

Rapport du projet Jeu de la vie :



PELTIER Emile
OUASSOU Moussa
PHAN Tran Thien An
PHAM Xuan Phuc
OUBANI Mariame

Sommaire:

1. Introduction
2. Choix de modélisation et d'implémentation
 - a. Choix de l'architecture
 - b. Fonctionnalités
 - c. Manuel de jeu
3. Difficultés rencontrées
4. Répartition du travail

1. Introduction:

Le Jeu de la vie est un automate cellulaire imaginé par John Horton Conway en 1970, un projet de simulation révolutionnaire conçu et réalisé par notre groupe de développeurs passionnés, cherche à capturer les subtilités de l'évolution biologique à travers le prisme de la programmation Python. Inspiré par des concepts biologiques fondamentaux tels que la reproduction, la mutation génétique et la sélection naturelle, notre jeu offre une expérience immersive où des entités autonomes, appelées affectueusement "Bobs", évoluent dans un monde virtuel.

À travers ce rapport, nous explorerons chaque aspect de notre création, de la représentation graphique de l'écosystème aux mécanismes complexes de reproduction, de perception et de mémoire des Bobs. Chaque ligne de code que nous avons écrite et chaque décision de conception que nous avons prise sont un témoignage de notre dévouement à créer un environnement virtuel évolutif.

Notre projet vise à offrir une expérience d'apprentissage pratique, où la programmation devient un moyen d'explorer les mystères de l'évolution. Nous espérons que notre jeu deviendra un outil éducatif stimulant, illustrant de manière ludique les concepts évolutifs pour les étudiants et les amateurs curieux.

2. Choix de modélisation et d'implémentation:

a. Architecture

Le projet est structuré de manière modulaire, avec des fichiers distincts pour différentes parties du jeu. Voici une brève analyse de l'architecture :

main.py: ce fichier s'occupe d'initialiser le module de pygame, l'écran, l'horloge et créer les setting et le jeu. Avant la boucle du jeu, il initialise le menu (qui est dans le fichier "GameControl/EventManager.py") et commence le jeu.

GameControl: Ce répertoire s'occupe de contrôler toutes les activités, les événements ainsi que les interactions entre les objets du jeu. Il est organisé en 5 fichiers python:

1. game.py: Ce fichier s'occupe d'initialiser les instances du jeu (gameController), les instances du graphique (world), ainsi que les paramètres. Il contient également deux méthodes, saveGame et loadGame, qui permettent de connecter et d'écrire les paramètres définis à ces instances du jeu et des paramètres, et vice versa. Ainsi, le jeu peut facilement sauvegarder et charger ses paramètres. La boucle du jeu (méthode run()) qui est incluse dans ce fichier a pour objectif de gérer l'horloge, les événements et le rendu sur l'écran. Elle comprend 4 méthodes importantes:

- **event():** elle s'occupe de tous les événements du jeu (c'est-à-dire: le jeu va interagir différemment quand on tape les boutons sur le clavier). Il contient les options qui permettent à l'utilisateur d'interagir avec le jeu, En utilisant le clavier:
 - Press “Escape”: quitter le jeu immédiatement
 - Press “Backspace”: ouvrir “In-game settings”
 - Press “Space”: Arrêt le jeu
 - Press “i” (i est le clé de numéro du clavier, i de 1 à 5): Sauvegarder le jeu dans option i
 - Press “S”: Simulation mode
- **update():** elle met à jour le mouvement du caméra
- **draw() et drawSimu():** elle appelle la classe World, qui fait le rendu du jeu
- **gameController.IncreaseTick() et gameController.updateRenderTick():** Ils mettent à jour les activités des objets du jeu

2. EventManager.py: Ce fichier gère les paramètres (input, chargement, etc.) à travers une interface. Il inclut une fonction permettant d'interagir avec le menu, les paramètres, ainsi que les options de chargement du jeu. Pour faciliter cette interaction, l'EventManager prend en charge les comportements des boutons. Les entrées du menu des paramètres sont bien gérées afin d'obtenir les valeurs appropriées pour chaque paramètre. De plus, ce fichier

permet l'arrêt du jeu, notamment pour la consultation des paramètres des Bobs.

3. **saveAndLoad.py:** Il propose des méthodes permettant de sauvegarder l'état actuel du jeu dans un fichier situé dans le répertoire 'save'. Les méthodes de chargement utilisent ce fichier pour restaurer les instances du jeu, les paramètres et réinitialiser les Bobs ainsi que la nourriture.
4. **setting.py:** Il comprend les attributs qui représentent les paramètres du jeu. En utilisant le modèle de conception "Singleton pattern", tous ces paramètres restent cohérents et synchronisés lorsqu'ils sont modifiés par d'autres méthodes.
5. **gameController.py:** Ce fichier représente le cœur du jeu, contenant les structures des objets tels que les Bobs, les tiles, et l'ensemble de tiles qui contiennent de la nourriture. Les attributs importants comprennent :
 - grid : une liste de listes de tiles qui représente tous les cas dans un carré 2D.
 - listFoods : un ensemble de tiles contenant de la nourriture.
 - listBobs : la liste des Bobs actuellement présents dans le jeu.
 - newBornQueue : une liste des Bobs qui viennent de naître au tick du jeu $t=i$; ils apparaîtront au tick suivant $t = i + 1$.
 - diedQueue : une liste des Bobs qui viennent de mourir au tick du jeu $t = i$; ils disparaîtront au tick suivant $t = i + 1$.

En ce qui concerne le tick, nous faisons une distinction entre le renderTick, qui est utilisé pour le rendu dans Pygame, et le currentTick, qui représente le tick du jeu. Pour la gestion des objets à chaque tick, les interactions se déroulent de la manière suivante :

Au début de chaque tick, la fonction increaseTick() ajoute tous les Bobs qui viennent de naître dans le newBornQueue à la liste listBobs, et elle supprime tous les Bobs qui sont dans le diedQueue. Ensuite, elle trie les Bobs dans l'ordre décroissant de vitesse. En parcourant tous les Bobs triés, elle permet à ces Bobs d'interagir entre eux. La classe utilise également le "Singleton pattern" pour assurer la synchronisation du jeu.

Tiles:

Les Bobs peuvent évidemment déplacer sur les cases. C'est le but de ce répertoire. La classe "Tiles" contient tous les objets des cases tandis que la classe "Bob" gère les Bobs.

1. tile.py:

Cette classe gère les cases du jeu, comprenant les attributs des coordonnées et la quantité d'énergie dans chaque case. Le concept de la nourriture est simplement assimilé à un niveau d'énergie. Par conséquent, il n'est pas nécessaire de créer une classe distincte pour la nourriture. À la place, l'énergie d'une case représente la nourriture, laquelle est régénérée chaque nouveau jour.

La classe Tile remplit une autre fonction essentielle : elle fournit des méthodes permettant de convertir les coordonnées des cases en positions à l'écran, ce qui est extrêmement utile pour le rendu graphique.

Enfin, lorsqu'un Bob doit trouver le chemin pour se nourrir ou s'échapper, la classe Tile propose des méthodes pour obtenir les cases adjacentes ainsi que la distance entre les cases.

2. bob.py:

La classe Bob se charge de gérer toutes les activités, de la création à la disparition, ainsi que des interactions entre un Bob et les autres objets. Elle comporte quatre méthodes principales. La première, "action", détermine les activités que le Bob effectue en un tick et leur ordre d'exécution. Ensuite, la méthode "interact" gère la manière dont Bob interagit avec d'autres éléments dans la même case. La troisième méthode, "scan", utilise la perception de Bob pour recueillir des informations. Par exemple, les prédateurs peuvent l'utiliser pour repérer une proie ou de la nourriture en utilisant leur vision. La quatrième méthode, "determineNextTile", aide le Bob à prendre des décisions sur la prochaine case vers laquelle il se déplace, en suivant les priorités suivantes : fuir les prédateurs, se diriger vers les sources de nourriture, chasser les proies et se déplacer au hasard tout en évitant les cases déjà visitées. Pour ce qui est de l'utilisation des points de mémoire par Bob, deux méthodes sont mises en place. La première se trouve dans la

méthode "action" pour aider Bob à se souvenir des cases qu'il a déjà visitées, et la seconde est dans la méthode "scan" pour aider Bob à se souvenir des sources de nourriture qu'il a déjà repérées. Si vous avez des questions sur l'implémentation spécifique de ces points de mémoire, n'hésitez pas à les poser.

View: Ce répertoire s'occupe de tous les affichages du jeu, incluant la carte, les bobs et la nourriture.

1. **world.py**

En utilisant les coordonnées des cases, ce module s'occupe de générer les images des cases, les bobs et la nourriture et les met sur écran. L'attribut PreviousTiles de chaque bob permet de déplacer doucement les bobs sur leur chemin.

En plus, en fonction du mode (normal ou simulation), le déroulement de l'affichage est différent

2. **camera.py**

Cette classe permet aux utilisateurs de contrôler les positions des images en appuyant sur les boutons UP, LEFT, DOWN, RIGHT

3. **texture.py**

Cette classe fournit les méthodes pour appeler les images sur le répertoire "asset/graphic". On va utiliser les images pour le rendu graphique dans la classe world.

b. Fonctionnalités

Menu Principal : Un menu principal affiche plusieurs fonctionnalités tels que:

- Lancement d'une nouvelle partie avec le bouton "NEW GAME"
- Chargement des cinq parties déjà sauvegardées avec le bouton "LOAD GAME"
- Accès aux différents paramètres du jeu ainsi qu'à d'autres fonctionnalités telles que STOP/PLAY MUSIC et INCREASE/DECREASE BRIGHTNESS via le bouton "SETTINGS"



- Quitter le jeu avec le bouton "QUIT"

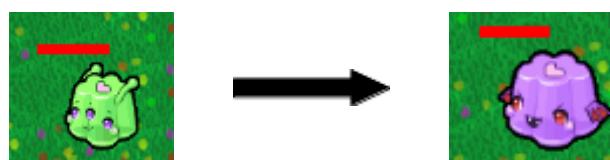
Le jeu:

Le jeu se déroule de manière claire, ce qui permet d'observer facilement les mouvements et les interactions entre les objets

- **La carte:** On a choisi le concept de bloc d'herbe (pareil au Minecraft) pour afficher clairement chaque case. Il est un peu ennuyant d'afficher juste une bloc d'herbe donc on a généré la carte en alternant les bloc d'herbe et de fleurs de manière aléatoire.
- **Personnages (Bob) :** Les entités du jeu, appelées "Bobs", sont rendues avec des textures. Les mouvements et les actions des Bobs sont gérés selon les exigences du projet, y compris les transitions, l'explosion (mourir), l'apparition (naître), la consommation d'énergie, la reproduction ainsi que la variation des caractéristiques.

Sur l'aspect graphique, un bob est sur 2 forme:

- Forme normale: Un petit bob en vert.
- Prédateur : les Bobs peuvent se transformer en prédateurs et chasser les Bobs présents dans leur champ de vision. Dans ce cas, le Bob change sa couleur du vert au violet

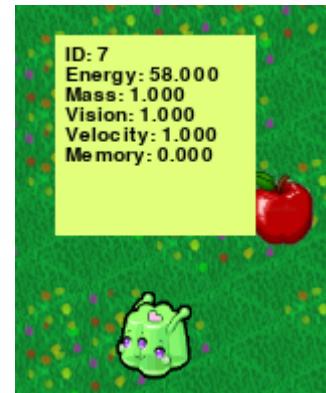


Mouvement: Le mouvement permet aux joueurs d'observer toutes les activités du bob.

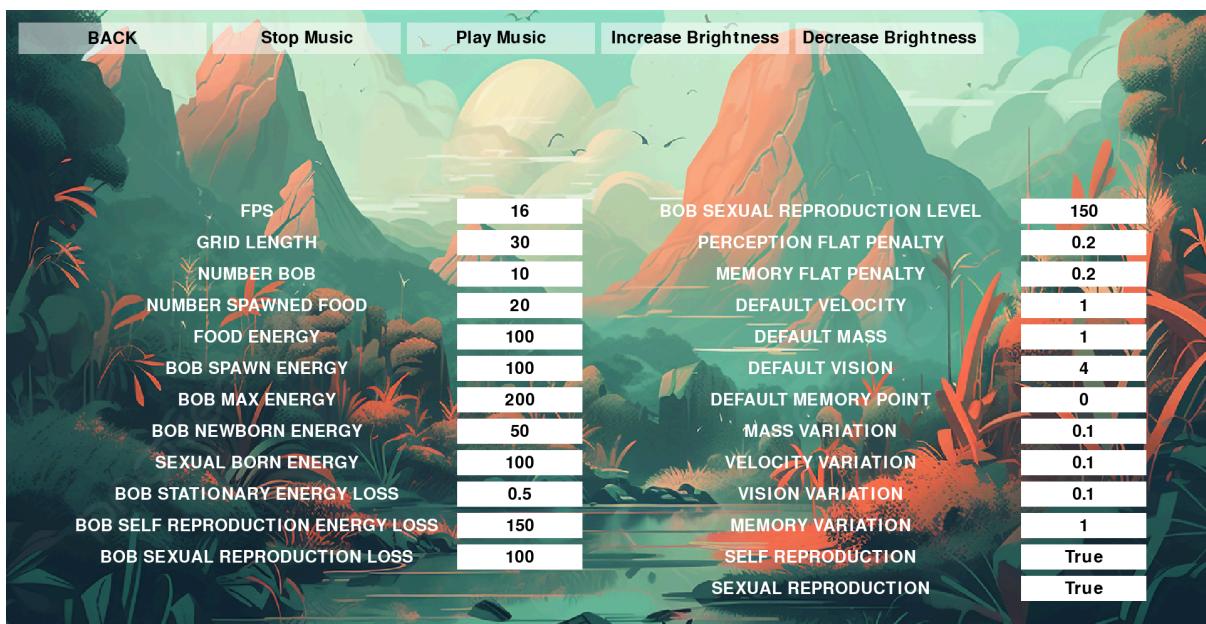
Caméra Dynamique : La caméra suit les directions que le joueur choisit avec son clavier

Le mode simulation: Ce mode permet d'accélérer les mouvement et l'interaction des bobs à chaque Rendertick, qui est utile pour l'observation de l'évolution des bobs au fil du temps. Ce mode permet aussi d'observer tous les objets dans la carte.

Pause: On propose une fonctionnalité pour arrêter tous les mouvements des bobs. Dans ce mode seulement, on peut consulter les caractéristiques des bobs par un/des petit(s) tableau qui apparaît à côté si on place la souris sur un bob. Pour cela, on n'a pas besoin d'exprimer graphiquement la grandeur ou la couleur du bob pour montrer les caractéristiques.



Setting et Ingame-setting: L'utilisateur peut utiliser la fenêtre de setting pour ajuster les paramètres du jeu qui inclut les paramètres comme-ci:



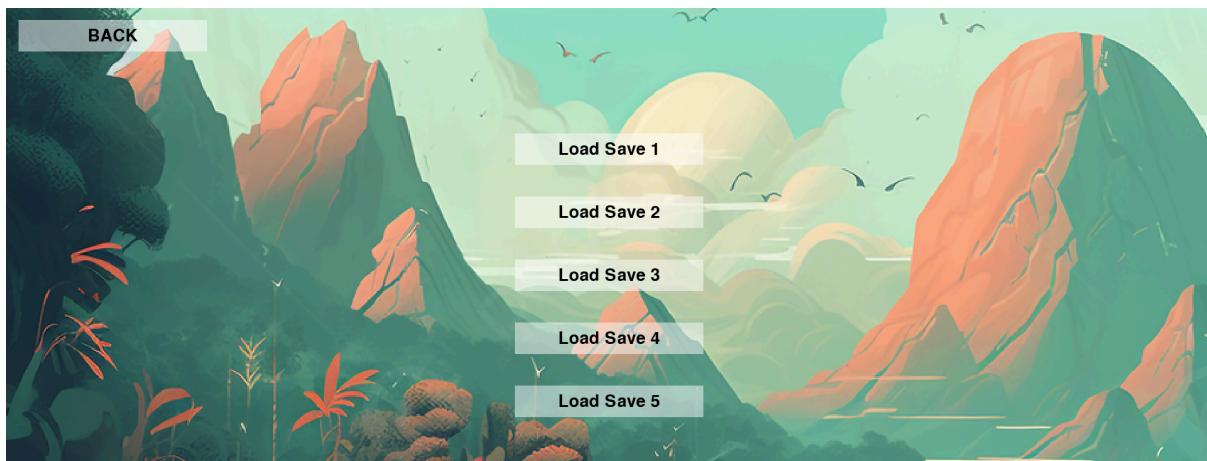
Setting



In Game Setting.

Save and Load:

À un point du jeu. On peut sauvegarder l'état courant du jeu à un des 5 emplacements de sauvegarde et le relancer quand on sélectionne dans le menu load l'emplacement de sauvegarde choisi.



Save and Load Options

Graphes:

Pour permettre d'observer correctement la simulation, on a besoin des résultats. Pour cela, on peut arrêter le jeu et voir l'évolution des nombre de bobs, nombre de bobs qui naît, nombre de bobs qui sont morts, la moyenne des masses, des énergies, de la vision, de la vitesse et le point de mémoire. Tous sont exprimé par les différents graphes.

c. Manuel du jeu:

Menu: Quand le programme commence, on va voir le menu qui affiche 4 boutons avec des fonctions différentes.



- “NEW GAME”: Commencer le jeu avec les paramètres défaut
- “LOAD GAME”: Ouvrir les options du chargement du jeu
- “SETTINGS”: Ouvrir les paramètres
- “QUIT”: Quitter le jeu

Settings:



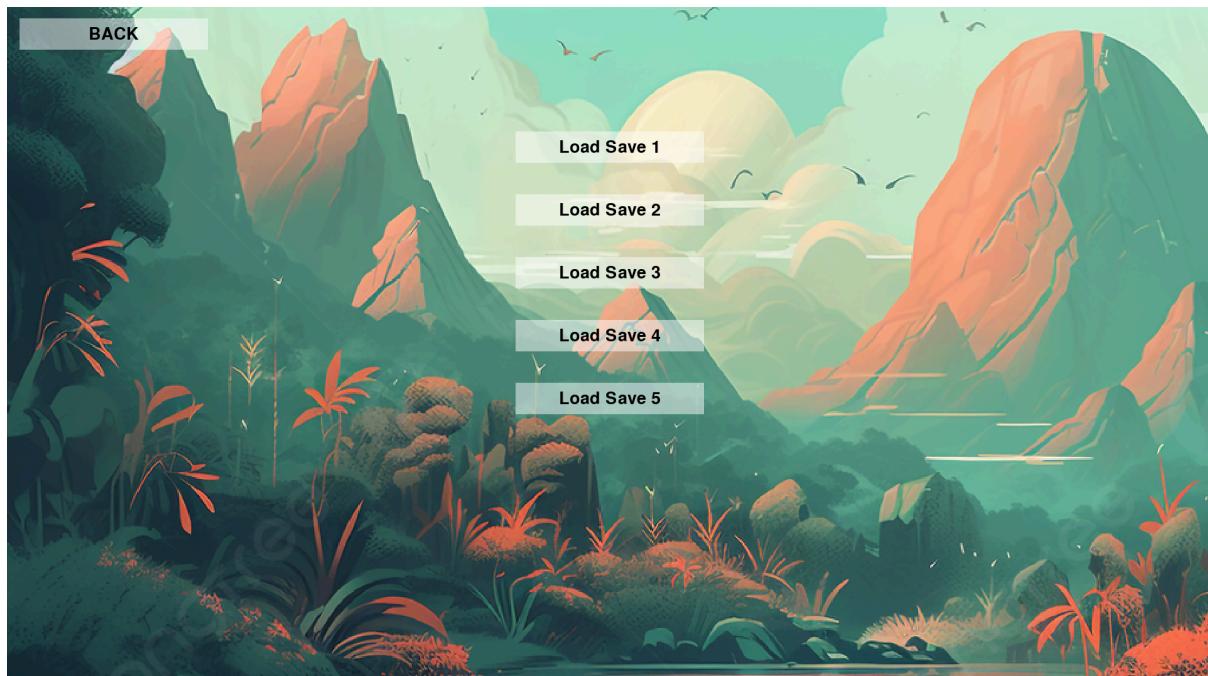
- “BACK”: Retourne au menu

- “Stop music, Play music”: L’ajustement du musique
- “Increase Brightness, decrease brightness”: L’ajustement de la luminosité
- Pour modifier chaque paramètre, on appuie sur le bouton correspondant au paramètres désiré. On tape la valeur appropriée et appuyez “Enter” pour enregistrer. *Chaque valeur a des limites sur les types:*
 - FPS: Un entier de 1 à 4 qui est la vitesse du jeu. En fait, le niveau de 4 exige moins de travaux de rendu graphique, ce qui est approprié pour la carte très large (200 x 200) par exemple
 - Grid Length: Un entier de 1 à 200: Le largeur n du grid nxn
 - Number Bob: Un entier de 0 à 200: Le nombre de bobs initial.
 - Number Spawed Food (Un entier de 0 à 1000): Nombre d’aliments qui apparaissent chaque jour.
 - Food Énergie (Un entier de 1 à 2000): L’énergie défaut d’un aliment
 - Bob Spawn Énergie (Un entier de 1 à 1000): Énergie initiale d’un bob.
 - Bob Max Énergie (Un entier de 1 à 1000): Énergie maximale d’un bob
 - Bob NewBorn Énergie (Un entier de 1 à 1000): L’énergie initiale du bob qui naît par parthenogenesis.
 - Sexual Born Énergie (Un entier de 1 à 1000): L’énergie du bob qui naît par reproduction sexuelle
 - Bob stationary energy loss (Un réel entre 0 et 10): L’énergie du bob quand il reste à un position.
 - Bob self production energy loss (Un entier de 0 et 1000): perte de l’énergie quand un bob reproduira un autre bob.
 - Bob sexual reproduction loss (Un entier de 0 et 1000): Perte de l’énergie quand 2 bob faire amour et reproduire un autre bob.
 - Bob sexual reproduction level (Un entier de 1 à 1000): Niveau d’énergie que 2 bobs dans le même cas peut faire amour
 - Perception flat penalty (Un réel entre 0 et 1): Perte d’énergie à cause de la vision
 - Memory flat penalty (Un réel entre 0 et 1): Perte d’énergie à cause du mémoire
 - Default mass, default velocity, default vision, default memory point: (réel entre 0 et 10): Les caractéristiques défauts des bobs.

- Mass. velocity, vision, memory variation (réel entre 0 et 10): La variation des caractéristiques des bobs.
- Self Production et Sexual production (Tapez 0 pour non, 1 pour oui)

Load Game:

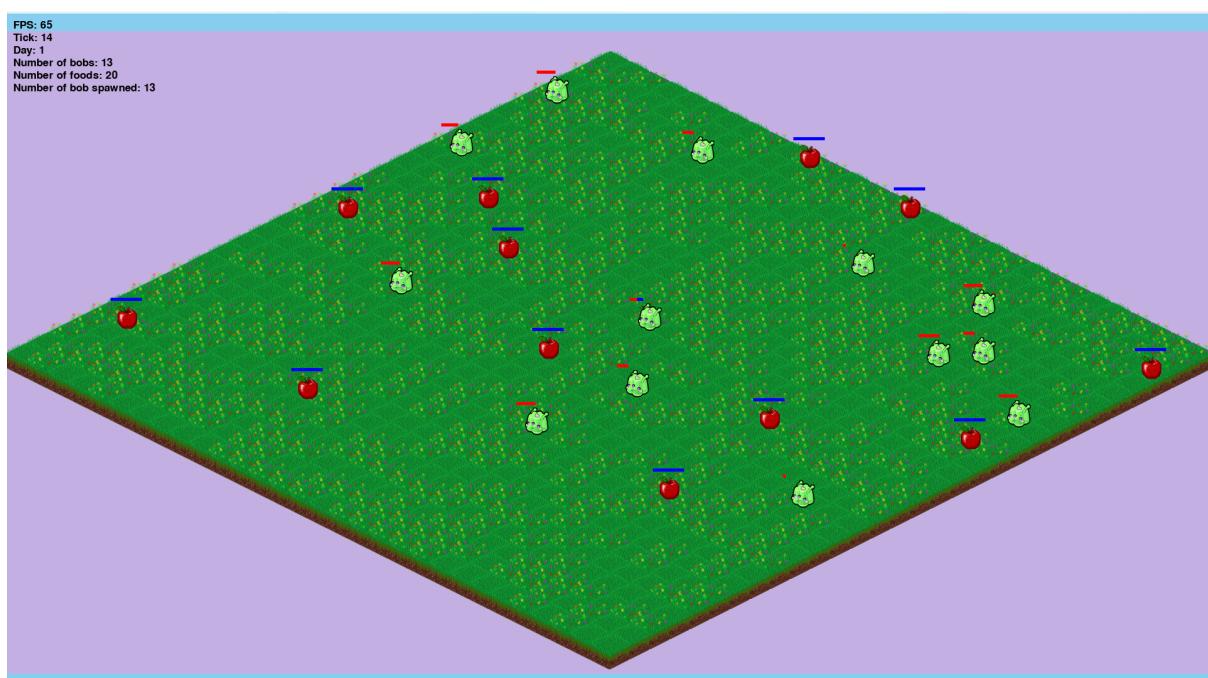
Après avoir appuyé sur le bouton de “Load Game”, on peut voir 5 options:



Cliquer sur l'option souhaité pour recharger et lancer le jeu immédiatement

Jeu:

Quand on commence le jeu, on va voir la carte, le bobs ainsi que quelques chiffres.

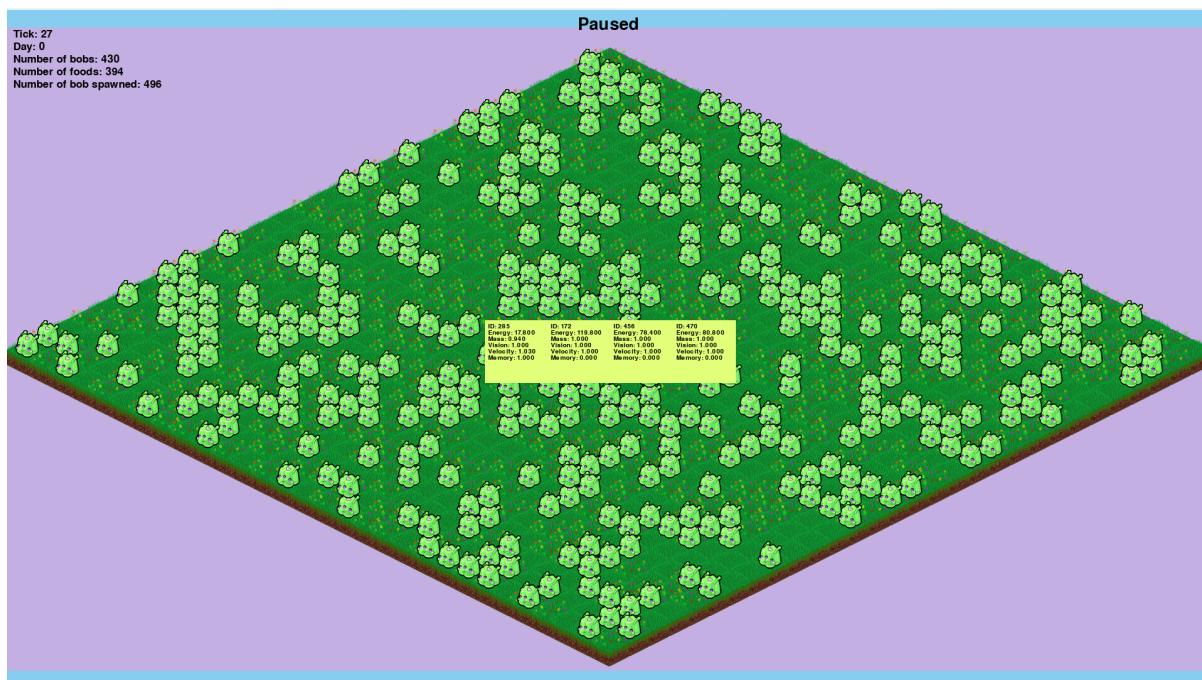


Voici les fonctions des quelques boutons:

- “S”: Basculer on/off le mode simulation
- “M”: Retourne au menu (sans sauvegarde)
- “Space”: Mettre en pause du Jeu
- “Backspace”: Ouvrir In Game Settings.
- “1”: Sauvegarder l'état courant à l'option 1
- “2”: Sauvegarder l'état courant à l'option 2
- “3”: Sauvegarder l'état courant à l'option 3
- “4”: Sauvegarder l'état courant à l'option 4
- “5”: Sauvegarder l'état courant à l'option 5
- “G” Consulter le graph

Pause:

Pause peut être utilisé dans le mode normal ou mode simulation. Dans le mode simulation, car les bobs se déplacent trop vite, on a décidé de supprimer la fonction pour consulter les caractéristiques des bobs dans ce mode. Pour cela, on propose de consulter les caractéristiques en mettant en pause le jeu. Pour consulter, on n'a juste besoin de mettre la souris dans la position des bobs. Si 2 bobs est dans le même cas, 2 tableaux vont apparaître.



Pause pour une population peuplé

Ingame Setting:

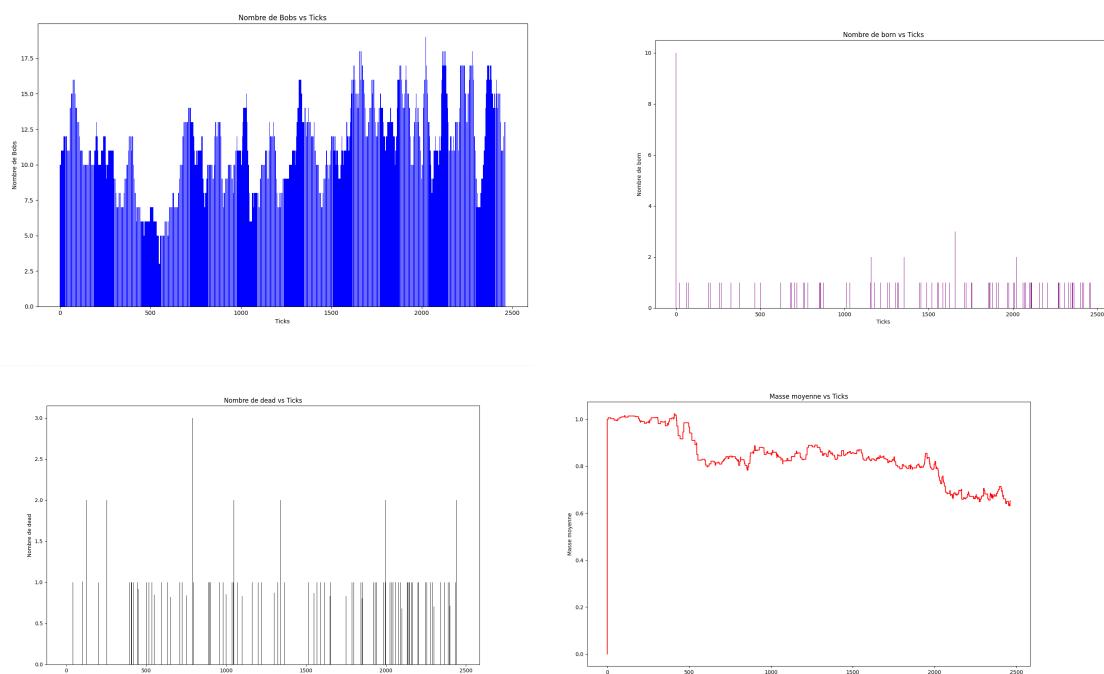
On peut également modifier les paramètres lors du déroulement du jeu

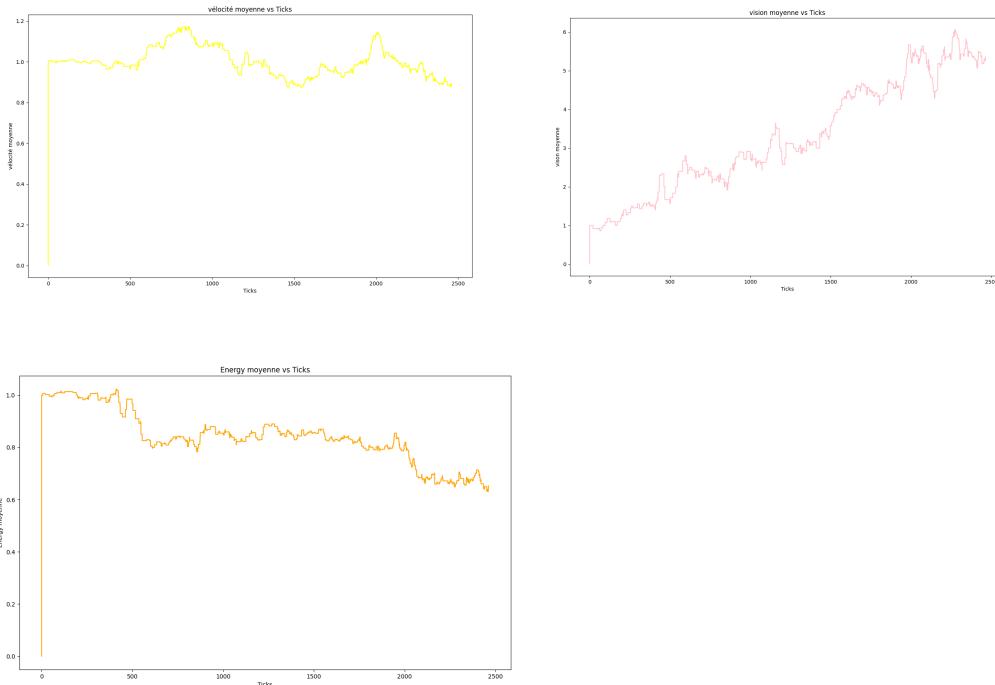


Ingame Setting

Graphes:

Au cours d'une simulation, on peut arrêter le jeu et afficher les graphes de "Nombre de bobs", "nombre de born bobs", "nombres de died bobs", les moyennes des caractéristiques des bobs au fil du temps:





Les graphes

3. Difficultés Rencontrées

- **Gestion de l'architecture :** On a dû revoir toute notre architecture au milieu du projet afin de pouvoir mieux gérer la partie logique du projet. Au début, l'architecture du jeu n'a pas été adaptée et non orientée objets. Nous avons dû refaire l'architecture en organisant les classes appropriées pour bien gérer les événements, l'interaction entre les objets du jeu.
- **Gestion d'interaction:** Pour que l'utilisateur puisse voir l'interaction des bobs comme la naissance, le mort et aussi les cas qui sont parcourus. Nous avons gérer l'animation du mouvement, de l'explosion,pour que l'utilisateur puisse observer et comprendre ce qu'il se passe dans le jeu.
- **Le mouvement:** Un bob peut parcourir plusieurs cas, pour cette raison, on créer un liste des cas qui sont parcourus intitulés previousTiles pour sauvegarder ces cas à chaque tick. Pour ça, on peut voir clairement et précisément les mouvements des bobs.
- **Le problème de performance/ Zoom In, Zoom out, ...:**
Le problème s'agit quand on initialise le jeu à une grande échelle. Si le nombre de cas est trop grand (200 par exemple), le FPS sont diminué dramatiquement (1 FPS). On propose des solution comme:

- Dessine les texture seulement dans la région couvert par l'écran
- Restructurer les surface comme un attribut du jeu.

Résultat: Le FPS reste stable de 18 FPS et le minimum garanti est 10 FPS pour une carte 200x200 avec 200 bobs

À cause du problème de performance, on ne peut pas ajouter la fonction zoom-in/out. Pourtant, en utilisant le mode Simulation, pause ou en déplaçant la caméra, l'utilisateur peut observer et accéder tous les bobs dans la carte

- **Intégration du Menu de Sauvegarde/Changement:** L'intégration des fonctionnalités de sauvegarde et de chargement dans le menu principal a nécessité une gestion complexe des données du jeu.
- **Consommation d'énergie:** Il y a quelques problèmes concernant la consommation d'énergie des Bobs. Nous ne savons pas si un Bob devrait consommer de l'énergie avant d'agir ou après avoir agi:
 - Avant d'agir : il existe la possibilité qu'un Bob ait une énergie ≤ 0 avant d'effectuer une quelconque action, en théorie, il mourrait dans ce cas. Cependant, nous ne pouvons pas gérer cela, et c'est également absurde car il aurait consommé de l'énergie et serait mort avant de pouvoir effectuer une action quelconque.
 - Après avoir agi : nous fixons toujours une limite maximale pour toutes les activités qui aident Bob à obtenir plus d'énergie. Ainsi, il ne peut pas dépasser cette limite, même s'il mange beaucoup. Comme nous considérons la reproduction asexuée comme un état énergétique, si l'énergie est consommée après l'action, Bob ne pourra pas effectuer de reproduction asexuée. Cependant, si nous supprimons cette limite, il y aura de nombreux problèmes liés à la consommation d'énergie.

Nous avons choisi la méthode de consommation d'énergie avant d'agir. Bob peut avoir des actions, une impulsion finale. S'il ne parvient pas à obtenir de la nourriture après ce tick, il mourra.
- Le problème concerne les **caractéristiques génétiques** de Bob générées par la reproduction sexuée. Dans cette section, vous nous avez dit de sauvegarder des caractéristiques telles que la perception et le point de mémoire en tant que nombres flottants pour assurer "l'évolution" et de les

arrondir au nombre entier le plus proche à chaque utilisation. Cependant, il y a un problème lié à la fonction de round en Python dans les cas où le nombre est sous forme x.5, après avoir fait quelques tests:

- Si x est un nombre impair (comme 1,5), il sera arrondi à 2.
- Si x est un nombre pair (comme 2,5), il sera arrondi à 2.

Nous pouvons faire une autre fonction “round” pour donner des valeurs comme en théorie, mais nous pensons qu'il vaut mieux de laisser tout à Python.

Texture: Nous nous sommes concentré sur la partie logique du jeu. Pourtant, il manque la ressource humaine pour trouver les images. Pour cette raison, le projet n'est pas assez esthétique pour satisfaire les yeux des joueurs.

4. Répartition du travail

PHAM Xuan Phuc (30%):

1. **Build Model:** Après de nombreux échecs et refontes, j'ai **construit un modèle orienté objet adapté** pour répondre aux exigences du jeu en organisant le jeu aux classes qui ont des fonctions varié: Le fichier *main.py*, les modules *Game Control*, *Tiles* et *view* avec les classes dedans.
2. **Gestion du jeu:** Je m'occupe de la plupart des fichiers dans le répertoire Game Control: Je m'occupe **gérer l'horloge, événement du jeu** (*tout game.py*), **gérer l'instance de l'état du jeu** (*tout gameController.py*), **Établir les paramètres** (*tout setting.py*)
3. **Save and Load:** Je m'occupe toute la partie **sauvegarder et recharger** en travaillant sur le fichier *saveAndLoad.py*. Je crée aussi **l'interface de LOAD GAME** pour choisir les options
4. **Tiles:** Je m'occupe de toute la classe de “Tile” qui **gère les cas du jeu, l'aliment et les bobs dedans**.
5. **Bobs:** Je m'occupe **d'initialiser les paramètres du bob et fournir des méthodes pour synchroniser l'état de chaque bobs à l'état du jeu** (dans Game Control et Tiles) **et au rendu graphique** (dans World).
6. **L'affichage:** Je m'occupe de **toutes les fonctions pour la rendu graphique du jeu, gestion de caméra et l'appels d'images**. Incluant l'affichage de la carte, les mouvements et l'interaction des Bobs

- 7. Menu et Setting:** Je m'occupe des contraintes des entrées d'inputs dans le setting et l'intégration de ces entrées à l'instance de paramètre. Ainsi que la fonctionnalité “In game setting” a été faite par moi.
- 8. Simulation mode et pause mode:** Je m'occupe des tous les comportement du jeu dans le mode simulation et le mode pause, incluant la consultation des informations des bobs.
- 9. Dessiner les images:** se trouve dans assets/graphic
- 10. Aider:**

Suite à la version incomplète de QUASSOU Moussa, j'aide à **débugger et compléter la partie de l'interface** (L'affichage de menu, les boutons, ...)

J'aide Émile PELTIER à **débugger et compléter les graphes**.

J'**observe et manipule tous les progrès du projet**.

PHAN Tran Thien An (25%):

- Être en charge de construire et développer la class Bob:
 - Basic level, Mass et Velocity: J'ai initialisé toutes les méthodes de base
 - Perception: J'ai fait les méthodes pour détecter les “predators”, “preys” et “foods” et la méthode principale scan().
 - Spacial Memory: J'ai ajouté 3 plus attributs, memorySpace qui est égal à 2*memoryPoint, un list visistedTiles, et un list foodTilesInMemo avec quelques méthodes concernant la mémoire.
 - Sexual Reproduction: J'ai implémenté cette partie en ajoutant des méthodes pour détecter les partenaires et mate()
 - Pour les mouvements, j'ai créé un fichier directions.py qui contient directionsDict et directionsList, cela permet de minimiser les méthodes moveForward() et runFrom().
- Générer des idées pour construire map, et class Tile: j'ai suggéré la structure de Map, d'intégrer food dans un tile, et des méthodes nécessaires de la classe tile pour la logique

PELTIER Emile (15%):

Au début du projet on a commencé par imaginer le jeu avec mes camarades puis à se familiariser avec l'outil pygame. Ensuite mon travail a été de commencer à implémenter les textures, et de créer le début de la classe Bob avec OUBANI Mariame. Qui a été restructuré et fini par PHAN Tran Thien An et PHAM Xuan Phuc. Ensuite dans l'élaboration de l'interface graphique du jeu réalisé avec OUASSOU Moussa, j'ai créé les cases de textes du menu Settings pour que l'on puisse changer les entiers des paramètres du jeu et j'ai créé le menu Load de l'interface de la même manière. J'ai conseillé l'utilisation de PICKLE à PHAM Xuan Phuc pour faire les fonctions save et load avec un exemple de code qui n'a pas été utilisé. J'ai été chargé de créer les graphiques qui permettent d'illustrer l'évolution des paramètres

du jeu, des bobs au cours d'une simulation. Tout d'abord, il a fallu que je me familiarise avec l'outil Matplotlib afin de pouvoir créer des graphes dont la représentation est adapté aux donnés que l'on souhaitait illustrer telles que le nombre de bob vivant, le nombre de bob mort, le nombre de naissance, la masse moyenne, la vitesse moyenne, et la vision moyenne d'un bob.

OUNANI Mariame (15%):

Dans le projet, j'ai été occupée à implémenter plusieurs fonctionnalités dans le jeu, telles que le contrôle lecture/pause de la musique et l'ajustement de la luminosité. J'ai également développé les différents graphiques des caractéristiques des Bobs, tels que ceux pour les Bobs nés à chaque tick, ceux pour les Bobs décédés et ceux pour les Bobs vivants. De plus, j'ai travaillé sur d'autres caractéristiques comme l'énergie moyenne, la masse moyenne, la vitesse moyenne et la vision moyenne. Le développement des graphiques a été réalisé en collaboration avec PELTIER Émile et OUASSOU Moussa. J'ai également commencé l'implémentation des textures et le développement de la classe bob et food. J'ai créé les cases de texte dans le menu Paramètres pour permettre la modification des paramètres du jeu, en collaboration avec Émile. J'ai également connecté l'interface utilisateur à la carte, de sorte que si l'on clique sur le bouton de démarrage du jeu, la carte s'affiche.

OUASSOU Moussa (15%):

Au sein de mon groupe de projet de création d'un jeu en langage de programmation Python, Après avoir constaté que mon collègue PHAN Tran Thien An avait déjà bien avancé sur l'implémentation de la grille du jeu, j'ai décidé de concentrer mes efforts sur un autre aspect crucial du projet. j'ai pris un rôle actif en réfléchissant et en travaillant de manière approfondie sur l'architecture du jeu avant même sa création. Ma contribution a débuté par la conception de l'interface graphique initiale, comprenant un menu principal avec les options "play", "settings" et "quit", une musique, ainsi qu'une image de fond ...etc. Mon collègue PHAM Xuan Phuc a ensuite repris cette interface, l'a modifiée et a ajouté de nouvelles fonctionnalités, renforçant ainsi l'expérience utilisateur.

Parallèlement, j'ai joué un rôle crucial en aidant le groupe à intégrer mon interface graphique au jeu. Malgré quelques difficultés rencontrées au cours de ce processus, nous avons collaboré avec PELTIER Emile et OUNANI Mariame de manière efficace pour surmonter ces obstacles. Cette expérience a souligné notre capacité collective à résoudre des problèmes techniques de manière collaborative.

PARACHE FRANÇOIS (0%)

Il nous a aidé à comprendre le sujet ainsi qu'à réfléchir sur les modèles. Il nous a aidé à installer python, pygame et github.