d'un pistolet. Pour que son parcours soit réaliste, elle devrait se déplacer à une vitesse de 150 mètres par seconde. À une cadence de 25 images par seconde, la balle se déplace donc de 6 mètres par image. Or la circonférence d'un corps humain est environ de 1,50 mètre, si bien que la balle manquerait très probablement les ennemis situés à 5 et 7,50 mètres. Voilà pourquoi on utilise la détection *a priori*.

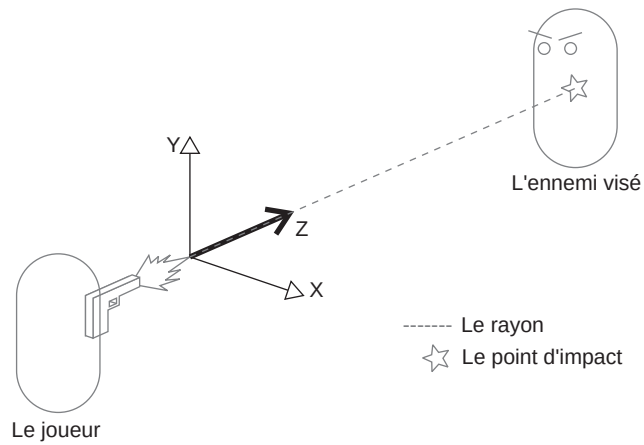
La détection de collision *a priori*

Au lieu de vérifier la collision avec un objet Balle, il s'agit de découvrir si une balle atteindra sa cible. En projetant un rayon depuis l'objet Pistolet (en tenant donc compte de son orientation) sur la même image que celle où le joueur appuie sur le bouton de tir, vous pouvez immédiatement vérifier quels objets coupent le rayon.

Cela est possible car les rayons sont tracés immédiatement. Vous pouvez comparer le rayon à un pointeur laser : lorsqu'on allume un laser, l'œil humain ne perçoit pas le déplacement de la lumière vers l'avant en raison de sa vitesse, si bien que le rayon laser semble tout simplement apparaître.

Dans le raycasting, les rayons fonctionnent de la même façon. Chaque fois que le joueur appuie sur le bouton de tir, un rayon est tracé dans la direction visée. Il est alors possible de récupérer des informations sur le collider que ce rayon touche. Une fois le collider identifié, un script peut alors désigner l'objet de jeu lui-même et scénariser son comportement en conséquence. Vous pouvez obtenir des informations très détaillées, comme le point d'impact, et les utiliser pour définir la réaction de la cible (faire reculer l'ennemi dans une direction particulière par exemple).

Figure 4.4



Dans cet exemple de jeu de tir, vous utiliseriez sans doute des scripts pour tuer ou repousser l'ennemi dont le collider touche le rayon. Comme le rayon est immédiat, cette réaction peut s'effectuer sur l'image suivant celle sur laquelle le rayon croise le collider de l'ennemi. La réaction étant enregistrée immédiatement, l'effet produit par le tir est réaliste.

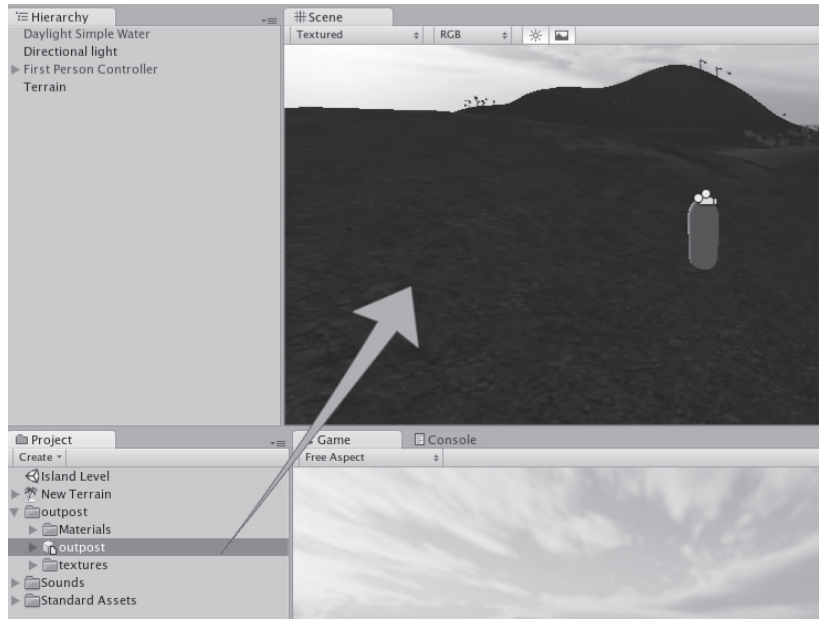
Il est également intéressant de noter que les jeux de tir affichent souvent des rayons normalement invisibles pour aider le joueur à viser. Ne confondez toutefois pas ces lignes avec le raycasting car ces rayons sont simplement utilisés pour effectuer le rendu des lignes de la visée.

L'ajout de l'avant-poste

Avant de commencer à utiliser à la fois la détection de collision et le raycasting pour ouvrir la porte, vous devez intégrer à la scène l'avant-poste que vous avez défini au Chapitre 2, "Environnements". Plus loin, vous écrirez le script qui contrôle les états d'animation de ce modèle.

Faites glisser le modèle d'avant-poste depuis le panneau PROJECT dans le panneau SCENE. Souvenez-vous que vous ne pouvez pas positionner le modèle lors du glisser-déposer, mais seulement une fois que celui se trouve dans la scène (autrement dit, après avoir relâché le bouton de la souris).

Figure 4.5



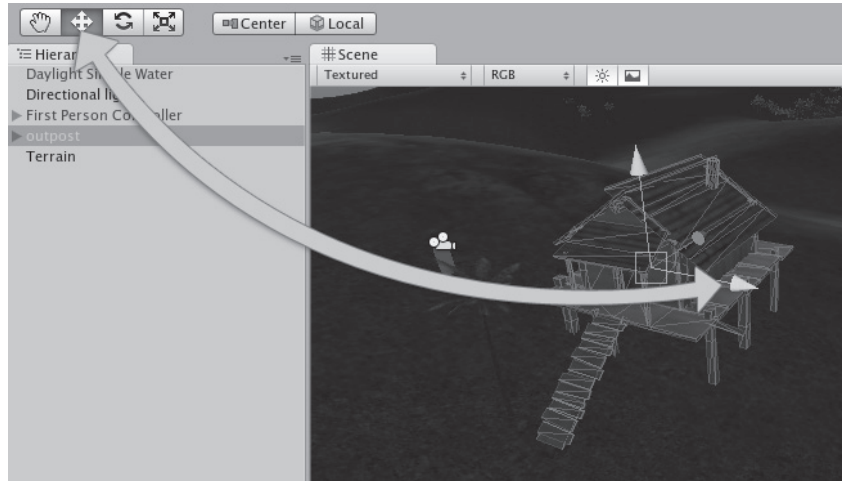
Une fois l'avant-poste dans le panneau SCENE, son nom apparaît dans le panneau HIERARCHY et il est automatiquement sélectionné. Vous êtes maintenant prêt à définir sa position et à le redimensionner.

Positionner le modèle

Comme la conception de votre terrain peut être différente de la nôtre, sélectionnez l'outil TRANSFORM, puis placez l'avant-poste dans une zone de terrain dégagée en faisant glisser ses poignées d'axe dans la scène.

Soyez prudent lorsque vous utilisez les poignées d'axe pour déplacer un modèle. Faire glisser le carré blanc où convergent les poignées modifie les trois axes à la fois, ce qui est à éviter en vue Perspective. Assurez-vous de faire glisser chaque poignée individuellement en gardant le curseur à l'extérieur du carré blanc.
Si vous souhaitez que l'objet reste plaqué sur la surface du terrain en vue Perspective, appuyez sur Cmd/Ctrl et faites glisser le carré blanc.

Figure 4.6



Nous avons placé l'avant-poste à (500, 30.8, 505) mais vous devrez peut-être le déplacer manuellement. Rappelez-vous qu'après avoir positionné l'objet à l'aide de ses poignées d'axe dans le panneau SCENE, vous pouvez saisir des valeurs spécifiques dans les champs Position du composant TRANSFORM dans le panneau INSPECTOR.

Redimensionner le modèle

Lorsque vous travaillez comme ici avec deux applications – le logiciel de modélisation dans lequel la ressource outpost a été créée et Unity –, il est courant que la taille du modèle importé ne corresponde pas aux unités métriques utilisées dans Unity et que vous deviez modifier ses dimensions.

Bien qu'il soit possible de définir les valeurs d'échelle dans les champs SCALE du panneau INSPECTOR, il est généralement préférable d'utiliser le paramètre SCALE FACTOR de la ressource originale. Sélectionnez le modèle OUTPOST dans le dossier Outpost du panneau PROJECT. Le composant FBXIMPORTER s'affiche alors dans le panneau INSPECTOR. Ses paramètres d'importation affectent toutes les occurrences du modèle que vous placez dans le panneau SCENE. Dans ce cas précis, modifiez la valeur SCALE FACTOR de 1 à 2, puis cliquez sur le bouton APPLY. Ainsi, la taille de l'avant-poste est suffisante pour que l'objet CHARACTER CONTROLLER passe par la porte et la taille de la pièce sera réaliste lorsque le personnage se trouvera à l'intérieur.

Colliders et tag de la porte

La porte doit être identifiée comme un objet particulier pour pouvoir s'ouvrir lorsqu'elle entre en collision avec le joueur. Pour cela, l'objet doit posséder un composant COLLIDER et un tag spécifique qui désignera l'objet dans le script. Cliquez sur la flèche grise située à gauche du nom de l'objet OUTPOST dans le panneau HIERARCHY pour afficher tous ses objets enfants.

Sélectionnez l'objet DOOR, puis placez le curseur sur le panneau SCENE et centrez la vue sur cet objet (touche F).

Figure 4.7