Rapport Jeu Vidéo

Collapse Brocoli

Réalisé par :

- **Halliche Tinhinane**
- **Sekri Lydia**
- **Oulebsir Mouna**
- **Belharet Ferhat**

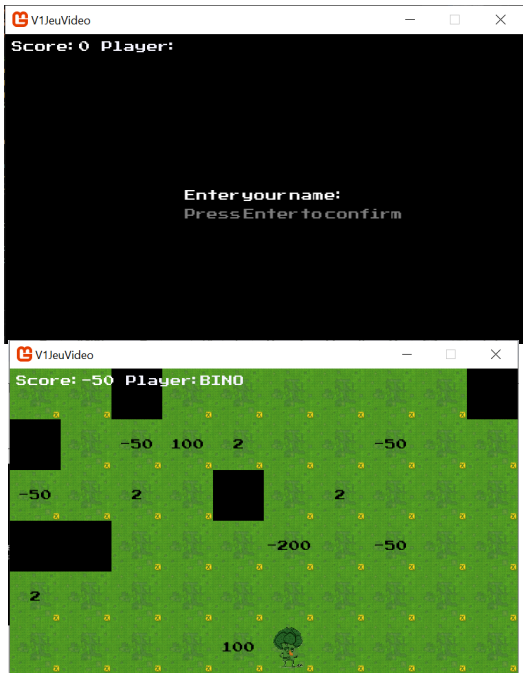
Encadré par : ***Nicolas Glade***

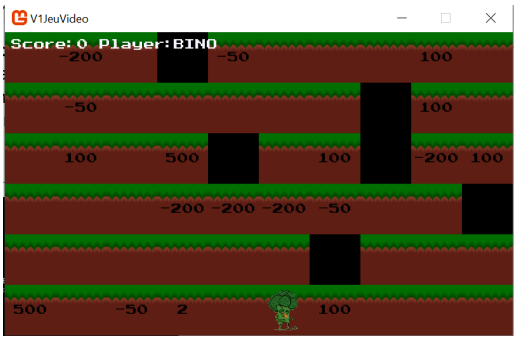
Année Universitaire : ***2024/2025***

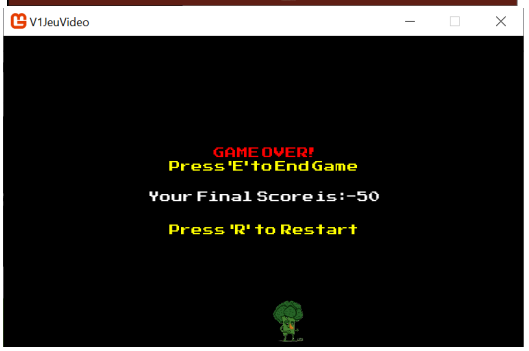
Sommaire :

1.Présentation du jeu :

Collapse Broccoli est un jeu où le joueur doit rester constamment en mouvement. Il progresse sur une grille, esquive les cases qui s'effondrent, collecte des points (positifs et négatifs), et traverse différentes plateformes, avec un score visible en temps réel.







2.Modélisation du projet :

Afin de diviser ce projets en classes on a opté pour cette Conception :

Class Game1: Représente le gestionnaire du jeu en se menant des méthodes pré faites lors de la création du projet Monogame

Class Player : Qui gère les assets de chaque player en terme de Texture , score , mouvement , et état du Player pendant la partie

Class platform: gère de son côté tout asset concernant les deux platforms du jeux et la disparition des cases aléatoires lors de la partie

Modélisation XML et XML SCHEMA : Afin de sauvegarder les scores du player sa position et son nom ainsi que des différents assets du jeu le XML est le parfait

format avec la bonne communication avec le jeu on peut charger les données de ce dernier mais aussi les mettre à jour selon chaque partie jouée

2.1 La classe Game1:

Nous avons utilisé une énumération essentielle pour notre jeu : l'énumération GameState, qui définit les différents états du jeu et facilite la gestion de la logique de jeu. Elle comprend :

- EnteringName : État où le joueur saisit son nom.
- Playing : État actif du jeu où le joueur interagit avec le gameplay.
- Transition : État de transition entre les plateformes ou niveaux.
- TransitionToGameOver : État de transition vers l'écran de "Game Over".
- GameOver : État final où le jeu est terminé, affichant les scores et les options de redémarrage.

Ensuite nous avons les différentes méthodes/fonctions:

Constructeur Game1(): Initialise les composants principaux du jeu, notamment le gestionnaire graphique (GraphicsDeviceManager) et la visibilité de la souris. Fournit une base pour configurer les paramètres graphiques et d'autres éléments avant le démarrage du jeu.

Méthode Initialize() : Configure les dimensions de la grille, initialise les plateformes, crée un joueur avec un score initialisé à zéro et définit l'état initial du jeu sur GameState.EnteringName.

Méthode LoadContent(): Charge le contenu graphique et audio du jeu. Elle crée une instance de SpriteBatch, appelle LoadGameContent() pour charger les ressources, et crée une texture noire pour les effets de transition.

Méthode LoadGameContent() :Charge dynamiquement les ressources nécessaires au jeu à partir d'un fichier XML. Cela inclut les textures, les polices, les sons, et les configurations des plateformes et du joueur. Elle initialise également les instances de Platform et Player avec les données chargées.

Méthode Update(GameTime gameTime) : Implémente une logique d'état (GameState) pour gérer les transitions entre différentes phases du jeu (saisie de nom, jeu en cours, transitions, game over), en fonction des entrées utilisateur

Méthode RestartGame() :Réinitialise le jeu en remettant le joueur et les plateformes à leur état initial, tout en réinitialisant le score dans le fichier XML.

Méthode ResetPlayerScore(): Réinitialise le score du joueur à zéro dans le fichier XML.

Méthode AskForPlayerName(KeyboardState keyboardState) : Permet au joueur de saisir son nom en utilisant le clavier.

Méthode DrawEnteringNameState(): Affiche l'écran de saisie du nom avec des instructions.

Méthode HandlePlayingState(GameTime gameTime, KeyboardState keyboardState) : Gère la logique du jeu pendant l'état de jeu actif. Cela inclut la gestion des mouvements du joueur, l'effondrement des cellules de la plateforme, et la vérification des collisions pour déterminer si le joueur est toujours sur une cellule intacte. Si le joueur tombe, elle change l'état du jeu en conséquence.

Méthode HandleTransitionState(GameTime gameTime) : Gère la transition entre les plateformes, en modifiant progressivement l'alpha pour des effets visuels.

Méthode Draw(GameTime gameTime): Dessine les éléments à l'écran en fonction de l'état actuel du jeu. Cette méthode gère l'affichage de l'écran de saisie du nom, du jeu en cours, des transitions, et de l'écran de "Game Over". Elle affiche également le score et le nom du joueur.

Méthode LoadSecondPlatform(): Charge la deuxième plateforme en initialisant une nouvelle instance de Platform avec les textures appropriées. Elle remet également la position du joueur à un emplacement par défaut sur la nouvelle plateforme.

Méthode SaveGameContent(): Sauvegarde l'état actuel du jeu dans un fichier XML. Cette méthode charge le document XML, trouve l'élément du joueur par son nom, et met à jour ou crée les éléments nécessaires pour enregistrer le score et la position initiale du joueur. Elle gère également les exceptions potentielles lors de la sauvegarde.

Méthode HandleTransitionToGameOver(GameTime gameTime)

Gère la transition vers l'état "Game Over". Cette méthode augmente progressivement la valeur de l'alpha de la transition en fonction du temps écoulé. Lorsque l'alpha atteint 1, elle joue un son de fin de round, sauvegarde l'état actuel du jeu et met à jour l'état du jeu pour le passer à "Game Over".

On trouve également des fonctions liées à la validation XML et à la transformation XSLT, que nous détaillerons un peu plus loin.

2.2 La Classe Player :

Nous avons une énumération **PlayerState ,** qui définit les états possible du joueur :

- Active : Le joueur est en jeu et visible.
- Invisible : Le joueur est en train de disparaître.
- Dead : Le joueur est mort et n'est plus visible.

Ensuite nous avons les différentes méthodes/fonctions:

Constructeur Player() : permets l'instanciation d'un joueur avec son nom sa position initiale son image / Texture et enfin son score initialisé à 0

Méthode Move() : Gère le déplacement du joueur sur la plateforme en fonction des touches pressées (haut, bas, gauche, droite). Le joueur se déplace d'une case à la fois, pendant tout le temps du jeu et sans dépasser les limites de la grille

Méthode CollectScoresOnPath() : Vérifie si la cellule actuelle ou le joueur est positionné contient un score. Si c'est le cas, elle l'ajoute au score total du joueur et le retire de la plateforme.

Méthode Draw() : Calcule la position du joueur en pixel et permets de le dessiner sur la fenêtre du jeu

Méthode Update() : permets de mettre à jour l'état du joueur pendant la partie selon l'énumération PlayerState , par exemple : Si le joueur est invisible, elle diminue son opacité et, une fois qu'elle atteint zéro, change l'état du joueur en "Dead" et le rend invisible.

2.3 La Classe Plateform :

Constructeur Plateform() : permet l'instanciation d'une plateforme avec une largeur/hauteur de grille, une taille de cellule, et une texture pour les cellules. Il initialise également les listes de cellules intactes et leurs points.
Méthode `InitializeCells()`: Crée toutes les cellules de la grille et attribue aléatoirement des scores à certaines d'entre elles. Les scores sont générés par la méthode GenerateRandomScore().

Méthode `GenerateRandomScore()`: Génère un score aléatoire parmi un ensemble de valeurs prédéfinies (+100, +500, *2, -50, -200).

Méthode `Draw()`: Dessine chaque cellule de la plateforme sur l'écran, en affichant également le score associé à chaque cellule, centré dans la cellule.

Méthode `CollapseRandomCells()`: Retourne le score d'une cellule donnée. Si la cellule a été effondrée, elle retourne 0.

Méthode `CollectScore()`: Marque une cellule comme "récoltée" en supprimant son score du dictionnaire des scores.

3. Modélisation XML et XML SCHEMA:

Méthode `ValidateXmlFileAsync()`: Valide un fichier XML par rapport à un schéma XSD de manière asynchrone. Elle configure les paramètres de validation, lit le fichier XML, et signale les erreurs ou avertissements via un gestionnaire d'événements.

Méthode `ValidationCallBack()`:Gère les événements de validation en affichant des messages d'avertissement ou d'erreur selon la sévérité de l'événement.

Méthode `XslTransform()`: Transforme un fichier XML en utilisant un fichier XSLT, produisant un fichier de sortie. Elle gère les exceptions et affiche un message de validation en cas de succès ou d'erreur.

Méthode Serialization<T>(T obj, string outputFilePath) : Sérialise un objet de type `T` en un fichier XML spécifié par `outputFilePath`. Elle utilise `XmlSerializer` pour effectuer la conversion et écrit le résultat dans le fichier. En cas de succès, un message de confirmation est affiché ; en cas d'erreur, un message d'erreur est affiché, indiquant la nature du problème rencontré lors de la sérialisation.

Méthode Deserialization<T>(string xmlFilePath) : Désérialise un fichier XML spécifié par `xmlFilePath` en un objet de type `T`. Elle lit le contenu du fichier à l'aide de `XmlSerializer` et retourne l'objet résultant. Si une erreur se produit pendant le processus, un message d'erreur est affiché et l'exception est relancée pour permettre une gestion appropriée de l'erreur par l'appelant.

Ces deux dernières méthodes ont été utilisées pour sauvegarder les données du jeu, et voici l'appel effectué dans Program.cs:

```
using V1JeuVideo;
using System.Threading.Tasks;
using System.Xml.Serialization;

class Program
{
    static async Task Main(string[] args)
    {
        string xml = "Content/GameConfig.xml";
        string xslt = "Content/GameConfig.xsl";
        string html = "Content/GameConfig.html";
        string xsd = "Content/GameConfig.xsd";


        Game1.XslTransform(xml, xslt, html);
        await Game1.ValidateXmlFileAsync(xsd, xml);

        string xsltBest = "Content/BestScore.xsl";
        string htmlBest = "Content/BestScore.html";
        Game1.XslTransform(xml, xsltBest, htmlBest);

        using var game = new Game1();
        game.Run();
        Game1.Serialization(game, @"Content/GameConfig.xml");
        Game1.Deserialization<Game1>(@"Content/GameConfig.xml");
    }
}
```

Étant donné que les fichiers XML, XML Schema et XSLT ont été ajoutés au dossier Content, voici leurs usage :

Le fichier XML GameConfig.xml est structuré pour contenir toutes les informations nécessaires à la configuration et au fonctionnement du jeu.

un élément racine <GameContent> qui encapsule toutes les données de configuration. La section <Textures> définit les textures utilisées dans le jeu, avec des éléments <Texture> spécifiant un attribut Name pour identifier chaque texture et un attribut Path pour le chemin d'accès à la ressource. De même, la section <Fonts> contient des éléments <Font> qui définissent les polices utilisées, tandis que la section <Sounds> inclut des éléments <Sound> pour les effets sonores, chacun ayant des attributs Name et Path.

Les paramètres de configuration du jeu sont regroupés dans la section <GameSettings>, qui inclut la largeur de la grille (GridWidth), la hauteur de la grille (GridHeight), et la taille des cellules (CellSize).

La section <Platforms> définit les plateformes du jeu à l'aide d'éléments <Platform>, chacun ayant un attribut Name et un attribut Texture pour spécifier la texture à utiliser.

Enfin, la section <Player> contient des informations sur le joueur, y compris son nom, son score actuel, la texture à utiliser, et sa position initiale sur la grille, définie par l'élément <InitialPosition>.

Exemple pour un seul joueur :

```
<Player>
  <Name> DINO</Name>
  <scores>
    <Score>1650</Score>
    <Score>2204</Score>
    <Score>2612</Score>
  </scores>
  <InitialPosition X="2" Y="4" />
</Player>
```

Exemple pour plusieurs joueurs:

```
<Player>
  <Name> DINO</Name>
  <scores>
    <Score>1650</Score>
    <Score>2204</Score>
    <Score>2612</Score>
  </scores>
  <InitialPosition X="2" Y="4" />
</Player>
<Player>
  <Name> ZAY</Name>
  <scores>
    <Score>2904</Score>
  </scores>
  <InitialPosition X="1" Y="0" />
</Player>
```

On a également défini un fichier GameConfig.xsd pour garantir que le fichier de configuration du jeu respecte une structure définie, facilitant ainsi la validation et l'intégrité des données lors du chargement et de la sauvegarde des informations du jeu.

Pour générer les pages HTML, nous avons créé deux feuilles de transformation XSLT : *GameConfig.xsl* transforme un document XML contenant des informations sur les scores des joueurs en une page HTML formatée. Pour chaque joueur, la page affiche son nom ainsi qu'une liste de ses scores, triés par ordre décroissant. La deuxième feuille, *BestScore.xsl*, transforme un document XML contenant des informations sur les joueurs et leurs scores en une page HTML présentant un classement des meilleurs scores. La page affiche un titre principal "Classement des meilleurs scores" et organise les données dans un tableau. Pour chaque joueur, le fichier extrait son nom et son meilleur score, déterminé en triant les scores par ordre décroissant. Ces transformations sont effectuées via l'appel à la méthode *xsltTransform()*.

4. Conclusion:

En résumé, **Collapse Brocoli** est un projet dont la structure est centrée autour de trois classes principales (Game1, Player et Platform), permet de séparer clairement les responsabilités et rend le code facile à maintenir.

L'intégration d'un système d'états de jeu (GameState) assure une gestion fluide de l'expérience utilisateur, depuis l'entrée du nom jusqu'à l'écran de fin de partie. La classe **Player** gère efficacement les interactions et l'état du personnage, tandis que la classe **Platform** dynamise le gameplay avec la gestion des cases qui s'effondrent et du système de points.

Une des grandes forces du projet réside dans l'utilisation des technologies XML. La modélisation des données en XML, associée à la validation via XSD et aux transformations XSLT, permet de garantir une sauvegarde fiable des données de jeu et la génération automatique des tableaux des scores en HTML. Cette approche assure l'intégrité des données tout en offrant une présentation claire des performances des joueurs.

Les mécanismes de sérialisation et de désérialisation permettent une gestion efficace de la persistance des données entre les sessions de jeu, tandis que les transformations XSLT facilitent une présentation ergonomique et accessible des scores.

Dans l'ensemble, ce projet nous a permis d’exploiter/appliquer les divers concepts de la programmation orientée objet c# et des technologies XML, tout en offrant une expérience de jeu fluide et agréable.