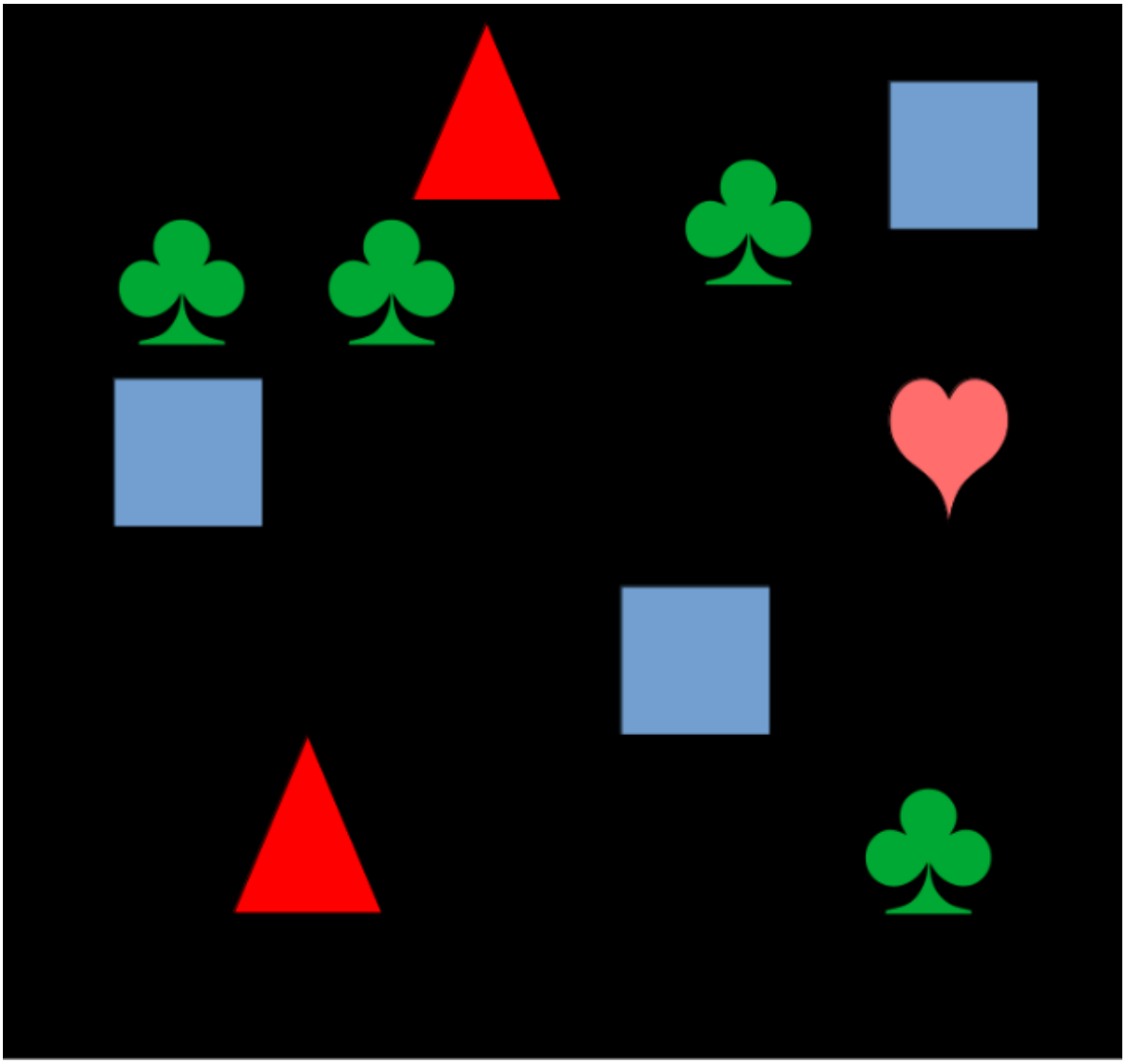
Conception



Type	Conception
Nom du projet	MicroSymbiosisTheory
Commentaire	Projet IPI, S2, Enib
Auteur	Maïa Le Corre et Mathieu Veber
Version	1.0
Date	13/04/2022

Table des matières

1 Rappel du cahier des charges

1.1 Contraintes techniques

- Le logiciel est associé à un cours, il doit donc fonctionner sur les machines de TP de l'ENIB pour que les élèves puissent le tester.
- Le langage utilisé en cours est Python. Le développement devra donc se faire en python.
- Les notions de programmation orientée objet n'ayant pas encore été abordées, le programme devra essentiellement s'appuyer sur le Paradigme de la programmation procédurale.
- Le logiciel devra être réalisé en conformité avec les pratiques préconisées en cours de MDD : barrière d'abstraction, modularité, unicode, etc...
- L'interface sera réalisée en mode texte dans un terminal

1.2 Fonctionnalités

F1 : Lancer une partie
- F1.1 : Jouer une partie
  - F1.1.1 : Afficher le jeu :
    - -Grille
    - -Score
    - -Entités
  - F1.1.2 : Se déplacer sur la grille ou manger
  - F1.1.3 : Tour des entités
  - F1.1.4 : Mort du joueur
- F1.2: Fin de partie
  - F1.2.1 : Afficher le score
  - F1.2.2 : Quitter le jeu

1.3 P1 : Prototype P1

Ce prototype porte essentiellement sur la création de la grille, sur l'affichage et sur le développement des Herbivores et Plantes.

Mise en œuvre des fonctionnalités : F1.1.1, F1.1.3

Livré dans une archive au format .zip ou .tgz

Contient un manuel d'utilisation dans le fichier readme.txt

1.4 P2 : Prototype P2

Ce prototype réalise toutes les fonctionnalités.

Ajout à P1 des fonctionnalités F1, F.1.1.2, F1.1.4, F1.2, F1.2.1, F1.2.2.

Livré dans une archive au format .zip ou .tgz

Contient un manuel d'utilisation dans le fichier readme.txt

2 Principes des solutions techniques

2.1 Langage

Conformément aux contraintes énoncées dans le cahier des charges, le codage est réalisé avec langage python. Nous choisissons la version 3.10.4

2.2 Architecture du logiciel

Nous mettons en oeuvre le principe de la barrière d'abstraction. Chaque module correspond à un type de donnée et fournit toutes les opérations permettant de le manipuler de manière abstraite.

2.3 Interface utilisateur.

L'interface utilisateur se fera via un terminal de type linux.

Nous reprenons la solution donnée en cours de IPI en utilisant les modules :

```
sys, os, copy, random, termios
```

2.3.1 Boucle de simulation.

Le programme mettra en œuvre une boucle de simulation qui gèrera l'affichage et les événements clavier.

2.3.2 Affichage

L'affichage se fait en communiquant directement avec le terminal en envoyant des chaînes de caractères sur la sortie standard de l'application.

2.3.3 Gestion du clavier

L'entrée standard est utilisée pour détecter les actions de l'utilisateur.

Le module `tty` permet de rediriger les événements clavier sur l'entrée standard.

Pour connaître les actions de l'utilisateur, il suffit de lire l'entrée standard.

2.4 Grille, personnages

Pour modéliser le plateau de jeu, une liste de liste (x*y) permet de stocker des caractères correspondant aux entités posés sur la grille

Par exemple:
carte =[
[‘ ‘, ‘■’, ‘ ’],
[‘ ‘,’ ‘,’ ‘],
[‘ ‘,’ ‘, ‘♣’]]

3 Analyse

3.1 Analyse des noms/verbes.

Verbes : nommer, choisir, jouer, afficher, déplacer, manger, se reproduire, mourir, finir, quitter.

Nom : joueur, grille, nom, score, joueur, entités, herbivores, carnivores, plantes.

3.2 Type des données.

type:Gamedata = struct

Map : [[str,str,str,str],[str,str,str,str]]

entities : {Herbivore, Carnivore, Plante}

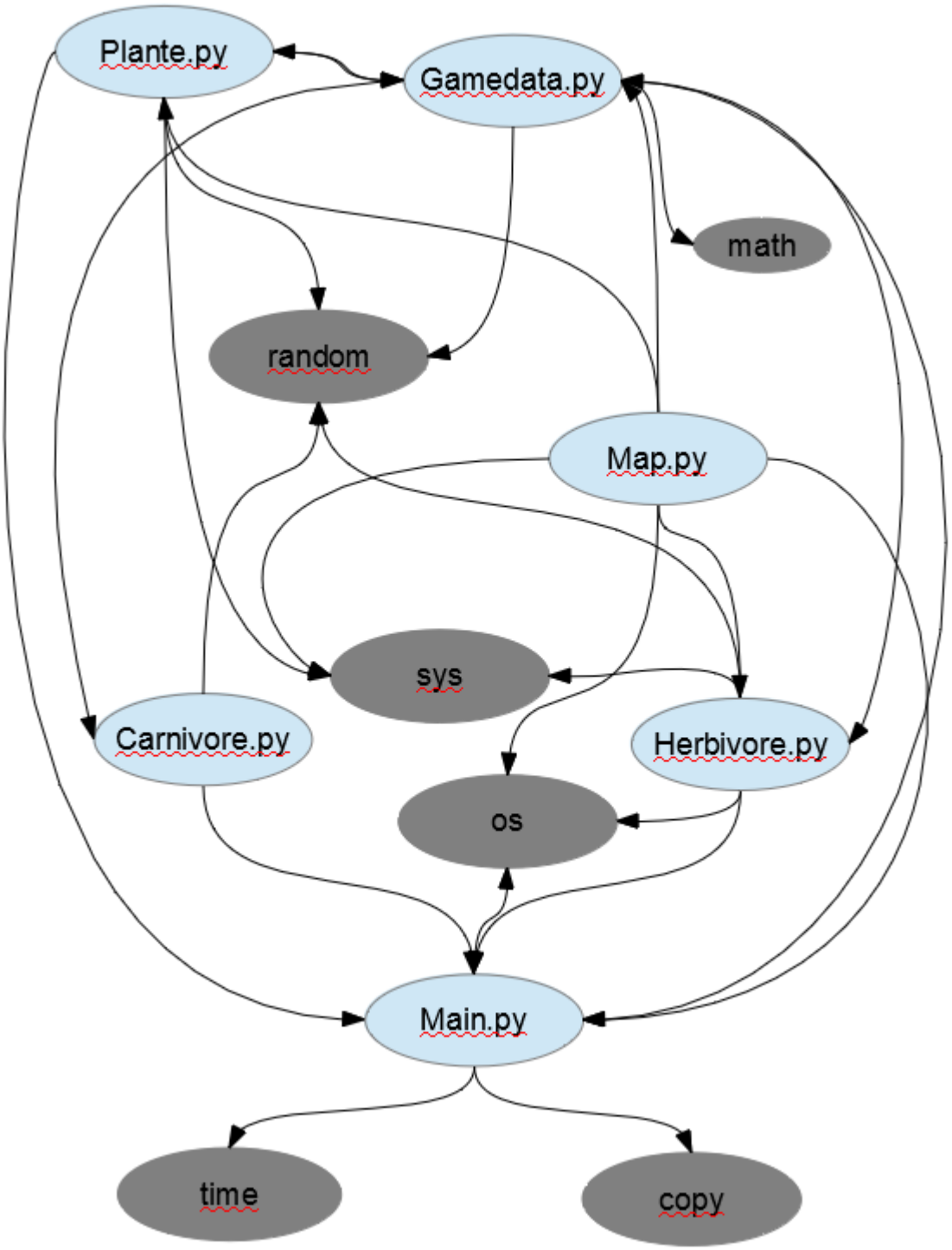
fstruct

type : Herbivore = struct
position : (int,int)
manger : bool
skin : str

type : Carnivore = struct
position : (int,int)
manger : bool
skin : str

type : Plantes = struct
position : (int,int)
skin : str

3.3 Dépendance entre module.



3.4 Analyse descendante

3.4.1 Arbre principal:

```
Main.main()
+-- Main.init()
|       +--Gamedata.create()
|       +--Gamedata.addHerbivore()
|       +--Gamedata.addPlante()
|       +--Gamedata.Carnivore()
|
+--Main.run()
        +--Main.show()
        +--Main.interract()
```

3.4.2 Arbre affichage

```
Main.show()
        +--Map.show()
```

3.4.3 Arbre interaction

```
Main.interact()
        |
        +--Herbivore.play()
        |       +--Gamedata.get_herbivore()
        |               +--Herbivore.getmanger()
        |               +--Herbivore.can_reproduce()
        |               |       +--Herbivore.reproduce()
        |               |       +--Herbivore.set_manger()
        |               +--Herbivore.can_eat()
        |               |       +--Herbivore.eat()
        |               |       +--Herbivore.set_manger()
        |               +--Herbivore.gotofood()
        |               +--Gamedata.update_herbivore()
        |
        +--Carnivore.play()
        |       +--Gamedata.get_carnivore()
        |               +--Carnivore.getmanger()
        |               +--Carnivore.can_reproduce()
        |               |       +--Carnivore.reproduce()
        |               |       +--Carnivore.set_manger()
        |               +--Carnivore.can_eat()
        |               |       +--Carnivore.eat()
```

```
|                 |         +--Carnivore.set_manger()
|                 +--Carnivore.gotofood()
|                 +--Gamedata.update_carnivore()
|
|
+--Gamedata.get_plante()
        +--Plante.play()
                +--Plante.randomposition()
                +--Gamedata.add_Plante()
```

4 Description des fonctions.

4.1 Programme Principal : Main.py

```
Main.main()
Main.init()
Main.run()
Main.show(gamedata)
Main.interact(gamedata, dt)
```

```
Main.main() ->None
      Description : fonction principale du jeu

      Paramètres : aucun

      Valeur de retour :aucune


Main.init() -> dictionnaire
      Description : initialisation du jeu

      Paramètres : aucun

      Valeur de retour : dictionnaire


Main.run()->rien

      Description : boucle de simulation

      Paramètres : aucun

      Valeur de retour :aucune
```

Main.**show(gamedata)**->rien

Description : Affiche le jeu

Paramètres :

gamedata : dictionnaire

Valeur de retour :aucune

Main.**interact(gamedata, dt)**-> dictionnaire

Description : gère tous les événements

Paramètres :

gamedata : dictionnaire

dt : float

Valeur de retour :dictionnaire

4.2 Map.py

Map.create(sizex, sizey)
Map.get_carte(carte)
Map.get_size(carte)
Map.show(carte)
Map.isinmap(newposition, carte)

Map.create() -> carte

Description: Crée une carte vide

Paramètres: rien

Valeur de retour: carte vide

Map.get_carte(carte)-> carte

Description: retourne la carte

Paramètres:

carte : matrice

Valeur de retour: carte

```
Map.get_size(carte) -> tuple
      Description: donne la taille de la carte

      Paramètres:

            carte : matrice

      Valeur de retour: tuple d'entiers

Map.isinmap(newposition, carte) -> bool
      Description: Permet de savoir si les coordonnes sont dans les
limites de la carte

      Paramètres:

            newposition : tuple

            carte : carte

      Valeur de retour: 0 si pas dans les limites, 1 si dans les
limites
```

4.3 Herbivore.py

Herbivore.create(position : tuple)
Herbivore.get_position(creature : dict)
Herbivore.get_manger(creature : dict)
Herbivore.set_position(creature : dict, newposition : tuple)
Herbivore.set_manger(creature : dict, etat : bool)
Herbivore.move(creature : dict, gamedata : dict, direction : str, allposition : list)
Herbivore.can_reproduce(creature : dict, gamedata : dict, allposition : list)
Herbivore.reproduce(creature : dict, gamedata : dict, allposition : list)
Herbivore.caneat(creature : dict, gamedata : dict)
Herbivore.eat(creature : dict, gamedata : dict)
Herbivore.gotofood(creature : dict, gamedata : dict, allposition : tuple)
Herbivore.show(creature : dict)

```
Herbivore.create(position : tuple) -> dict
      Description: Créer une entité herbivore

      Paramètres:

            position : tuple d'entiers

      Valeur de retour: dictionnaire comprenant l'apparence, la
position, l'état de faim
```

Herbivore.**get_position**(creature : dict)->tuple

Description: renvoie la position de l'entité

Paramètres:

creature : dictionnaire

Valeur de retour: tuple d’entiers

Herbivore.**get_manger**(creature : dict)-> bool

Description: renvoie l'état de faim

Paramètres:

creature : dictionnaire

Valeur de retour: 1 si a mangé, 0 si n’a pas mangé

Herbivore.**set_position**(creature : dict, newposition : tuple)->dictionnaire

Description: change la position

Paramètres:

creature : dictionnaire

newposition : tuple

Valeur de retour: dictionnaire

Herbivore.**set_manger**(creature : dict, etat : bool) -> dict

Description: change l'état de faim

Paramètres:

creature : dictionnaire

etat : bool

Valeur de retour: dictionnaire

Herbivore.**move**(creature : dict, gamedata : dict, direction : str, allposition : list) -> dict

Description: Permet de bouger l'entité

Paramètres:

creature : dictionnaire

gamedata : dictionnaire

direction : str

allposition : list de tuple

Valeur de retour: dictionnaire

Herbivore.**can_reproduce**(creature : dict, gamedata : dict, allposition : tuple) -> bool

Description: Permet de savoir si l'entité peut se reproduire

Paramètres:

creature : dictionnaire

gamedata : dictionnaire

allposition : list de tuple

Valeur de retour: 1 si peut se reproduire, 0 si ne peut pas

Herbivore.**reproduce**(creature : dict, gamedata : dict, allposition : tuple) -> dict

Description: Créer une nouvelle entité du même type sur une case libre autour

Paramètres:

creature : dictionnaire

gamedata : dictionnaire

allposition : list de tuple

Valeur de retour: dictionnaire, une nouvelle entité

Herbivore.**caneat**(creature : dict, gamedata : dict) -> bool:

Description: Permet de savoir si l'entité peut manger

Paramètres:

creature : dictionnaire

gamedata : dictionnaire

Valeur de retour: 1 si peut manger, 0 si ne peut pas

Herbivore.**eat**(creature : dict, gamedata : dict) -> dict:

Description: Permet de manger une entité à proximité

Paramètres:

creature : dictionnaire

gamedata : dictionnaire

Valeur de retour: dictionnaire, gamedata modifié sans l'élément qui a été mangé

Herbivore.**gotofood**(creature : dict, gamedata : dict, allposition : tuple) -> dict:

Description: Déplace l’entité vers la nourriture la plus proche

Paramètres:

creature : dictionnaire

gamedata : dictionnaire

allposition : list de tuple

Valeur de retour: dictionnaire, avec la position modifié

```
Herbivore.show(creature : dict)->None
Description: Affiche l'entité

      Paramètres:

            creature : dictionnaire

      Valeur de retour: rien
```

4.4 Carnivore.py

Carnivore.create(position : tuple)
Carnivore.get_position(creature : dict)
Carnivore.get_manger(creature : dict)
Carnivore.set_position(creature : dict, newposition : tuple)
Carnivore.set_manger(creature : dict, etat : bool)
Carnivore.move(creature : dict, gamedata : dict, direction : str, allposition : list)
Carnivore.can_reproduce(creature : dict, gamedata : dict, allposition : list)
Carnivore.reproduce(creature : dict, gamedata : dict, allposition : list)
Carnivore.caneat(creature : dict, gamedata : dict)
Carnivore.eat(creature : dict, gamedata : dict)
Carnivore.gotofood(creature : dict, gamedata : dict, allposition : tuple)
Carnivore.show(creature : dict)

```
Carnivore.create(position : tuple) -> dict
      Description: Créer une entité carnivore

      Paramètres:

            position : tuple d'entiers

      Valeur de retour: dictionnaire comprenant l'apparence, la
position, l'état de faim
```

```
Carnivore.get_position(creature : dict)->tuple
      Description: renvoie la position de l'entité

      Paramètres:

            creature : dictionnaire

      Valeur de retour: tuple d'entiers
```

Carnivore.**get_manger**(creature : dict)-> bool

Description: renvoie l'état de faim

Paramètres:

creature : dictionnaire

Valeur de retour: 1 si a mangé, 0 si n'a pas mangé

Carnivore.**set_position**(creature : dict, newposition : tuple)->dictionnaire

Description: change la position

Paramètres:

creature : dictionnaire

newposition : tuple

Valeur de retour: dictionnaire

Carnivore.**set_manger**(creature : dict, etat : bool) -> dict

Description: change l'état de faim

Paramètres:

creature : dictionnaire

etat : bool

Valeur de retour: dictionnaire

Carnivore.**move**(creature : dict, gamedata : dict, direction : str, allposition : list) -> dict

Description: Permet de bouger l'entité

Paramètres:

creature : dictionnaire

gamedata : dictionnaire

direction : str

allposition : list de tuple

Valeur de retour: dictionnaire

Carnivore.**can_reproduce**(creature : dict, gamedata : dict, allposition : list) -> bool

Description: Permet de savoir si l'entité peut se reproduire

Paramètres:

creature : dictionnaire

gamedata : dictionnaire

allposition : list de tuple

Valeur de retour: 1 si peut se reproduire, 0 si ne peut pas

Carnivore.**reproduce**(creature : dict, gamedata : dict, allposition : list) -> dict

Description: Créer une nouvelle entité du même type sur une case libre autour

Paramètres:

creature : dictionnaire

gamedata : dictionnaire

allposition : list de tuple

Valeur de retour: dictionnaire, une nouvelle entité

Carnivore.**caneat**(creature : dict, gamedata : dict) -> bool:

Description: Permet de savoir si l'entité peut manger

Paramètres:

creature : dictionnaire

gamedata : dictionnaire

Valeur de retour: 1 si peut manger, 0 si ne peut pas

Carnivore.**eat**(creature : dict, gamedata : dict) -> dict:

Description: Permet de manger une entité à proximité

Paramètres:

creature : dictionnaire

gamedata : dictionnaire

Valeur de retour: dictionnaire, gamedata modifié sans l'élément qui a été mangé

Carnivore.**gotofood**(creature : dict, gamedata : dict, allposition : list) -> dict:

Description: Déplace l'entité vers la nourriture la plus proche

Paramètres:

creature : dictionnaire

gamedata : dictionnaire

allposition : list de tuple

Valeur de retour: dictionnaire, avec la position modifié

Carnivore.**show**(creature)->None

Description: Affiche l'entité

Paramètres:

creature : dictionnaire

Valeur de retour: rien

4.5 Plante.py

Plante.create(position)
Plante.get_skin(plante)
Plante.get_position(plante)

```
Plante.create(position : tuple) ->  plante
      Description: Créer une entité plante

      Paramètres:

            position : tuple d'entiers

      Valeur de retour: dictionnaire comprenant l'apparence et la
position

Plante.get_skin(plante : dict) -> str

      Description: renvoie l'apparence de la plante

      Paramètres:

            plante : dict

      Valeur de retour: str


Plante.getposition(plante : dict) -> tuple

      Description: renvoie l'apparence la position

      Paramètres:

            plante : dict

      Valeur de retour: tuple
```

4.6 Gamedata.py

Gamedata.create()
Gamedata.addplante(gamedata : dict, position : tuple)
Gamedata.addCarnivore(gamedata : dict, position : tuple)
Gamedata.addHerbivore(gamedata : dict, position : tuple)
Gamedata.get_allposition(gamedata : dict)
Gamedata.get_allposition_nearby(gamedata : dict, position : tuple)
Gamedata.get_herbivore(gamedata : dict)

Gamedata.get_carnivore(gamedata : dict)
Gamedata.get_plante(gamedata : dict)
Gamedata.get_herbivore_position(gamedata : dict)
Gamedata.get_carnivore_position(gamedata : dict)
Gamedata.get_plante_position(gamedata : dict)
Gamedata.kill_plante(gamedata : dict, plantetokill : int)
Gamedata.kill_carnivore(gamedata : dict, carnivoretokill : int)
Gamedata.kill_herbivore(gamedata : dict, herbivoretokill : int)
Gamedata.distance(position1, position2)
Gamedata.count_nearby_entities(gamedata : dict, creature : dict, thingtocount)
Gamedata.isinmap(newposition : tuple,carte : list)
Gamedata.valid_move(gamedata : dict, allposition : list, newposition : tuple)
Gamedata.randomposition(gamedata : dict)

Gamedata.**create**() -> gamedata : dict

Description: Créer une entité gamedata

Paramètres: aucun

Valeur de retour: dictionnaire avec la carte et des entités

Gamedata.**addPlante**(gamedata : dict, position : tuple) -> dict

Description: ajoute une plante dans l'emplacement voulu

Paramètres:

gamedata : dictionnaire

position : tuple

Valeur de retour: gamedata modifié

Gamedata.**addCarnivore**(gamedata : dict, position : tuple) -> dict

Description: ajoute un carnivore dans l'emplacement voulu

Paramètres:

gamedata : dictionnaire

position : tuple

Valeur de retour: gamedata modifié

Gamedata.**addHerbivore**(gamedata : dict, position : tuple) -> dict

Description: ajoute un herbivore dans l'emplacement voulu

Paramètres:

gamedata : dictionnaire

position : tuple

Valeur de retour: gamedata modifié

Gamedata.**randomposition**(gamedata : dict) -> tuple

Description: renvoie une position aléatoire et valide sur la map

Paramètres:

gamedata : gamedata

Valeur de retour: tuple

Gamedata.**get_allposition**(gamedata : dict) -> list:

Description: renvoie une liste de toutes les positions de toutes les entités sur la carte

Paramètres:

gamedata : gamedata

Valeur de retour: liste de tuple

Gamedat.**get_allposition_nearby**(gamedata : dict, position : tuple) -> list

Description: renvoie une liste de toutes les positions autour de l'entité

Paramètres:

gamedata : gamedata

Valeur de retour: liste de tuple

Gamedata.**get_herbivore**(gamedata : dict)->list
Description: renvoie une liste de tous les herbivores
Paramètres:

gamedata : gamedata

Valeur de retour: liste de dictionnaire

Gamedata.**getcarnivore**(gamedata:dict) ->list

Description: renvoie une liste de tous les carnivores
Paramètres:

gamedata : gamedata

Valeur de retour: liste de dictionnaire

Gamedata.**getcarnivore**(gamedata:dict) ->list

Description: renvoie une liste de toutes les plantes
Paramètres:

gamedata : gamedata

Valeur de retour: liste de dictionnaire

Gamedata.**get_herbivore_position**(gamedata : dict)->list
Description: renvoie une liste de toutes les positions des Herbivores
Paramètres:

gamedata : gamedata

Valeur de retour: liste de tuple

Gamedata.**get_carnivore_position**(gamedata : dict)->list
Description: renvoie une liste de toutes les positions des carnivore
Paramètres:

gamedata : gamedata

Valeur de retour: liste de tuple

Gamedata.**get_plante_position**(gamedata : dict)->list

Description: renvoie une liste de toutes les positions des plantes

Paramètres:

gamedata : gamedata

Valeur de retour: liste de tuple

Gamedata.**kill_plante**(gamedata : dict, elementtokill : int)->Gamedata

Description: permet de retirer l'entité du dictionnaire

Paramètres:

gamedata : gamedata

elementtokill : int

Valeur de retour: gamedata sans l'élément mort

Gamedata.**kill_herbivore**(gamedata : dict, elementtokill : int)->Gamedata

Description: permet de retirer l'entité du dictionnaire

Paramètres:

gamedata : gamedata

elementtokill : int

Valeur de retour: gamedata sans l'élément mort

Gamedata.**kill_carnivore**(gamedata : dict, elementtokill : int)->Gamedata

Description: permet de retirer l'entité du dictionnaire

Paramètres:

gamedata : gamedata

elementtokill : int

Valeur de retour: gamedata sans l'élément mort

Gamedata.**distance**(position1: tuple, position2 : tuple):

Description: donne la distance entre deux éléments

position1 : tuple

position2 : tuple

Valeur de retour: renvoie un entier

Gamedata.**count_nearby_entities**(gamedata : dict, creature : dict, thingtocount: list) -> int

Description: Donne le nombre d'entités présentent autour d'une entité

gamedata : dictionnaire

creature : dict

thingtocount : list

Valeur de retour: int

4.7 Joueur.py

Joueur.create(position : tuple)
Joueur.get_position(creature : dict)
Joueur.get_manger(creature : dict)
Joueur.set_position(creature : dict, newposition : tuple)
Joueur.set_manger(creature : dict, etat : bool)
Joueur.move(creature : dict, gamedata : dict, direction : str, allposition : list)
Joueur.caneat(creature : dict, gamedata : dict)
Joueur.eat(creature : dict, gamedata : dict)
Joueur.show(creature : dict)

5 Calendrier et suivi de développement.

5.1 P1 :

5.1.1 Fonctions à développer.

Fonctions	codées	testées	commentaires
Main.main()			
Main.init()			
Main.run()			
Main.show(gamedata)			
Main.interact(gamedata, dt)			
Map.isinmap(newposition, carte)			
Map.create(sizex, sizey)			
Map.get_carte(carte)			
Map.get_size(carte)			
Map.show(carte)			
Gamedata.create()			
Gamedata.addplante(gamedata : dict, position : tuple)			
Gamedata.addCarnivore(gamedata : dict, position : tuple)			
Gamedata.addHerbivore(gamedata : dict, position : tuple)			
Gamedata.get_allposition(gamedata : dict)			

Gamedata.get_allposition_nearby(gamedata : dict, position : tuple)			
Gamedata.get_herbivore(gamedata : dict)			
Gamedata.get_carnivore(gamedata : dict)			
Gamedata.get_plante(gamedata : dict)			
Gamedata.get_herbivore_position(gamedata : dict)			
Gamedata.get_carnivore_position(gamedata : dict)			
Gamedata.get_plante_position(gamedata : dict) Gamedata.kill_plante(gamedata : dict, plantetokill : int)			
Gamedata.kill_carnivore(gamedata : dict, carnivoretokill : int)			
Gamedata.kill_herbivore(gamedata : dict, herbivoretokill : int)			
Gamedata.distance(position1, position2)			
Gamedata.count_nearby_entities(gamedata : dict, creature : dict, thingtocount)			
Gamedata.isinmap(newposition : tuple,carte : list)			
Gamedata.valid_move(gamedata : dict, allposition : list, newposition : tuple)			
Gamedata.randomposition(gamedata : dict)			

5.2 P2 :

5.2.1 Fonctions à développer.

Fonctions	codées	testées	commentaires
Herbivore.create(position : tuple)			
Herbivore.get_position(creature : dict)			
Herbivore.get_manger(creature : dict)			
Herbivore.set_position(creature : dict, newposition : tuple)			
Herbivore.set_manger(creature : dict, etat : bool)			
Herbivore.move(creature : dict, gamedata : dict, direction : str, allposition : list)			
Herbivore.can_reproduce(creature : dict, gamedata : dict, allposition : list)			
Herbivore.reproduce(creature : dict, gamedata : dict, allposition : list)			
Herbivore.caneat(creature : dict, gamedata : dict)			

Herbivore.eat(creature : dict, gamedata : dict)			
Herbivore.gotofood(creature : dict, gamedata : dict, allposition : tuple)			
Herbivore.show(creature : dict)			
Plante.create(position)			
Plante.get_skin(plante)			
Plante.get_position(plante)			

5.3 P3 :

5.3.1 Fonctions à développer.

Fonctions	codées	testées	commentaires
Carnivore.create(position : tuple)			
Carnivore.get_position(creature : dict)			
Carnivore.get_manger(creature : dict)			
Carnivore.set_position(creature : dict, newposition : tuple)			
Carnivore.set_manger(creature : dict, etat : bool)			
Carnivore.move(creature : dict, gamedata :			

dict, direction : str, allposition : list)			
Carnivore.can_reproduce(creature : dict, gamedata : dict, allposition : list)			
Carnivore.reproduce(creature : dict, gamedata : dict, allposition : list)			
Carnivore.caneat(creature : dict, gamedata : dict)			
Carnivore.eat(creature : dict, gamedata : dict)			
Carnivore.gotofood(creature : dict, gamedata : dict, allposition : tuple)			
Carnivore.show(creature : dict)			

5.4 P4

5.4.1 Fonctions à développer.

Fonctions	codées	testées	commentaires
Joueur.create(position : tuple)			
Joueur.show(creature : dict)			
Joueur.eat(creature :			

dict, gamedata : dict)			
Joueur.caneat(creature : dict, gamedata : dict)			
Joueur.move(creature : dict, gamedata : dict, direction : str, allposition : list)			
Joueur.set_manger(creature : dict, etat : bool)			
Joueur.set_position(creature : dict, newposition : tuple)			
Joueur.get_manger(creature : dict)			
Joueur.get_position(creature : dict)			