JUSTIFICATION TECHNIQUE

Promotion 2027 - Groupe 4 : Advenier Flore et Houssemaine Pascal

23/05/2025

Sommaire

I - Introduction

Dans le cadre du cours Programmation avancé dispensé à l'ENSC en première année, nous avons réalisé le jeu informatique ENSemenC. Dans ce jeu, le joueur gère un terrain destiné à l'agriculture. Le but du jeu est de faire pousser différentes espèces de plantes malgré les aléas de la météo et des événements aléatoires urgents nécessitant une intervention imminente. Le jeu ne fixe pas d'objectif de victoire précise, il est donc libre au joueur de se fixer ses propres objectifs. Cependant la partie peut se terminer plus tôt que prévu si 3/4 des plantes ou plus sont en décroissance ou qu'au moins la moitié de ses plantes sont mortes à partir d'un terrain contenant au moins 10 plantes.

Nous nous sommes organisés durant ce semestre S6 pour une réalisation fluide et structurée du projet. Nous avons également essayé d'au mieux suivre les règles et autres contraintes,en faisant des choix, tout en ajoutant des options ou petites subtilités afin de rendre le jeu le mieux construit et le plus agréable à jouer.

II - Structure du projet

II - 1 Choix pour le jeu

Lors de la conception du jeu nous avons suivi les règles et fonctionnalités obligatoires pour le projet comme une plante meurt si ses préférences ne sont pas respecté à 50%. Nous avons cependant fait des choix sur les zones grises du projet.

Par exemple, pour calculer la survie des plantes, nous avons choisi de prendre en compte la luminosité, l'humidité et la température. Si les besoins en luminosité et les besoins en humidité ne sont pas respectés, la plante meurt. Si la température excède de quatre fois la température minimale ou maximale de la plante, elle meurt aussi. Si nous ne sommes pas dans ces cas extrêmes, un score de vie de la plante est calculé (détaillé dans les méthodes).

Les plantes que nous avons créées ont toutes des caractéristiques et besoins différents. Les onze plantes que nous avons créées ont quatres mode de croissance (de semi à plante mûre). Cependant, la plante "nulle" et la plante invasive que nous avons créées, elles, n'ont pas d'évolution possible et ne peuvent pas mourir.

Au niveau de la structure du jeu, nous avons créé quatres mondes, sous forme de quatres terrains différents (tourbière, argile, sable ou roche) parmi lesquels le joueur peut choisir en début de partie. Nous avons organisé ces terrains en 6 parcelles et chaque parcelle a 12 emplacements pour planter une plante. A chaque tour un des deux modes (classique ou aléatoire est choisi). Il y a une chance sur six d'entrer en mode urgence.

Le mode classique propose une description textuelle du terrain , avec des illustrations. Au début d'un tour, des animaux, néfastes (sanglier ou escargot) ou bénéfiques (abeille et ver de terre), arrivent. Après ceci, le joueur peut effectuer autant d'actions qu'il veut (en fonction de l'argent qu'il possède). Les actions possibles sont :

- Protéger le terrain
- Arroser les plantes
- Ombrager le terrain
- Désherber les plantes invasives et les plantes mortes
- Planter une plante

Ces actions sont faites sur une parcelle à la fois sauf pour l'action protéger qui elle protège les six parcelles à la fois.

Le mode urgence propose une vue globale du terrain sur lequel il faut agir pour résoudre les situations.

II - 2 Structure du code

Pour structurer notre code de la façon la plus lisible possible, nous l'avons séparé en dossiers pour chaque classe de base et ses classes dérivées et en fichier libre pour chaque classe sans hiérarchie. Nous avons ensuite un fichier Programme.cs qui demande quel type de terrain le joueur veut jouer dans (le joueur poursuivra toute la partie dans le même type de terrain) puis qui instancie la classe Simulation avec le terrain choisi. C'est ensuite la classe simulation qui déploie pas association aux autres classes tout le jeu avec une boucle while dans la méthode simuler qui effectue chaque tour de jeu tant que la condition d'arrêt n'a pas été atteinte.

Voici notre diagramme UML qui représente la structure de notre jeu (si l'affiche ne fonctionne pas, nous avons joint le diagramme dans notre répertoire github).

Plus globalement, notre jeu peut être divisé en plusieurs éléments. Nous avons un dossier Plantes dans lequel les 14 plantes héritées sont répertoriées, un dossier Terrains avec les 4 terrains possibles hérités, un dossier Urgences avec les deux types d'urgences et un dossier Animaux avec nos quatres animaux hérités. En plus des ces dossier nous avons des classes en plus telles que Année, Saison, Parcelle et Mois. Toutes ces

classes sont appelées dans notre classe simulation elle-même appelée par le programme principal.

II - 3 Normes du projet

Pour respecter les normes générales et les requêtes du projet, nous suivons les normes PascalCase pour les Méthode, Classes et Propriété et camelCase pour les variables et les attributs. Le code est absent d'accents mis à part pour quelques string comme Décembre par exemple.

III - Explication des Classes

III - 1 Classes de temporalité et de météo

Mois

Nous avions choisi de modéliser un tour de jeu comme étant un mois. Nous voulions également que chaque mois paraisse particulier et que cela modifie les paramètres du jeu. La classe Mois permet donc de répertorier différentes valeurs utiles au jeu comme la température moyenne en celsius, le taux d'ensoleillement (entre 0 et 1 en croissance),et la pluviométrie en cm par mois.

En fonction du nom du mois lors de l'instanciation on accède aux différentes valeurs. Nous avons également complété la classe par un ***toString*** afin de pouvoir faire un résumé des valeurs du mois facilement

On peut noter qu'on avait envisagé une autre approche au début en influant sur l'aléatoire par des fluctuations des différentes variables mais nous l'avons abandonné pour faciliter l'exécution.

```
//les modification de météo d'un mois à l'autre

        int fluctuationPluviometrie = rng.Next(-2, 2 + 1);

         MeteoDuMois.Pluviometrie = Math.Abs(MeteoDuMois.Pluviometrie+ fluctuationPluviometrie);

         int fluctuationTemperature = rng.Next(-2, 2 + 1);

      MeteoDuMois.Temperature += fluctuationTemperature;

      MeteoDuMois.Ensoleillement*= rng.NextDouble();

      int risqueModeUrgence = rng.Next(0, 5);
```

Ancien code abandonné

Saison

Lors de la conception nous nous disions qu'il pourrait être intéressant de faire les même différenciation à un cran au dessus pour les saisons. Cependant nous n'avons pas eu le temps d'implémenter une influence des saisons mais la classe reste là si jamais on voudrait complexifier le jeu en le rendant encore plus asymétrique.

Saison est constituée d'une liste de mois qui s'instancie en fonction du nom de saison attribué de manière similaire à la classe Mois

Annee

La classe Annee sert donc à contenir une liste de liste de mois. La classe contient la méthode ***DonnerLeMois*** qui permet de modifier deux indices qui identifient le mois en question dans la double liste. La méthode **ChangerDeMois** permet de modifier ces deux indices là. Une méthode ToString faisant appel au contenu des contenant de la double liste permet de donner les informations du mois actuel. Cette classe sera seulement instancié une fois en début de simulation.

III - 2 Classes de répartition spatiale

Terrain

Pour notre jeu, nous avons créé une classe ***Terrain*** parente et quatres classes héritées : ***TerrainSableux***, ***TerrainRocheux***, ***TerrainArgileux*** et ***TerrainTourbiere***. La classe terrain est appelée plusieurs fois dans le jeu. Au début, dans le programme principal, elle est appelée pour initialiser le terrain de jeu en fonction du type de terrain choisi par le joueur. Durant la partie, elle est appelée pour afficher le terrain que ce soit en mode classique ou urgence.

La classe parente a les classes suivantes :

```
4 references
public bool TerrainProtege {get; set;}
6 references
public double HumiditeTerrain {get; set;} //Allant de 0(
5 references
public double AbsorbtionDeLeau {get; set;} // allant de
2 references
public double EnsoleillementTerrain {get; set;} //Allant
66 references
public List<Parcelle> Parcelles {get; set;} //Dans chaqu
9 references
public string TypeTerrain {get; set;} = string.Empty; //
```

Cela permet d'avoir les caractéristiques du terrain (***AborbtionDeLeau***, ***HumiditeTerrain*** et ***EnsoleillementTerrain***) qui varient en fonction du type de terrain choisi. Le type de terrain est initialisé dans les classes héritées.

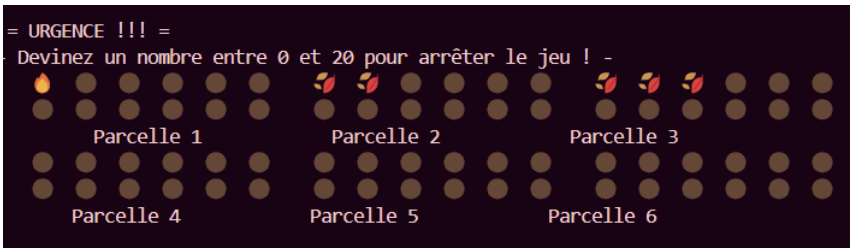
Dans le constructeur de ***Terrain***, la liste ***Parcelles*** est initialisée. Les caractéristiques sont aussi initialisées en fonction du type. Le booléen ***TerrainProtege*** est lui utilisé pour vérifier si l'action "Protéger le terrain a été appelée par le joueur" et pour garder cette information dans l'affichage du terrain.

Les méthodes créées dans cette classe peuvent être réparties en deux catégories : des méthodes d'affichage et une méthode de protection. La méthode protéger ne prend rien en paramètre, elle renvoie uniquement le booléen ***TerrainProtege*** à true. Les autres méthodes sont ***ToClassiqueString***, ***ToActionString*** et ***ToUrgenceString***. Nous avons décidé d'écrire trois équivalents de méthode ToString. Cependant, ces méthodes sont appelées par leur nom et non par un Console.WriteLine.

- ***ToClassiqueString*** est utilisé en mode classe et affiche le descriptif du terrain et des plantes grâce à des descriptions textuelles et des emoticônes.

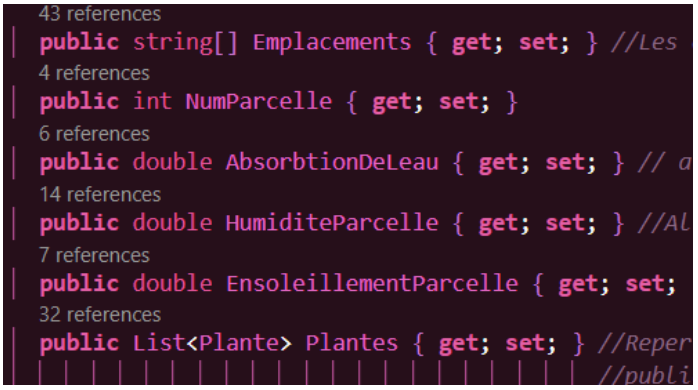
- ***ToActionString*** est uniquement appelée après que le joueur ait réalisé une action. Elle affiche le même résultat que ***ToClassiqueString*** avec un descriptif du terrain plus synthétique.

- ***ToUrgenceString*** est utilisé en mode urgence et affiche tout le terrain en vue globale pour que le joueur puisse interagir avec les problèmes à résoudre.



Parcelle

Notre classe Parcelle constitue les six “sous-terrains” du terrain. Chacune est composée de douze unités d’espace dans lesquelles nous plantons les plantes. Cette classe est primordiale car elle contient en méthodes toutes les possibles actions du joueur (hors protéger le terrain).



Voici les attributs crées. Les caractéristiques de la parcelle sont les mêmes que celles du terrain (***AborbtionDeLeau***, ***HumiditeParcelle*** et ***EnsoleillementParcelle***) au moment de l’initialisation. Cependant, puisque nos actions sont réalisées sur une parcelle à la fois, il était nécessaire de reprendre ces attributs. Les attributs ***Emplacements*** et ***Plantes*** servent à répertorier les plantes, respectivement pour l’affichage des dessins des plantes et pour retenir en mémoire les plantes plantées, leur position et leurs caractéristiques.

Pour ce qui est des méthodes, comme dit précédemment, nous en avons 6 (dont 4 qui représentent les actions des joueurs :

- ***Planter*** est utilisée pour demander au joueur quelle plante il veut planter sur quelle parcelle. Ces deux questions sont imbriquées dans des boucles do…while avec des conditions de robustesse pour éviter que le jeu s’arrête si le joueur entre une valeur non acceptée. Les onze cas sont différenciés par un switch…case.

- ***Desherber*** permet de remplacer les plantes mortes 🍂 et mauvaises herbes 🌱 en emplacement vide.

- ***Ombrager*** diminue simplement l'ensoleillement de la parcelle et donc des plantes dessus.

- ***Arroser*** permet d'augmenter l'humidité de la parcelle et donc des plantes. Il y a trois intensité d'arrosage parmi lesquelles le joueur peut choisir : 1, 2 et 3. Cela ajoute 0.1, 0.2 ou 0.3 à l'humidité de la parcelle. Ce choix est demandé au joueur de façon robuste grâce à une boucle do…while.

- ***InfiltrationDeLaPluie*** ajoute un taux d'humidité de la parcelle de façon proportionnelle au mm de Pluie tombés dans le mois.

```
HumiditeParcelle += moisActuel.Pluviometrie * AbsorbtionDeLeau / 10;
if (HumiditeParcelle > 1)
{
    HumiditeParcelle = 1;
}
```

- ***InfluenceSoleil*** a la même fonction mais en modifiant l'ensoleillement de la parcelle.

```
// on initialise l'ensoleillement de la parcelle
EnsoleillementParcelle = moisActuel.Ensoleillement;
//l'ensoleillement asseche la parcelle et reduit l'humidité de la
HumiditeParcelle -= HumiditeParcelle * moisActuel.Ensoleillement;
if (HumiditeParcelle < 0)
{
    HumiditeParcelle = 0;
}
```

III - 3 Classes des plantes

Plante

Cette classe est composée d'une grande variété d'attributs et est la classe parente des quatorze autres plantes. Cependant, il est nécessaire de dire que nous avons créé des attributs que nous n'avons pas utilisés dans le jeu. Nous les avons tout de même gardés pour illustrer l'évolution de notre réflexion. Par exemple, ***SaisonSemi*** et ***DureeDeMaturation*** nous ont pas servi.

```
//CARACTERISTIQUES DE LA PLANTE
    public string? NomPlante {get; protected set;}
    public List<string>? ImagesPlante {get; set;}
    public bool NatureCommercialisable {get; protected set;}
    public int NombreProduits {get; protected set;}
    public int ValeurProduit {get; set;} //Les plantes ont une valeur monétaire (🔔) par produit récolté.

    //PREFERENCES DE LA PLANTE
    public Saison? SaisonSemi {get; protected set;}
    public double BesoinHumidite {get; set;} //Compris entre 0 et 1.
    public double BesoinLuminosite {get; set;} //Compris entre 0 et 1.
    public double TemperaturePrefereeMin {get; protected set;}
    public double TemperaturePrefereeMax {get; protected set;}

    //DONNEES INITIALES QUI CHANGENT AU COURS DE LA PARTIE
    public double VitesseCroissance {get; set;} //Symbolise l'état de la plante, compris entre -1 et 1 :  -1 = mort, 0 = pas de changement, entre 0.5 et 1 = double la taille lors d'un tour de jeu.
    public Parcelle? ParcellePlante {get; set;} //Indique sur quelle parcelle du terrain se situe la plante.
    public int NiveauMaturation {get; set;} // Indique à quel niveau dans les emoticone la plante est, 0 pour morte 4 pour à son max

    //CONTRAINTES

    public int DureeDeMaturation {get; set;} //En mois.
```

Cette classe contient une seule méthode : ***VerificationEtatPlante***. Cette méthode permet de savoir si la plante est morte (ses besoins en humidité et ensoleillement ne sont pas respectés ou la température excède de quatre fois la température minimale ou maximale de la plante) et calcule sa vitesse de croissance en fonction de la température si elle n'est pas morte.

Les quatorze classes héritées que nous avons créées peuvent être divisées en trois catégories : les plantes à planter, l'invasive et la plante "null". La plante invasive 🌱 et la plante null 🟤 partagent les mêmes caractéristiques : elles sont immortelles et sont illustrés par une seule

image. Les onze autres plantes ont des valeurs d'attributs changeantes mais elles fonctionnent de la même manière. Nous avons donc : blé, maïs, tournesol, nénuphar, tulipe, bamboo, cactus, fraise, carotte, piment et raisin. L'évolution de la maturité de splantes est représentée par ***ImagesPlantes*** : {" 🍂 ","🌽.0","🌽.1","🌽.2","🌽.3"}. Il y a 5 modes de croissance : morte, semi jusqu'à mûre.

III - 4 Classes des perturbateurs

Animaux

Nous avons créé quatres animaux, deux destructeurs (sanglier et escargot) et deux bénéfiques (abeille et ver de terre). Ces quatres classes ont des méthodes et attributs différents. Leur unique point commun réside dans le fait qu'ils influencent une seule parcelle choisie aléatoirement.

- Grâce à ses pouvoirs pollinisateurs, l'abeille fait maturer chaque plante (hors plante invasive et morte) d'un niveau. Pour cela, elle parcourt toutes les plantes de la parcelle.
- Si les conditions d'humidité de la parcelle sont strictement inférieures à 1, les vers augmentent cette humidité de 0.1.
- Le sanglier court sur une parcelle et détruit une plante sur deux.
- Les escargots ne viennent manger les plantes commercialisables que si l'humidité du terrain est supérieure à 0.3.

Il est donc possible de faire face aux possibles effets néfastes des animaux en mettant une barrière pour le sanglier et en gardant une humidité faible pour les escargots.

Urgences

Nous avons créé deux urgences : ***Enfant*** et ***Feu***. Ces deux-ci ont la même probabilité d'apparition. Les deux classes ont le même fonctionnement et partagent le même constructeur et attributs. Néanmoins, nous n'avons pas créé ces classes de façon héritée car les

deux idées nous sont venues à des moments différents. Dans les deux cas, nous avons une méthode ***Urgence*** qui affiche un message et lance un petit jeu. Au cours des manches de jeu, le feu ou l'enfant progressent sur le terrain.

- La classe ***Enfant*** demande au joueur de battre l'enfant au pierre feuille ciseaux et l'enfant détruit les plantes dès qu'il avance.
- De la même façon, dans la classe ***Feu*** le feu détruit au fur et à mesure des saisies du joueur : le joueur doit deviner un nombre entre 0 et 20 (avec des indices de l'ordinateur).

III - 5 Classe de déroulé du jeu

Simulation

Notre classe ***Simulation*** est le centre de notre jeu. Elle prend comme attributs toutes les variables principales de notre programme et certaines spécifiques à la boucle de jeu (***rng***, ***conditionArret*** et ***ArgentJoueur***).

```
3 references
private Random rng = new Random(); //est utilisé pour l'al
4 references
private Annee anneeSimulation;
6 references
private bool conditionArret = false;
4 references
private int NbrTour { get; set; }
15 references
private int ArgentJoueur { get; set; }
42 references
private Terrain TerrainSimulation { get; set; }
4 references
public List<List<Plante>>? EnsemblePlantes { get; set; } //
2 references
private PlanteNull PlanteNull { get; set; } //Plante nulle
1 reference
public Simulation(Terrain terrainJeu) //Prend en paramètre
{
```

Son constructeur permet d'initialiser les plantes de chaque parcelle par des plantes nulles et de remplir l'***EnsemblePlante*** qui

offre une vue d'ensemble sur le terrain. Les méthodes sont au nombre de deux : ***CalculerConditionArret*** et ***Simuler***.

- ***CalculerConditionArret*** calcule le nombre de plantes mortes, le nombre de plantes en mauvais état (avec une vitesse de croissance inférieure à -0.5 et vérifie le solde du joueur (si le solde est supérieur à -5). Il demande aussi au joueur s'il veut continuer à jouer. Ces quatres critères sont susceptibles de mettre la ***conditionArret*** à true.

- ***Simuler*** est faite d'une grande boucle do…while qui vérifie la condition d'arrêt. Au début de chaque tour, on change de mois, fait apparaître les plantes invasives (si le nombre de tours est un multiple de 5), vérifie l'état des plantes (les affiche mortes, les cueille et les fait grandir si besoin - une plante peut maturer mais également revenir à un état de croissance plus faible) et l'on affiche les informations nécessaire sua tour (météo du mois). Uns fois cela fait, un random décidera de si le tour sera en mode urgence ou classique. Dépendant du cas, un autre random choisi l'urgence ou l'animal à venir sur le terrain. En effet, un tour classique commence par l'apparition d'un animal et cela se finit avec les actions du joueur. Le joueur peut faire autant d'actions qu'il veut - jusqu'à qu'il lui reste 1 ou mois dans son solde. Cependant, l'action protéger étant assez coûteuse, pour la réaliser, il est nécessaire d'avoir les 15 clochettes. Enfin, la boucle se termine en demandant au joueur s'il veut continuer ou arrêter.

IV - Déroulement du projet

IV - 1 Répartition du travail

Pour structurer notre projet et se répartir au mieux le travail, nous avons commencé par créer un document listant toutes les choses à faire. Nous avons détaillé la structure du programme jusqu'aux types des attributs et paramètres des méthodes. Une fois ce travail fait, nous nous

sommes répartis le travail une tâche à la fois. Bien sûr, la structure du programme a été modifiée au cours du temps et nous nous sommes retrouvés plusieurs fois pour réviser cette trame.

Chacun faisait une tâche à la fois et tenait l'autre au courant des ses avancées. Nous avancions globalement à un même rythme assurant ainsi une répartition équitable du travail. Lorsque l'un de nous bloquait sur une tâche (une méthode, un débogage, etc…) nous nous retrouvions en présentiel pour en discuter et y réfléchir ensemble.

Pour nous aider au moment de la rédaction du justificatif technique, nous nous sommes imposé de commenter un minimum toutes nos fonctions au moment du code. Bien sûr, tous ces commentaires ont été révisés et modifiés pour le rendu final mais cela nous a tout de même beaucoup aidé.

IV - 2 Utilisation de GitHub

Pour travailler collaborativement, à distance ou en présentiel, nous avons dû utiliser l'outil GitHub. Nous avons commencé par créer une branche chacun (branche "Pascal" et branche "Flore"). Après chaque modification nous commettions et synchronisons les changements sur notre branche permettant ainsi de sauvegarder le travail au fur et à mesure et de regarder l'avancement de l'autre. Puisque nous nous somme toujours répartis le travail en tâches distinctes, nous avancions chacun de notre côté sans soucis. Le plus souvent, quand c'était possible, lorsque nous avions fini notre tâche nous nous retrouvions en présentiel pour merger les branches et résoudre les conflits ensemble. Pour diviser clairement les tâches, Pascal a également utilisé des branches spécifiques pour des tâches spécifiques comme la création des classes temporelles et une branche pour faire uniquement du débogage et de la mise en forme finale.

Lorsque nous faisons des modifications de la mise en forme du code ou des ajouts de commentaires, nous modifions directement dans le main et synchronisons ces changements sans l'avis de l'autre.

Globalement, nous avons trouvé l'utilisation de GitHub bien plus simple et intuitive que le premier projet.

V - Problèmes rencontrés

V - 1 Problèmes résolus

- Débogage des problèmes d'Index :

Nous avons eu beaucoup de soucis avec les indices des listes. En effet, pour les ***Emplacements*** et les ***Plantes*** nous aurions dû utiliser des tableaux vu que leurs dimensions sont fixes. A cause de cette optimisation et des problèmes de déclaration de listes, nous avons eu des soucis d'index. Pour résoudre ce problème, nous avons beaucoup utilisé le débogueur pour avoir une meilleure visibilité sur le projet et la logique du jeu.

- Problème de la plante nulle:

Nous avons eu un problème de typage dans les parcelles c'est-à-dire dans les listes de plantes. En effet, lorsqu'on faisait une requête sur les différents éléments de la parcelle on ne pouvait pas le faire tant que toute la parcelle n'était pas remplie de plantes. Puisqu'une liste contient seulement des éléments d'un même type nous ne pouvions pas avoir un élément nul autre qu'une Plante. Pour palier à ce problème, nous avons donc décidé de créer une plante "immortelle" qui correspond à un espace vide.

VI - 2 Problèmes non résolus

- Changement de mois qui provoque un Index Out of Range :

Même si la méthode ***ChangerDeMois*** de la classe Annee donnais les bon index (***MoisActuel*** et ***SaisonActuel***) il en résultait que ***AnneeActuel[SaisonActuel].MoisDeLaSaison[MoisActuel]*** faisaient

référence à plusieurs bon mois avec parfois des répétition de ces mêmes mois pour ensuite indiquer un index out of range.

VI - Pistes d'amélioration

Le changement de mois paraissait comme une méthode facile et on est donc partie du principe qu'il allait bien fonctionner. Cependant nous avons dû faire preuve de plus de rigueur et nous assurer de son bon fonctionnement plus tôt et pas vers l'approche de la deadline lorsque nous déboggions plus profondément. Après quelques tentatives et améliorations nous avions voulu nous consacrer à d'autre partie du projet malgré que ce problème n'était pas résolu. Nous avons motivé ce choix par le fait qu'on pourrait aussi faire fonctionner le jeu avec un seul mois et sans variations des paramètres. Cela reste tout de même regrettable et nous devrons mieux vérifier même des codes qui nous paraissent simples plus tôt dans le projet.

Les classes ***Annee***, ***Saison*** et ***Mois*** possèdent tous des liens de hiérarchie il aurait éventuellement été intéressant d'essayer de le construire avec de l'hérédité surtout que ce type de classe peut avoir son utilité dans d'autres projets.

Nous aurions toujours pu mieux gérer notre temps mais notre volonté à rendre quelque choses du meilleur de nos capacités nous à pousser à un sprint final qui sera toujours présent puisqu'un jeu peut toujours devenir meilleur. Nous avons aussi compris la nécessité de simplifier nos modélisations et de ne pas être trop ambitieux en voulant trop modéliser le monde réel, ce que nous tâcherons d'implémenter la prochaine fois dès le début.

VII - Conclusion

Nous sommes fiers et contents de notre jeu. Le cahier des charges est respecté et nous avons même ajouté des fonctions créatives. Nous avons bien aimé le fait de modéliser un monde mettant en jeu plusieurs paramètres et objets. Ce projet nous a permis de développer nos compétences en programmation orienté objet et nous avons senti une réelle différence dans notre aisance à coder au fil du projet. Cependant nous aurions pu améliorer quelques points afin d'avoir un jeu complètement fonctionnel mais cette expérience nous a donné la confiance nécessaire pour pouvoir attaquer à nouveaux des projets de cette envergure.

Tout jeu pouvant toujours être amélioré, nous voyons les points faibles et d'amélioration possible. Nous restons tout de même fier du projet que nous rendons de par son fonctionnement, ses touches personnelles et les acquis que ce jeu nous a permis de perfectionner ou d'acquérir.