GameBlitz : Course à obstacle

A person jumping over a structure

Description automatically generated

# Objectif

Créer un jeu qui est une variation du classique Serpent et Échelles.

Ceci n’est pas évalué, mais donne un aperçu de la forme de projet qu’on débutera sous peu. Rendez-vous le plus loin possible !

# Testez

Dans tous les fichiers des classes respectives, testez les fonctionnalités de vos objets ! (if \_\_main\_\_...).

Assurez-vous d’avoir le hotreload !

Classe Pion

## Visuel

Créer un pion en utilisant 2 cylindres et une sphère qui ressemble à ceci. La largeur de la base devra être 0.25 unité (hauteur approximative de 1 unité). La (les) couleur(s)/texture(s) doivent être configurable via le constructeur. Sauriez-vous utiliser le graphe de scène (SceneGraph) pour profiter de la hiérarchie (similairement aux Node de Godot) ?

A yellow object with a round base

Description automatically generated

## Fonctionnalités

Codez et testes les méthodes suivantes :

* void set\_offset(Vec3 offset) : valeur de décalage de la position qui sera toujours additionnée à l’interne par les 2 méthodes suivantes.
* *void teleport(Vec3 destination)* : déplacement le pion instantanément à la destination.
* *void animate\_to(Vec3[] list\_position, callable on\_end\_animate)* : Le pion doit glisser de sa position actuelle jusqu’à list\_position[0], puis de list\_position[0] à list\_position[1], etc… jusqu’à la fin.
  + Créer un callable on\_end\_animate pour signaler que *animate\_to* est terminée.
* *tuple* *compute\_rule()* : Selon l’état interne du pion, retourne un tuple (duet) de forme (int,int). Retourner (1,6) aucun état affecte le joueur.

## État

Dans l’entête du fichier, créez un enum nommé EEtat sous forme de flag (voir la liste dans le tableau plus bas).

* void add\_state(EEtat nouvel\_etat) : ajoute un état au joueur.
* EEtat get\_state() : retourne l’état actuel.
* string[] get\_state\_string() : retourne une liste de string contenant le nom de l’état affecté par le joueur. Peut être vide.
* void clear\_state() : remet l’état normal (0).

TESTEZ COMME IL FAUT AVANT DE CONTINUER !

# Classe Board

## Cases

Créez un plateau 6x6 (36 cases) avec des couleurs/textures alternées. Utilisez des cubes de 1x1.

A screenshot of a game

Description automatically generated

## Cases états

À l’aide d’une structure de votre choix, « pimentez » procéduralement le niveau en désignant des cases spéciales qui affecteront l’état du joueur. Ces cases doivent être allouées aléatoirement au moment de l’initialisation. Un seul affect par case est permis pour les cases [1,35].

|  |  |  |  |
| --- | --- | --- | --- |
| État | Couleur rec. | Nb de cases | Effet |
| CouchPotato | Brun foncé | 2 | Divise en 2 |
| Lazy | Jaune | 2 | Impose le dé [0,2] |
| Optimistic | Bleu foncé | 2 | Multiplie par 2 |
| RedBull | Vert pâle | 2 | Additionne 2 |
| Grumpy | Rouge foncé | 2 | Multiplie par -1 |
| Cafeinated | Brun pâle | 2 | Additionne 1 |
| 2YOMorph | Noir | 2 | Multiplie par -1 |
| DoNotCare | Bleu pâle | 1 | Impose le dé [2,8] |

## Fonctionnalités

* vec3 : tile\_to\_pos(int index\_case) : Donne la position 3D d’une tuile selon l’index de la case. La position est celle du dessus au centre du cube.
* *get\_state(*int index\_case*)*: retourne le type de case
* *is\_winning(int index\_case) :* retourne vrai si c’est la dernière case (index 35).
* *is\_at\_start(int index\_case) :* retourne vrai si c’est la première case (index 0).
* *can\_move\_to(int index\_case) :* retourne vrai si l’index est valide(entre 0 et 35).

TESTEZ COMME IL FAUT AVANT DE CONTINUER !

# Classe Arbiter

## État de jeu

En regardant la documentation de Ursine Engine, regardez comment faire un simple texte à l’écran. Le texte devra dire : « Commencement de la partie ! » et être placé en haut à gauche.

### État des joueurs

Sur le côté droit, on doit voir l’état de chaque joueur. Le texte doit être coloré selon la couleur du joueur.

## Fonctionnalité

* void game\_start() : Texte état du jeu : Commencement de la partie ! Appuyez sur espace!.
* void player\_ready(int id\_joueur, tuple (int,int)) : Texte état du jeu : Joueur id\_joueur, lancez le dé (min,max).
* int throw\_dice(tuple (int,int) ) : Lance un nombre au hasard (min et max définis par le duet en paramètre. Texte état du jeu : Lancé entre min et max : X avec min, max et X les valeurs appropriées.
* void player\_arrive(int id\_joueur, string[] status) : Selon si on a une case qui affecte l’état, Texte état du jeu : Joueur id\_joueur, aucun nouveau état ! ou Joueur id\_joueur : un nouveau état : nouveau\_etat !. La liste d’états du joueur doit se mettre à jour.
* void player\_winning(int id\_joueur) : Met à jour le texte avec Bravo id\_joueur !.

TESTEZ COMME IL FAUT AVANT DE CONTINUER !

# Classe GameEngine

Il est temps d’utiliser tous les objets ensemble pour créer le jeu ! Voici les règles (testez à chaque étape !)

## État initial

Les deux joueurs sont sur la case de l’index 0 avec une séparation (utilisez set\_offset sur les pions).

## Boucle de jeu

* Fiez-vous aux fonctionnalités de l’arbitre pour comprendre le flow du jeu. On change d’étape en appuyant toujours sur *espace*. (L’état initial est défini dans game\_start())
* À tour de rôle, chaque joueur lance le dé avec la touche « espace ». En vérifiant que le joueur puisse avancer à la destination, glisser le pion de case en case jusqu’à sa destination.
* Arrivé à sa destination, vérifiez si la case est spéciale et affectez l’état/terminez la partie. **Si le joueur arrive sur la case de départ, il perd tous ses états !**

### Arrêt de jeu

Dès qu’un des joueurs arrive exactement à la dernière case, arrêter la partie et mettre à jour le texte avec l’ID du gagnant.