Musicz - un jeu d'automation

Rapport

Description du projet

Objectifs

Créer un jeu d'automation (comme. Factorio, Satisfactory, shapez.io, etc.) dans le thème de la musique. Le joueur doit livrer des notes individuelles, puis des accords et enfin des suites de notes pour créer des courtes mélodies.

Problématique

Vous avez un voisin, M. usic qui est un voisin très bruyant. Il joue de la musique toute la journée et toute la nuit. Vous avez décidé de lui rendre la pareille en créant un jeu d'automation musical. Vous devez créer des mélodies pour le rendre fou.

Puisqu'il s'agit d'un jeu vidéo à titre de divertissement, même si l'on ne peut exclure une charge pédagogique potentielle au niveau de l'aprentissage basique du solfège, il n'y a pas de problématique réelle que nous tentons de résoudre, si ce n'est celle fictive décrite plus haut. Le but du jeu est d'offrir une expérience unique, celle de créer une usine à bruit, progressivement devenant plus complexe, à la manière des jeux d'automations.

Exigences fonctionnelles

- Le joueur doit recevoir des objectifs spécifiques à accomplir pour pouvoir progresser dans le jeu.
- Le jeu doit avoir une interface fonctionnelle à la souris.
- Le jeu doit posséder du son.
- Le jeu doit contenir des instructions, sous la forme d'un texte explicatif ou autre.
- Le joueur peut placer des éléments de production sur une grille
  - Générateur de notes
  - Changement de pitch (fréquence), monte ou descend la note selon une gamme donnée
  - Changement de tempo (rythme), accélère ou ralentit la note, croche, noire, blanche, etc
  - Instrument, altère la note selon un instrument donné
  - Combineur de notes, produit des accords qui sont ensuite considérés comme une seule note
  - Combineur de notes alternés, produit une séquence
  - Sortie haut parleur, crée le son
- Le joueur peut supprimer des éléments de production existants

Exigences non-fonctionnelles

- Le jeu doit avoir des performances acceptable. 30-60fps minimum.
- Le jeu doit être compatible avec différents systèmes d'exploitation. Windows et Linux.
- Le jeu sera crée avec Godot 4.x.
- Le jeu sera crée avec .NET et donc C# comme language.
- Le joueur ne doit pas perdre sa progression lors de l'exécution du jeu.

Architecture préliminaire

Comme notre projet est constitué d'une application qui tourne uniquemenent en local, et n'a rien de connecté, notre architecture est composée du pipeline CI/CD de tests pour valider le fonctionnement

du jeu lors d'ajouts de nouvelles fonctionalités, ainsi que celui de build pour créer les différents fichiers de build lors de la création d'un tag pour faire une release.

En raison de la nature du développement de jeu vidéo ainsi que de notre manque d'expérience avec le moteur Godot, nous itérerons rapidement sur les fonctionnalités afin d'avoir un prototype très rapidement puis de les raffiner au fur et à mesure.

Patterns utilisés

En ce qui concerne le jeu, le pattern du singleton sera utilisé sous la forme d'un `GameManager` présent de manière statique et unique dans chaque scène. N'importe quel script pourra l'accéder et disposer de méthodes utiles pour la gestion des scènes, des paramètres, des sons joués, etc. Toute tâche trop complexe pour le `GameManager` sera déléguée à un manager dédié, tel que le `ProgressionManager`, `AudioManager`, etc. Cette utilisation du pattern permet une grande flexibilité, nécessaire pour un projet aussi court, au coût d'une plus grande exposure des propriétés du `GameManager`.

Nous utilisons également le pattern du visiteur afin que chaque objet note consulte la place sur laquelle elle se trouve et exécute une action en fonction de celle-ci; Si elle se trouve sur une case `Speaker`, elle est jouée, si elle se trouve sur une case `Transit`, elle est déplacée pourvu que la case suivante soit libre, avec une méthode en callback lors du passage sur la case.

Un autre pattern utilisé fréquemment est celui de l'observateur, que ce soit au travers d'événements Godot, appelés signaux, ou par nos propres méthodes. Ceci permet un code bien plus élégant s'adaptant assez bien à notre volonté d'animer l'entiereté du jeu à un rythme donné, ce rythme s'accélérant selon la montée en niveaux.

Choix musicaux

Nous avons choisi la notation américaine pour la durée des notes : `Whole`, `Half` `Quarter` etc. afin de gagner en clarté, notation également utilisée dans le vieux continent sous la forme demi-ton, quart de ton, etc.

Pour le pitch des notes, leur hauteur, nous utilisons la notation sous forme de simple lettre, allant de A pour Do jusqu'à G pour Si, au contraire de la notation allemande qui substitue H à B. Ceci permet une grande simplicité d'écriture tout en ayant un but pédagogique, le joueur apprenant à reconnaître les notes de musique.

Dans la recherche d'une manière élégante de jouer des notes, nous nous sommes renseignés sur des addons Godot permettant de jouer des fichiers MIDI, mais après plusieurs essais infructueux, nous avons choisi un autre addon permettant de jouer des samples, des fichiers WAV, ce qui est moins commode d'un point de vu programmeur mais en raison du court délai, le résultat est plus qu'agréable, si l'on peut tolérer la répétition des notes jouées.

Choix Godot

Nous avons initialement décidé de partir sur un système de `Tilemap`, outil très puissant permettant de dessiner des éléments selon un quadrillage strictement donné, que ce soit des carrés, des hexagones, etc. Mais après une première implémentation et malentendu quant aux capabilités de cet outil, nous avons dû changer sur une solution hybride, afin de contenir les informations fixes du jeu, informations ne pouvant être contenues directement via les couches de données custom de la `Tilemap`.

En raison des contrôles choisis, par la souris, nous avons dû réfléchir à la manière dont les éléments d'UI viendraient à être interagir avec le joueur. Entre les boutons d'outils, débloqués au fur et à mesure de la progression, du label affichant la progression actuelle et l'objectif à réaliser, mis à jour par le `ProgressionManager`, et un simple bouton permettant d'accéder au menu, qui sert d'instruction et possède également un bouton pour quitter le jeu. Utiliser la large palette de classes d'UI de Godot était

un choix évident, même si ceci demande un temps d'aprentissage, leur complexité étant très grande pour un projet aussi court.

Un autre point majeur du développement a été l'utilisation de ressources communes, via la classe `Resource` de Godot. En effet entre les boutons d'UI, le menu d'instruction et les cases placées, il existe une certaine redondance d'information, ceci est également valable pour faire coincider une note à une durée donnée qui correspond à un sprite représentant cette donnée, qui est également représentable sous forme de note.

Mockups

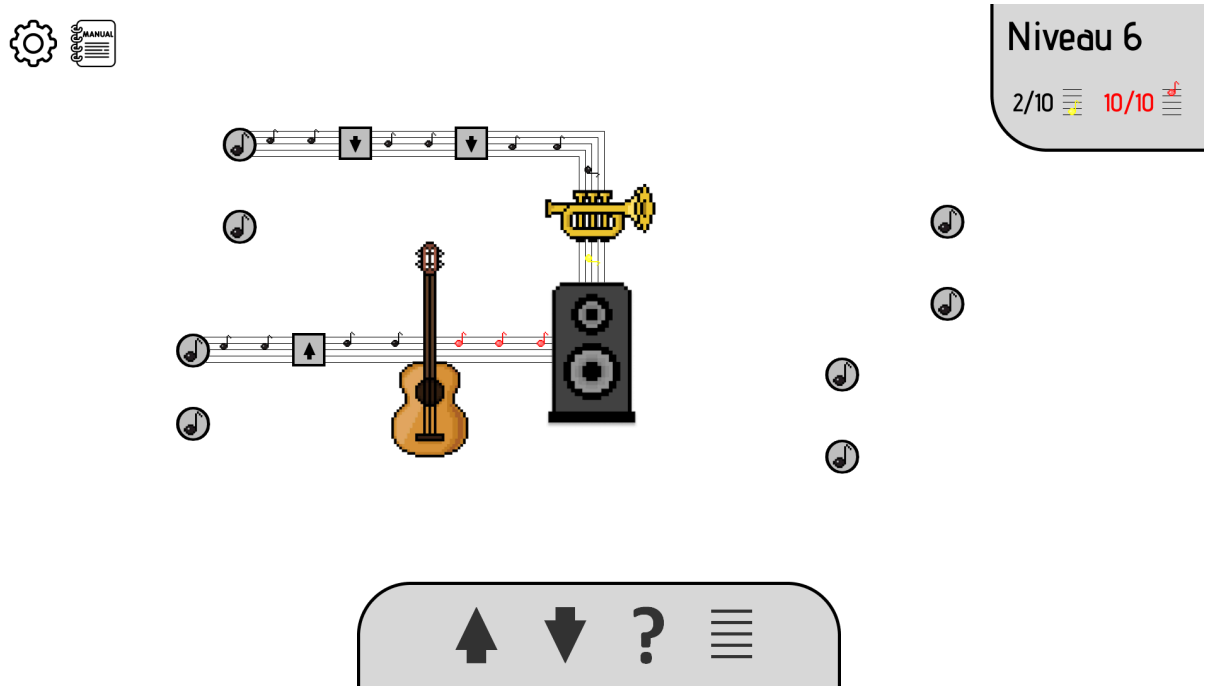

Figure 1: Mockup du jeu

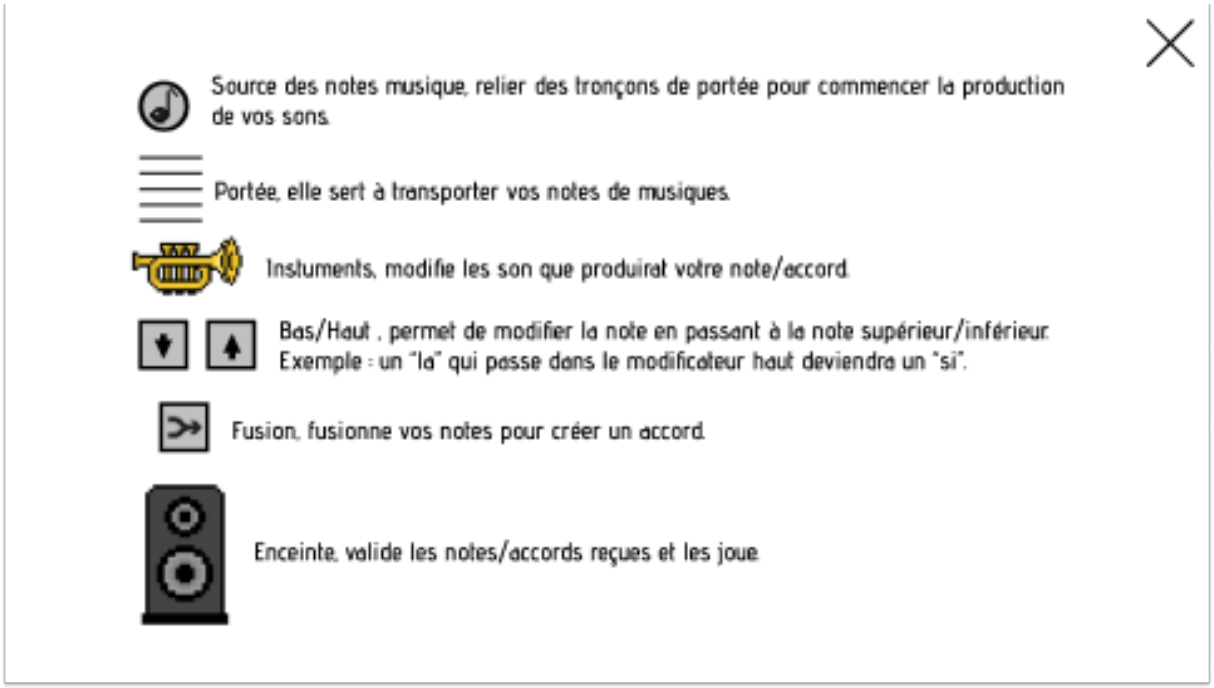

Figure 2: Mockup de la page explicatif

Description des choix techniques

Bases techniques du jeu

Le choix principal que nous avons fait est celui du moteur de jeu à utiliser. Les 3 options que nous avons envisagé sont Unity, Godot et PixiJS. Godot a été retenu pour plusieurs raisons: Premièrement, il est plus facile à intégrer à un pipeline CI/CD que Unity, principalement du au fait que ce dernier a une solutionn propriétaire payante, et au fait que Godot est open-source. Godot est également un des moteurs de jeu le plus populaire en ce moment pour les, en partie à cause du fiasco récent de marketing

de Unity. PixiJS était une solution envisagée et intéressante, mais comme il s'agit d'un moteur de jeu beaucoup plus léger et moins connu nous avons préféré rester avec Godot.

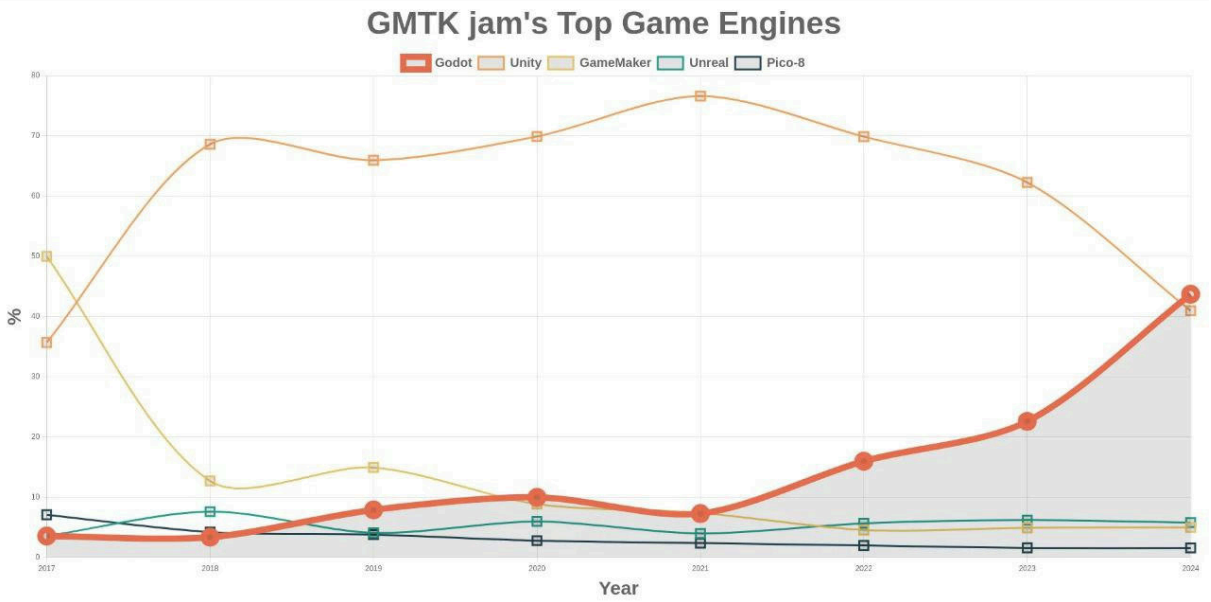

Figure 3: approximation du pourcentage de game engines utilisés lors de la GMTK game jam

Godot nous permet d'utiliser d'utiliser de nombreux outils pour la mise en place d'un jeu en 2D et nous permet facilement d'exporter le jeu final pour plusieures plateformes. Au sein de Godot, nous avons décidé d'utiliser C# .Net 8.0 au lieu du gdScript, langage propriétaire de Godot. Ce choix a été fait en partie par réticence d'apprendre un langage avec une seule utilité, et en partie à cause de la ressemblance du C# avec Java que nous avons déjà dù utiliser dans le cadre cursus à la HEIG, assurant une base solide. La choix de la version de .Net est pour avoir la compatibilité avec Godot 4.x qui est la version la plus récente.

Nous avons trouvé une librairie de tests unitaires pour Godot qui supporte le gdScript ainsi que le C# qui s'appelle gdUnit4 et avons décidé de l'intégrer à notre processus de travail, pour valider le fonctionnement du jeu lors d'une Pull request.

Outils utilisés

L'outil que nous avons choisi pour créer le rapport, ainsi que tout autre documentation requise est Typst, en raison des possibilités de mise en page qu'il offre, sa relative simplicité d'utilisation, ainsi que sa familiarité avec certains membres de l'équipe. Pour l'édition et la compilation de ces documents, nous utilisons VSCode équipé de l'extension tinymist Typst.

Pour la création des mockups nous avons décidé de suivre la recommendation faite dans le cadre de ce cours et d'utiliser Figma.

Le développement est fait sur l'éditeur de Godot pour le ore du jeu, ainsi que Rider pour l'édition des scripts en raison de l'habitude et de la facilité de prise en main d'utiliser les outils de JetBrains.

Dernièrement, notre landing page est faite avec GitHub pages.

Description du processus de travail

Planning

Un github project a été créé pour ce projet. Il dispose des catégories suivantes :

- Backlog : ensemble de tâches à faire pour le projet
- Ready : tâches prêtes à être assignées, non dépendantes d'autres tâches
- In progress : tâches en cours de réalisation. À ce moment là une issue github est créee et assignée à la personne en charge d'un tâche donnée.
- In review : tâches terminées et en attente de review
- Done : tâches terminées et validées

En plus de ces catégories nous avons catégorisé les tâches en fonction de leur importance pour le projet :

- required : Tâche requise par le projet
- optional : Tâche non requise par le projet mais apporterait une plus value indéniable
- nice to have : Tâche optionnelle ayant d'avantage trait à l'expérience de l'utilisateur plus que les fonctionnalités, des signes et des feedbacks, principalement.
- bug : Il s'agit d'un bug à investiguer et/ou à corriger
- refactor : Tâche de refactoring pour l'amélioration future du projet

Git workflow

- Organisation du git: branche `main` avec le workflow pour créer un build
- Les PR ouvertes lancent des teste unitaires qui doivent passer pour pouvoir merge la PR
- Branches de features éventuellement suffixées avec les initiales de la personne en charge
- Branche `pages` pour la landing page, hébergée sur Github Pages
- Les branches mergées et non utilisées sont supprimées

CI/CD

- Des tests unitaires sont lancés lors de la creation d'une PR et bloquent le merge si ils ne passent pas
- Lorsqu'un tag est créé, un build est lancé et une release qui contient la version linux et windows du jeu est créée

Améliorations futures

Des prérequis initialement discutés, un seul n'a pu être réalisé à temps, celui de la combinaison de différentes notes sous forme de séquence, en raison de la complexité d'implémentation de celui-ci.

Un refactoring du code serait également bienvenu pour s'assurer de la versatilité de celui-ci pour des futures modifications, telles que celle mentionnée ci-dessus.

En terme d'ajouts pur et durs, des autres instruments, une plus grande variété de notes, des meilleurs effets visuels et sonores seraient tout aussi bien accueillis et pourraient facilement être ajoutés à l'architecture existante, ne demandant que des modifications mineures.

Finalement, l'expérience utilisateur reste à travailler, expérience au coeur du développement de jeu vidéo. Mieux expliciter les instructions, ce qui est demandé, la manière dont les objets interagissent entre eux demanderaient un temps bien plus considérable que celui alloué pour ce projet.