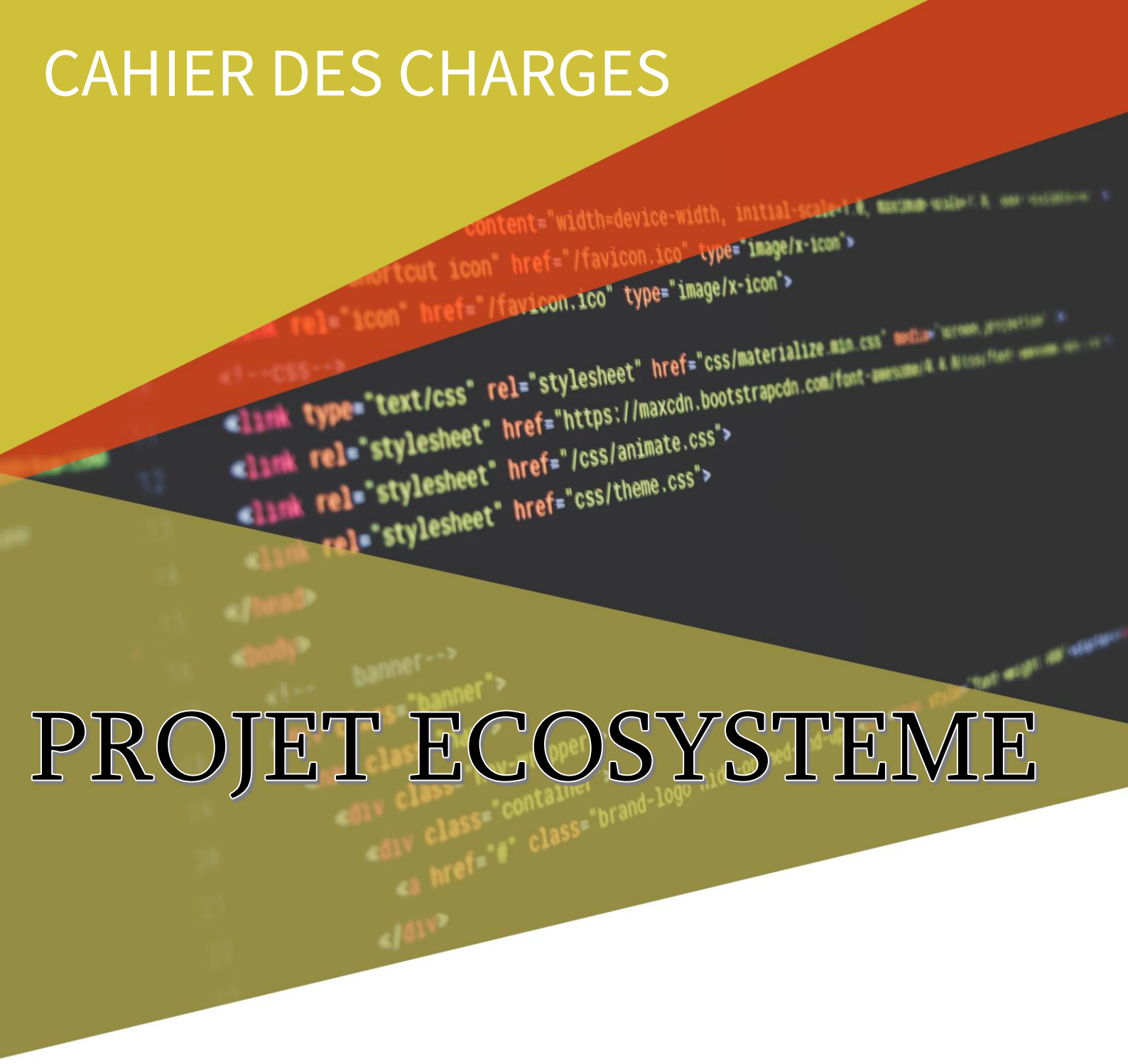
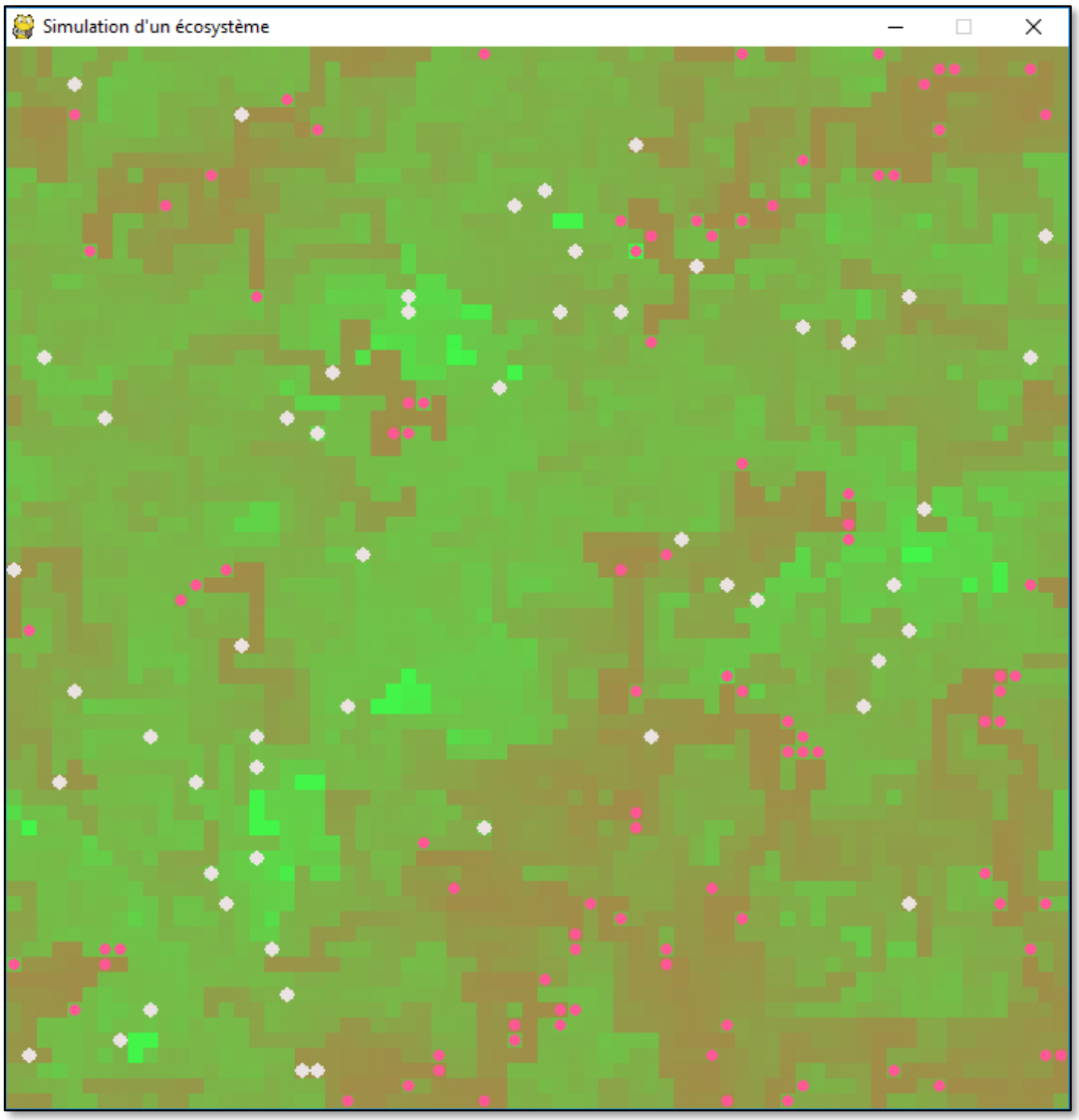
CAHIER DES CHARGES



PROJET ECOSYSTEME

Marc Guillemot, Terminale F
Aurélien Kittel, Terminale A
Cyprien Bourotte

SOMMAIRE



I. Objectifs :

A quoi servira ce code ? Résultats attendus ?

Le but de ce projet est de créer un écosystème composé d'herbe, de moutons et de loups capable de se reproduire.

Nous avons donc décider de le traiter sur le langage Python. Ainsi, nous avons essayé d'ajouter plusieurs autres fonctionnalités tels que la détection de loups et de la plus grande quantité d'herbe à proximité. De plus, nous allons créer une classe « créature » mettant en relation la classe loup et la classe mouton. Un autre fichier « monde » nous permettra de mettre en relation le fichier herbe et les fichiers créature, loup et mouton. De plus, nous aurons un fichier générant une interface graphique et la gérant au fur et à mesure : « GUI ». Enfin, un fichier « main » servira à lancer le programme et mettre en relation tous les fichiers : créature, mouton, loup, herbe, monde et GUI.

Tout cela nous permettra donc de représenter un petit écosystème en temps réel à travers l'interface graphique.

II. Fonctionnalités :

De quoi aurons-nous besoin pour réaliser ce projet ?

Pour ce projet, nous aurons besoin de créer plusieurs classes, et, par conséquent, créer plusieurs fichiers.

Nous aurons également besoin de créer une interface graphique.

Nous définirons également les attributs de chaque classe de manière publique afin d'éviter de devoir écrire des setters/getters.

Bien que cette simulation s'exécute de manière automatique, nous devrons laisser la possibilité de pouvoir influer sur cette dernière en pouvant jouter des loups ou encore des moutons.

III. Contraintes :

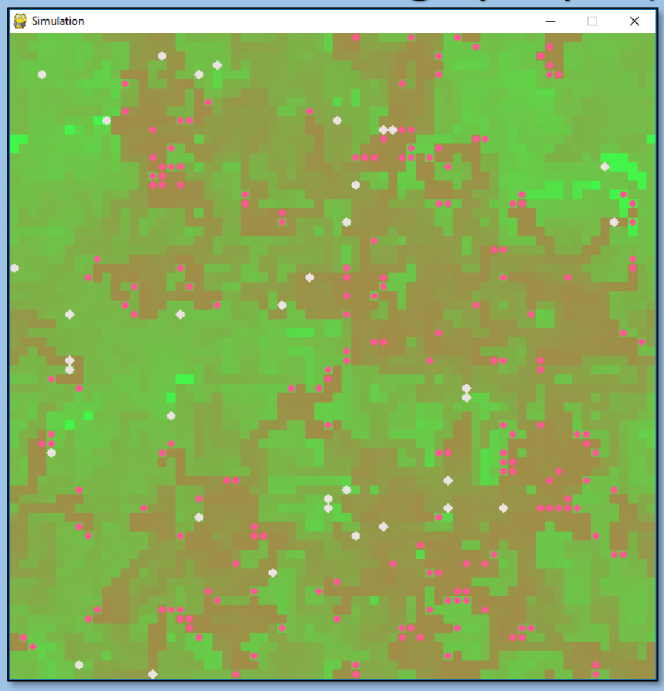
Quelles contraintes devrons-nous respecter ?

Nous devrons réaliser ce projet par groupe de 3. Nous devrons également créer plusieurs fichiers python contenant les différentes classes (au minimum 5) puis nous devons les mettre en relations via les importations ainsi que de les lancer tous en même temps depuis un fichier main.

IV. Diagrammes représentant l'interrelation des fichiers

« MAIN » : le lanceur

« GUI » : classe de l'interface graphique (Pygame)



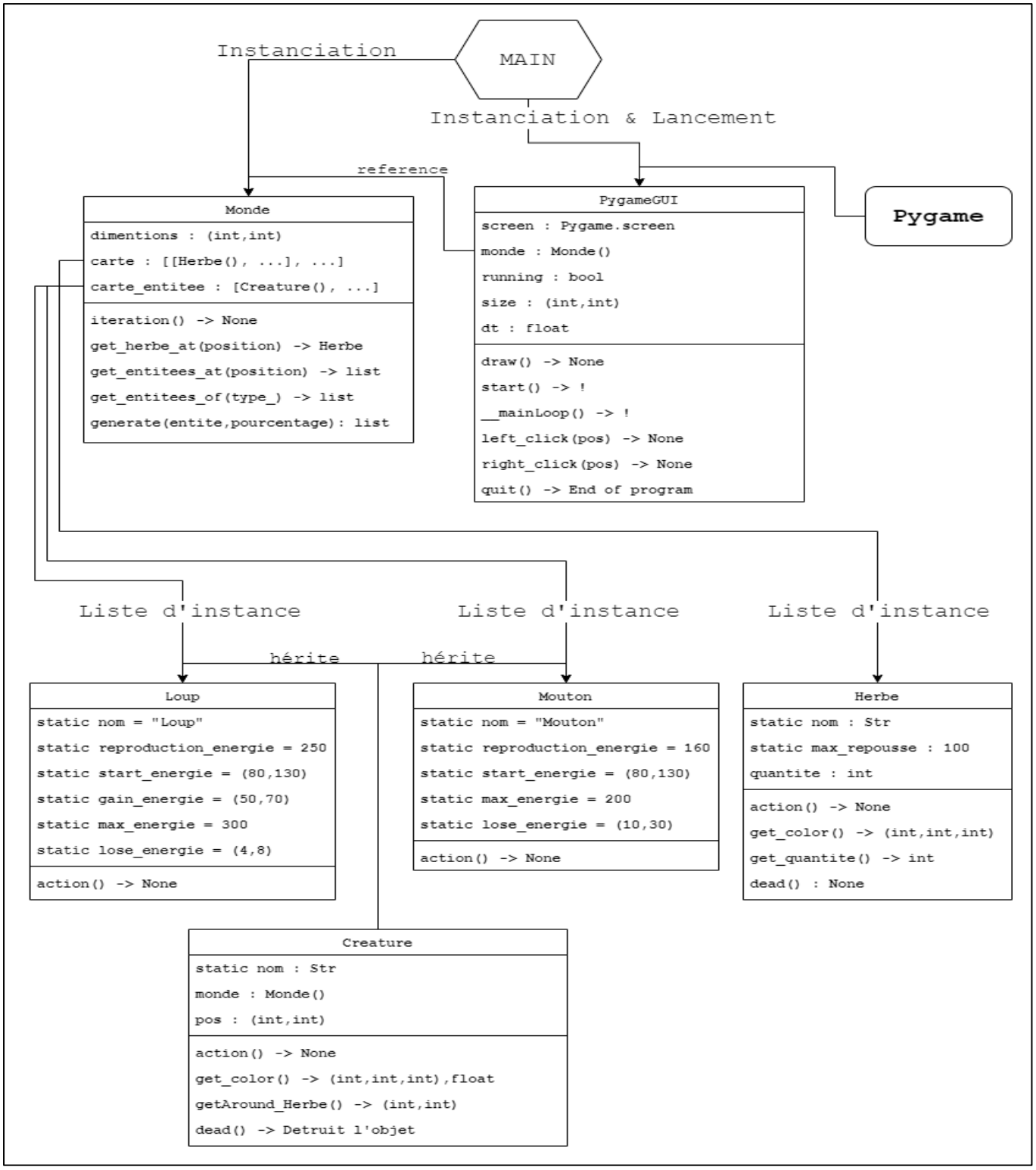
« MONDE » : classe créant et gérant le monde et ses occupants.

« CREATURE »

« MOUTON »

« LOUP »

« HERBE »


Instanciation
MAIN
Instanciation & Lancement
reference
Monde
dimentions : (int,int)
carte : [[Herbe(), ...], ...]
carte_entitee : [Creature(), ...]
iteration() -> None
get_herbe_at(position) -> Herbe
get_entitees_at(position) -> list
get_entitees_of(type_) -> list
generate(entite,pourcentage): list
PygameGUI
screen : Pygame.screen
monde : Monde()
running : bool
size : (int,int)
dt : float
draw() -> None
start() -> !
__mainLoop() -> !
left_click(pos) -> None
right_click(pos) -> None
quit() -> End of program
Pygame
Liste d'instance
Liste d'instance
Liste d'instance
hérite
hérite
Loup
static nom = "Loup"
static reproduction_energie = 250
static start_energie = (80,130)
static gain_energie = (50,70)
static max_energie = 300
static lose_energie = (4,8)
action() -> None
Mouton
static nom = "Mouton"
static reproduction_energie = 160
static start_energie = (80,130)
static max_energie = 200
static lose_energie = (10,30)
action() -> None
Herbe
static nom : Str
static max_repousse : 100
quantite : int
action() -> None
get_color() -> (int,int,int)
get_quantite() -> int
dead() : None
Creature
static nom : Str
monde : Monde()
pos : (int,int)
action() -> None
get_color() -> (int,int,int),float
getAround_Herbe() -> (int,int)
dead() -> Detruit l'objet

V. Pseudo-Code

Quelles sont les différentes étapes que le code devra respecter ?

Phase 1 : création des classes (Programmation Orientée Objet)

- ⇨ **MOUTON // LOUP**

Ces classes servent à gérer les caractéristiques et les actions des loups et des moutons au fur et à mesure de la simulation.

Importation des modules nécessaires

Création de la classe avec ses instances de classe, ses attributs (publics) et sa méthode comme détaillé ci-dessus.

Ainsi, tous deux possèdent une méthode action gérant les actions de ces derniers : nourriture, mouvement et reproduction.

Ainsi, on fait appel aux fonctions getAround() et getAroundHerbe(), définies dans le fichier et la classe créature, pour se diriger vers l'endroit le plus fourni en terme de quantité d'herbe à 4 blocs (sans les diagonales) pour les moutons et vers l'endroit où il y a des moutons pour le loup sur 8 blocs (avec les diagonales) sinon aléatoirement.

Ces classes permettent également de faire perdre de l'énergie à chaque itération aux créatures grâce à la méthode lose_energie() contenue dans la classe créature.

De même, la méthode dead() dans créature, organise la mort des entités.

- ⇨ **CREATURE**

Cette classe est la classe mère des classes loup et mouton. Elle rassemble donc tous les points communs de ces classes et notamment leurs méthodes.

Importation des modules nécessaires.

Création de la classe avec ses instances de classe, ses attributs (publics) et sa méthode comme détaillé ci-dessus.

Ainsi, comme dit ci-dessus, la classe contient donc les méthodes dead(), getAround() et getAroundHerbe() mais également la méthode getColor() créant la couleur et la taille des points représentant les créatures pour l'interface graphique, que nous verrons plus tard.

- ⇨ **HERBE**

Cette classe sert à gérer les caractéristiques et les actions de l'herbe au fur et à mesure de la simulation.

Importation des modules nécessaires

Création de la classe avec ses instances de classe, ses attributs (publics) et ses méthodes comme détaillé ci-dessus.

Ainsi, la classe herbe possède une méthode getColor() générant les variations de la couleur de l'herbe au fur et à mesure des itérations. Ces dernières sont générées grâce à un code RGB, ainsi qu'un ratio permettant de rendre plus fluide le passage d'une couleur à une autre.

De même, la classe herbe possède une méthode action() régénérant à chaque itération d'un certain pourcentage chaque bloc d'herbe ainsi que d'une méthode dead() gérant la « mort » d'un bloc d'herbe.

Phase 2 : Mise en relation des données des classes (Programmation Orientée Objet)

- ⇨ **MONDE**

Cette classe sert à générer le monde, la simulation.

Importation des modules nécessaires

Création de la classe avec ses instances de classe, ses attributs (publics) et ses méthodes comme détaillé ci-dessus.

Ainsi, la classe herbe possède une méthode itération() qui exécute UNE itération du monde au cours de laquelle chaque entité (herbe, animaux…) exécute son action ; mais elle compte également deux méthodes get_entites_at() et get_entites_of() retournant les entités d'un certain type ou retournant les entités situées à une position donnée.

Enfin, la classe Monde possède une dernière méthode generate() qui génère un pourcentage d'entités (herbe, mouton, loup…) sur des positions dans le monde.

Phase 3 : Création d'une interface graphique

⇨ **PYGAMEGUI**

Cette classe sert à générer l'interface graphique.

Importation des modules nécessaires

Création de la classe avec ses instances de classe, ses attributs (publics) et ses méthodes comme détaillé ci-dessus.

Ainsi, la classe PygameGui possède une méthode draw(), dessinant le contenu de la fenêtre Pygame : herbe, loups et moutons.

La classe PygameGui possède également une méthode start() servant à lancer la boucle principale définie dans la méthode _mainLoop(). Cette boucle est notamment constituée de détection d'events ainsi que de réactions à avoir pour chacun d'eux. Ces réactions sont également définies dans des méthodes nommées left_click() et right_click().

Enfin, la classe PygameGui possède une dernière méthode quit() permettant de quitter à tout moment la fenêtre Pygame ce qui n'est pas possible initialement.

Phase 4 : Lancement général (Programmation instructive)

Ce fichier sert à lancer le programme en entier et donc de mettre en relation tous les fichiers et toutes les classes.

Importation des modules nécessaires

Lancement du programme grâce à la condition :

```
if __name__ == '__main__':
    monde = Monde((70,70))
    gui = PygameGui(monde,(700,700))
    gui.start()
```

VI. Problèmes :

Quels pourraient être les problèmes rencontrés ?

- ⇨ Mauvaises mises en relation et importations des différents fichiers les uns dans les autres.
- ⇨ Création des différentes entités au fur et à mesure du jeu.
- ⇨ Problèmes liés à l'interface graphique Pygame dus à la récente découverte de ce module.
- ⇨ Faire attention à la nomination des différentes classes et variables dans les différents fichiers ainsi qu'à leur utilisation.

VII. Limites

Quelles limites ce projet peut-il rencontrer ?

Ce programme possède des limites sur le fait qu'il ne peut pas prévoir ou prendre en compte le placement des loups et des moutons au départ ainsi que de la quantité d'herbe. Cela peut donc engendrer une simulation qui se termine très rapidement sans avoir véritablement eu le temps de commencer et pouvant ne pas très bien représenter l'auto-suffisance d'un écosystème.

VIII. Planning

Quelles dates nous fixons-nous pour les différentes étapes ?

Date de début du projet : 18 septembre 2020.

- Date maximum de lancement du développement du code : 20 septembre 2020.
  - ➔ Lancé le 18 septembre 2020.

- Date de rendu : 09 octobre 2020
  - ➔ Fini le : 2 octobre 2020.