

Projet Cartes Poker

Conception Agile

Université Lumière Lyon 2

Larmat Jean Pierre
Pierre Gatien
2024



Présentation du projet

Le Planning Poker est une technique de gestion agile utilisée pour estimer la complexité d'un ensemble de tâches (appelée Backlog) dans un projet qui utilise la méthode Scrum, notamment en développement logiciel. Cet outil collaboratif repose sur la participation des membres d'une équipe pour favoriser des estimations fiables.

Le projet consiste en la création d'une application interactive, un jeu qui reproduit les caractéristiques et les règles du Planning Poker. Ce jeu est destiné à des équipes de développement logiciel. Le jeu sera multijoueur en local, permettant ainsi aux membres de l'équipe de participer ensemble autour d'une même table, tout en manipulant les interactions via une souris.

Motivation des choix

La motivation principale derrière ce projet est de rendre le processus de Planning Poker plus engageant, interactif et amusant pour les équipes qui l'utilisent. En permettant des interactions locales et en intégrant des fonctionnalités telles que la personnalisation des modes de jeu et la gestion des fichiers JSON via la sérialisation et la désérialisation, nous cherchons à améliorer l'expérience utilisateur.

Pour le développement de cette application, nous avons choisi d'utiliser Unity, et par conséquent le langage C#. Unity est un moteur de jeu puissant et largement utilisé dans l'industrie du jeu vidéo, offrant une excellente base pour des projets de développement interactifs comme celui-ci. De plus Unity, étant un moteur de jeu extrêmement populaire dans le domaine du développement de jeux vidéo, nous offre une excellente opportunité d'apprendre les bases et de nous entraîner à utiliser ce logiciel ce qui nous permet d'acquérir une expérience pratique en manipulation de moteurs de jeu, compétence essentielle pour notre futur parcours académique au sein du programme GAMAGORA et notre intérêt pour le développement de jeux vidéo.

Présentation Moteur de Jeu Unity

Unity est une plateforme de développement de logiciels interactifs largement utilisée par les développeurs du monde entier. Un moteur de jeux est un logiciel conçu pour simplifier et accélérer le processus de développement de jeux vidéo. Les moteurs de jeux fournissent généralement des fonctionnalités de base telles que la gestion des graphiques, de la physique, de l'audio, des entrées utilisateur, de l'intelligence artificielle et des animations. Cela permet aux développeurs de se concentrer sur la conception du jeu lui-même plutôt que sur le développement de systèmes fondamentaux.

Unity offre un ensemble d'outils et de fonctionnalités prédéfinies permettant aux développeurs de créer des jeux sans avoir à construire chaque aspect du jeu à partir de zéro.

Au cœur de Unity se trouve le concept de GameObjects, des éléments conteneurs fondamentaux qui composent le monde du jeu. Ces GameObjects peuvent être des objets

physiques, des personnages, des décors ou tout autre élément interactif. Ils sont enrichis par l'ajout de composantes, qui définissent leur comportement, leur apparence et leurs interactions avec l'environnement du jeu.

Les scénarios, ou scènes, sont des environnements dans lesquels ces GameObjects interagissent. Les développeurs utilisent Unity pour concevoir ces scènes en plaçant et en configurant les GameObjects, en définissant les interactions et les règles du jeu.

Unity facilite le développement grâce à une interface intuitive, ainsi qu'une vaste quantité de fonctionnalités prêtes à l'emploi, telles que la physique, les graphismes,, la gestion des animations, le son, la gestion des entrées utilisateur et bien plus encore. De plus, Unity prend en charge plusieurs plateformes de déploiement, permettant aux développeurs de créer des jeux pour PC, appareils mobiles, etc, le tout à partir d'un seul projet.

Scripting en Unity

Pour contrôler le comportement des GameObjects et créer des interactions dynamiques, les développeurs utilisent des scripts écrits en C#. C# est un langage de programmation Orienté Objet inspiré de Java développé par Microsoft, principalement utilisé pour le développement d'applications dans le framework .NET. Dans Unity, les scripts en C# sont attachés aux GameObjects et sont exécutés à différents moments pendant le déroulement du jeu, permettant ainsi aux développeurs de manipuler plus précisément le comportement de leurs jeux et pas dépendre que des fonctionnalités prédefinies du logiciel.

Ainsi, la gestion des scripts et comportement des objets repose sur un cycle de vie précis, fourni par la classe MonoBehaviour. Ce cycle est structuré autour de plusieurs méthodes prédefinies, qui permettent d'initier, de contrôler des actions chaque frame et de configurer les interactions entre les composants

Fonctionnalités à tenir en compte

Durant le développement de notre application on a utilisé plusieurs méthodes et fonctionnalités spéciales qui faut comprendre pour interpréter le code.

Méthodes Monobehaviour

Les principales méthodes utilisées dans ce cycle incluent Awake(), Start(), Update(), OnEnable(), OnDisable() entre autres.

Ces fonctions clés permettent de structurer le comportement des objets dans Unity en répondant aux besoins spécifiques de chaque étape du cycle de vie d'un script. Une utilisation appropriée de "Awake()", "Start()", "Update()", "OnEnable()", "OnDisable()" garantit une organisation claire du code, une exécution efficace des tâches, et une maintenance globale du projet.

-Awake() : Initialisation essentielle

La première à être appelée lorsqu'un objet contenant un script est instancié. Elle intervient même si l'objet est désactivé au début de la scène. Elle est principalement utilisée pour initialiser les dépendances essentielles, comme les références à d'autres composants ou ressources qui doivent être configurés avant tout autre action.

-Start() : Configuration après chargement

La méthode "Start()" est exécutée une fois, après que la scène est entièrement chargée et que l'objet est activé. Elle est complémentaire à "Awake()", car elle permet d'initialiser les données ou comportements qui dépendent de l'état global de la scène ou d'autres objets.

-Update() : Gestion des comportements continus

La méthode "Update()" est appelée une fois par frame et constitue le cœur des interactions dynamiques dans Unity..

-OnEnable() et OnDisable() : Activation et désactivation

Les méthodes "OnEnable()" et "OnDisable()" sont appelées respectivement lorsque l'objet ou le script est activé ou désactivé. Ces méthodes sont particulièrement utiles dans les systèmes modulaires où des fonctionnalités doivent être activées ou désactivées dynamiquement, ce qui est le cas beaucoup dans notre projet, on active et désactive constamment des différentes UI.

Fonctionnalités particulières

-Coroutines : Gestion des actions asynchrones

Les coroutines permettent d'exécuter des actions sur plusieurs frames ou avec des délais définis, sans bloquer l'exécution du reste du jeu. Contrairement à "Update()", elles ne s'exécutent pas en boucle mais selon une logique séquentielle. Par exemple, dans notre projet on les utilise surtout pour les animations des cartes.

-Prefabs:

Les prefabs dans Unity sont des modèles réutilisables qui permettent de créer des instances d'objets préconfigurés dans une scène. Ils servent à stocker un GameObject et tous ses composants, ce qui facilite la duplication, la gestion et la modification cohérente de plusieurs instances. Utilisé dans notre projet pour la génération des balises de nom des joueurs dans la scène.

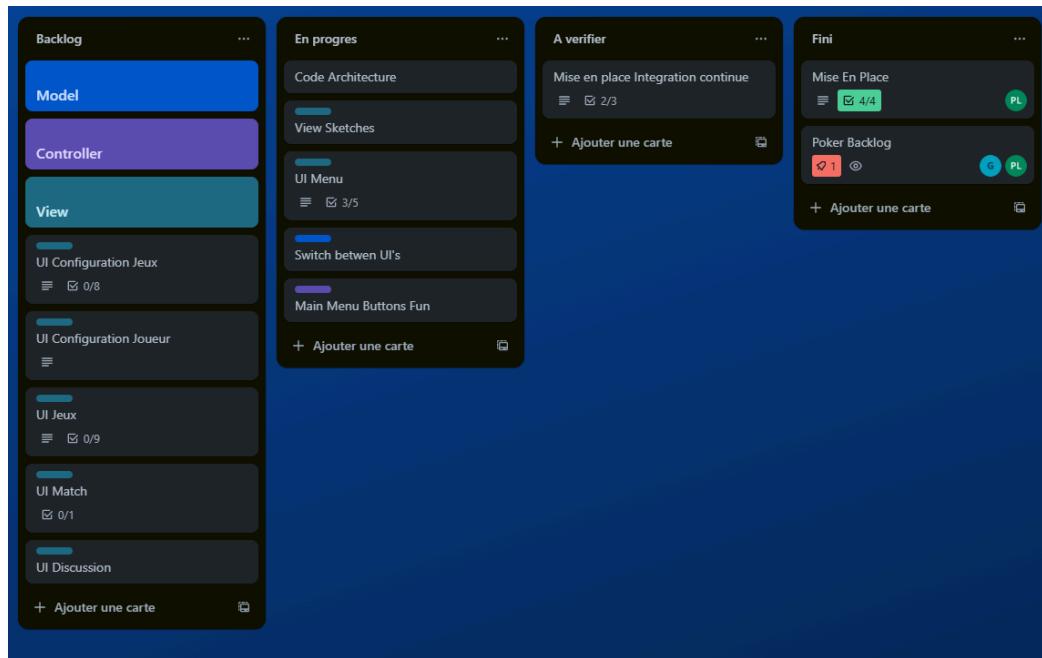
Structure Project

Méthodes agiles utilisées

Nous avons utilisé tout au long de la conception de ce projet certaines méthodes de conception agile. Nous avons travaillé en pair programming pour avancer sur le code

efficacement. En parallèle, nous avons mis en place un tableau Kanban pour organiser et visualiser les tâches au fur et à mesure du temps. Ce tableau nous a aidés à prioriser les fonctionnalités, à suivre l'avancement des différentes étapes du projet et à identifier rapidement les blocages éventuels. Le tableau Kanban a été mis en place grâce au site web Trello, qui propose des outils simples pour éditer et visualiser le tableau. Voici un aperçu du Trello.

Kanban :



Architecture application

Organisation du code:



Pour structurer le projet Unity, nous avons mis en place une organisation claire et méthodique des scripts, visant à simplifier la gestion, la maintenance et l'évolution. Les scripts sont regroupés dans des dossiers spécifiques en fonction de leur rôle au sein du projet.

Par exemple, pour la scène du menu, nous créons un répertoire portant le même nom, dans lequel sont stockés tous les scripts associés à cette scène. De plus, chaque scène est subdivisée en plusieurs répertoires dédiés aux différentes interfaces utilisateur (UI) définies. Chaque répertoire contient les scripts nécessaires au fonctionnement de l'interface correspondante.

Structure des dossiers du projet :

- **Game** : Contient les scripts liés à la logique principale du jeu. Il est subdivisé en plusieurs sous-dossiers correspondant aux fonctionnalités spécifiques :

- UI_Char_Creation : Regroupe les scripts permettant de gérer la création de pseudo, notamment via le script "Char_Creation_Controller.cs".

- UI_Coffee : Contient "Coffee_Controller.cs", un script dédié à la gestion des fonctionnalités liées à l'interface "Coffee" dans le jeu.

- UI_End_Game : Inclut "End_Game_Controller.cs", qui gère les comportements spécifiques à la fin du jeu.

- UI_Game : Ce sous-dossier héberge les scripts qui contrôlent l'interface utilisateur principale et les interactions liées aux tâches du jeu..

- **Autres fichiers importants** : Des scripts tels que "Card_Controller.cs" et "Game_Cards_Animations_Controller.cs" assurent des fonctionnalités spécifiques au animation et le contrôle des valeurs de chaque cartes.

- **Menu** : Ce dossier regroupe les scripts liés aux interfaces des menus :

-UI_Game_Mode : Contient "Description_Controller.cs", qui gère les descriptions des modes de jeu.

-UI_Game_Settings : Regroupe les scripts associés aux paramètres du jeu, comme par exemple les scripts "Add_Custom_Task.cs", "Choice_Timer_Controller.cs" et également "Game_settings_NextButton_Controller.cs".

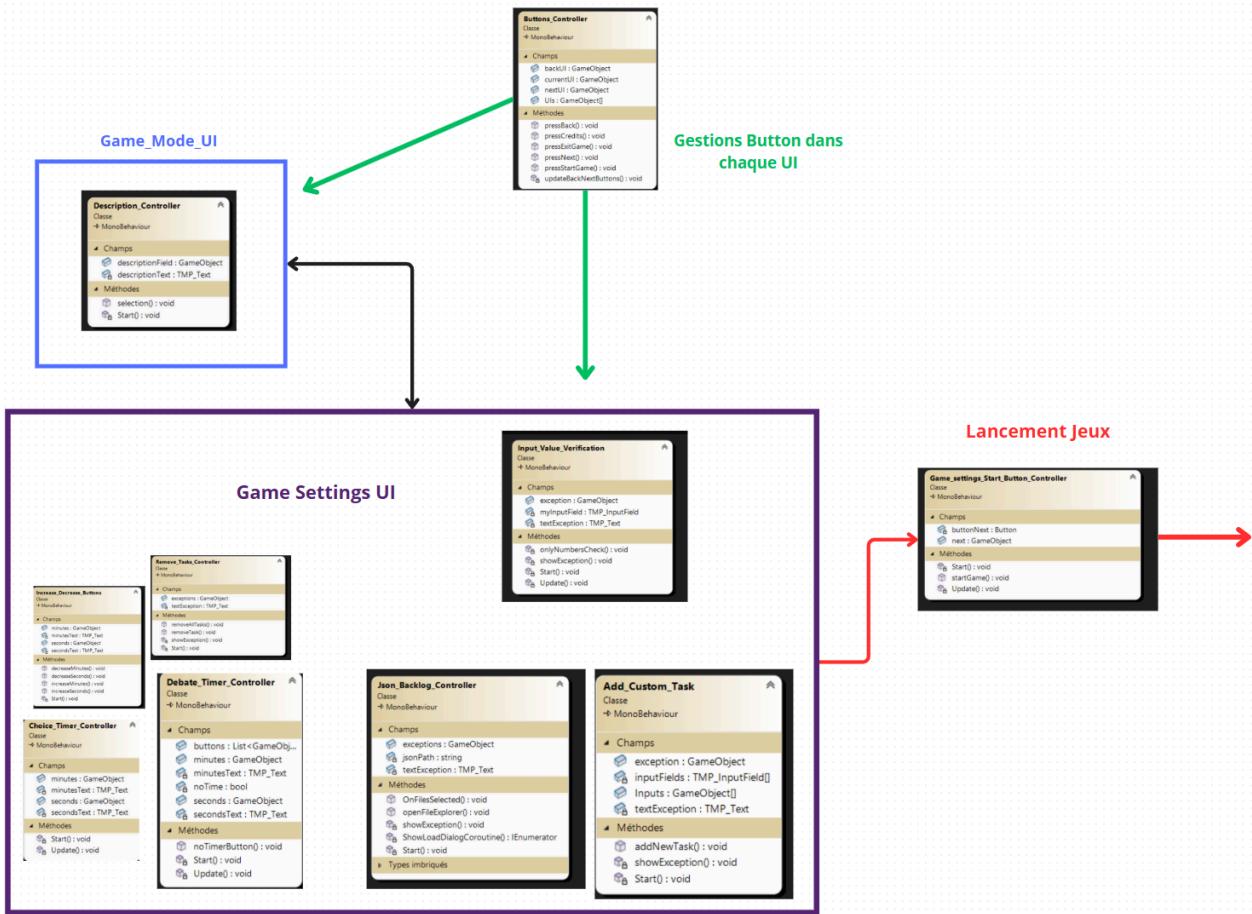
- **UI_Main_Menu** : Héberge "Buttons_Controller.cs", qui gère les boutons du menu principal.

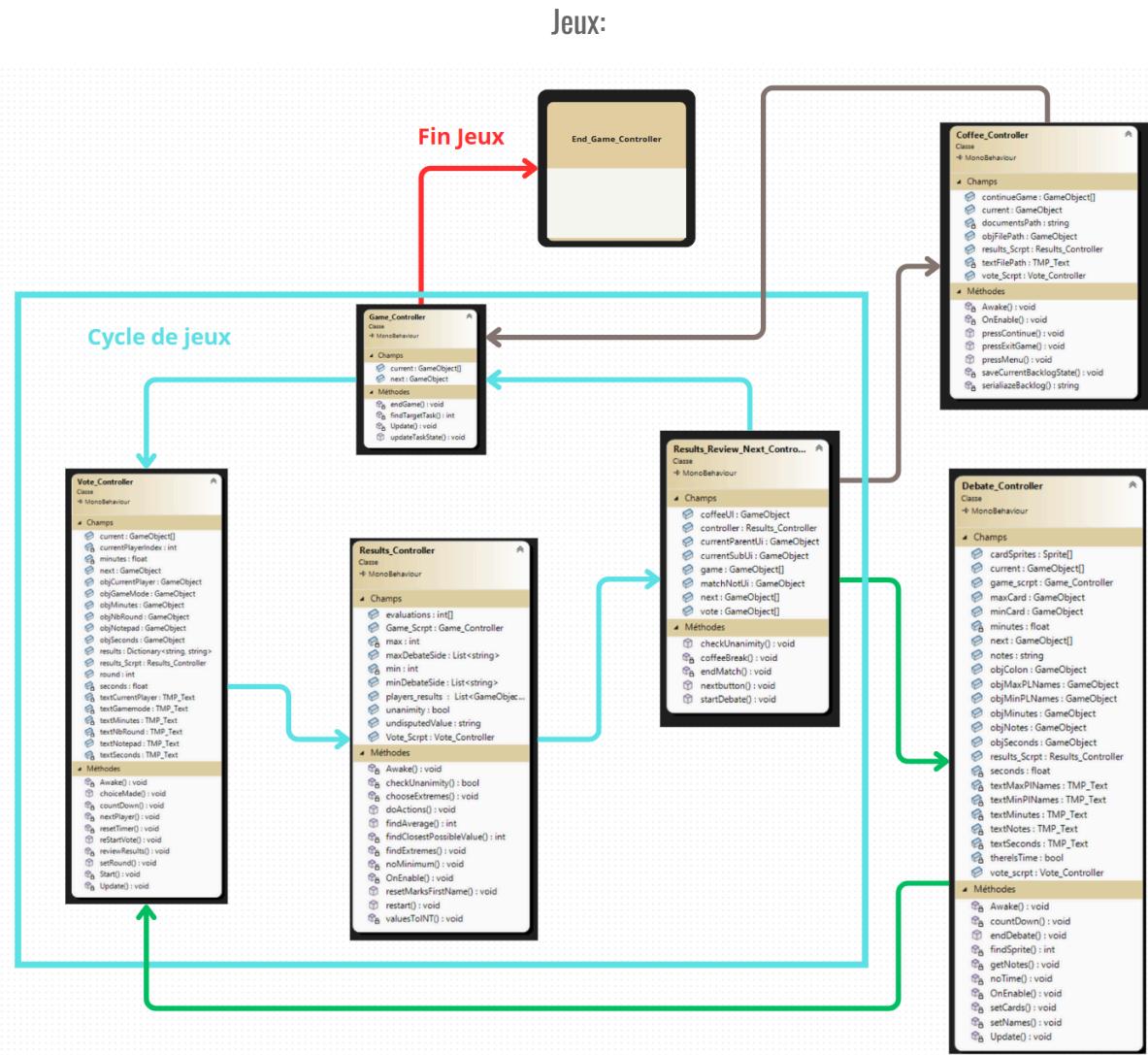
- **Scripts généraux** :

Des scripts tels que "Back_Ground_Singleton.c" et "GameSettings.cs" permettent de gérer des fonctionnalités globales au projet, comme les paramètres persistants ou les informations globales partagées entre scènes.

Diagrammes de Classes:

Menú Principale:





Résumé de l'architecture

L'architecture de notre projet repose sur deux scènes principales dans Unity, chacune étant conçue pour répondre à des étapes spécifiques du fonctionnement de l'application. Chaque scène est composée de plusieurs **GameObjects parents** représentant différentes interfaces utilisateur (UI). Ces GameObjects UI seront activés et désactivés dynamiquement en fonction du déroulement de l'application.

Première Scène : Menu Principal

La première scène est dédiée au menu principal et à la configuration du jeu. Elle inclut les fonctionnalités suivantes :

-Paramétrage du jeu : L'utilisateur peut choisir le mode de jeu souhaité et consulter les règles associées.

-Configuration des joueurs : Permet de définir le nombre de joueurs participant.

-Gestion du chronomètre : Options pour configurer le temps alloué à chaque phase, comme le choix des joueurs ou les débats.

-Backlog des tâches : Affichage des tâches non encore évaluées, prêtes à être intégrées dans le jeu.

Cette scène agit comme un point de départ, permettant à l'équipe de configurer les paramètres avant de lancer le jeu. Le jeu ne pourra démarrer que lorsqu'il y a minimum 2 joueurs et minimum 1 tache pas valide dans le backlog.

Deuxième Scène : Jeu

La deuxième scène est le cœur du projet, où les joueurs interagissent activement pour évaluer les tâches. Les étapes clés de cette scène incluent :

-Création des pseudonymes : Chaque joueur entre son pseudonyme pour l'identification dans le jeu.

-Sélection des tâches : Les joueurs choisissent une tâche à évaluer parmi le backlog.

-Phase de vote : Chaque joueur vote pour évaluer la complexité de la tâche.

-Affichage des résultats : Les choix des joueurs sont affichés, et selon le contexte, les actions suivantes sont proposées :

~Lancer un débat pour clarifier les divergences.

~Valider l'évaluation et passer à une autre tâche.

~Prendre une pause café si nécessaire.

Le jeu continue jusqu'à ce que toutes les tâches du backlog aient été évaluées, marquant ainsi la fin de la session.

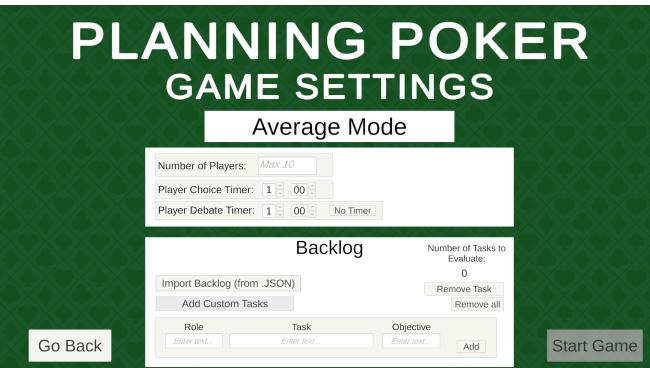
Interface de l'application



Le menu principal au démarrage comporte de boutons simples pour démarrer ou quitter



Lorsque l'utilisateur veut démarrer une partie, il doit d'abord choisir le mode parmi ceux proposés (strict ou moyenne) grâce à un menu déroulant. Il y a également un texte correspondant au mode choisi expliquant le fonctionnement de celui-ci.



Le menu de configuration des options de la partie. Le backlog peut être écrit à la main. Cet écran nous permet aussi d'importer un backlog à partir d'un json sélectionné dans un explorateur de fichier intégré. Une fois les règles mises en place, le bouton start game peut être cliqué.



Ici, les noms des joueurs sont définis les uns après les autres dans le champ. Le nombre de joueurs à nommer dépend bien sûr du nombre renseigné juste avant dans la configuration.



Cette interface en début de partie affiche la tâche qu'il va falloir évaluer. A gauche de l'écran est affiché le nom des joueurs présents. Grâce au bouton reroll, on peut changer la tâche à évaluer. Lorsque la tâche convient, confirmer.

La partie a commencé, chaque joueur clique chacun son tour sur la carte qu'il veut sélectionner. Le temps est indiqué en haut à gauche de l'écran, et il y a un notepad.

Quand tous les joueurs ont terminé de choisir l'estimation qui leur convient, on a un menu de review avec les résultats obtenus. Les extrêmes seront marqués par les symboles du yin(max) et yang(min)

L'écran de débat lorsque les membres de l'équipe ne sont pas d'accord sur la même valeur. Il est possible d'écrire en bas dans les notes, et le contenu de ces notes sera affiché au manches suivantes dans la partie notepad.



Finalement, quand les joueurs ont fini de valider des tâches, on arrive à la fin du jeu ou on peut vérifier tâche par tâche et sauvegarder le json ou on veut.



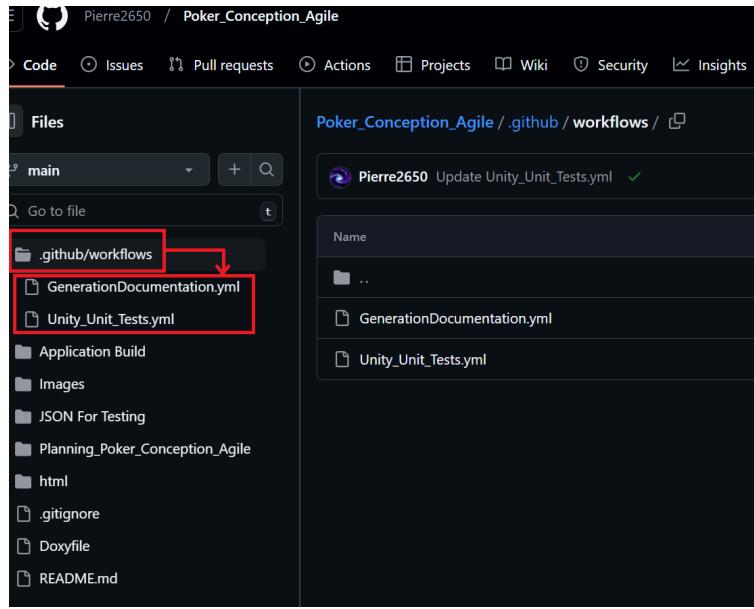
Aussi lorsque les joueurs ont tous choisi la carte café, on nous indique qu'un fichier json contenant le nouveau backlog est sauvegardé dans nos documents. À partir de là on peut choisir de continuer, revenir ou quitter grâce aux boutons.

Integration Continue

Pour mettre en place l'intégration continue avec des tests unitaires dans notre projet, nous avons choisi d'utiliser GitHub Actions, une plateforme intégrée de CI/CD qui permet d'automatiser le processus de build, de test et de déploiement de nos projets.

GitHub Actions

GitHub Actions est une solution de workflow automation fournie par GitHub, qui permet de créer et de gérer des pipelines de CI/CD directement depuis le dépôt GitHub. Chaque workflow dans GitHub Actions est constitué de fichiers YAML situés dans un répertoire .github/workflows (qui est caché) de votre dépôt principal. Ces fichiers définissent les étapes du pipeline, telles que l'installation des dépendances, les tests unitaires, et le déploiement.



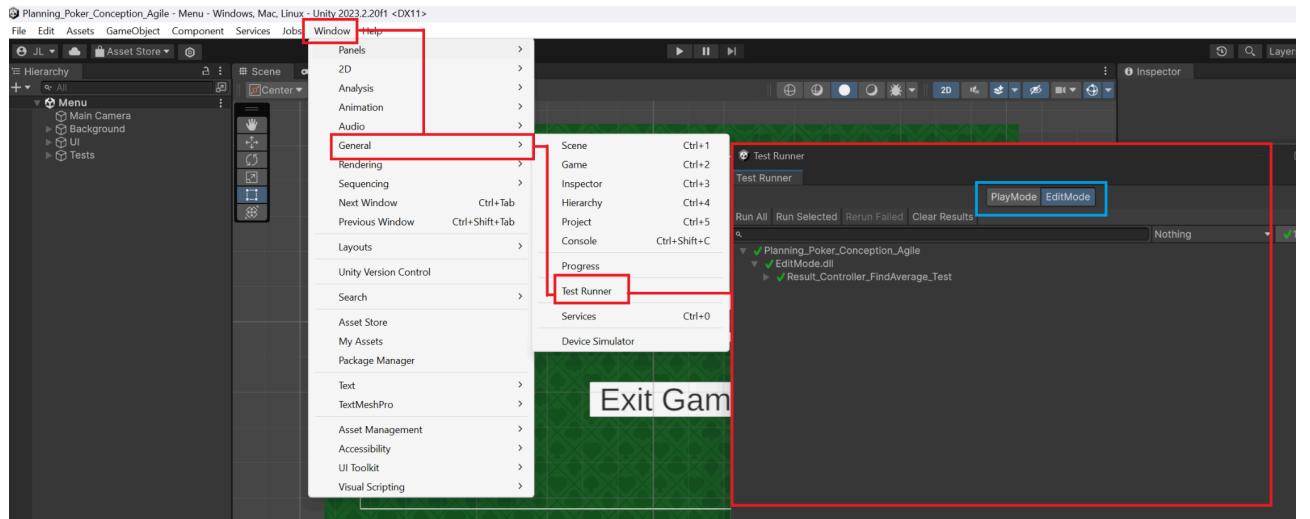
Un workflow dans GitHub Actions est un ensemble d'étapes qui décrivent ce que le système doit faire à chaque push ou pull request. Chaque étape exécute une tâche spécifique comme l'installation des dépendances, le test de code, ou la compilation. Ainsi on a créé 2 workflow un pour les tests unitaire et 1 pour la génération de documentation.

Tests Unitaires en Unity

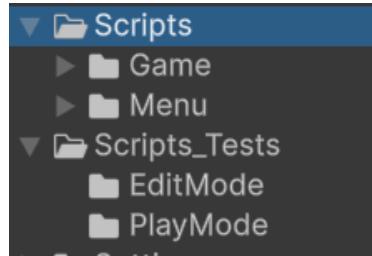
Unity offre des fonctionnalités intégrées pour effectuer des tests unitaires directement depuis son interface. Les tests unitaires peuvent être réalisés en utilisant deux modes :

Edit Mode : Les tests sont exécutés dans l'éditeur Unity. C'est utile pour tester le code sans démarrer le jeu complet.

Play Mode : Les tests sont exécutés pendant que le jeu est en cours d'exécution. C'est utile pour vérifier que le jeu se comporte comme prévu en réponse à diverses entrées utilisateur.



Organization des fichiers



C'est ainsi qu'on a trouvé une page appelée GameCI, une communauté de développeurs qui crée des API open source conçues spécifiquement pour l'intégration continue dans les projets de jeu vidéo. Grâce à leur site, nous avons pu configurer facilement un fichier YAML pour GitHub Actions spécifiquement destiné à exécuter des tests unitaires en mode Edit. GameCI propose des exemples de workflows prêts à l'emploi pour l'intégration continue, nous permettant ainsi de démarrer rapidement avec les bonnes pratiques.

```

name: Test project

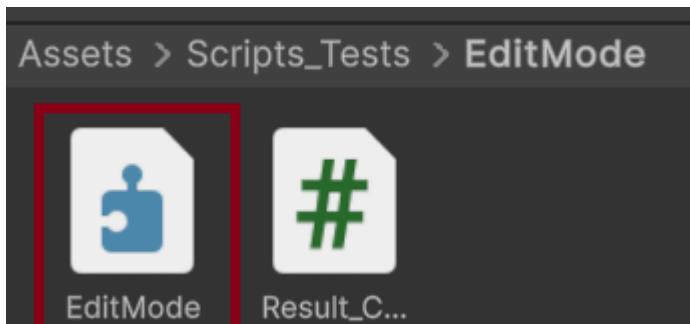
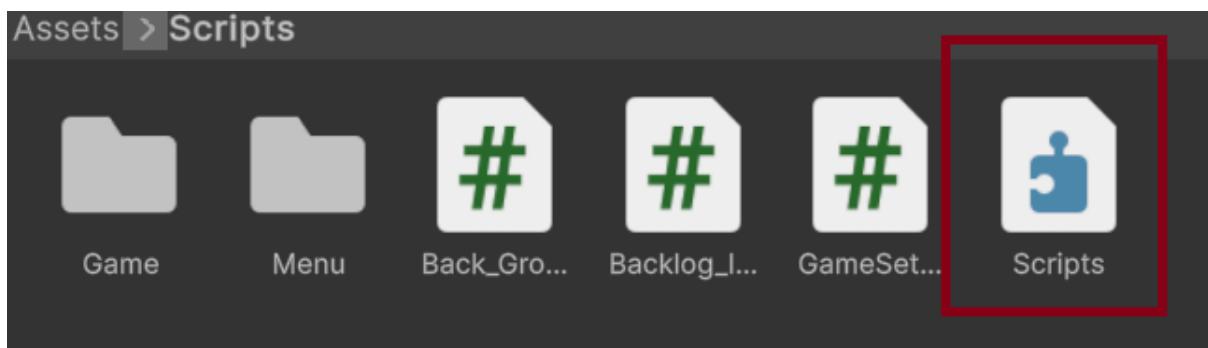
on: [push, pull_request]

jobs:
  testAllModes:
    name: Test in ${matrix.testMode}
    runs-on: ubuntu-latest
    strategy:
      fail-fast: false
  
```

Le seul truc de très important à tenir en compte lorsqu'on implémente leur code est la gestion de la license unity. En effet pour l'environnement on a besoin de passer un compte avec licence unity dans le fichier YML. Pour l'accès aux licences Unity, nous avons dû configurer ainsi des secrets GitHub, en particulier en utilisant le secret UNITY_LICENSE_KEY fourni par l'API GameCI. Cela permet de gérer les licences Unity de manière sécurisée sans exposer des clés en clair dans le code source.

Finalement le dernier facteur à tenir en compte pour la bonne mise en place des test est la manipulation des fichier assembly en unity et c#. Lors de la création de tests unitaires dans Unity, des fichiers d'assembly sont générés pour chaque test. Les fichiers d'assemblage contiennent des informations sur les modules de code qui vont être testés. Il est essentiel de configurer un assembly pour tous les scripts de l'application, en les reliant correctement aux tests unitaires. Cela garantit que tous les scripts sont inclus dans l'assemblage et que les tests peuvent être exécutés efficacement

Fichiers Assembly

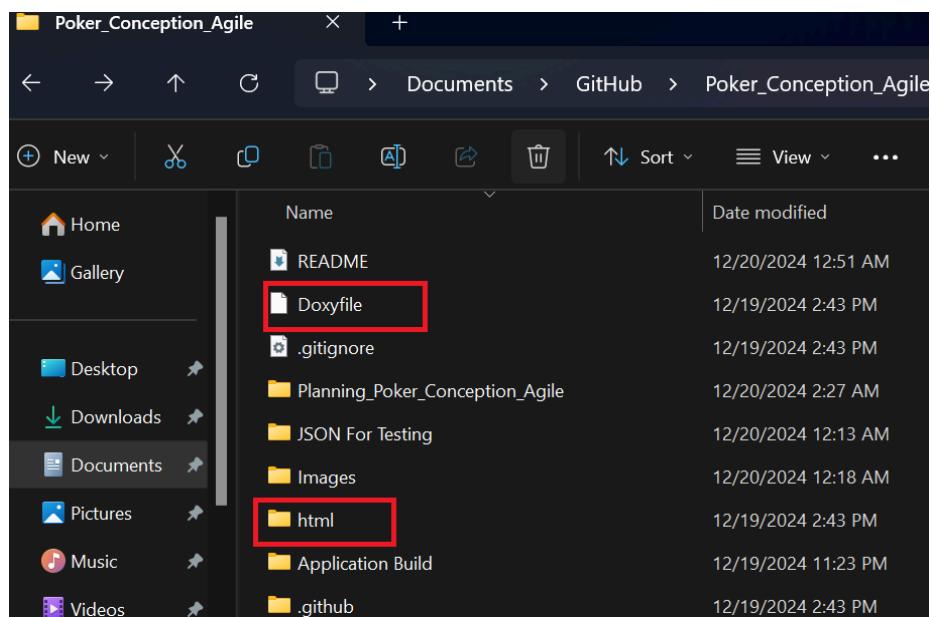
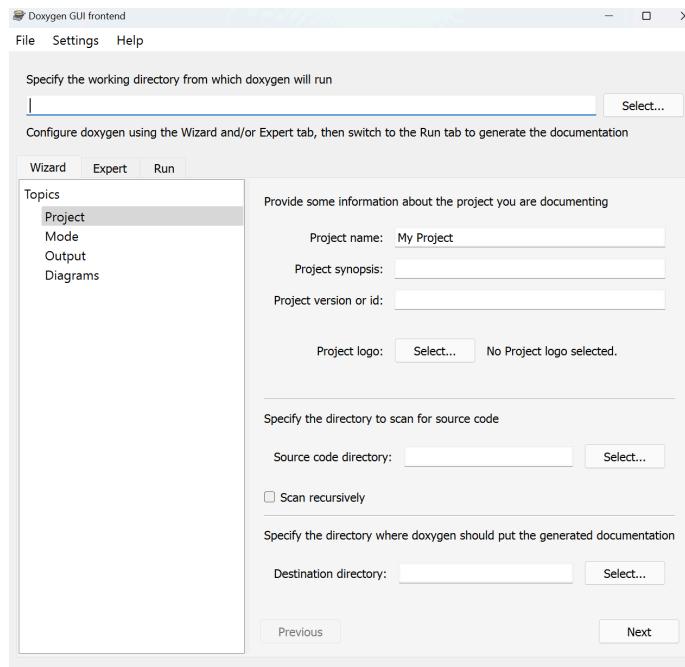


Generation Documentation

Pour automatiser la génération de documentation dans le cadre de l'intégration continue de notre projet, nous avons choisi d'utiliser Doxygen, un outil puissant et largement utilisé pour documenter le code source. Doxygen est un générateur de documentation qui extrait des informations directement depuis les commentaires dans le code source pour produire une documentation claire et structurée. Il prend en charge plusieurs langages, notamment C++, C#, Java, Python, et bien d'autres. Les commentaires dans le code doivent suivre une syntaxe spécifique en utilisant des balises comme @brief ou @class pour annoter les classes, fonctions, et variables. Par exemple :

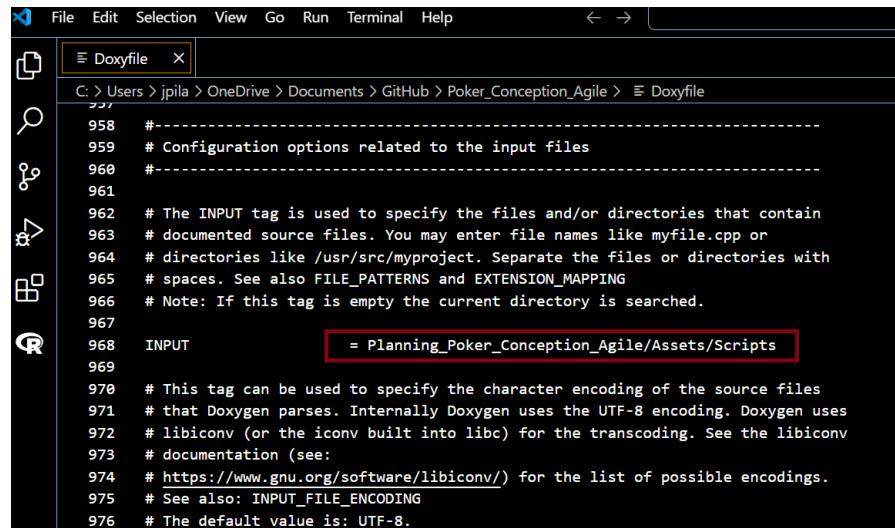
```
Back_Ground_Singleton.cs  Result_Control...verage_Test.cs
C# Miscellaneous Files
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  /**
6   * @brief Class Description: Script Qui va permettre de maintenir le background entre différentes Scènes.
7   */
8  public class Back_Ground_Singleton : MonoBehaviour
9  {
10
11     /**
12      * @brief Cette classe garantit qu'il n'existe qu'une seule instance de l'objet de fond dans la scène,
13      * Le background persiste entre les différentes scènes grâce à la méthode 'DontDestroyOnLoad'.
14      *
15      * @var Back_Ground_Singleton instance
16      * @brief Instance unique du singleton de la classe Back_Ground_Singleton.
17      */
18     public static Back_Ground_Singleton instance;
19
20     private void Awake()
21     {
22         instance = this;
23         DontDestroyOnLoad(this.gameObject);
24     }
25
26
27 }
```

Nous avons téléchargé et installé Doxygen dans le répertoire de notre projet. Cette installation a automatiquement créé deux éléments clés. Un répertoire html qui contient la documentation générée sous forme de fichiers HTML consultables via un navigateur et un fichier Doxyfile, qui est le fichier de configuration principal de Doxygen. Il détermine les paramètres à utiliser pour générer la documentation.



Le Doxyfile est un fichier texte contenant toutes les configurations nécessaires pour contrôler le fonctionnement de Doxygen. Après l'installation, nous avons configuré ce fichier en suivant l'exemple proposé par notre professeur. Une configuration importante que

nous avons personnalisée est le chemin (INPUT) vers le répertoire contenant notre projet Unity. Ce paramètre indique à Doxygen où chercher les fichiers sources à documenter.



```
File Edit Selection View Go Run Terminal Help
Doxyfile X
C: > Users > jpila > OneDrive > Documents > GitHub > Poker_Conception_Agile > Doxyfile
958 #-----
959 # Configuration options related to the input files
960 #
961
962 # The INPUT tag is used to specify the files and/or directories that contain
963 # documented source files. You may enter file names like myfile.cpp or
964 # directories like /usr/src/myproject. Separate the files or directories with
965 # spaces. See also FILE_PATTERNS and EXTENSION_MAPPING
966 # Note: If this tag is empty the current directory is searched.
967
968 INPUT = Planning_Poker_Conception_Agile/Assets/Scripts
969
970 # This tag can be used to specify the character encoding of the source files
971 # that Doxygen parses. Internally Doxygen uses the UTF-8 encoding. Doxygen uses
972 # libiconv (or the iconv built into libc) for the transcoding. See the libiconv
973 # documentation (see:
974 # https://www.gnu.org/software/libiconv/) for the list of possible encodings.
975 # See also: INPUT_FILE_ENCODING
976 # The default value is: UTF-8.
```

Finalement Pour automatiser la génération de la documentation à chaque modification du projet, nous avons intégré Doxygen dans notre pipeline d'intégration continue en utilisant GitHub Actions. Nous avons copié le fichier YAML fourni sur Moodle par le professeur dans le répertoire .github/workflows de notre dépôt. Ce workflow exécute Doxygen pour générer la documentation. Ainsi la CI va créer à chaque fois une nouvelle branche avec un dossier HTML ou on aura notre page avec notre code commenté.

