Bourquin Olin Christopher, Demont Kilian, Guggisberg Simon,
Troeltsch Jonas

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.

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 conctitué 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 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.

Mockups

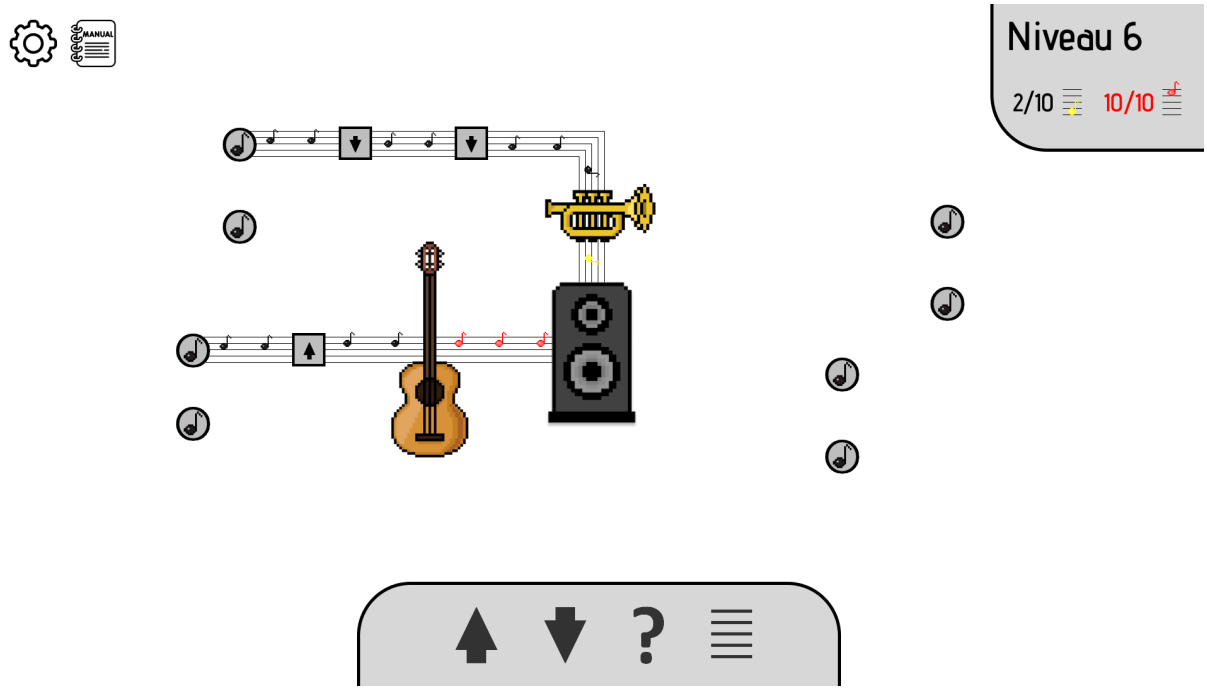

Figure 1: Mockup du jeu

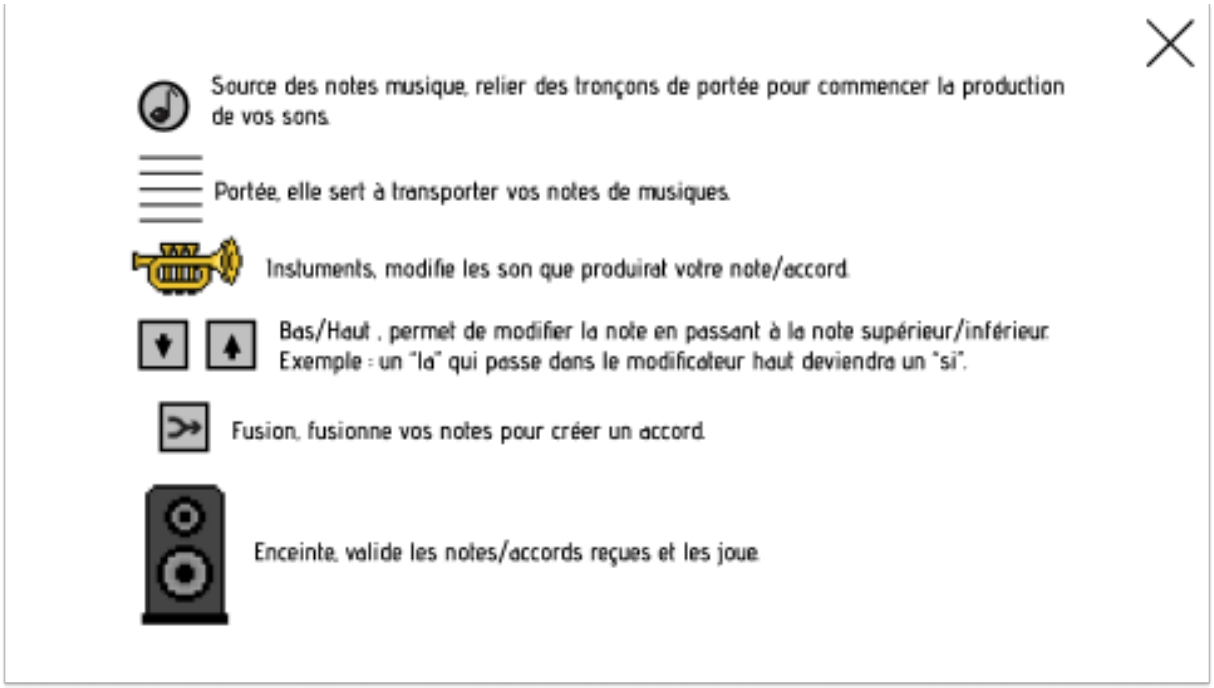

Figure 2: Mockup de la page explicatif

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.

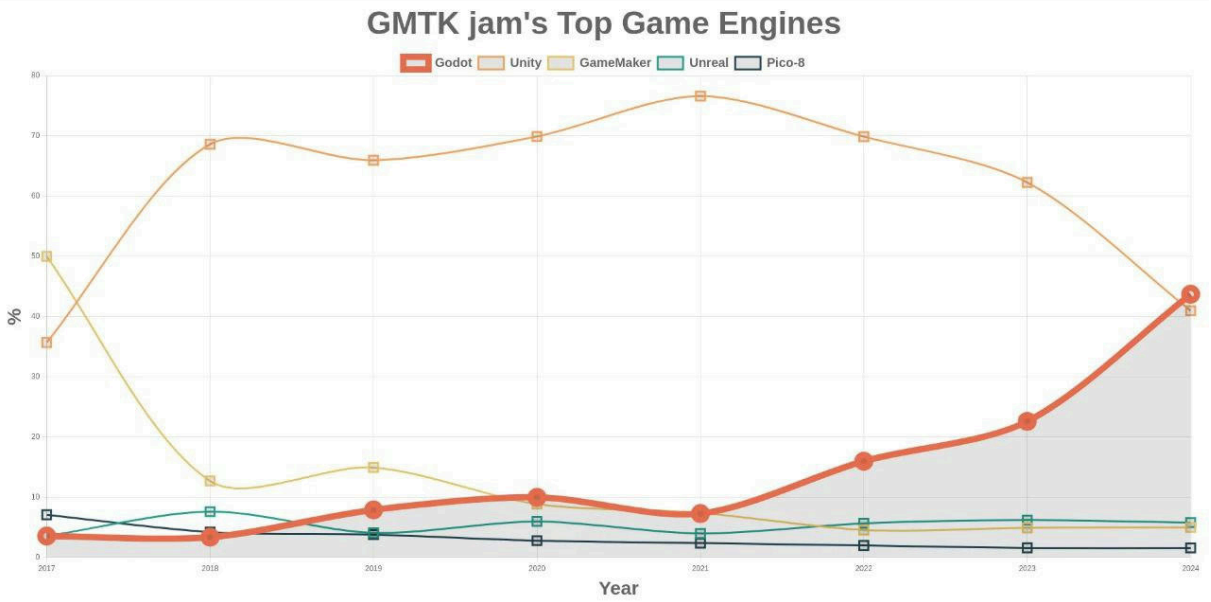

Figure 3: approximation du pourcentage de game engines utilisés lors de la GMTK game jam

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 familiarité du C# avec Java que nous avons déjà dù utiliser dans le cadre de notre formation. 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 à cause de l'habitude que nous avons d'utiliser les outils JetBrains.

Dernièrement, notre landing page est faite avec GitHub pages.

Description du processus de travail

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
- Branches personnelles de test avec les initiales de la personne en charge

- Branche `pages` pour la landing page, hébergée sur Github Pages

CI/CD

- 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