



Programmation avancée - Projet 2025

ENSemenC

Julien Bernard
Nathan Adelmard



Sommaire

- Interprétation du sujet.....2**
 - 1. Avant propos.....2
 - 2. Interface utilisateur VS diversité des plantes et parcelles..... 2
 - 3. Grille de jeu.....2
- Notre jeu..... 3**
- Les classes.....7**
- Description des classes.....8**
 - 1. MenuChoix.....8
 - 2. Joueur..... 9
 - 3. Plante.....9
 - 4. CepageX..... 13
 - 5. Parcelle..... 14
 - 6. ParcelleX..... 17
 - 7. SauvegardeManager..... 17
 - 8. GestionSaison..... 19
 - 9. Partie..... 20
 - 10. AffichageChargement..... 24
 - 11. ModeUrgence.....25
 - 12. AffichageParcelle.....26



Interprétation du sujet

1. Avant propos

Avant de commencer ce projet, nous nous sommes concertés assez longuement pour déterminer ce qui était le plus important pour nous de développer. En effet, le sujet est à la fois vaste et précis. Sans cadrage préalable de notre part, il est compliqué de démarrer ce projet sans soucis. Nous avons donc commencé par choisir le sujet des vignes car c'est un sujet qui nous intéressait et que nous voulions faire apprendre de manière ludique à l'utilisateur du jeu.

2. Interface utilisateur VS diversité des plantes et parcelles

Étant en école d'ingénieur en cognitive, l'interface utilisateur ainsi que l'expérience utilisateur sont des critères importants pour nous. De ce fait, nous avons naturellement été amenés durant ce projet à passer beaucoup de temps sur la création des différents menu et affichages, leurs aspects générales, leurs accessibilité, leurs rendu graphiques etc... Bien qu'un affichage console soit contraignant, nous avons trouvé des idées et nouvelles fonctions pour se rapprocher un maximum d'une interface graphique d'un jeu plus classique. Mais cette grande implication dans le bon développement d'un environnement stable et visuellement agréable ne nous a pas permis de développer la diversité des plantes et des sols comme nous aurions aimé le faire dans les temps impartis. Cependant, les bases sont posées et ajouter des variantes de plantes sera facilité par ces bases.

3. Grille de jeu

Avant de commencer le développement nous ne savions pas s'il était nécessaire de faire une grille de jeu qui représente une parcelle. Cette question peut sembler sans importance, et pourtant ne pas en faire impliquerait de ne pas pouvoir gérer individu par individu ou alors ceci de manière très peu ergonomique. Nous ne voulions pas non plus devoir gérer des pourcentages de plantes ou juste des nombres de plantes contrôlés de manière globale. Nous



avons envie de proposer un contrôle totale et individuelle de chaque plante. C'est pourquoi nous nous sommes tournés vers une représentation 2D d'une parcelle.

Notre jeu

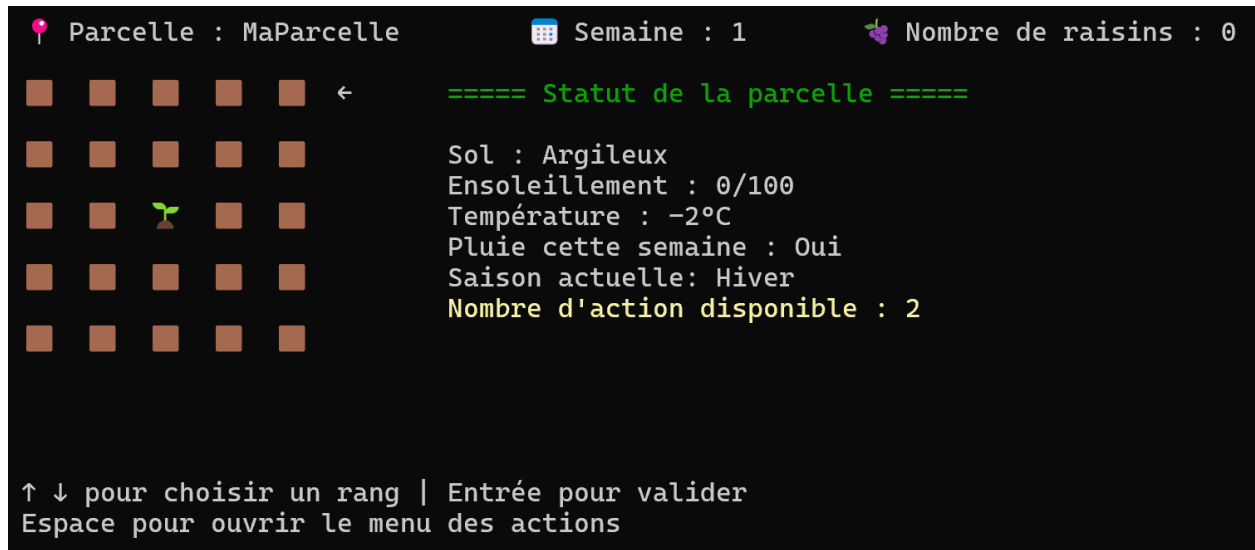
Nous allons maintenant aborder les fonctionnalités principales de notre jeu. Une fois lancé, un menu d'accueil s'affiche et nous permet de lancer une nouvelle partie, charger une partie déjà existante, voir les règles du jeu et quitter le jeu.

```
=====
                        🍷 Le Jeu Viticole 🍷
=====
Utilisez les flèches ↑ ↓ pour naviguer, Entrée pour valider.

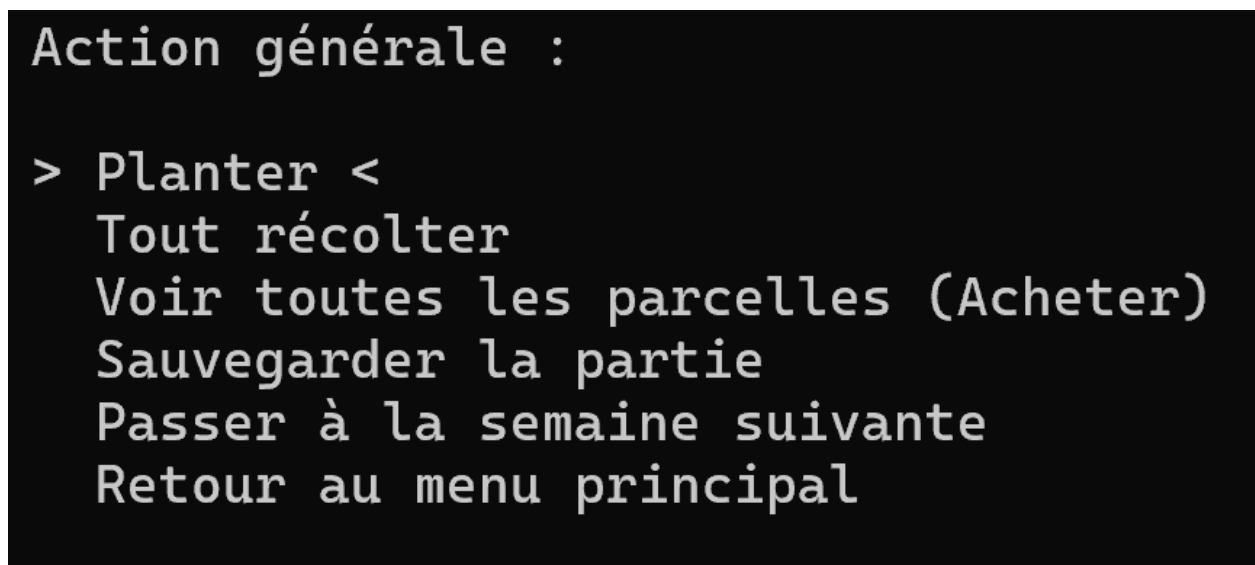
> Nouvelle Partie <
  Charger une Partie
  Règles du jeu
  Quitter
```

Implémenter un système de sauvegarde était important pour nous car un jeu de gestion qui recommence à chaque partie n'a que très peu d'intérêt. Il est d'ailleurs recommandé de run le jeu dans un terminal externe à Visual Studio Code pour une meilleure expérience.

Lancer une partie permet de lui donner un nom et d'initier notre première parcelle gratuite. Il est recommandé de faire une grande parcelle car la première est gratuite et les suivantes sont plus chères en fonction de la taille. Il est aussi possible de nommer cette parcelle et de choisir le type de sol. Une fois initialisée, notre parcelle apparaît dans la console avec un résumé de son statut, incluant la météo et la saison. Chaque type de parcelle possède une couleur de sol différente. Un cépage apparaît sur la grille sous l'apparence suivante : “🌱” et va clignoter pour laisser apparaître un autre emoji qui donne l'état de la plante. Pour référence, la plante peut être saine (✅), malade (🦠), desséchée (💧) et malade et desséchée (⚠️). Il nous semblait important de pouvoir facilement visualiser les problèmes réglables d'un cépage dans l'affichage général plutôt que de devoir cliquer sur chaque plante pour accéder à ces informations.



En appuyant sur la touche “Espace” du clavier il est possible d’ouvrir le menu d’action général.

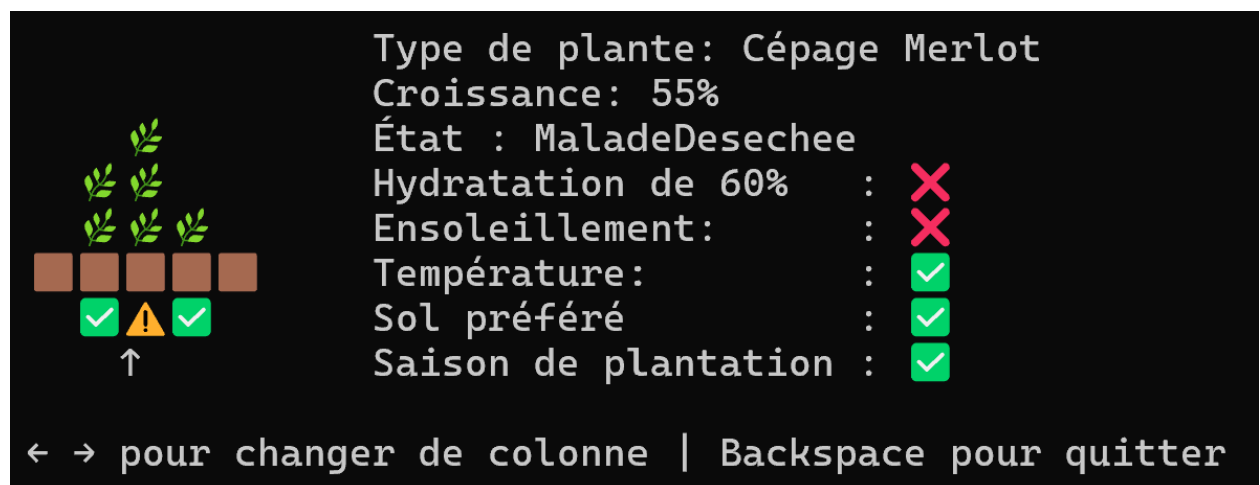


Ce menu permet de Planter, Tout récolter, Voir toutes les parcelles et en acheter, Sauvegarder la partie, Passer à la semaine suivante et revenir au menu principal. Nous allons nous intéresser dans un premier temps à la fonction de planter puis nous reviendrons plus tard sur les autres options de ce menu. Une fois que nous avons appuyé sur entrée pour valider l’action de planter, la parcelle s’affiche à nouveau et un curseur à droite des cases s’affiche. Il est possible de naviguer dans toute la parcelle avec les flèches directrices du clavier. Avec les touches correspondantes aux chiffres il est possible de sélectionner un type de cépage qu’on veut

planter. A savoir que chaque cépage a des sols préférés, des saisons de plantation préférées etc... Une fois le type et la case sélectionnée nous pouvons appuyer sur “Entrée” pour valider la plantation.



Un curseur placé à droite de la parcelle nous permet de naviguer entre chaque rang en appuyant sur “Entrée”. Cette action nous amène dans la vue de rangé détaillé et un deuxième curseur permet de se déplacer de gauche à droite pour venir survoler chaque plante et faire afficher les caractéristiques et le statut de la plante sélectionnée.



Cet affichage est lui aussi dynamique, on y retrouve en dessous de chaque cépage l'emoji caractérisant son état actuel. Chaque plante est représentée par des emojis de feuilles qui seront de plus en plus en grand en fonction de la croissance de chaque plante.



Il est possible d'appuyer sur "Entrée" pour ouvrir le menu d'action sur un cépage sélectionné. Ici nous pouvons : arroser, traiter et récolter

Revenons maintenant au menu d'action générale. La fonction tout récolter permet de parcourir tous les cépages et de récolter uniquement ceux qui ont atteint 100% de croissance. L'option pouvoir voir toutes les parcelles permet d'afficher la liste des parcelles que détient le joueur dans sa partie. Il est aussi possible d'en acheter de nouvelles contre des raisins. Le joueur peut aussi sauvegarder la partie et revenir à l'écran principal. Et enfin il peut passer à la semaine suivante.

Lors d'un passage à la semaine suivante, il y a une probabilité que le mode urgence se déclare. Dans ce cas de figure, des oiseaux attaquent les vignes et le joueur doit réussir un QTE (Quick Time Event) à plus de 50% sous peine de voir ses récoltes réduites de 50%. Un QTE consiste à réaliser une action courte et précise dans un temps imparti court. Ce mode urgence repose donc sur les réflexes et la vivacité du joueur à l'image d'un agriculteur qui ferait fuir des oiseaux un peu trop gourmands.



Les classes

Voici la liste de nos classes avec leurs sous classes :

- Jeu
- Partie
- AffichageParcelle
- AffichageChargement
- Joueur
- Parcelle
 - ParcelleArgileuse
 - ParcelleCalcaire
 - ParcelleGraveuleuse
- Plante
 - CepageCabernetSauvignon
 - CepageChardonnay
 - CepageMerlot
 - CepagePinotNoir
 - CepageSyrah
- CataloguePlantes
- MenuChoix
- ModeUrgence
- Saisons
- SauvegardeManager



Description des classes

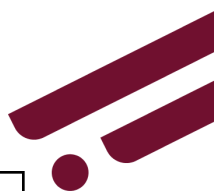
1. MenuChoix

Cette classe est l'une des premières qui a été développée car elle est un point central de notre jeu. En effet, elle est utilisée 7 fois dans le jeu pour créer des menus fonctionnels, dynamiques et intuitifs pour les joueurs.

Propriété	Rôle / fonctionnement (en bref)
<code>options :</code> <code>List<string></code>	Contient la liste des libellés à afficher, l'index dans la liste correspond au choix retourné.
<code>titre :</code> <code>string?</code>	Titre optionnel affiché au-dessus du menu ; <code>null</code> ou chaîne vide \Rightarrow pas de titre.

Constructeur	Effet principal
<code>MenuChoix(List<string> options, string? titre = null)</code>	Stocke la liste reçue dans <code>options</code> et, si fourni, le texte de <code>titre</code> . Aucune logique supplémentaire ; le contrôle d'affichage se fait plus tard via <code>Afficher()</code> .

Méthode et paramètres	Description	Return
-----------------------	-------------	--------



<code>Afficher()</code>	Boucle d'affichage interactive : <ul style="list-style-type: none"> • Efface l'écran et montre le titre (via <code>AfficherTitre</code>). • Parcourt <code>options</code> pour afficher chaque entrée, en entourant celle actuellement sélectionnée par <code>></code> <code><</code>. • Intercepte les flèches <code>↑</code> <code>↓</code> pour déplacer <code>selection</code> circulairement. • Sort de la boucle quand Entrée est pressée. 	Index (int) de l'élément choisi dans <code>options</code> .
<code>AfficherTitre()</code>	Si <code>titre</code> n'est ni <code>null</code> ni vide, l'écrit dans la console avant les options.	void (aucune valeur)

2. Joueur

Propriété	Rôle / fonctionnement
Nom : <code>string</code>	Identifiant du joueur, sert pour l'affichage et pour nommer la sauvegarde.
NombreDeRaisins : <code>int</code>	Stock de raisin récolté (monnaie d'échange pour acheter de nouvelles parcelles). Initialisé à 0, puis incrémenté lors des récoltes.

3. Plante

Les plantes sont les éléments centraux du jeu et leur implémentation en ce qui concerne l'équilibrage est très compliquée car ce n'est pas ce qui nous est appris à l'ENSC. Cependant avec nos recherches personnelles et les connaissances de Julien Bernard au sujet du monde viticole, nous avons pu mettre au point cette classe plante. Il faut bien faire la distinction entre les propriétés d'une plante et les besoins dont elle a besoin pour survivre. En effet ces aspects ne sont pas forcément explicites dans le sujet alors voici notre interprétation suivant notre sujet et nos idées. Une plante est caractérisée, entre autres, par une espérance de vie, une vitesse de




croissance, une probabilité de tomber malade, une production potentielle, et une consommation d'eau hebdomadaire. Tandis que ces besoins sont les suivants et sont au nombre de 6 :

- Ne pas être malade
- Ne pas être desséchée
- Recevoir assez de lumière
- Être planté sur son sol préféré
- Avoir été plantés durant la (les) bonne(s) saison(s)
- Être dans la plage de température préférée

Si la moitié de ces besoins ne sont pas respectés, alors la plante meurt. De plus, chacun de ces aspects peuvent ajouter ou diminuer le taux de croissance hebdomadaire.

La classe plante est d'ailleurs une classe abstraite car elle ne sera jamais instanciée, nous verrons dans la partie suivante les sous classe de cette superclasse pouvant être instanciés.

Propriété	Rôle / fonctionnement
Nom : <code>string?</code>	Nom du cépage (affiché dans l'UI, sert aussi aux messages).
Parcelle : <code>Parcelle</code>	Référence (non-nullable) vers la parcelle qui contient la plante, permet de connaître le sol, la météo courante, etc.
EsperanceDeVie : <code>int</code>	Compteur d'espérance de vie en semaine, décrémente chaque tour : $\leq 0 \Rightarrow$ mort.
Hydratation : <code>int</code>	Niveau d'eau 0-100 %, diminue de <code>ConsommationEauHebdo</code> chaque semaine.
ConsommationEauHebdo : <code>int</code>	Quantité d'eau perdue automatiquement chaque tour.
SaisonMomentPlantation : <code>string</code>	Saison au moment où la plante a été semée (utilisée pour les bonus / survie).



<code>EstDansSaisonPreferee : bool</code>	<code>true</code> si <code>SaisonMomentPlantation</code> figure dans <code>SaisonsPlantation</code> .
<code>VitesseCroissance : int</code>	Points de croissance ajoutés chaque semaine, avant bonus/malus.
<code>ProbaMaladie : double</code>	Probabilité (0-1) de tomber malade à chaque tour.
<code>ProductionPotentielle : int</code>	Nombre de raisins gagnés quand la plante est récoltée (croissance = 100 %).
<code>Etat : enum EtatPlante</code>	État courant : <code>Saine</code> , <code>Malade</code> , <code>Desechee</code> , <code>MaladeDesechee</code> , <code>Morte</code> .
<code>Croissance : int</code>	Pourcentage de croissance (0-100).
Besoins invariables	
<code>SaisonsPlantation : List<string></code>	Saisons autorisées pour la plantation.
<code>SolPreferee : string?</code>	Type de sol idéal.
<code>BesoinsEau : int</code>	Seuil d'hydratation minimal ; $< \Rightarrow$ dessèchement.
<code>BesoinsLumiere : int</code>	Ensoleillement minimal pour bonus / survie.
<code>TemperaturePreferee : (int Min, int Max)</code>	Fourchette de température idéale.
Compteurs internes	
<code>semainesDesechees</code> (<i>privé</i>)	Nombre de semaines consécutives en état <code>Desechee</code> .
<code>semainesMalade</code> (<i>privé</i>)	Nombre de semaines consécutives en état <code>Malade</code> / <code>MaladeDesechee</code> .



Méthode (paramètres)	Description du fonctionnement	Return
<code>AvancerSemaine(int ensoleillement, int temp)</code>	Boucle de vie hebdomadaire : <ul style="list-style-type: none">• Décrément d'eau (<code>Hydratation</code>) et mise à jour de l'état via <code>MettreAJourEtatHydratation</code>.• Vérifie la survie globale (<code>VerifierSurvie</code>), tue la plante si < 3 besoins satisfaits.• Test probabilité de maladie, peut passer de <code>Saine/Desechee</code> à <code>Malade/MaladeDesechee</code>.• Applique la croissance de base + bonus (<code>CalculerBonusCroissance</code>).• Vieillesse (<code>EsperanceDeVie--</code>).• Compte les semaines malade : $\geq 3 \Rightarrow$ mort.	<code>void</code>
<code>Arroser(Parcette parcelle, int quantite = 70)</code>	Ajoute de l'eau (+ <code>quantite</code> limité à 100 %). Met à jour l'état, consomme une action sur la parcelle, affiche un message.	<code>void</code>
<code>Traiter(Parcette parcelle)</code>	Si malade \Rightarrow passe à <code>Saine</code> ou <code>Desechee</code> (si combiné à la sécheresse). Consomme 1 action.	<code>void</code>
<code>Recolter(Joueur joueur, Plante plante)</code>	Si <code>Croissance == 100</code> , remet la croissance à 10 % et crédite <code>ProductionPotentielle</code> au joueur, sinon affiche un message.	<code>void</code>
<code>VerifierSurvie(int ensoleillement, int temp)</code>	Calcule le nombre de besoins respectés ≥ 3 et renvoie <code>true</code> , sinon <code>false</code> .	<code>bool</code>
<code>CalculerBonusCroissance(int lumiere, int temp)</code> <i>(protected virtual)</i>	Additionne / soustrait des points bonus selon hydratation, lumière, température, sol, saison et état. Si la plante est malade ou desséchée, le total de bonus est divisé par deux.	<code>int</code> bonus
<code>MettreAJourEtatHydratation()</code> <i>(private)</i>	Gère le passage à <code>Desechee</code> , réinitialise / incrémente <code>semainesDesechees</code> , tue la plante après 3 semaines consécutives.	<code>void</code>



4. CepageX

Les différents cépages se contentent d'appeler le constructeur de la classe mère Plante. Voici un exemple de l'un des cépages de notre catalogue, le Cépage Merlot :

Propriété initialisée	Valeur dans ce cépage
Nom	"Cépage Merlot"
SolPreferee	"Argileux"
VitesseCroissance	15
ProbaMaladie	0,12
BesoinsEau	70
BesoinsLumiere	50
TemperaturePreferee	(15 ; 30) °C
EsperanceDeVie	100 semaines
ProductionPotentielle	30 raisins
ConsommationEauHebdo	40 %
SaisonsPlantation	{ "Printemps", "Été" }

Constructeur	Effet principal
<code>CepageMerlot(string saisonActuelle)</code>	Appelle <code>base(saisonActuelle)</code> puis renseigne les valeurs du tableau ci-dessus. Aucun autre traitement.



5. Parcelle


La classe Parcelle permet de représenter l'endroit où seront plantés nos cépages. C'est une classe abstraite qui a comme sous classe directe les différents types de sols de notre univers.

Propriété	Rôle / fonctionnement
Nom : string	Identifiant de la parcelle dans l'interface (affiché dans les menus).
Largeur : int	Nombre de colonnes de la matrice, limite la position x des plantes.
Hauteur : int	Nombre de lignes, limite la position y.
MatriceEtat : Plante?[,]	Grille 2D contenant soit une référence de plante, soit null (emplacement vide).
MatriceEtat_Prox y : Plante?[][]	Tableau « jagged » (tableau de lignes) utilisé uniquement pour la (dé)-sérialisation JSON ; convertit automatiquement vers/depuis MatriceEtat . Ceci sera revu dans la classe qui s'occupe de la sauvegarde.
TypeSol : string?	Type de sol fixe (argileux, graveleux, calcaire...), sert à comparer avec les sols préférés des cépages.
Ensoleillement : int	Valeur de l'ensoleillement courante (0-100) de la parcelle. Mise à jour à chaque semaine et dépend de la météo.
Temperature : int	Température courante (°C).



<code>Pluie : bool</code>	<code>true</code> si la semaine est pluvieuse (arrosage gratuit +20% mais ensoleillement = 0), sinon <code>false</code> .
<code>BlocTerre : string</code>	Emoji représentant visuellement la parcelle (par défaut " ").
<code>NombreActionDispo : int</code>	Compteur d'actions restantes pour la semaine. Ce compteur n'est pas dans joueur car chaque parcelle doit avoir un nombre limité d'action. Sinon on pourrait voir naître des stratégies consistant à acheter plusieurs parcelles mais en remplir qu'une pour bénéficier des actions des autres.
<code>BonusAction : int</code>	Bonus calculé = nombre de plantes présentes, ajouté au quota de base.
<code>BaseActions : const int = 3</code>	Nombre d'actions accordé chaque semaine avant bonus (constant).

Constructeur	Effet principal
<code>Parcelle(string nom, int largeur, int hauteur)</code>	Initialise <code>Nom</code> , <code>Largeur</code> , <code>Hauteur</code> , puis crée <code>MatriceEtat</code> vide de taille <code>[hauteur, largeur]</code> . Les autres propriétés restent à leurs valeurs par défaut (météo non définie, sol fixé dans la sous-classe).




Méthode et paramètres	Description du fonctionnement	Return
void ReinitialiserActions()	Compte le nombre de cases non-nulles dans MatriceEtat et stocke le résultat dans BonusAction . En effet, chaque plante ajoute une action à l'utilisateur chaque semaine. Fixe NombreActionDispo à BaseActions + BonusAction . Appelée à chaque début de semaine.	void
void UtiliserAction()	Décrémente NombreActionDispo d'une unité dès qu'une action (planter, arroser, etc.) est effectuée.	void
<i>(accesseurs de</i> MatriceEtat_Proxy)	Get : reconstruit un tableau jagged pour la sérialisation. Set : reconstruit la grille 2D après désérialisation.	Plante?[][] (get) / void (set)



6. ParcelleX

Même chose ici que pour les sous classes de Plante, toutes les sous classes de Parcelle se contentent d'appeler le cons de Parcelle. Voici un exemple de parcelle argileuse :

Propriété héritée	Valeur appliquée dans la sous-classe	Effet
TypeSol	"Argileux"	Indique aux plantes qu'il s'agit d'un sol argileux, utilisé pour les bonus/malus de croissance.
BlocTerre	" 	Représentation graphique dans la console (cases de la parcelle marron/terre battu).
<i>(toutes les autres propriétés de Parcelle restent inchangées et fonctionnent comme décrit précédemment.)</i>		

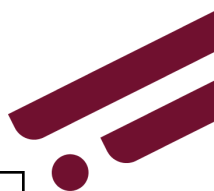
7. SauvegardeManager

Cette fonction est un module statique qui encapsule toute la logique de persistance du jeu : il sérialise une instance complète de Partie en JSON, la sauvegarde sur disque dans un dossier dédié, nommé "sauvegardes", permet l'énumération des fichiers créés pour alimenter le menu "charger une partie" de l'écran de démarrage, puis désérialise à l'identique pour restaurer l'état du gameplay, si la personne veut continuer sa partie plus tard. Le code gère lui-même la création du répertoire, l'écriture et la lecture des fichiers, de sorte que le reste de l'application n'a qu'à invoquer trois méthodes (sauver, lister ou charger) sans se préoccuper du format ni du support physique.



Propriété	Rôle / fonctionnement
<code>var json</code>	Reçoit le texte JSON produit par <code>JsonConvert.SerializeObject(partie, ...)</code> . Elle sert de tampon : on y stocke la version sérialisée de l'objet <code>Partie</code> afin de pouvoir l'écrire sur disque, l'envoyer ou l'afficher.
<code>var fichiers</code>	Liste complète des chemins de tous les fichiers <code>.json</code> trouvés dans le dossier sauvegardes.
<code>string json</code>	Contient le texte JSON lu depuis le fichier de sauvegarde
<code>new JsonSerializerSettings</code>	Crée une instance de configuration pour Json.NET ; c'est là que l'on précise comment la sérialisation/désérialisation doit se comporter.
<code>TypeNameHandling.Auto</code>	Ecrit le nom complet du type, afin de restaurer correctement les sous-classes au chargement.
<code>ReferenceLoopHandling.Ignore</code>	Ignore les références circulaires pour éviter les boucles infinies ou exceptions.
<code>SerializeObject et DeserializeObject</code>	Permet la transformation en Json des données du jeu et inversement.

Méthode et paramètres	Description	Return
-----------------------	-------------	--------



<code>Sauvegarder(Partie partie, string nomSauvegarde)</code>	Permet la sauvegarde de la partie par la sérialisation des données.	void (aucune valeur)
<code>ListerSauvegardesDis ponibles()</code>	Interroge le dossier <code>sauvegardes</code> pour récupérer tous les fichiers <code>.json</code> , retire l'extension et renvoie la liste de noms.	List<string>
<code>Charger(string nomSauvegarde)</code>	Vérifie l'existence du fichier demandé puis lit le JSON, puis déséréalise en <code>Partie</code> avec les mêmes options (gestion des types + ignore des boucles). En cas d'absence de fichier, informe l'utilisateur et renvoie <code>null</code> .	Partie? (? : car peut être nullable)

8. GestionSaison

Cette classe permet juste de gérer les saisons de manière assez basique. Le seul point un peu compliqué ici, et auquel nous avons été confronté est le fait que la saison de départ d'une parcelle est tirée au hasard et que par conséquent le roulement peut avoir 4 nature différentes.

Nom	Rôle / fonctionnement
<code>startIndex : int</code> (privé)	Valeur entre 0 et 3 choisie aléatoirement au constructeur, détermine la saison de départ : 0 = Printemps, 1 = Été, 2 = Automne, 3 = Hiver. Nous avons introduit cette variable après s'être rendu compte que les parties allaient commencer par des saisons différentes.



<code>dureeParSaison</code> : <code>int</code> (<i>privé</i>)	Nombre de semaines qui composent une saison. On force un minimum de 1 pour éviter toute division par zéro.
<code>SaisonDeDepart</code> : <code>string</code>	Propriété calculée qui traduit <code>startIndex</code> en nom lisible : Printemps / Été / Automne / Hiver.

Constructeur	Effet principal
<code>GestionSaisons(int dureeParSaison = 13)</code>	<ul style="list-style-type: none">• Normalise le paramètre (<code>Math.Max(1, dureeParSaison)</code>) et le range dans <code>dureeParSaison</code>.• Tire aléatoirement <code>startIndex</code> (0 - 3) pour fixer la saison de départ.

Méthode & paramètres	Description du fonctionnement	<code>return</code>
<code>GetSaison(int semaine)</code>	<ul style="list-style-type: none">• Calcule combien de "blocs" de <code>dureeParSaison</code> se sont écoulés : <code>blocs = (semaine-1)/dureeParSaison</code>.• Ajoute ce décalage à <code>startIndex</code>, applique <code>% 4</code> pour boucler sur les 4 saisons.• Convertit l'index cyclique en libellé (Printemps / Été / Automne / Hiver).	Nom de la saison correspondant à la semaine demandée (<code>string</code>).

9. Partie

La classe `Partie` permet de gérer une partie de jeu en proposant toutes les méthodes nécessaires à l'avancement de chaque cycle de semaine. C'est `Partie` qui crée la première parcelle, applique la météo hebdomadaire, fait vieillir et gérer les plantes, calcule le passage des saisons, réinitialise/compte les actions disponibles et offre les services de haut niveau (planter, tout récolter, acheter de nouvelles parcelles). En d'autres termes, elle encapsule la logique du monde tandis que les autres classes (parcelles, plantes, menus) s'occupent d'unités plus petites,




ainsi le reste du code peut simplement interroger ou appeler Partie sans avoir à recoller manuellement tous ces morceaux de logique.

Propriété	Rôle / fonctionnement
<code>Joueur : Joueur?</code>	Le joueur qui possède la ou les parcelles de la partie (null tant qu'aucune partie n'est chargée / créée).
<code>Parcelles : List<Parcelle></code>	Collection des parcelles contrôlées par le joueur. L'élément à l'index 0 est celle créée au démarrage.
<code>Semaine : int</code>	Compteur global, commence à 1 et s'incrémente via <code>SemaineSuivante()</code> .
<code>gestionSaisons : GestionSaisons</code>	Gère le défilement des saisons (cycle de 13 semaines par défaut).
<code>SaisonActuelle : string</code>	Nom lisible de la saison courante (mis à jour à chaque changement de semaine).
<code>ParcelleEnCours : Parcelle?</code>	Parcelle actuellement affichée / contrôlée dans l'UI.
<code>ChoixTypeParcelle : MenuChoix</code>	Menu réutilisé pour demander le type de sol lors de la création d'une nouvelle parcelle.
<code>CoutSurface : int</code>	Tarif en "raisin-monnaie" par case lors de l'achat d'une nouvelle parcelle.
<code>rnd : Random (static)</code>	Générateur commun pour la météo.

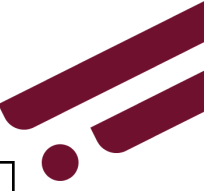


Constructeur	Effet principal
<code>Partie()</code>	Constructeur vide (nécessaire pour la désérialisation). Aucune logique.
<code>static</code> <code>CreerNouvellePartie(Joueur joueur)</code>	<ul style="list-style-type: none">• Instancie une <code>Partie</code>.• Crée un <code>GestionSaisons</code>, enregistre la saison de départ.• Demande à l'utilisateur le type de la première parcelle, puis appelle <code>InitialiserParcelles</code> pour la créer.• Lance un premier <code>AppliquerMeteo()</code> pour initialiser Température, Pluie, etc.• Retourne l'objet prêt à être utilisé.

Méthode & paramètres	Description du fonctionnement	Retour
<code>private Parcelle</code> <code>InitialiserParcelles(int typeDeParcelle)</code>	Demande à l'utilisateur : nom (≤ 30 car.), largeur & hauteur (4-10). Instancie la bonne sous-classe de <code>Parcelle</code> selon <code>typeDeParcelle</code> . Ajoute la parcelle à <code>Parcelles</code> et la définit comme <code>ParcelleEnCours</code> .	La nouvelle <code>Parcelle</code> .



<code>SemaineSuivante()</code>	Mise à jour de la semaine +1, On récupère la saison courante avec <code>SaisonActuelle</code> , réinitialise les actions de <code>ParcelleEnCours</code> , applique la météo à chaque parcelle, fait progresser chaque plante de chaque parcelle (hydratation, croissance, mortalité) et enlève celles qui meurent.	<code>void</code>
<code>Planter(Plante p, int y, int x)</code>	Si la case (y,x) de <code>ParcelleEnCours</code> est libre : on y installe la plante, diminue le nombre d'action. Sinon message d'erreur.	<code>void</code>
<code>ToutRécolter(Joueur j, Parcelle parcelle)</code>	Balaye la matrice : chaque plante à 100 % de croissance déclenche <code>Recolter</code> , donnant des raisins au joueur.	<code>void</code>
<code>AppliquerMeteo()</code>	Pour chaque parcelle : tire <code>Pluie</code> (30 %), on calcule l'ensoleillement (0 si pluie sinon valeur aléatoire entre 30-100), puis la température aussi aléatoirement selon <code>SaisonActuelle</code> en fonction des différents minimum et maximums, et stocke ces valeurs sur la parcelle (pour affichage & logique).	<code>void</code>




<code>private</code> <code>AcheterParcelle(Joueur</code> <code>joueur)</code>	Dialogue d'achat : type de sol → nom → dimensions. Calcule un coût (<code>largeur*hauteur*CoûtSurfac</code> <code>e</code>). Si le joueur a assez de raisins, on débite le joueur, on crée la parcelle, l'ajoute à la liste et la sélectionne comme étant la parcelle en cours.	<code>void</code>
<code>AfficherToutesLesParcelle</code> <code>s(Joueur joueur)</code>	Construit un menu listant chaque parcelle (nom, sol, taille) + option "Acheter". Si un index de parcelle est choisi, on la définit comme <code>ParcelleEnCours</code> , sinon on appelle <code>AcheterParcelle</code> .	<code>void</code>

10. AffichageChargement

Cette classe est

Propriété	Rôle / fonctionnement
<code>rnd : Random</code> <code>(static)</code>	Générateur aléatoire unique à la classe, sert au tirage de la probabilité de lancer un le <code>ModeUrgence</code> .

Constructeur	Effet principal
<i>(aucun constructeur public)</i>	La classe ne possède pas de constructeur exposé, l'unique champ <code>rnd</code> est initialisé via l'initialiseur <code>static</code> au chargement du type.

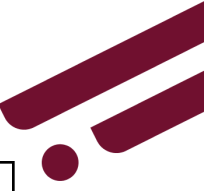


Méthode & paramètres	Description du fonctionnement	Return
<code>Afficher(int semaineActuelle, Parcelle parcelle)</code>	<ul style="list-style-type: none"> • Affiche une animation de chargement : • Vide la console, écrit “ ⌚ Passage à la semaine ...”, puis dessine une barre horizontale composée de “ ” qui grandit sur 40 pas ($40 \times 40 \text{ ms} \approx 1,6 \text{ s}$). • À la fin, tire un nombre aléatoire entre 0-100, avec 26 % de chances on lance <code>ModeUrgence.Lancer(parcelle)</code> pour déclencher le QTE lié aux oiseaux. • Termine par un message “✅ Semaine passée !” puis met en pause 1 seconde. 	<code>void</code>

11. ModeUrgence

Propriété	Rôle / fonctionnement
<code>rnd : Random (static)</code>	Génère les nombres aléatoires pour décider quelle flèche doit être appuyée et si le mode urgence est déclenché.
<code>Fleches : (ConsoleKey, string)[] (static)</code>	Tableau contenant les quatre flèches directionnelles associées à leur symbole affiché, utilisé pour générer les QTE de manière aléatoire.

Méthode & paramètres	Description du fonctionnement	Return
<code>Lancer(Parcelle parcelle, int rounds = 10, int barLength = 20, int speedMs = 30)</code>	Déclenche un mini-jeu (QTE) pour défendre une parcelle contre les oiseaux. À chaque round, une flèche aléatoire est demandée, et l'utilisateur doit la presser avant la fin d'un timer visuel en forme de barre décroissante. Si l'utilisateur réussit au moins la moitié des rounds, les récoltes sont préservées ; sinon, la croissance de toutes les plantes est divisée par 2. Les paramètres ont été ajustés pour que tout le monde puisse réussir assez	<code>void</code>



	facilement. De plus, la tolérance de 50% de réussite n'est pas très punitive.	
--	---	--

12. AffichageParcelle

Passons maintenant à la classe la plus importante pour notre projet, celle qui est responsable de la quasi-totalité des visuels ainsi qu'une partie des appels à toutes les méthodes des différentes classes que nous avons vu précédemment. Comme nous l'avons dit dans l'introduction, les visuels et l'ergonomie sont des points centraux dans notre projets, notre but est de faire oublier à l'utilisateur qu'il est dans un terminal de commande. C'est la raison pour laquelle nous allons nous attarder un peu plus que pour les restes des classes, et au vu de la complexité de certaines méthodes la forme des tableaux descriptifs ont quelque peu changer. Nous avons d'ailleurs découvert la fonctionnalité de `Console.SetCursorPosition(x,y)` en faisant quelques recherches pour pouvoir écrire à droite de notre grille par exemple de manière plus simple.

Propriété	Rôle / fonctionnement
<code>parcelle</code> : <code>Parcelle</code>	Référence à la parcelle actuellement affichée. Permet d'accéder à ses dimensions, plantes, météo, etc.
<code>menuAction</code> : <code>MenuChoix</code>	Instance du menu affiché pour les actions. Stockée pour réutilisation.
<code>joueur</code> : <code>Joueur</code>	Référence au joueur actif, utilisée notamment pour afficher les ressources ou effectuer des récoltes.
<code>partie</code> : <code>Partie</code>	Référence à l'objet <code>Partie</code> , contient le contexte global (semaine, saison, parcelles).



<code>jeu : Jeu</code>	Référence à l'objet principal <code>Jeu</code> , utile pour accéder aux fonctions de contrôle du déroulement du jeu.
------------------------	--

Constructeur	Effet principal
<code>AffichageParcelle(Parcelle parcelle, Joueur joueur, Partie partie, Jeu jeu)</code>	Initialise les 4 références principales (parcelle, joueur, partie, jeu) à partir des arguments fournis. Prépare l'objet pour l'affichage dynamique.

Méthodes :

<code>public void AfficherParcelleCurseur(...)</code>
<p>Description :</p> <p>Cette méthode est responsable de l'affichage graphique en console d'une parcelle de vigne. Elle construit ligne par ligne une représentation visuelle de la grille contenant les plantes, leur état de santé, ou les emplacements vides. L'affichage peut s'adapter selon deux modes :</p> <ul style="list-style-type: none">• mode plantation (avec curseur sur n'importe quelle case de la grille),• mode inspection (avec sélection de ligne avec curseur uniquement).
<p>Paramètres :</p> <ul style="list-style-type: none">• <code>ligneSelectionnee</code> : rangée actuellement sélectionnée (utile en navigation standard).• <code>lignes / colonnes</code> : dimensions de la parcelle.• <code>afficherEtat</code> : indique si l'on souhaite afficher l'état de santé des plantes (<code>true</code>) ou simplement leur présence (<code>false</code>).

- **planter** : si activé, on affiche un curseur de sélection pour planter une nouvelle vigne. Il est activé dans le mode pour planter.
- **caseSelectionnee** : coordonnées **[x, y]** de la case actuellement ciblée pour planter.

Fonctionnement :

1. Efface la console et affiche les informations générales (nom de la parcelle, semaine en cours, raisins du joueur).
2. Pour chaque case de la grille, on détermine :
 - Si elle est vide \Rightarrow on affiche le type de sol + flèche si sélectionnée. (Mode plantage pour les cases du milieu))
 - Si elle contient une plante :
 - en mode **afficherEtat** \Rightarrow on montre un emoji correspondant à son état (✅, 🦠, 💧, ⚠️). Cette variable est voué à varier toutes les secondes pour avoir un affichage dynamique. Cela sera abordé dans la classe Jeu.
 - sinon \Rightarrow on affiche un plant classique 🌱.
3. Flèche de ligne : si l'on n'est pas en mode plantation, une flèche latérale \leftarrow est affichée sur la ligne sélectionnée.

```
public void AfficherPlantage()
```

Description :

Cette méthode gère toute l'interface d'interaction utilisateur pour planter une nouvelle vigne (plante) dans une parcelle. Elle combine l'affichage visuel de la grille, des flèches de navigation, de la sélection de cépage et de la validation du placement.

Fonctionnement principal :

1. Récupération de la liste des cépages disponibles via `CataloguePlantes.GetToutes(...)`.
2. Initialisation des coordonnées de sélection dans la grille (`caseSelectionnee`) ainsi que de la plante sélectionnée (`indexPlanteSelectionnee`). On crée un booléen qui nous servira pour sortir de la boucle while.
3. Boucle interactive jusqu'à validation de l'action :
 - Affiche en continu :
 - la parcelle avec les icônes des plantes (via `AfficherParcelleCurseur`),
 - la liste des plantes disponibles à droite (via `AfficherPlantesDisponibles`),
 - les instructions de navigation et de sélection.
 - Intercepte les touches directionnelles pour déplacer la sélection sur la grille.
 - Intercepte les touches 1 à 5 pour choisir un cépage.
 - Valide le plantage avec Entrée, si une plante a été sélectionnée :
 - la plante est ajoutée à la grille via `partie.Planter(...)`,
 - une confirmation est affichée.

Particularité :

Afin de permettre un affichage dynamique et réactif de la grille pendant la phase de sélection de plantation, nous avons dû adapter la manière d'écouter les entrées clavier de l'utilisateur. Plutôt que de bloquer l'exécution avec une simple attente d'entrée (`Console.ReadKey()`), nous avons utilisé la méthode `Console.KeyAvailable` pour détecter les touches de manière non bloquante dans une boucle temporelle. Cette approche permet de rafraîchir l'affichage en

continu (toutes les ~100 ms), même en l'absence d'interaction, et d'assurer une expérience utilisateur fluide avec des effets visuels dynamiques, comme l'alternance de l'état des plantes ou la mise à jour instantanée du curseur de sélection.

```
public int? AfficherDetailRangee(int y)
```

Description :

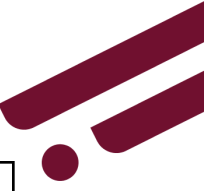
La méthode `AfficherDetailRangee` permet d'explorer en détail une rangée spécifique d'une parcelle dans le jeu. Elle représente visuellement l'évolution des plantes de la ligne sélectionnée à travers des paliers de croissance (100%, 75%, etc.) affichés en haut de l'écran sous forme d'icônes (🍇 ou 🌿). Elle affiche également l'état de santé de chaque plante (saine, malade, morte, etc.) et permet à l'utilisateur de sélectionner une plante à l'aide des flèches directionnelles, puis d'en consulter toutes les caractéristiques détaillées à droite de l'écran.

Paramètre :

Le seul paramètre de la méthode est `int y`, qui représente l'index de la ligne (rangée) de la parcelle à afficher. Cette ligne est parcourue colonne par colonne pour afficher la croissance et l'état de chaque plante. L'utilisateur pourra ensuite se déplacer latéralement pour sélectionner une colonne dans cette rangée.

Fonctionnement :

Une fois la méthode appelée, la console est nettoyée et une boucle d'interaction est lancée. Pour chaque palier de croissance (100, 75, 50, 25, 0), les plantes sont représentées en fonction de leur niveau de développement. Ensuite, après avoir affiché le sol, l'état de santé de chaque plante est affiché avec un emoji. L'utilisateur peut déplacer un curseur avec les touches ← et → pour sélectionner une plante, puis appuyer sur `Entrée` pour valider ou `Backspace` pour sortir de l'appel et revenir à la vue générale de la grille. Si une plante



est sélectionnée, ses informations détaillées (croissance, état, hydratation, adéquation avec l'environnement) sont affichées dynamiquement à droite de l'écran. La méthode retourne l'index de la colonne choisie ou `null` si l'utilisateur quitte. On se servira de la valeur de la colonne sélectionnée pour ouvrir un menu d'action pour le cépage en particulier.

```
public string AfficherMenuActionDetail(int x)
```

Description :

La méthode `AfficherMenuActionDetail` affiche un petit menu interactif permettant au joueur de choisir une action ciblée à effectuer sur une plante sélectionnée. Ce menu s'ouvre lorsqu'une cellule contenant une plante a été choisie dans l'interface détaillée de la parcelle. Il propose trois actions possibles : arroser, traiter ou récolter.

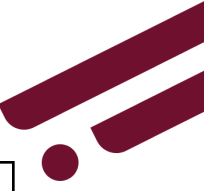
Fonctionnement :

La méthode instancie un objet `MenuChoix` avec la liste des actions possibles, puis appelle sa méthode `Afficher()` qui gère l'affichage en console et la navigation clavier. Une fois que le joueur valide son choix, l'index correspondant est utilisé pour retourner le libellé de l'action choisie sous forme de chaîne ("`Arroser`", "`Traiter`" ou "`Récolter`"). Cette chaîne sera ensuite utilisée pour déclencher l'action dans la logique de jeu principale.

```
public string AfficherMenuActionGeneral()
```

Description :

La méthode `AfficherMenuActionGeneral` permet d'afficher un menu interactif regroupant toutes les actions générales que le joueur peut effectuer pendant une semaine de jeu. Ce menu apparaît lorsque l'utilisateur appuie sur la touche Espace depuis l'interface de la parcelle. Il centralise les actions qui ne concernent pas une seule plante, mais toute la



parcelle ou l'état général de la partie.

Fonctionnement :

Une liste d'options est d'abord construite (planter, récolter, changer de parcelle, sauvegarder, passer la semaine ou revenir au menu principal). Cette liste est transmise à une instance de `MenuChoix` qui s'occupe de l'affichage console et de la navigation. L'index du choix sélectionné est ensuite utilisé dans un `switch` pour renvoyer le nom exact de l'action correspondante sous forme de chaîne de caractères. Cette chaîne est ensuite utilisée dans la boucle principale du jeu pour déclencher le bon traitement.

```
private void AfficherInfosParcelle(int x, int y)
```

Description :

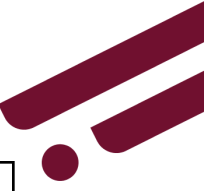
La méthode `AfficherInfosParcelle` permet d'afficher un encadré récapitulatif contenant les caractéristiques de la parcelle actuellement sélectionnée. Ces informations sont affichées à droite de la grille principale, afin de fournir au joueur un aperçu immédiat des conditions environnementales (ensoleillement, pluie, température, etc.) ainsi que de la saison et du nombre d'actions restantes.

Paramètre :

La méthode prend deux entiers : `x` et `y`, qui représentent les coordonnées dans la console où commencera l'affichage des informations. Cela permet de positionner dynamiquement la section d'information à un endroit fixe, en dehors de la grille de jeu.

Fonctionnement :

Le texte est affiché ligne par ligne à des positions fixes en fonction de `x` et `y`. Après un titre encadré et coloré en vert foncé, chaque ligne présente une donnée : type de sol, ensoleillement (sur 100), température actuelle (en degrés), pluie (oui/non), saison en cours et nombre d'actions disponibles (en jaune). Des `Console.SetCursorPosition()` sont



utilisés pour garantir un affichage clair et bien aligné. La méthode n'a pas de valeur de retour, elle effectue simplement une écriture dans la console.

```
private void AfficherPlantesDisponibles(int x, int y,  
List<Plante> plantes, int? selection)
```

Description :

La méthode `AfficherPlantesDisponibles` affiche dans la console la liste des cépages disponibles à la plantation. Elle est utilisée lors du choix d'une plante, notamment dans le menu de plantation, pour présenter les différentes options accessibles au joueur en fonction de la saison.

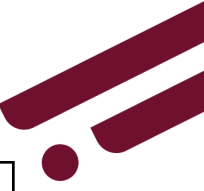
Paramètres :

Elle prend quatre paramètres :

- `x` et `y` : les coordonnées où débutera l'affichage (à droite de la grille en général) ;
- `plantes` : la liste des instances de `Plante` disponibles à planter ;
- `selection` : un index optionnel (`int?`) qui permet de surligner l'élément sélectionné (avec un fond gris) si une plante est déjà choisie

Fonctionnement :

La méthode commence par écrire le titre "🌱 Plantes disponibles :" en vert à la position (`x`, `y`). Ensuite, elle itère sur la liste de plantes en positionnant dynamiquement le curseur pour afficher chaque ligne à (`x`, `y + 2 + i`). Si l'indice courant correspond à l'élément sélectionné, l'entrée est mise en évidence avec un fond gris et une écriture noire. Chaque



ligne affiche l'indice (1-based) et le nom du cépage, aligné à droite pour garder une présentation homogène. Elle ne retourne rien, son effet est purement visuel.

13. Jeu

La classe `Jeu` constitue le point d'entrée du jeu viticole. Elle orchestre l'affichage du menu principal, le lancement d'une nouvelle partie ou le chargement d'une partie existante, et pilote ensuite la boucle principale de jeu. Elle assure également l'affichage des règles, la sauvegarde des données et permet de revenir au menu à tout moment.

Propriété	Type	Rôle / Fonctionnement
<code>isPartieEnCours</code>	<code>bool</code>	Contrôle si une partie est en cours ; empêche le retour prématuré au menu principal.
<code>enJeu</code>	<code>bool</code>	Active ou interrompt la boucle <code>BoucleJeu</code> .
<code>isChargement</code>	<code>bool</code>	Sert de flag temporaire pour gérer le passage à la semaine suivante.
<code>menuPrincipal</code>	<code>MenuChoix</code>	Menu d'accueil permettant de lancer une nouvelle partie, charger, lire les règles, etc.



Signature	Effet principal
<code>Jeu()</code>	Initialise le menu principal avec 4 options : Nouvelle Partie, Charger une partie, Règles et Quitter.

Méthode	Description	Return
<code>void Lancer()</code>	Affiche le menu principal via <code>menuPrincipal.Afficher()</code> et agit selon le choix sélectionné.	<code>void</code>
<code>void NouvellePartie()</code>	Demande un nom de sauvegarde, initialise le joueur et la partie, puis lance la boucle principale de jeu.	<code>void</code>
<code>void ChargerPartie()</code>	Liste les sauvegardes disponibles, permet d'en charger une, puis relance la boucle <code>BoucleJeu</code> .	<code>void</code>
<code>void AfficherRegles()</code>	Affiche les règles générales du jeu en console.	<code>void</code>
<code>void Quitter()</code>	Quitte proprement le programme avec un message de fin.	<code>void</code>
<code>void BoucleJeu(...)</code>	Gère le cœur du gameplay, semaine après semaine : météo, actions (planter, récolter, etc.), et menus.	<code>void</code>

