TP 2 : Combat Sumo (/80)

Figure 1: not representative of final product

Finalité

- 1.5 périodes de laboratoire sont prévues pour ce travail.
- Ce travail compte pour 20% de la session.
- Vous familiariser avec le développement de fonctionnalité à partir de requis et d'instructions

Prérequis

- Avoir réalisé avec succès le formatif 3 et 4.

À remettre

- La remise devra être faite en classe à la semaine 6 en présentant le résultat à l'enseignant

Notes importantes

- Se baser sur le package « **TP2 – Sumo Ball Battle** » fourni avec le TP2
- Se placer en équipe de 1 ou 2 pour le TP

Description du jeu final attendu

- Le jeu se déroule sur une île centrale, où vous devez renverser vos ennemis et éviter d'être vous-même délogé de l'île pendant le plus longtemps possible
- Le jeu se déroule en **vagues**, où les ennemis apparaissent, et tous les ennemis doivent être battus avant de passer à la prochaine vague
- Le personnage du joueur et les ennemis doivent tous être construits à partir de **sphères**
- Tous les déplacements du joueur et des ennemis doivent êtres **basés sur la physique**
- Des **power-ups** doivent apparaitre en jeu et offrir des avantages / inconvénients au joueur (ou au ennemis si désiré)
- La partie devient **graduellement** de plus en plus difficile (caractéristique des ennemis augmentent)
- La partie se termine lorsque le joueur est délogé de l'arène (effet du GameOver au choix)

Remarques Générales

- Tout code en lien avec la physique doit être placé dans FixedUpdate()
- Utiliser ShaderGraph pour créer des matériels pour notre joueur, et enemis
- Tout déplacement direct (modification de transform) doit être normalisé dans le temps
- Les constantes utilisées doivent être sous forme de variables (pas « hard-coded »)
- Le code doit être commenté de façon adéquate

Nouveautés (capacité à acquérir par soi-même)

- Utiliser des enums pour dénoter différentes variations d'un objet
- Utiliser le UV d'une sphère pour dans ShaderGraph
- Utiliser Random.insideUnitCircle pour le spawn

Comportement des différents archétypes :

Positionnement et Comportement de la caméra :

- La caméra doit pointer vers un **point focal**, gameobject positionné en **(0,0,0)**
- Les touches gauches-droites doivent faire pivoter ce point focal **autour de l'axe Y** à l'aide d'un script **RotateCamera.cs** attaché au point focal
- La caméra est un **children** du point focal dans la hiérarchie

Déplacement et Comportement du joueur :

- Le personnage doit être une **sphère**
- Un seul script **PlayerController.cs** doit gérer le comportement du personnage
- Ses déplacements doivent être basés sur la **physique**
- Une force peut seulement être appliquée dans la direction où regarde la caméra dans le **plan XZ** (pas de composantes en Y) ou dans le sens opposé.

Positionnement et Comportement des ennemis :

- Les ennemis doivent être une prefab instanciés par un script **LevelController.cs**
- Les ennemis sont des sphères à caractéristiques **variables**
- Le **PhysicsMaterial** des ennemis doivent être rebondissant (Bounciness = 1.0)
- Se déplacent dans la direction du **joueur** et tentent de le faire tomber de l'arène
- Auront des caractéristiques et un nombre dépendant de la **difficulté** du jeu
- Exemples de caractéristiques :
    - Taille (localScale)
    - Masse
    - Vitesse
    - Autres

Comportement des powers-ups :

- Au moins **deux types** de power-ups avec des effets **différents**
- Un power-up aléatoire est spawné dans l'arène au **début de vague**
- Active le power-up lorsque le joueur y touche
- Ont un effet **temporaire** sur le joueur, soit par un timer, ou un nombre d'utilisations
- Exemples d'effets :
    - Augmente la taille / masse du player
    - Remplace le PhysicMaterial du joueur
    - Ajoute une force additionnelle aux ennemis lors de collisions avec eux
    - Une vie additionnelle qui sauve le joueur s'il tombe
    - Autres effets au choix et à l'inspiration du programmeur

Scripts Nécessaires et fonctions (/55):

RotateCamera.cs (/4)

- Détecte les inputs et pivote le point focal de droite à gauche

PlayerController.cs (/15)

- Applique une force sur le joueur selon l'input vertical, dans la direction où la camera regarde
- Contient une variable static public Gameobject player, qui référence le gameobject.
- Une fonction EnablePowerUp(type) qui a un effet différent selon le type de power-up
- Désactive l'effet d'un power up selon un timer, ou un nombre d'utilisation (unique pour chaque type)
- Contrôle le matériel du joueur

EnemyController.cs (/10)

- Fonction InitializeEnemy(…) permet au spawner d'assigner des caractéristiques à l'ennemi en fonction de la difficulté croissante
- Initialise la variable de son matériel avec la difficulté passée par le spawner
- Applique une force sur l'ennemi dans la direction du joueur

PowerUp.cs (/6)

- Une variable Enum contient le type de power-up dont il s'agit (au moins 2)
- OnTriggerEnter avec le joueur, appelle la fonction associée sur le PlayerController, puis se détruit

LevelController.cs (/15)

- Spawn un, ou des ennemies par « vague »
- Garde un compteur du nombre d'ennemis de la vague présente
- Une Fonction public void EnemyOutOfBound() est appelé de l'extérieur, et décroit le compteur, démarrant une nouvelle vague si le compteur arrive à 0
- Traque la difficulté du jeu, qui doit augmenter chaque fois que la vague précédente est terminée
- La difficulté doit avoir un impact sur la difficulté de chaque vague, soit par le nombre d'ennemis, par leur caractéristiques (ou les deux), ou autre au choix de l'étudiants
- À chaque nouvelle vague, spawn un power-up aléatoire à un emplacement aléatoire dans l'arène
- Contient une variable public static LevelController instance, qui se référence lui-même
- As une variable public bool isGameOver, que les autres objets référencent pour déterminer l'état de la partie

OutOfBoundsTrigger.cs (/5)

- Si un ennemi entre dans le trigger, détruit l'ennemie et appelle la fonction EnemyOutOfBound() sur LevelController.instance
- Si le joueur entre dans le trigger, appelle la fonction GameOver() sur LevelController.instance

Matériaux avec ShaderGraph (/25) :

PlayerMat (/10)

- Basé sur une texture (node SampleTexture2D)
- Doit devenir rouge pour une courte durée lorsque le joueur entre en collision avec un ennemi (hit effect utilisant node Blend_OverWrite)
- Doit changer temporairement lorsque sous l'effet d'un powerUp (au choix)

EnemyMat (/10)

- Basé sur une texture (node SampleTexture2D)
- Avoir une apparence aidant le joueur à identifier la force/difficulté de l'ennemi
- Une seule variable changeante (difficulty) doit avoir un impact sur les différents effets visuels de l'ennemi.
- Au moins 3 effets visuels différents selon la difficulté

PowerUpMat (/5)

- Avoir une apparence animée procéduralement qui oscille continuellement dans le temps
- Animation au choix

Bonus : Effets Sonores et/ou Particules (/5):

- En jeux : Musique de fond (au choix) (+1)
- Lorsque la partie échoue (gameover) (+1)
- Lorsque considéré pertinent par le joueur (+ 1 point par effet)