Module Moteur de jeux: Unreal Engine

Travaux Pratiques 9

Intelligence Artificielle (base)

Illustrations réalisées sous UE 4.24.0

Objectif: Dans cette séance, nous allons voir comment intégrer un comportement à un personnage. Il s'agira de mettre en place un comportement associé à des ennemis. A la fin de cette séance, vous serez capable de :

- Créer une entité IA capable de contrôler un personnage.
- Créer et utiliser des arbres de décisions et *BlackBoard*.
- Utiliser la perception IA pour donner la vue à un acteur.
- Créer un comportement pour faire errer un ennemie et lancer une attaque.

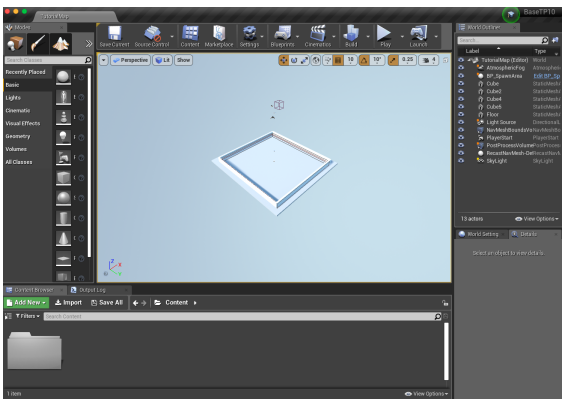
Travail à rendre sur moodle : à la fin de la séance vous devez déposer une archive contenant les captures d'écran permettant de noter l'avancement de chaque question avec idéalement une vidéo montrant le résultat final de votre travail.
A noter : les questions précédés de M2 (resp. DU) seront prises en compte pour la notation des étudiants en M2 (resp. DU) uniquement, les autres étudiants peuvent passer la question.

Partie I: Démarrage

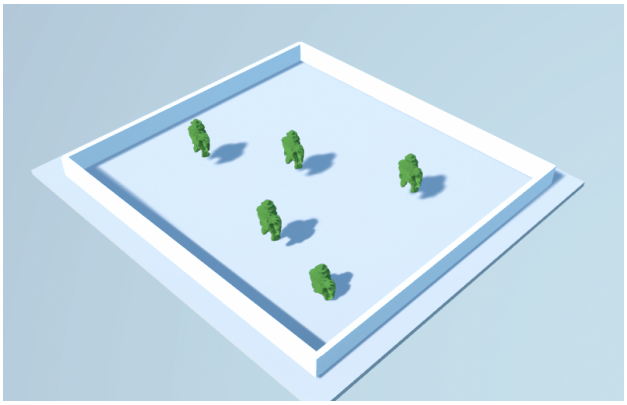
1.1 Récupérez le projet de base accessible sur moodle à l'adresse suivante :

`https://moodle.univ-lyon2.fr/mod/resource/view.php?id=85681`

Vous devriez obtenir un projet ressemblant à l'illustration (a) suivante :



(a)

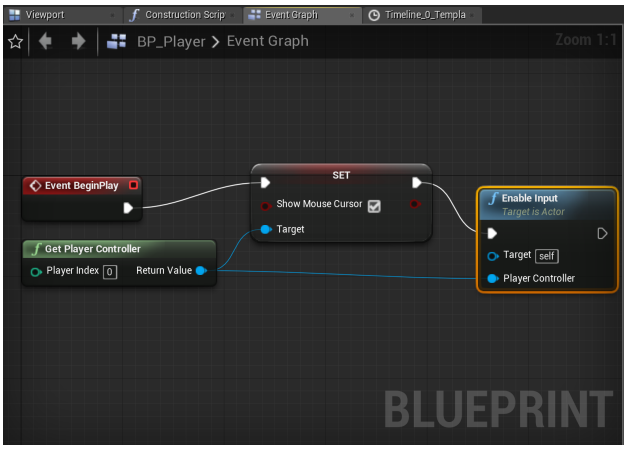


(b)

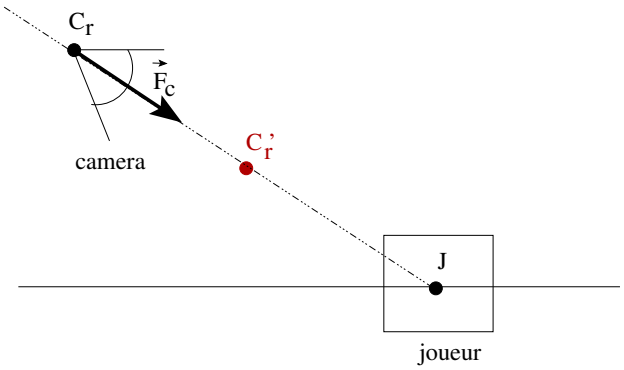
1.2 Lancez le jeu et vérifiez que vous pouvez bien rajouter des personnages (Alcapone) en cliquant sur le sol (comme sur l'image (b)).

1.3 Repérez les éléments de l'arborescence du projet et en particulier la classe `BP_Player` dans le répertoire `Content/Blueprints` qui contient la caméra. C'est sur cet objet que nous travaillerons pour les questions suivantes.

M2 1.4 En vous inspirant de la partie 3 de la feuille du TP4, associez les touches *Flèche Haut/Bas* pour faire reculer (resp. avancer) la caméra. Pour cette question, vous pouvez d'abord vérifier que la liaison des touches fonctionne bien. Si vous avez des difficultés avec l'activation des touches, vous pouvez forcer l'activation de la détection des événements claviers en rajout les expressions ci-dessous (dans la classe `BP_Player`) :



(a)



(b)

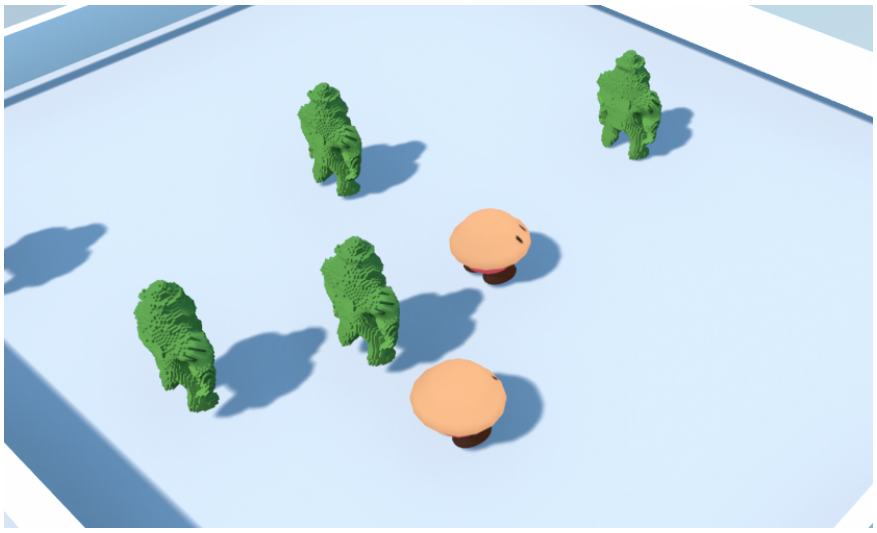
$$C'_r = C_r + k||\overrightarrow{C_rJ}|| * \overrightarrow{F_c}$$

M2 1.5 Pour implémenter le zoom, il s'agit de définir une fonction `deplaceCamera` qui permettra de rapprocher ou éloigner la caméra par rapport à la scène tout en gardant la même direction. Le distance de déplacement sera à transmise à travers un paramètre dans la fonction.

Indication : Utilisez la formule et le schémas (b) ci-dessus.

M2 1.6 Utilisez un *timeline* et votre fonction précédente afin de rendre le niveau de zoom progressif.

1.7 On souhaite rajouter un autre type de personnage dans la scène. Pour cela, rajoutez un nouvel input (dans les réglages du projet), et l'associer au clique droit avec la touche SHIFT. Dans l'onglet *Event Graph*, utilisez ensuite l'événement précédant pour générer des `BPBuffin`.



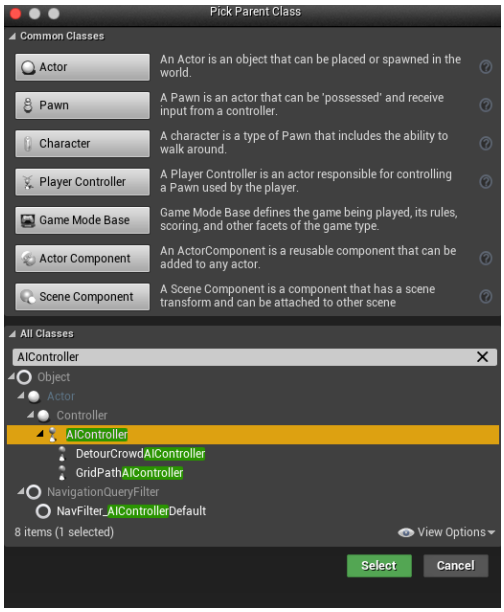
Partie II: Création d'un contrôleur IA

Dans cette partie, nous allons créer un personnage avec une IA, pour cela nous aurons besoin des éléments suivants :

- **Corps** : Il s'agit de la représentation physique du personnage, dans notre TP il s'agit du personnage d'Alcapone.
- **Âme** : L'âme est l'entité contrôlant le caractère, cette dernière peut être le joueur ou l'IA.
- **Cerveau** : Le cerveau est la partie de l'IA qui prend des décisions. Cette partie peut être faite par différents moyens en C++, *Blueprint* ou par arbre de décisions.

Le corps est déjà créé dans le projet de base et dans la partie suivante nous allons implémenter un contrôleur IA qui correspond à l'âme du personnage. Le rôle d'un contrôleur est de recevoir des actions en entrées et de les appliquer (ou non) suivant le fait qu'elles soient prises en charge et autorisées. Pour créer un contrôleur, nous allons créer une nouvelle classe dans le répertoire `IA` que vous allez rajouter ici dans l'arborescence suivante : `Content/BaseTP10/Characters/IA`.

2.1 Dans le répertoire précédent, créez une nouvelle classe `IA_Alcapone` et sélectionnez comme classe parent la classe `AIController` (voir image suivante).



2.2 Il s'agit ensuite d'indiquer que le personnage d'Alcapone va utiliser le contrôleur défini dans la question précédente. Pour cela, ouvrez le personnage `BP_Al` situé dans le répertoire `Content/Characters/BluePrints` (image (a) ci-dessous) et modifiez le champ associé au contrôleur IA (image (b) ci-dessous). Par la même occasion, précisez aussi que le personnage est généré (image (b) ci-dessous).

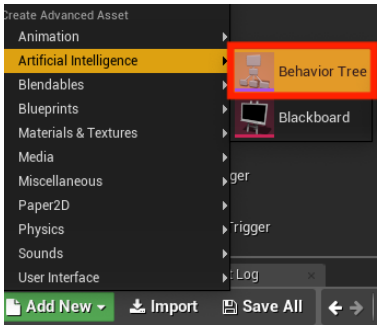
(a) (b)

Compilez la classe et fermez l'objet.

2.3 Importez le personnage que vous avez animé lors le séance du TP8 et faire en sorte qu'elle remplace le personnage d'alcapone lords de la génération de personnage.

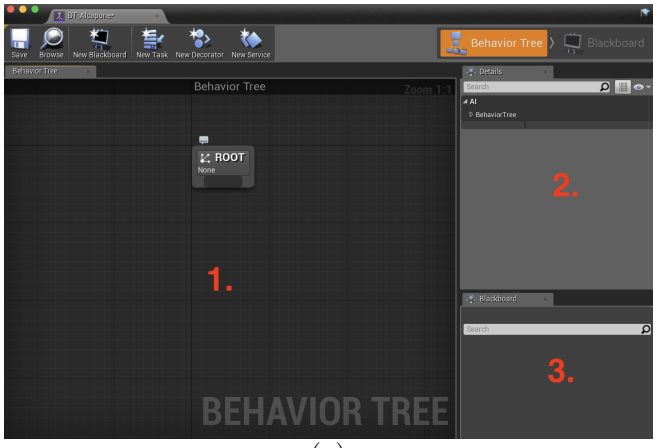
Partie III: Arbre de comportement

3.1 Allez dans le répertoire `Content/Characters/AlcaponeVoxels/IA` et créez une nouvelle classe `Behavior Tree`.

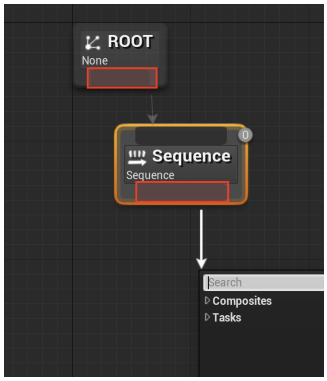


L'éditeur de l'arbre de comportement est composé de 3 panneaux principaux (voir image (a)) :

1. **Behavior Tree** : l'arbre de comportement en lui même.
2. **Details** : les détails des noeuds sélectionnés apparaîtrons ici.
3. **Blackboard** : ce panneau montrera les clés et valeurs (affiché uniquement lors du lancement du jeu).



(a)

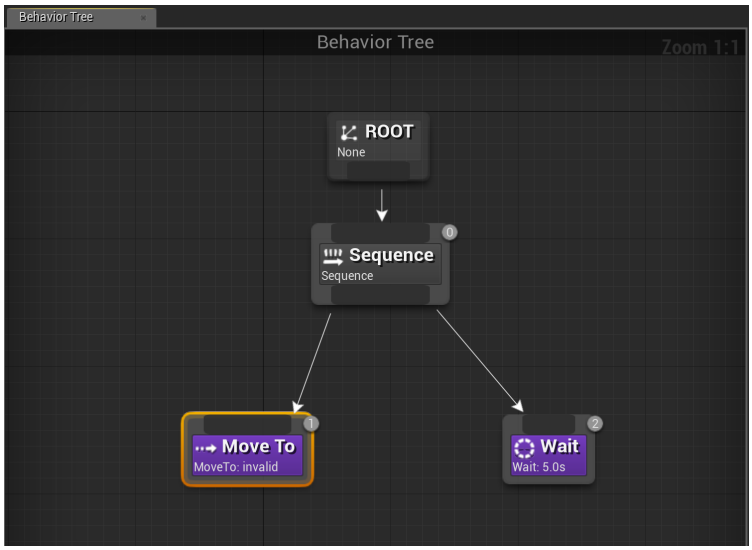


(b)

À partir d'un noeuds du graphe (image (b) précédente), il est possible de rajouter des éléments par glissé-déposé en positionnant la souris sur les zones inférieures noires.

Dans la suite cette partie, nous allons éditer l'arbre de façon à bouger le personnage sur une position aléatoire.

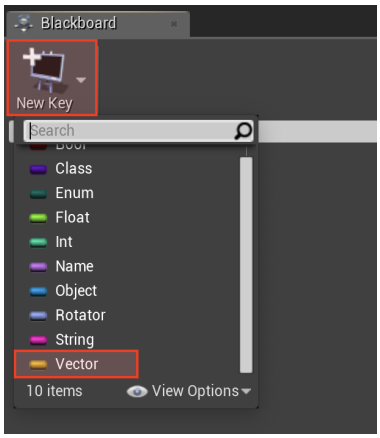
3.2 Créez l'arbre suivant qui est composé d'un séquenceur, composé lui-même de deux actions de déplacement et d'attente.



Comme vous pouvez le voir, `moveTo` accepte seulement des valeurs définies à travers le *blackbord*. Nous allons donc en définir dans la question suivante.

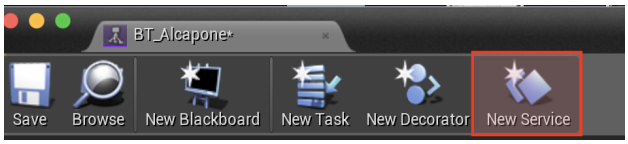
3.3 Allez à nouveau dans le panneau de l'explorateur de contenu, et rajoutez une classe `Blackboard` depuis le menu *Add Neaw/Artificial Intelligence/Blackboard*.

3.4 Rajoutez une variable de type *vector* et nommez là `positionCible`.

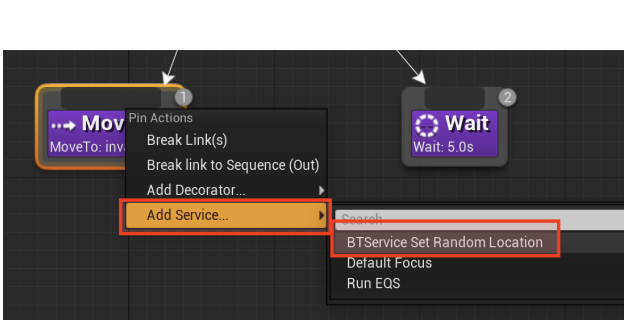


Afin de donner une valeur aléatoire, nous allons créer un nouveau service que nous rattacherons ensuite à un noeud.

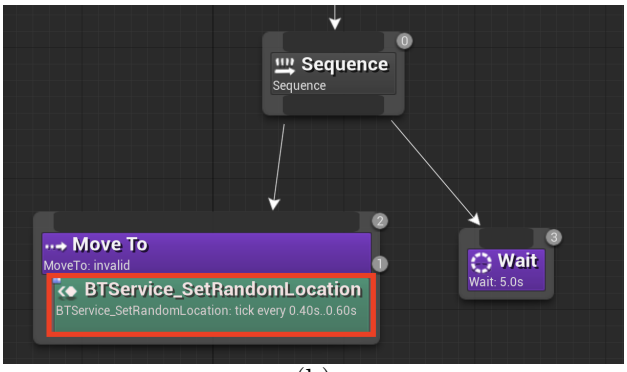
3.5 Créer un nouveau service en sélectionnant la classe `BT_Alcapone`, puis bouton *New Service* (voir image ci-dessous). Vous utiliserez le nom `BTService_SetRandomLocation`



3.6 Maintenant vous pouvez rattacher le service que vous venez de créer, dans le graph de comportement. Pour cela sélectionnez le noeud `MoveTo` et rajoutez le service comme décrit sur l'image (a) ci-dessous.



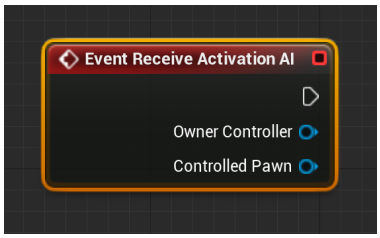
(a)



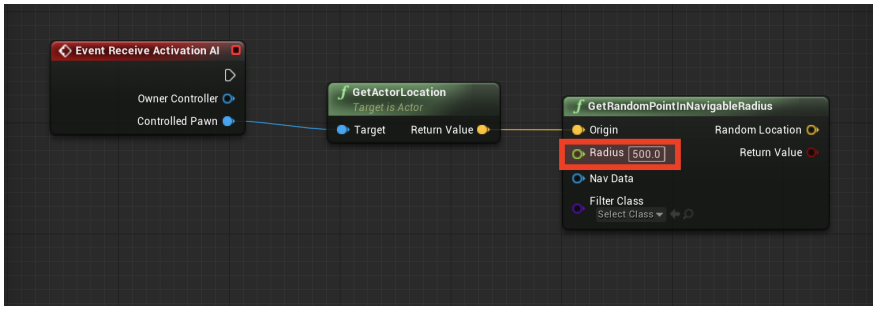
(b)

Actuellement le service ne fait rien il faut donc lui rajouter la fonctionnalité de la génération de la position aléatoire.

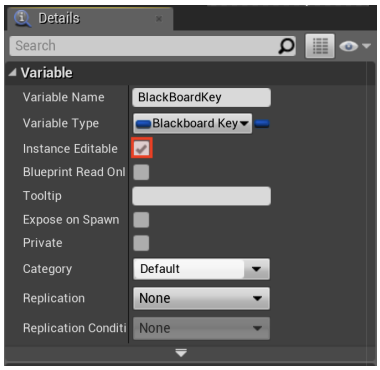
3.7 Dans l'*event graph* du service précédant (classe `BTService_SetRandomLocation`), rajoutez l'événement suivant :



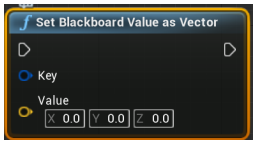
3.8 Rajoutez le calcul suivant qui va permettre d'obtenir la position aléatoire sur un rayon de taille 50 autour du personnage.



3.9 Afin de pouvoir définir une clé dans le *Blackboard*, il faut rajouter une variable de type *BlackboardKey* (toujours dans la classe `BTService_SetRandomLocation`, voir image ci-dessous).



3.10 Utilisez ensuite la fonction `SetBlackboardValueAsVector` de façon à mettre à jour la clé représentée par la variable que vous avez rajoutée dans la question précédente.

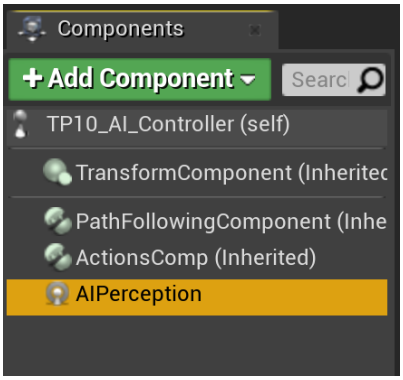


3.11 Enfin, pour que la comportement soit bien effectif, il reste encore à faire en sorte que le contrôler (`TP10_AI_Controller`), charge au lancement du jeu l'arbre de comportement (`TP10_BehaviorTree`). Pour cela dans l'*EventGraph* du contrôleur, cherchez une fonction d'exécuter l'arbre de comportement et faire en sorte qu'elle soit exécutée qu'une seule fois au lancement du jeu.

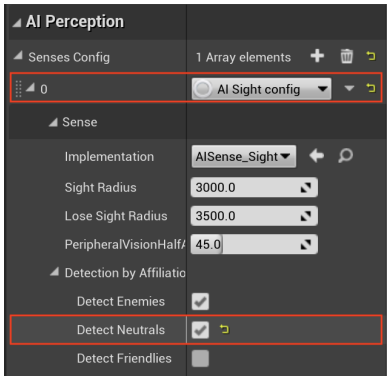
Partie IV: Mise en place d'une perception et d'un ennemi cible

Une perception (`AIPerception`) est un composant qu'il est possible de rajouter à un acteur et qui permet d'obtenir une faculté de perception visuelle ou auditive. Dans cette partie nous allons mettre en place cette perception pour faire en sorte qu'un acteur se déplacent vers un autre acteur dès qu'ils a une acteur apparaît dans son champ visuel.

4.1 La perception s'ajoute sur votre contrôleur (`TP10_AI_Controller`) dans l'onglet *Components* (image (a)).



(a)

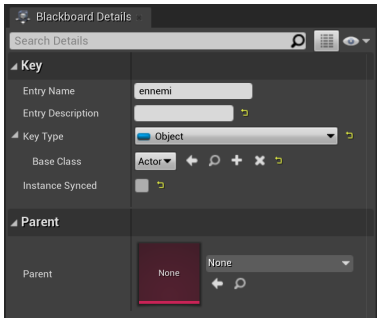


(b)

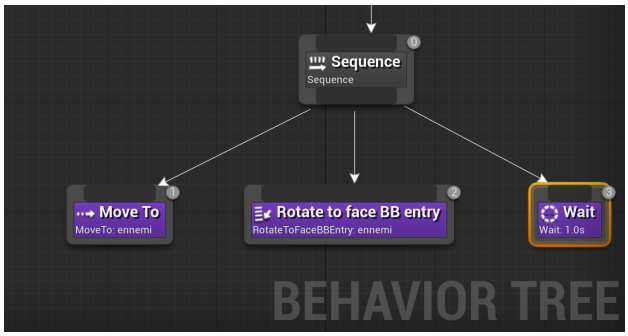
4.2 Dans l'onglet détails du composant précédant, vous pouvez rajoutez une configuration de perception (image (b) ci-dessus). Par défaut le composant précédant donne une perception d'un énemi, mais pour l'instant vous pouvez sélectionnez l'option permettant de détecter les éléments neutres.

Maintenant que nous avons rajouté la perception, il s'agit de rajouter clé ennemi dans le `Blackboard`. Ce dernier représentera la cible sur lequel votre personnage se dirigera.

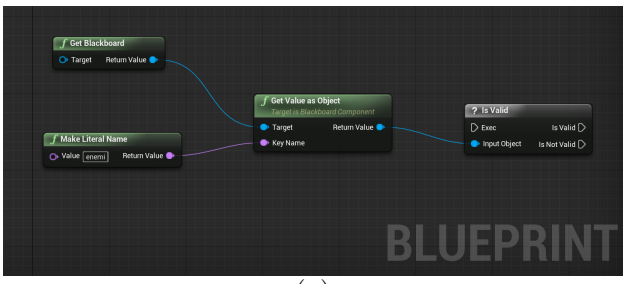
4.3 Rajoutez une nouvelle clé de type `Objet` dans votre objet `TP10_AI_BlackBoard` dont la classe de base sera de type `Actor`.



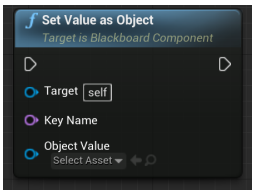
4.4 Modifiez l'arbre de façon à avoir la configuration suivante :



4.5 Dans le contrôleur, rajoutez le code suivant (image (a) ci-dessous) qui permet de détecter si un acteur a déjà un ennemi cible ou pas. Si l'acteur n'a pas de cible vous devez parcourir tous les acteurs du champ de vision en utilisant la fonction `GetCurrentlyPerceivedActors` et en définissant la variable du `TP10_AI_BlackBoard` avec l'acteur repéré (vous pourrez utiliser la fonction suivante (image (b)) qui permet d'enregistrer un objet à travers une clé dans un objet cible (le `Blackboard`).

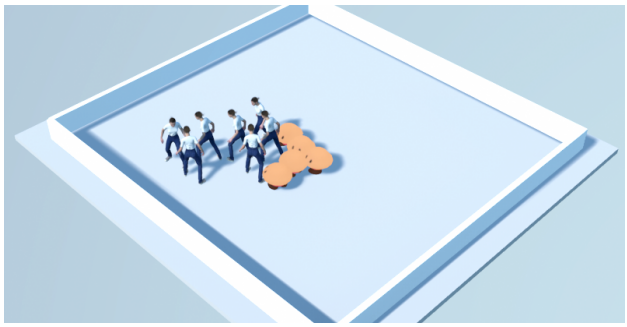


(a)



(b)

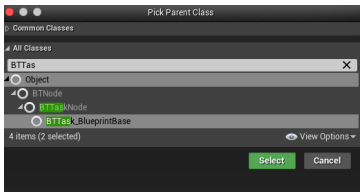
4.6 Testez que désormais vous avez les acteurs qui se déplacent vers une autre cible. Vous devriez obtenir quelque chose ressemblant à l'image suivante.



Partie VI: Création d'une attaque

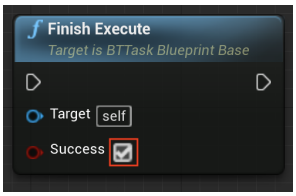
Pour créer une attaque, il est nécessaire de créer un objet associé à la tache d'attaque. Ensuite cet objet pourra être rajouté dans l'arbre de comportements. Pour la suite nous allons créer l'objet associé à l'attaque qui aura notamment le rôle de déclencher l'animation de l'attaque en elle même.

5.1 Le rajout d'un objet d'attaque se fait dans la fenêtre de l'exploration des contenus (*Content Browser*. Rajoutez un nouvel élément de type `BTTaskBlueprintBase` dans le dossier `BaseTP10/-Characters/AI` (voir image (a) ci-après). **Attention :** assurez-vous que la classe crée est hérite bien de `BTTaskBlueprintBase` (surtout si vous changez de nom car un bug supprime l'héritage en cas de changement de nom).



(a)

(b)

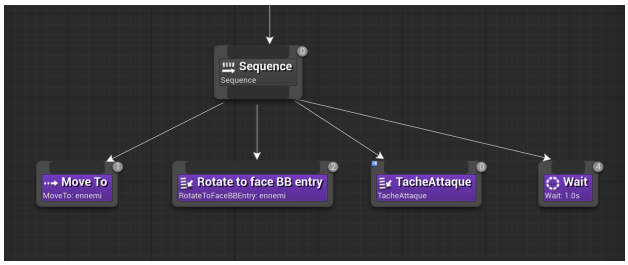


(c)

5.2 Ouvrez l'objet précédant et rajoutez un événement `EventReceiveExecuteAI` qui permet de déclencher des actions lorsque la tache est exécutée (image (b) ci-dessus).

5.3 Convertissez l'objet concerné par l'attaque en `BP_Muffin`, et mettre à jour son attribut `isAttacking`. Ensuite, il est nécessaire de rajouter un bloque spécifiant que l'attaque est terminée (bloque illustrée sur l'image (c)) et cochez l'option `success`.

5.4 Enfin pour mettre en place l'attaque, il faut la rajouter dans l'arbre des comportements :



M2 5.5 Détecter quand un acteur a été attaqué et dans ce cas lancer une animation faisant tomber l'acteur.

Partie VI: Pour ceux qui ont fini

Dans cette partie l'objectif sera de mettre en place un comportement permettant à un acteur de sortir d'un labyrinthe.

6.1 Écrire sur papier l'algorithme de la main droite qui permet de sortir d'un labyrinthe.

6.2 Dessiner un labyrinthe dans votre projet.

6.3 Implémentez les fonctions permettant de détecter si un mur/obstacle est présent à droite.

6.4 Même question avec un mur devant.

6.5 Implémentez l'algorithme de la première question et testez qu'un acteur arrive à sortir du labyrinthe.