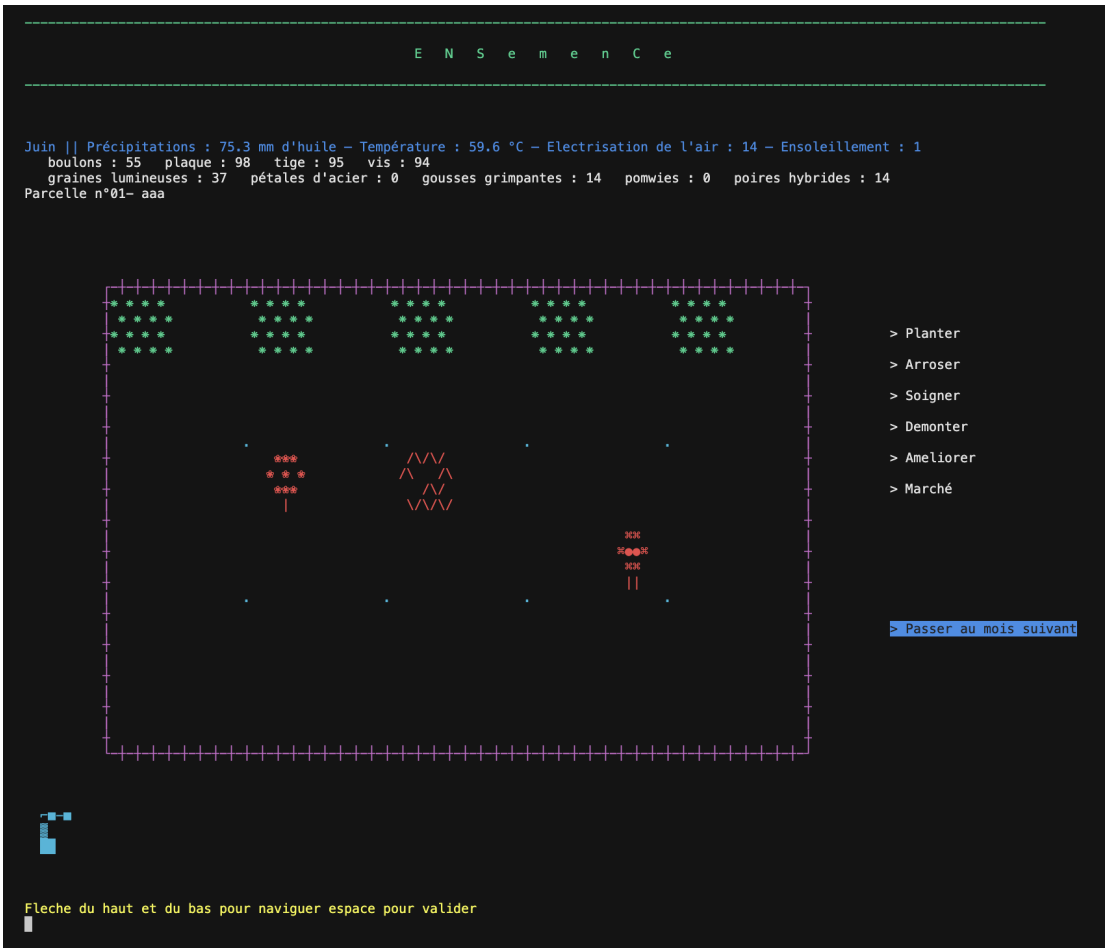
ENSC Bordeaux INP – Module CO6SFPAO

Projet : Programmation avancée

Sujet : ENSemenCe



Année universitaire 2024-2025

Promotion 2027 - Groupe 3

Quentin Lopez

Julia Zitouni--Flambard

Sommaire

Introduction : l'univers du jeu

Pour le projet ENSemenCe, nous avons voulu détourner l'univers potager du projet vers un monde dystopique entièrement mécanisé. C'est pourquoi toutes les plantes que nous avons rendues disponibles sont mécaniques, composées de boulons, vis, tiges et plaques de fer. Ainsi, on peut les construire ou les démonter. Nous avons tout de même conservé l'esprit principal du projet : ces plantes produisent différents fruits ou fleurs, selon leur état (qui lui-même dépend des conditions météorologiques). Le joueur peut vendre ces produits contre des ressources, qu'il utilise ensuite pour entretenir ses plantes et faire grandir son potager. Cependant, contrairement à des plantes normales, ces plantes mécaniques ont une obsolescence programmée… Elles vieillissent, leurs composants s'usent et elles deviennent plus fragiles avec le temps.

Pour pimenter la gestion de ce potager, différents éléments viennent perturber la vie de ces plantes : des virus les abîment et les rendent particulièrement fragiles, des créatures surgissent parfois pour détruire

les plantes, et au contraire des fées peuvent occasionnellement venir donner des composants.

Voulant rendre le jeu à la fois aussi réaliste et aussi modulable que possible, nous avons porté une attention particulière aux graphismes, à l'interface utilisateur, au confort de jeu… Il est également à noter que la majorité des paramètres de jeu ont été passés en constantes, donc potentiellement modifiables pour personnaliser l'expérience de jeu (bien que cette personnalisation n'ait pas été implémentée à ce stade).

1. Déroulement d'une partie

A - Déroulement général

Le jeu commence par l'affichage des règles qui est géré dans une fonction de la classe graphique. Puis la boucle de jeu se lance directement après avoir instancié le monde avec la fonction Simuler. Pour instancier le monde il suffit de proposer des dimensions en X et en Y pour une parcelle. Si l'on souhaite on peut également modifier le nombre de composants au départ avec un facteur multiplicateur (cela permet d'effectuer des tests). La simulation démarre avec des plantes sur une rangée avec un espacement de 1.

La fonction Simuler constitue la boucle principale du jeu, où chaque itération représente un mois de simulation. À chaque nouveau mois, le programme effectue des tirages aléatoires pour déclencher l'apparition de la fée ou activer le mode urgence, selon le nombre de mois écoulés. En effet nous avons ajouté un délai aussi bien pour les fées que pour les créatures pour éviter d'avoir les événements dès les premiers mois. Ensuite, une boucle interne permet d'interagir avec le joueur tant que le joueur n'a pas exécuter l'action de passer au mois suivant ou qu'une situation d'urgence est activée. Durant cette phase, le jeu est affiché puis on récupère les touches saisies et ceux en boucle.

B - Tour de jeu : normal

La navigation dans le jeu repose sur une hiérarchie de menus définie dans la classe Constantes sous la forme d'un Dictionary<string, Dictionary<int, string>>, où chaque menu contient des actions associées. Deux propriétés de la classe Monde, MenuSelectionnee et SectionSelectionnee, permettent de suivre le menu actuel et l'élément sélectionné. La méthode GererEntreeClavier gère le déplacement du curseur à l'aide des flèches haut et bas, en modifiant dynamiquement SectionSelectionnee, qui est affichée en bleu à l'écran.

Lorsque le joueur appuie sur "Entrée", la méthode ValiderEntre est appelée. Elle détermine le comportement à exécuter en fonction du menu ou de la section sélectionnée : soit une action est exécutée dans le monde, soit la navigation change de menu en accédant à la première entrée du sous-menu sélectionné. Cette navigation utilise la méthode PrendreSectionSuivantePrecedente. Un cas particulier est la section "Retour au menu principal", qui remplace le menu courant par la valeur de Origin, permettant ainsi de basculer entre le menu principal classique et sa version spécifique au mode urgence (dans lequel un sous-menu "Pièges" devient disponible). L'action "Passer au mois suivant" change l'état du booléen Passer, ce qui provoque l'avancement du tour dans la boucle principale du jeu. Les autres actions se divisent en deux catégories : celles qui s'exécutent immédiatement si elles sont valides (ex. achat, vente, amélioration), et celles qui nécessitent une sélection de case.

Dans ce dernier cas, la méthode SelectionCase est appelée. Elle positionne le curseur dans la grille, vérifie si l'action est possible (via ActionPossible, qui met à jour la propriété CaseSelectionneePossible), et empêche l'exécution de l'action si les conditions ne sont pas réunies. Ensuite, elle gère les entrées clavier : les flèches modifient les coordonnées via CoordCase, tandis qu'un appui sur "Entrée" exécute l'action si elle est possible. Par exemple :

- Planter est possible uniquement si la case est vide, respecte les espacements nécessaires, et si la plante ne gêne pas les autres.

- Démonter et Arroser nécessitent la présence d'une plante sur la case.
- Poser un piège demande la présence d'une créature et un niveau de piège suffisant.

Enfin, certaines actions peuvent être bloquées avant même d'atteindre la sélection de case, si les composants nécessaires sont insuffisants ou si la condition de niveau (comme pour les pièges) n'est pas remplie.

Parmis les autres actions possibles sans sélectionner une case qui sont appelé dans la méthode valide entrée nous pouvons acheter des composants avec des boulons, améliorer le matériel de la parcelle pour augmenter l'efficacité de certaines action, réduire leur coût ou proter les plante (l'amélioration de la clôture bloque les créature de niveau strictement inférieur), et arroser de manière automatique.

Ainsi dans le mode normal le joueur navigue dans les différents menus et exécute autant d'actions autant qu'il le souhaite, jusqu'à ce qu'il passe au mois suivant. Dans le cas présent cela enclenche la fonction passer tour enclenchant une action sur chaque parcelle (bien que le jeu dispose d'une seule parcelle actuellement, nous avions anticipé l'éventualité d'acheter une parcelle). Au sein de chaque parcelle, pour chaque plante cela enclenche la pousse qui actualise son état (ce qui se traduit visuellement par une couleur). Cette fonction s'occupe également de la récolte s'il s'agit d'un arbre fruitier ou d'une fleur, avec un bonus de production si elle est en bonne état, et de supprimer les plantes mortes.

C - Tour de jeu : mode urgence, fées et virus

Le mode urgence est activé à l'aide d'un booléen nommé Urgence, déclenché par la méthode TirageModeUrgence. Cette méthode effectue un tirage aléatoire pour déterminer l'apparition d'une créature. Si une créature apparaît, un niveau aléatoire compris entre 1 et 4 lui est attribué. Si ce niveau est supérieur ou égal à celui de la clôture, la créature n'apparaît pas et le mode urgence ne s'enclenche pas. Dans le cas contraire, la créature se manifeste sur

un bord aléatoire du plateau via sa méthode propre PositionnerSurBord, supprimant la plante présente sur sa case à l'aide de la méthode mange.

Simultanément, le menu en cours est remplacé par le menu spécifique au mode urgence, ainsi que le menu d'origine utilisé pour revenir en arrière. Dans ce nouveau menu, l'option "Passer au mois suivant" est remplacée par l'action "PasserAction". À chaque action validée par le joueur, la créature se déplace aléatoirement, détruisant les plantes sur son passage, et ce, autant de fois que le nombre de déplacements dont elle dispose.

Pour faire disparaître la créature, le joueur a deux options : poursuivre ses actions jusqu'à ce que la créature épuise son nombre d'actions, ou bien placer un piège sur sa case. Ce piège doit être d'un niveau au moins égal à celui de la créature et nécessitera des ressources. Cette action est accessible via un menu spécifique dans le menu général du mode urgence. Par ailleurs, la commande "PasserAction" permet de déclencher un tour de la créature sans que le joueur réalise une action. Comme mentionné précédemment, le mode urgence ne peut être activé durant les premiers mois du jeu. De plus, à mesure que le joueur améliore la clôture, les probabilités d'apparition d'une créature diminuent. Cependant, la clôture ne peut être renforcée que jusqu'au niveau 3, tandis que les créatures peuvent atteindre le niveau 4. Cela empêche donc de bloquer totalement leur apparition.

Concernant les fées, la mécanique est plus simple. Chaque mois, une fée a une probabilité d'arriver sur le plateau sur un bord de manière aléatoire et donne des composants de chaque type. L'affichage marque un temps d'arrêt et un message informe le joueur de l'action de la fée. Puis le tour reprend.

Enfin, à chaque tour les plantes ont une certaine probabilité d'attraper un virus. Si elles en attrapent un, son niveau est tiré au hasard entre 1 et 3. En fonction de ce niveau, elles auront besoin d'une certaine quantité de programmes antivirus pour s'en débarrasser. Pour faire, le joueur doit aller au marché échanger des fleurs contre des antivirus, puis les administrer à la plante via le menu Soigner. Tant qu'elles sont malades, les plantes sont très fragiles et leur état général en subit les conséquences.

D - Tests réalisés sur les différentes fonctionnalités

Pour s'assurer du bon fonctionnement du code, tout au long de la création du jeu, nous avons choisi de commencer par les fonctions d'affichage : ainsi, à chaque fois que nous ajoutons une fonctionnalité, nous pouvions facilement la tester. C'est pourquoi, logiquement, l'ajout des fonctions de navigation dans les différents menus a suivi. Nous allons voir ici quelques tests que nous avons pu réaliser au fonds mesure de la création du jeu.

Tout d'abord, sur les fonctions d'état de plantes, le test était assez simple à réaliser, puisque l'état de la plante se visualisait rapidement sur l'écran d'affichage, par un changement de couleur. Nous avons donc pu remarquer que l'état de la plante par était bien affecté par les conditions météorologiques, et en fonction des préférences de la plante. De même les fonctions Planter étaient assez rapides à tester : avec les fonctions de sélection d'action (que nous testions donc par la même occasion), nous vérifions que les conditions d'emplacement nécessaire étaient respectées, et que les ressources nécessaires étaient bien soustrait aux réserves du joueur. Tout cela était possible, grâce à un affichage interactif actualisé à chaque fois que le joueur effectue une action aussi simple que se déplacer dans le menu.

À l'ajout du mode urgence, nous avons pu le tester assez rapidement, du fait de deux défauts que nous avons corrigés par la suite : la probabilité d'un tour urgence était trop élevée pour pouvoir jouer correctement, et les tours en mode urgence survenaient dès l'ouverture du jeu. C'est suite à ça que nous avons notamment instauré un nombre minimum de mois passés dans le jeu avant qu'un tour urgence ne puisse survenir.

De façon générale, et comme nous l'avons mentionné ailleurs, le fait de mettre les variables de prix, de probabilité d'apparences, et autres, dans une classe Constantes dédiée nous permettait de faire des tests assez rapidement, en modifiant simplement la valeur de la variable. Pour les bonnes fées par exemple, il nous suffisait de mettre leur probabilité d'apparition à 1 pour les voir en action.

2. Modélisation objet réalisée

A - Diagramme de classes UML

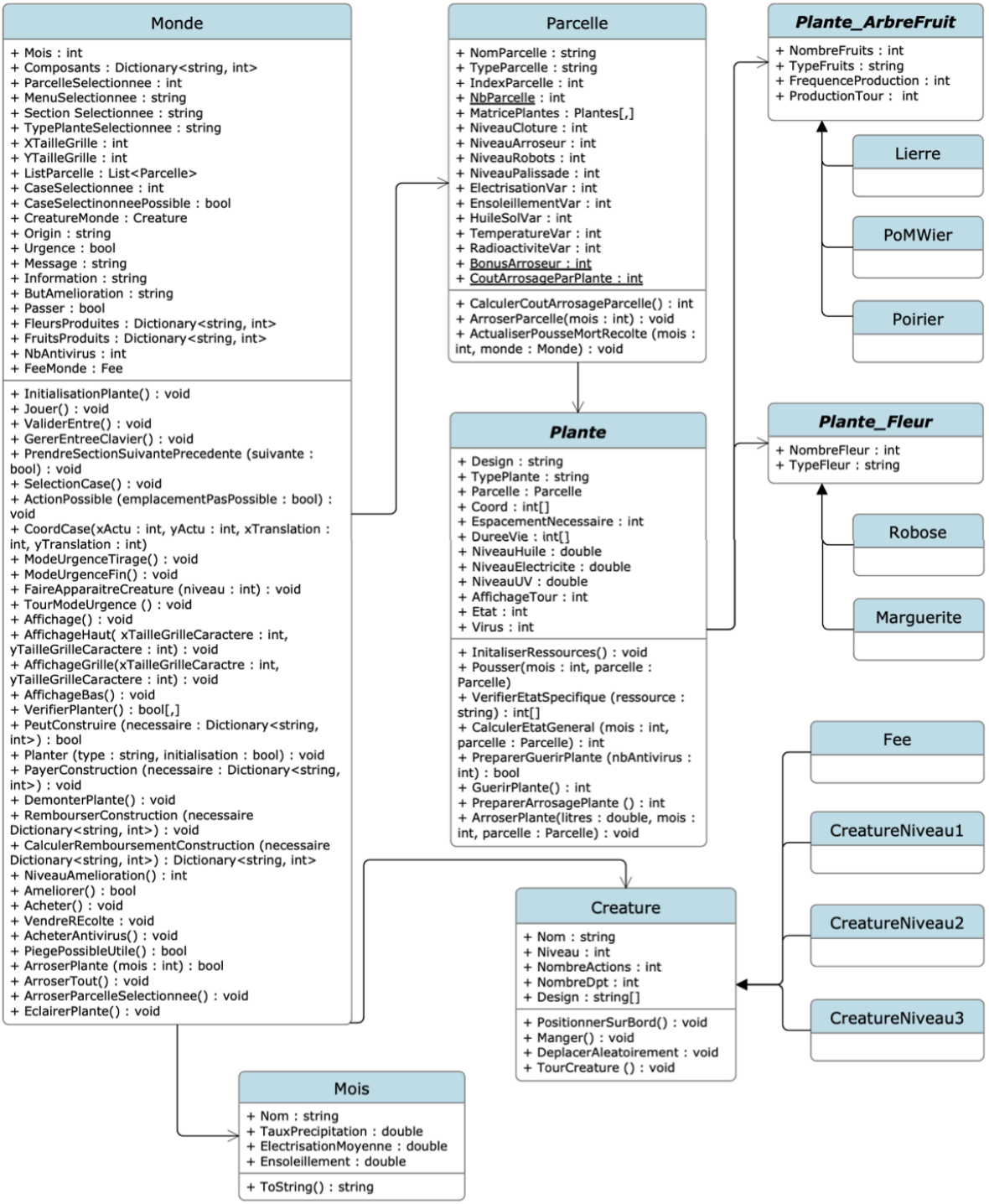


Diagramme UML des principales classes du jeu

Voici le diagramme de classes UML, représentant les principales classes de ce programme. Ce diagramme n'inclut que les classes directement reliées à la classe Monde, et n'inclut donc pas les classes annexes telles que Graphiques, Constantes, et autres. En effet, ces classes servaient davantage à stocker des informations et variables pour rendre le jeu plus agréables à jouer et le code plus lisible, qu'à créer des objets. C'est pourquoi, pour conserver un diagramme lisible nous avons choisi ici de nous concentrer sur les classes de monde, de parcelle, de plantes, de mois et de créatures, qui regroupent les propriétés et méthodes principales du jeu.

B - Structure générale du code

Le dossier dotnet contient l'ensemble des fichiers nécessaires au bon fonctionnement de la simulation. Le point d'entrée du programme se trouve dans le fichier Program.cs, qui déclenche le lancement de la simulation.

Un fichier Graphique regroupe l'ensemble des éléments visuels du jeu, incluant les designs des différents composants ainsi que des fonctions d'affichage spécifiques. Cela permet d'alléger le code de la classe Monde et de séparer au maximum l'affichage bien que la fonction Monde dispose de quelques fonctions d'affichages et de modifier facilement l'aspect de la simulation sans impacter son déroulement. Par exemple, nous avons un dictionnaire Palette contenant la couleur de chaque élément.

Le fichier Constantes centralise les paramètres du jeu, facilitant leur gestion et leur modification. Il comprend le dictionnaire pour la navigation, les paramètres descriptifs des plante s, les coûts de construction des différents éléments (plante, amélioration, piège), les coûts et gains liés au marché, les probabilités d'apparition des créatures ainsi que d'autres paramètres.

La gestion des créatures est assurée par le fichier Creature, qui contient la classe de base ainsi que ses dérivées : CreatureNiveau2, CreatureNiveau3, CreatureNiveau4 et Fée. Les sous-clousse CreatureNiveau1, 2 3 et 4 ne contionnent pas de méthode spécifique, elle permettant d'insitancier des créatures sans spécifier les paramètres. Néanmoins nous avons choisi d'en faire

des sous-classe dans l'éventualité où l'on souhaiterait redéfinir certaines actions.

Pour la flore, le fichier Plante contient la classe de base, tandis qu'un autre fichier dédié à ArbreFruit contient cette classe dérivée avec ses sous-classes, représentant chaque type d'arbre fruitier. Deux classes supplémentaires permettent de créer des instances de plantes et de les stocker dans un dictionnaire, uniquement dans le but d'accéder à leurs paramètres de base.

Un fichier distinct contient la classe Fleur, une autre spécialisation de Plante, avec une structure similaire : des sous-classes pour chaque type de fleur et des classes destinées à gérer l'accès aux données de base de chaque variété.

La gestion du temps repose sur le fichier Mois, qui inclut la classe du même nom ainsi qu'une classe associée contenant un dictionnaire de tous les mois. Ce système permet de gérer la météo et ses effets sur le jeu.

Enfin, nous avons la classe Monde avec la fonction principale : simulation. Elle comprend donc le déroulement des tours, des méthodes pour l'apparition d'évènement (mode urgence et fée), des méthodes pour récupérer les saisies de l'utilisateur ainsi que des méthodes pour effectuer les différentes actions sur le potage.

C - Focus sur la classe "Monde" et l'affichage

L'affichage du jeu est géré par une méthode principale nommée Affichage, qui elle-même s'appuie sur trois sous-méthodes : AffichageHaut, AffichageGrille et AffichageBas.

La méthode AffichageHaut est chargée de l'affichage des informations situées en haut de l'écran. Elle comprend le titre du jeu, les données liées au mois en cours, le nombre de boulons, le nombre de fruits, le nom de la parcelle ainsi que l'événement en cours.

La méthode AffichageBas est dédiée à l'affichage des informations contextuelles selon l'action en cours. Il s'agit essentiellement d'une succession de conditions (if) permettant d'adapter l'affichage aux différents menus et sous-sections. Par exemple, si le joueur est dans le menu "Planter" et qu'il a

sélectionné une plante, l'affichage indiquera le coût correspondant. Pour cela, une méthode du module Graphique, nommée AfficherDictionnaire, est utilisée afin d'écrire le contenu d'un dictionnaire sous forme de ligne, notamment pour afficher les coûts associés à certaines actions.

La méthode AfficherGrille est la plus complexe. Elle repose sur différents types de coordonnées, comme illustré sur la figure de référence : les coordonnées absolues dans la console (yConsole), les coordonnées dans la grille (yGrille) et les coordonnées dans une case (yCase), exprimées en nombre de caractères. Lors de l'exécution du programme, c'est la variable yConsole qui effectue le parcours ligne par ligne. Le traçage débute à partir d'un repère défini (YGrilleStart), et les autres types de coordonnées sont ajustés dynamiquement pour détecter, entre autres, les interlignes.

Les coordonnées relatives à la grille permettent d'accéder aux plantes présentes, tandis que celles à l'intérieur d'une case servent à tracer les motifs de chaque plante. Elles permettent aussi de masquer certaines lignes lorsque la plante est trop petite pour être entièrement affichée.

Enfin, le menu latéral est tracé à l'aide d'un dictionnaire associant chaque ligne à une section à afficher. Grâce à la variable de parcours en caractères, il suffit de vérifier si la ligne courante existe en tant que clé dans le dictionnaire, et d'en afficher la valeur. Ce mécanisme assure une gestion indépendante du menu par rapport à la grille principale.

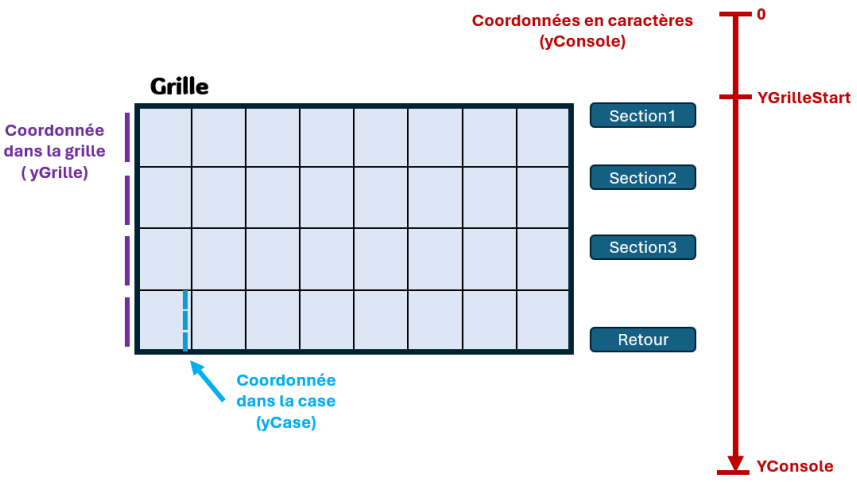


Figure présentant les différentes coordonnées en y

La fonction AfficherGrille se distingue également par sa complexité liée à la diversité des éléments qu'elle prend en compte pour chaque case. En effet, elle doit déterminer si la case correspond à la position du curseur, à celle d'une créature, ou à une plante. Chacun de ces cas entraîne un affichage spécifique.

Elle intègre également une gestion fine des couleurs à travers l'utilisation du dictionnaire Palette issu de la classe Graphique recensant les couleurs de chaque élément.

Enfin, la fonction prend aussi en charge le tracé des matériaux présents sur la grille, tels que les arroseurs, palissades et clôtures. Pour cela, elle fait appel aux méthodes appropriées de la classe Graphique, assurant une séparation claire entre la logique de tracé et les données de jeu.

Pour les plantes, l'affichage se fait également grâce à des constantes dans la classe de graphiques. L'apparence des plantes est conditionné à leur âge (plus elles poussent, plus les lignes du design s'impriment), et à son état (la plante est verte si elle est en bon état, jaune/orange si elle est en état moyen et rouge si elle est en danger de mort).

3. Gestion de projet

Au démarrage du projet, nous avons d'abord mené une phase d'idéation pour choisir l'univers que nous souhaitions développer. Cette étape nous a permis de définir un cadre cohérent. Ensuite, nous avons dressé une liste des fonctionnalités à implémenter, afin de répondre aux exigences du cahier des charges.

Nous nous sommes accordés sur les classes, ce qui nous a permis de structurer l'architecture du code de manière claire et d'organiser notre dépôt GitHub en conséquence. À ce sujet, une confusion initiale a mené à la création de deux dépôts GitHub. Bien que nous ayons finalement travaillé sur un seul dépôt, nous n'avons pas réussi à supprimer l'autre. Outre cela, nous n'avons pas rencontré de difficulté majeure sur github.

Après cette phase préparatoire, nous avons réparti les tâches.
Une personne s'est chargée d'effectuer le diagramme la réalisation du diagramme UML et de recenser les caractéristiques de l'univers du jeu : météo, types de plantes, composants, etc., afin de constituer une base de données cohérente et de l'implémenter dans le code. Cette étape a nécessité un travail collaboratif important sur un document Google partagé.

Une autre personne s'est occupée des fonctions liées à l'affichage, à la navigation dans les menus et à l'interaction utilisateur, pour assurer un enchaînement fluide et logique des actions pendant la partie.

Concernant l'échéance du rendu, une confusion personnelle, ainsi qu'une forte accumulation de soutenances à préparer sur la période nous a forcés à accélérer fortement le rythme durant la dernière semaine pour finir le projet à temps.

Conclusion : bilan critique du projet

Au début du projet, il nous a fallu un certain temps pour pleinement prendre en main le sujet : à la fois très précis par endroit, et très vague ailleurs, cette semi-liberté était à la fois un avantage et un inconvénient. Bien que cela nous ait permis de faire preuve de créativité, il nous a été difficile de définir nos priorités et de cerner clairement les attentes du cahier des charges.

Malgré cela, nous avons apprécié le fait de pouvoir créer un univers de toutes pièces. Ce processus de conception libre nous a permis de nous investir dans un cadre original et personnel. Par ailleurs, par rapport à notre premier projet, l'utilisation des classes nous a réellement aidés à structurer le code de manière plus lisible et modulaire, rendant la navigation dans le projet plus fluide. Un aspect que nous aurions pu améliorer est la gestion des éléments publics et privés dans nos classes. Nous avons parfois manqué de rigueur dans l'encapsulation, en exposant certaines variables ou méthodes qui n'auraient pas nécessairement besoin de l'être. Mais nous avons plutôt choisi de nous

concentrer sur le reste du code, sans nous rajouter une difficulté à ce sujet, car dans le cadre d'un jeu, l'enjeu est minime.

En dehors de notre erreur concernant l'échéance, nous avons rencontré des difficultés à nous immerger pleinement dans le développement. Bien que des séances aient été dédiées au projet, leur répartition sur plusieurs semaines entraînait une perte de temps à chaque reprise, le temps de se replonger dans la logique du code. Nous aurions préféré consacrer deux journées entières consécutives au projet, ce qui aurait permis une concentration continue, mais cela s'est révélé compliqué au vu du nombre de projets à gérer en parallèle.

Malgré ces contraintes, nous sommes satisfaits du résultat final. Nous avons réussi à réaliser un jeu fonctionnel, qui remplit en grande partie les objectifs du cahier des charges, et cela constitue pour nous une réussite.