Scott’s Escape - Documentation du code

# Module de perception des bruits

## Conception

La gestion du son suit un schéma simple : un émetteur génère un bruit portant des informations (objet émetteur, position d’émission, puissance du son) qui est capté par un manager puis répartit à des récepteurs qui décident ou non d’agir en fonction. Ceci va permettre d’émuler le fait que des gardes entendent ou pas les sons en fonction de la distance au joueur, mais aussi rendre plus facile la visualisation de ces sons.

## Classes

1. Noise
2. Noise Emitter
3. Noise Receiver
4. Noise Manager

## Mise en place

Pour que la perception des bruits fonctionne correctement, il est nécessaire que chaque agent émettant du son possède un composant héritant de *Noise Emitter* et redéfinisse la méthode *ComputeNoise* correspondant à la manière d’émettre le son de l’agent. De plus, il faut que chaque agent capable de percevoir et réagir à des sons possède un composant *Noise Receiver*. Enfin, il faut qu’un objet porte le composant *Noise Manager* pour synchroniser le tout.

# Module de gestion du niveau d’ADN

## Conception

L’objectif de la gestion du niveau d’ADN du personnage est ensuite d’y rattacher l’évolution de ses caractéristiques (vie, taille, vitesse, etc) et l’apparition/disparition de certaines compétences (grappin, attaque, etc). Cependant cette partie ne va définir que la gestion du niveau d’ADN, les caractéristiques et les compétences étant définies plus loin dans ce document.

Ainsi, le joueur à la possibilité de manger certains éléments d’une certaines valeur faisant monter une jauge. Une fois cette jauge remplie elle revient à 0 et le joueur gagne un niveau. La jauge demande alors plus de valeur de nourriture pour être remplie. Si le joueur prend trop de dégâts ou lors de certains évènements (tuer un ennemi), il perd un niveau. Si le joueur est déjà au niveau 1, il meurt. Le niveau du joueur se situe toujours entre 1 et 3. Il faut 1 de nourriture pour passer au niveau 2, 3 de nourriture pour passer au niveau 3 et 7 de nourriture pour maximiser le niveau 3

## Classes

1. PlayerDNALevel
2. FoodController : Interactable

## Mise en place

La classe PlayerDNALevel doit être un composant du Player, au même niveau que son PlayerEntityController. La classe PlayerDNALevel est entièrement dépendante de PlayerEntityController et n’a pas besoin d’être manipulée ; elle est à considérer Read-only. Les différents évènements et valeurs la concernant sont accessibles via les évènements OnDnaLevelChanged (donne la quantité d’ADN via un float), OnCurrentLevelChanged (donne le niveau actuel via un int) et OnDies.

La classe FoodController doit être un composant de tous les objets considérés comme de la nourriture absorbable par le joueur.