

## TP2 : Combat sumo (/80)



*Attention: non représentatif du produit final.*

### Finalité

- 3 périodes de laboratoire sont prévues pour ce travail (environ une semaine et demie).
- 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 les laboratoires 3 et 4.

### À remettre

- La remise, **incluant une démo en classe** du résultat à l'enseignant, devra être faite **via GitHub avant le 9 octobre 2025, 23h59**
- Vous devrez me donner un accès à votre *repo* GIT du TP2 à [mathieu.piette@cegeplimoilou.ca](mailto:mathieu.piette@cegeplimoilou.ca) (Voir annexe 2 pour les instructions complète sur la création et le partage d'un *repo* GIT à votre coéquipière ou coéquipier, et l'enseignant du cours.)

### Notes importantes

- Se baser sur le *package* « **TP2 – Combat sumo** » fourni avec le TP2
- Le TP est à **réaliser seul** ou **en équipe de 2**

(ATTENTION : pour celles et ceux qui réaliseront le travail en équipe, il est obligatoire de compléter et remettre la grille de répartition des tâches, présentée en annexe 1).

## Description du jeu final attendu

- Le jeu se déroule sur une île centrale, où vous devez renverser les ennemis et éviter d'être vous-même délogé de l'île 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 être **basés sur la physique**
- Des **power-ups** doivent apparaître en jeu et offrir des avantages / inconvénients au joueur (ou aux 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 matériaux pour notre joueur, et ennemis
- 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

## Éléments à intégrer à votre projet

- 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*
- Effets sonores et/ou particules (au choix des développeurs) :
  - En jeux : Musique de fond (au choix)
  - Lorsque la partie échoue (*gameover*)

## Comportement des différents archétypes (évalué lors de la démo) :

Positionnement et comportement de la caméra (/5) :

- 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 **objet enfant** du point focal dans la hiérarchie

Déplacement et comportement du joueur (/7) :

- 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 (/7) :

- Les ennemis doivent être une *Prefab* instanciés par un script ***LevelController.cs***
- Les ennemis sont des sphères à caractéristiques **variables**
- Le **PhysicsMaterial** des ennemis doit 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* (/6) :

- Au moins **deux types** de *power-ups* avec des effets **différents**
- Un *power-up* aléatoire apparaît 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 et la masse du joueur
  - 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 (/40):

### RotateCamera.cs (/3)

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

### PlayerController.cs (/10)

- Applique une force sur le joueur selon l'*input* vertical, dans la direction où la caméra 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'utilisations (unique pour chaque type)
- Contrôle le matériel du joueur

### EnemyController.cs (/7)

- 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 (/5)

- 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 (/10)

- Fait apparaître un, ou des ennemis par « vague »
- Garde un compteur du nombre d'ennemis de la vague présente
- Une Fonction *public void EnemyOutOfBound()* est appelée de l'extérieur, et décroît 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 leurs caractéristiques (ou les deux), ou autre au choix de l'étudiant
- À chaque nouvelle vague, fait apparaître un *power-up* aléatoire à un emplacement aléatoire dans l'arène
- Contient une variable *public static LevelController instance*, qui se référence lui-même

- A 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* (/15) :

#### PlayerMat (/6)

- 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 (/6)

- 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 (/3)

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

## Annexe 1 – Répartition des tâches

Chaque membre de l'équipe doit remplir cette grille pour détailler sa contribution au travail.

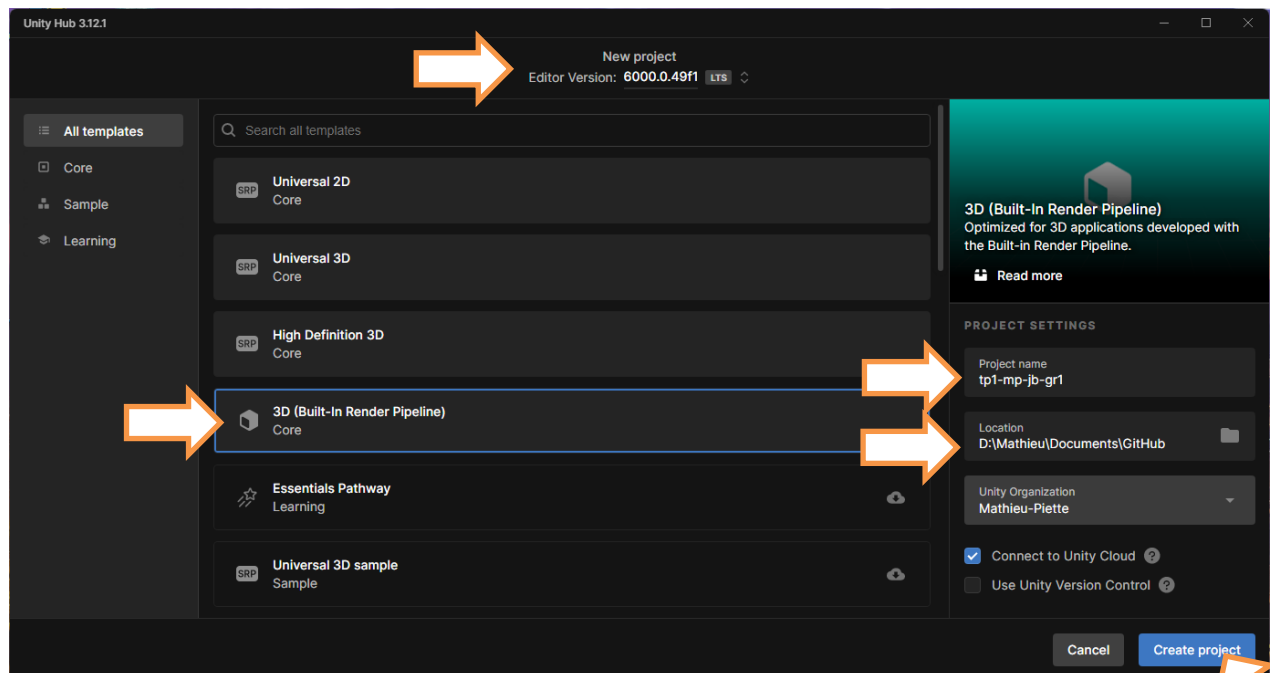
ATTENTION : Utilisez des verbes à l'infinitif (ex. : Programmer le script *PlayerController*, ajouter des sons, créer les animations, etc.).

Nom et prénom	Tâches réalisées
[Étudiant 1]	<ul style="list-style-type: none"><li>• [...]</li><li>• [...]</li><li>• [...]</li></ul>
[Étudiant 2]	<ul style="list-style-type: none"><li>• [...]</li><li>• [...]</li><li>• [...]</li></ul>

## Annexe 2 – Création d'un projet Unity, d'un repo GitHub, et partage du repo.

### 1. Création d'un nouveau projet

- 1) Ouvrez **Unity Hub**, puis sélectionnez **New Project**
- 2) Sélectionnez la bonne version de Unity (**6000.0...**)
- 3) Choisissez le modèle **3D (Built-In Render Pipeline)**, puis nommez votre projet en inscrivant **tp2-[initiales étudiant 1]-[initiales étudiant 2]-[groupe]**. Par exemple : « **tp2-xy-yz-gr1** »<sup>1</sup>.
- 4) Choisissez l'emplacement local de sauvegarde de votre projet, puis cliquez sur **Create project**
- 5) **Attendez que le projet soit créé et s'ouvre (cela peut prendre plusieurs minutes...)**



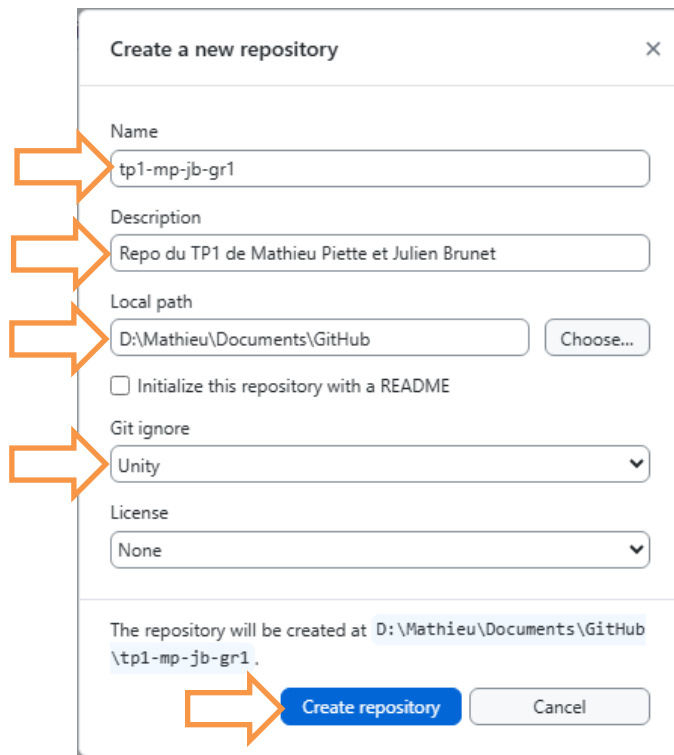
### 2. Création d'un Repository GitHub et démarrage

- 1) Une fois la création du projet terminée, fermez Unity et Unity Hub
- 2) Ouvrez GitHub Desktop, et créez un nouveau repository du même nom que votre projet Unity. Par exemple : « **tp2-xy-yz-gr1** ».
- 3) Ajoutez une description, puis spécifiez le chemin d'accès.

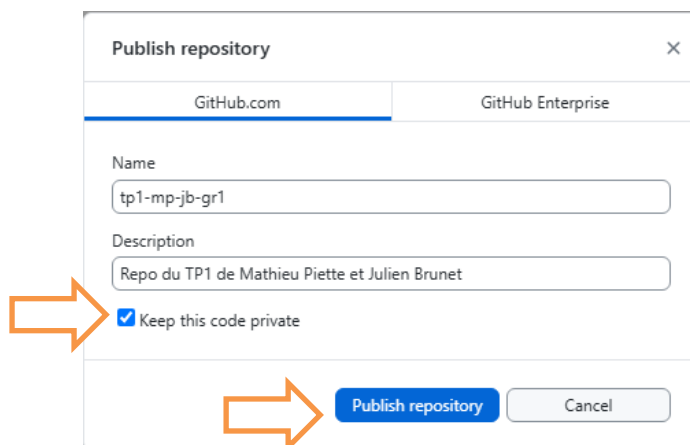
**ATTENTION :** Le chemin d'accès doit être la racine où vous stockez l'ensemble de vos projets Unity, pas le dossier du projet.

<sup>1</sup> Comme nos projets sont hébergés sur GitHub et que le nom de projet doit pouvoir former une URL fonctionnelle, nommez vos projets selon la convention kebab-case.

- 4) Sous *Git ignore*, sélectionnez Unity.
- 5) Cliquez sur *Create repository* (les captures présentent l'exemple du TP1, à adapter pour le TP2).



- 6) De retour dans l'interface de GitHub Desktop, cliquez sur *Publish repository*
- 7) Assurez-vous que *Keep this code private* est bien coché, puis cliquez sur *Publish repository*.

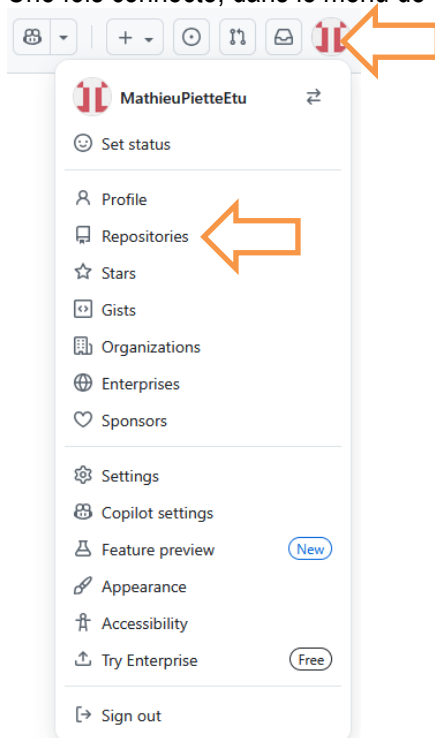


### 3. Partage de votre repo à une coéquipière ou coéquipier, ainsi que l'enseignant du cours.

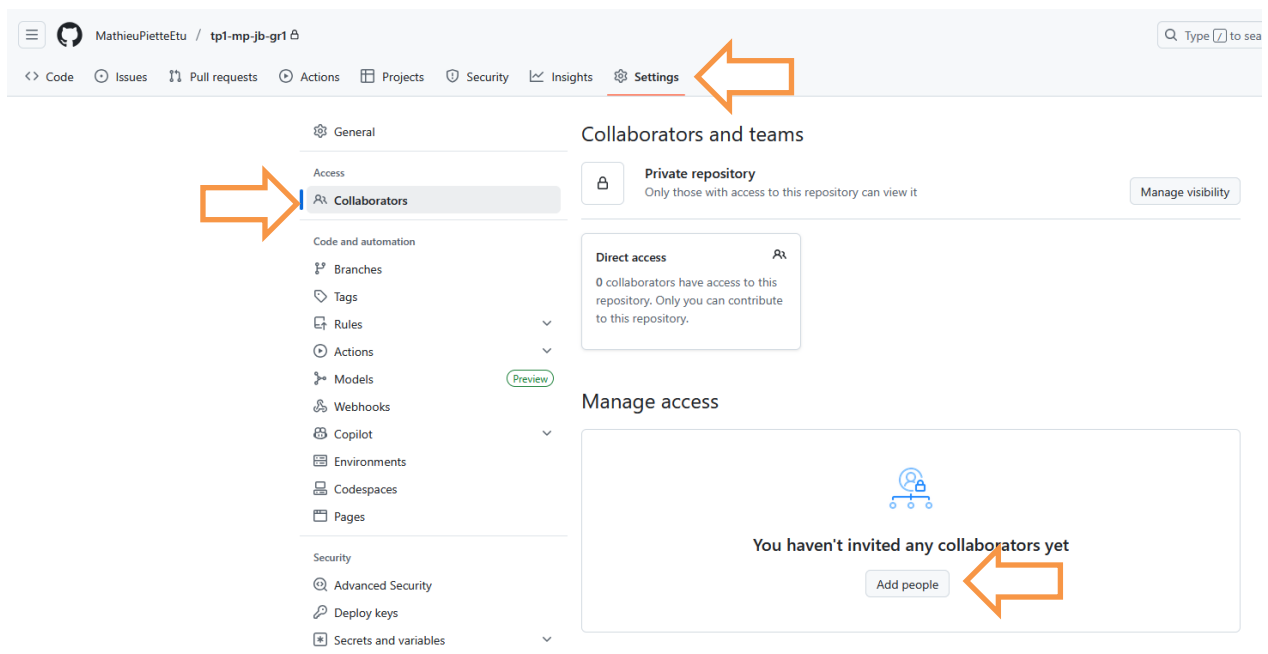
- 1) Rendez-vous sur l'interface Web de GitHub (<https://github.com/login>)



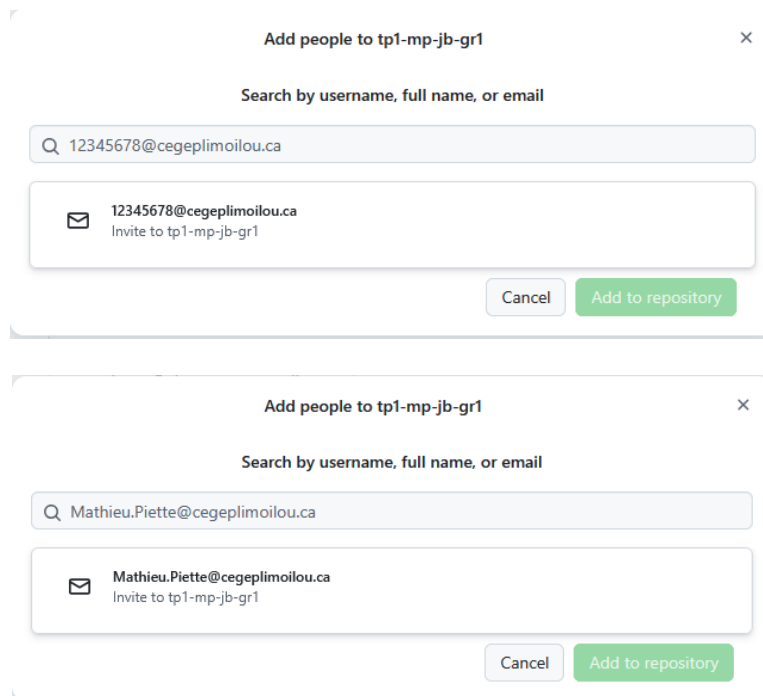
2) Une fois connecté, dans le menu de droite, accédez à vos *Repositories*



3) Lorsque vous êtes dans votre Repo, cliquez sur *Settings*, *Collaborators*, puis *Add people*.



4) Ajoutez ensuite le courriel de votre coéquipière ou coéquipier (si le travail est réalisé en équipe), ainsi que mon courriel (mathieu.piette@cegeplimoilou.ca).



The image shows two screenshots of a GitHub dialog box titled "Add people to tp1-mp-jb-gr1". The dialog has a search bar with the placeholder text "Search by username, full name, or email". In the first screenshot, the search bar contains the email "12345678@cegeplimoilou.ca". Below the search bar, a list of search results is shown, with the first result being "12345678@cegeplimoilou.ca" with a subtext "Invite to tp1-mp-jb-gr1". At the bottom right of the dialog are two buttons: "Cancel" and "Add to repository". The second screenshot is identical but shows the search bar with the email "Mathieu.Piette@cegeplimoilou.ca" and the search result "Mathieu.Piette@cegeplimoilou.ca" with the subtext "Invite to tp1-mp-jb-gr1".

Les personnes invitées recevront une invitation pour collaborer sur le repo, et pourront écrire sur ce dernier.

Bon travail!