YJ2-C

BEALES Archie
BENMEZIANE Yacine
CHEZE Rémi
JIN Christophe

Rapport de Projet

Projet de programmation IF14Y010

L2 Informatique 2023 - 2024

Université Paris Cité

Sommaire :

Introduction

Dans le cadre de notre cursus en deuxième année de licence informatique, nous avons été chargés de réaliser un projet en groupe consistant à recréer le jeu "Piece Out". Ce projet, qui a mobilisé les compétences de quatre étudiants, s'inscrit dans l'objectif de mettre en pratique les connaissances théoriques acquises en programmation et en algorithmie.

"Piece Out" est un jeu de puzzle captivant qui met à l'épreuve la logique et la stratégie des joueurs. Le jeu se compose de plusieurs pièces de formes variées que le joueur doit manipuler et disposer de manière à résoudre les puzzles proposés. Notre mission était de reproduire fidèlement ce jeu en respectant ses mécaniques et ses règles tout en développant une interface utilisateur intuitive et agréable.

Pour mener à bien ce projet, nous avons adopté une méthodologie de travail collaborative, en répartissant les tâches selon les compétences et les intérêts de chaque membre de l'équipe. Cela nous a permis de maximiser notre efficacité et d'assurer une progression harmonieuse du projet. Ce rapport détaille les différentes étapes de développement du jeu, des spécifications initiales à la réalisation finale, en passant par la conception, la programmation et les tests.

Nous espérons que ce document offrira une vision claire et complète de notre démarche, des défis rencontrés et des solutions mises en place pour reproduire le jeu. Nous tenons à remercier nos encadrants pour leur soutien et leurs conseils tout au long de ce projet.

Cahier des charges

Réussie
Non fini

1. Reproduction du Jeu
    - Recréer fidèlement les mécaniques de jeu de "Piece Out" :
        - Translation, rotation et inversion des pièces similaires à celles de Tetris.
        - Objectif de configurer les pièces dans une disposition préétablie.
    - Assurer que les mouvements des pièces soient limités et conformes aux règles spécifiées (rotation orientée, translation limitée à certaines directions).
2. Améliorations Graphiques et Esthétiques
    - Intégrer des animations de fond.
    - Ajouter de la musique et des effets sonores.
    - Mettre en place des textures changeantes.
    - Afficher des informations comme le nombre de coups et le temps écoulé.
3. Recherche de Solutions
    - Implémenter des algorithmes de recherche pour vérifier l'existence de solutions et déterminer la plus rapide.
    - Utiliser ces algorithmes pour offrir une aide au joueur.
4. Spécifications Techniques
    1. Mécaniques de Jeu
        - Tutoriel initial avec situations de jeu illustrées.
        - Pièces rotatives et translatables selon des règles prédéfinies.
        - Symétries et mouvements spécifiques pour chaque type de pièce.
    2. Esthétique
        - Vue séparée du modèle pour intégrer les améliorations visuelles.
        - Fluidité des pièces.
        - Conception sonore immersive avec musique et effets sonores appropriés.

Livraison

- **Prototype fonctionnel** après les 6 premières semaines.
- **Version finale** avec toutes les améliorations après 12 semaines.
- **Rapport de projet** détaillant toutes les étapes du développement, les défis rencontrés, les solutions mises en œuvre, et les résultats obtenus.

La Logique

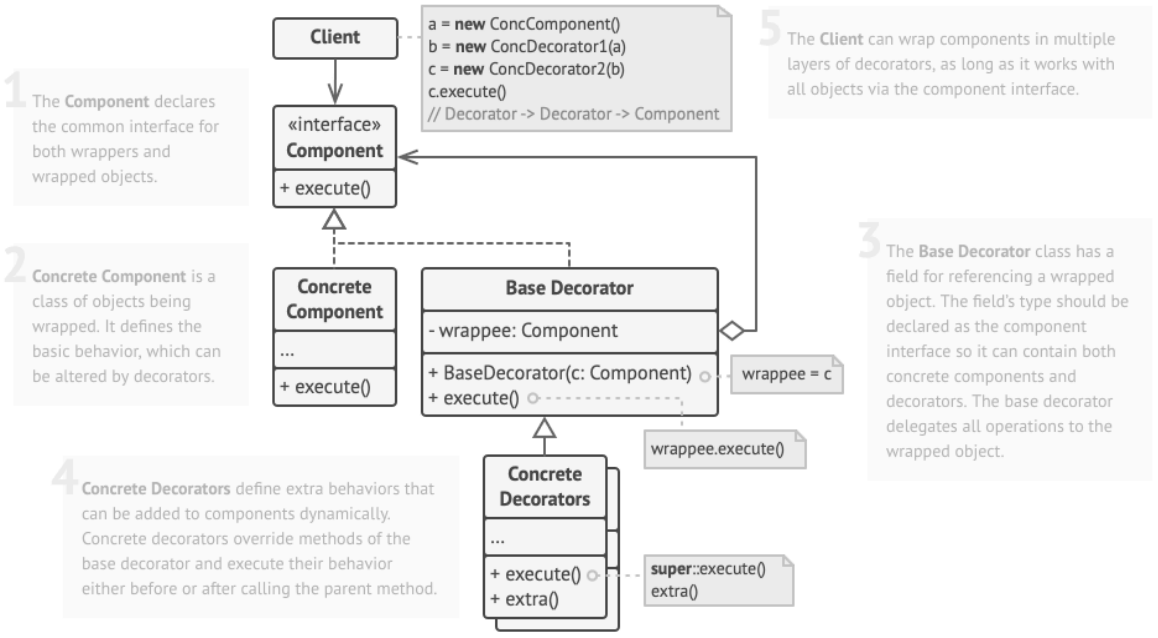
Choix des classes:

Nous avons pris le temps de bien choisir l'organisation du projet pour bien respecter les normes de la programmation orientée objet, l'équipe a pris soin a que les classes soient organisées de manière à séparer les préoccupations, facilitant ainsi la maintenance et l'évolution du code. Cette architecture modulaire permet de développer et de tester indépendamment les différentes parties du jeu, tout en assurant une intégration harmonieuse de l'ensemble des composants. Nous avons collé en annexe un diagramme UML du projet.

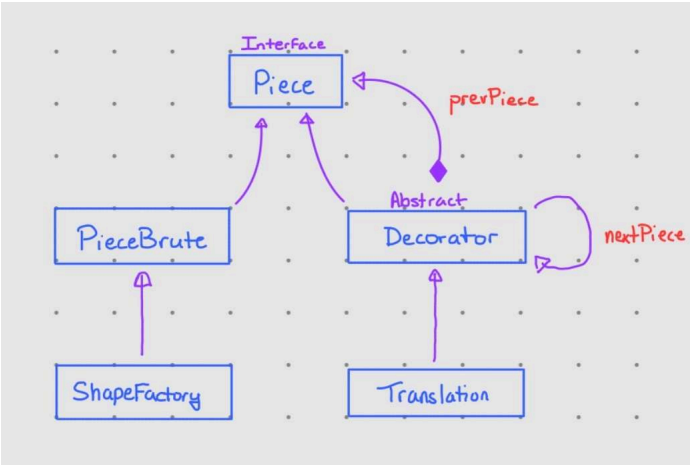
Modèle décorateur:

Les pièces du jeu constituent un véritable défi en termes de conception. Il est nécessaire de pouvoir stocker la forme, la position, les différents mouvements, les collisions, ainsi que le point fixe de chaque mouvement. Pour résoudre ce problème, nous avons utilisé le modèle décorateur (decorator pattern), un patron de conception similaire à une liste doublement chaînée. Ce modèle permet de représenter chaque mouvement comme une addition au mouvement précédent, offrant ainsi une flexibilité et une modularité accrues dans la gestion des mouvements des pièces.

Un exemple concret de modèle décorateur est les vêtements, par exemple quand il fait froid on met un pull, s'il pleut on porte une veste sur le pull, ces vêtement sont des décorateurs mais ne changent pas la personne en dessous.
Voici la structure du pattern décorateur:



Dans notre projet les décorateurs sont implémentés comme ceci:



Nous avons une interface “Piece” qui sert de base. “ShapeFactory” permet de créer une “PieceBrute” comportant toutes les données dont nous avons besoin pour une pièce. Ceci nous permet d’éviter d’avoir des duplicatas de valeurs tel que la position de la pièce qui sont répétées dans les décorateurs. Nous avons aussi une classe abstraite “Decorator” qui sert de base pour les décorateurs concrets et qui implémente “Piece”. Cette base nous sert donc de base pour pouvoir créer différents décorateurs (comme ci-dessous avec les translations “Translation”) où nous pouvons leur attribuer différentes actions. L’attribut “prevPiece” permet de pouvoir retourner voir les précédents décorateurs tandis que “nextPiece” permet d’ajouter de nouveaux décorateurs à l’objet.

Stockage JSON:

Pour stocker des données 3 méthodes possibles nous sont venues à l’esprit : utiliser des fichiers textes, une base de données ou des fichiers JSON.
Les fichiers texte sont simples d’utilisation et compatibles sur n’importe quel système mais ils ne fournissent pas de structure pour formater les données ni pour rechercher ou analyser des données.

Une base de données est un stockage structuré et sécurisé permettant une gestion des données en temps réel. Le problème c’est que ça ne convient absolument pas à notre jeu qui se veut jouable hors ligne, de plus faire une base de données à un certain coût et demande de la maintenance.

Nous avons donc choisi d’utiliser des fichiers JSON puisqu’ils sont conçus pour stocker des données dans un format qui soit léger et facile à lire. De plus, Java s’intègre très bien avec JSON ce qui rend notre travail beaucoup plus facile à itérer.
Notre stockage se découpe en trois parties :

- **Levels:**

  Chaque niveau est représenté dans ce fichier JSON avec plusieurs paramètres : l’identifiant utilisé par le jeu pour de la recherche, son nom pouvant être utilisé pour

l'affichage, la difficulté pouvant être utilisé pour séparer les niveaux en plusieurs catégories, le nombre de mouvements minimum pour finir le niveau et les données pour générer la map.
Chaque niveau a aussi un paramètre qui détermine s'il doit être bloqué dans le menu de sélection de niveau, lorsqu'un niveau a été joué, la classe "LevelManager" met à jour le fichier pour changer le paramètre afin de sauvegarder la progression des joueurs.

- **Pièces:**

  Ce fichier stocke toutes les pièces qui apparaissent dans chaque map, on définit les propriétés suivantes pour chacune : leur forme, leur couleur, leur position et leur type d'action. De plus, on y stocke aussi les données permettant d'identifier si la bonne pièce est bien arrivée à la fin du niveau ainsi que dans la bonne position.

- **Shapes:**

  Le fichier "Shapes" définit chaque pièce que le jeu est capable d'utiliser, chaque pièce est formée d'une boîte de collision et de son "sprite", l'image qui la représente.

Une fois que le jeu est lancé, la classe "LevelManager" se charge de récupérer les informations dont elle a besoin. D'abord elle récupère un JSONArray, un tableau d'objets JSON qui est stocké à la base du fichier, puis elle récupère le JSONObject, l'objet qu'elle veut récupérer (ex : une map ou une pièce) en cherchant la bonne clé identifiant qui correspond à l'objet que l'on recherche. Pour finir, le jeu récupère les informations de l'objet JSON dont il a besoin pour générer les pièces ou la map.

Complétion de niveau:

- **Targets:**

Target est un objet qui traque une pièce pour déterminer si elle est dans son état correct voulu en fin de partie. Cette classe contient les informations nécessaires représentant les conditions que la pièce doit respecter. Notamment, la position destinée et la forme finale. Cet Objet est créé à partir du json ou l'on communique:

1. La pièce à traquer
2. Les coordonnées de la destination
3. Le nombre de rotation à appliquer
4. Si l'on flip (symétrie axiale) la pièce ou pas et l'axe du flip.

- **Condition de victoire:**

C'est dans PieceLogic.java où les pièces et les targets du niveau actuel sont stockés, ainsi que la condition de victoire. Alors il suffit de vérifier que l'on a pour chaque target qui s'y trouve, les informations qu'il porte correspondent aux coordonnées et la forme actuelle de la pièce qu'il traque.

- **Evaluation du niveau:**

Chaque niveau admet une solution avec un nombre minimum de coups, fourni lors de la création du niveau dans `levels.json`. Si l'on arrive à viser ce nombre, on obtient une note de 3 étoiles. Plus on s'en éloigne, plus l'évaluation baisse.

Système Undo/Redo:

- **Definition d'un coup (Move):**

Un coup dans le jeu est une interaction avec une pièce, plus précisément quand on active une des actions qui la décorent. Dans ce cas, une définition très évidente d'un coup est le décorateur en question (qui vient d'être activé).

- **Action inverse (Reverse Personal Action):**

Chaque action parmi les décorateurs a naturellement une action inverse (ou symétrique). Exemple: Translation gauche → Translation droite, Rotation horaire → Rotation anti horaire…etc. Avoir une telle action implémentée permet d'annuler (undo) des coups pendant la partie en cas d'erreur, même si la pièce ne contient pas le décorateur symétrique.

- **Le Undomanager (inspiré du TP10 POOIG):**

Pour pouvoir gérer la chaîne des coups successifs effectués pendant la partie, on fait appelle au Undomanager de swing[1].

```
import javax.swing.undo.UndoManager;
```

Où la chaîne de modifications (Édits) et les manipulations qui l'affectent (undo et redo) sont déjà implantées. Il suffit que la classe Move étende la classe Abstract Undoable Edit afin de se servir de ce manager.

```
Class Move extends javax.swing.undo.AbstractUndoableEdit
```

[1] https://docs.oracle.com/javase/8/docs/api/javax/swing/undo/UndoManager.html

La Vue

Pièces et Carte:

Pour le système d'affichage des pièces et du niveau en général, nous avons découpé cela plusieurs étages. Tout d'abord avec l'arène, contenant un tableau de cellules de taille dynamique (donné par le fichier JSON correspondant). Ces cellules permettent l'affichage de du niveau, par un fond de différentes couleurs selon si la cellule est une zone jouable. Nous affichons ensuite "par dessus" des objets cible ("Target") qui servent de ligne d'arrivée pour chaque pièce du niveau. Finalement, apparaissent au dernier étage les pièces avec des images correspondant à chaque pièce possible et y ajoutons les images des décorateurs correspondants à leur bon emplacement.

Les menus:

Il existe deux menus dans notre jeu : le menu principal appelé "Launcher" qui permet de choisir le nombre de joueurs pour lancer une partie ou de quitter, le nombre de joueurs est très important car il permet de donner le nombre d'instances qu'il va falloir créer pour la partie.
Ainsi après avoir choisi le/les joueurs doivent choisir un niveau dans le "LevelSelectMenu" parmi ceux débloqués, les joueurs ne peuvent jouer qu'à partir du premier niveau ou à partir d'un niveau déjà fait, chaque bouton du menu de sélection de niveau est un component custom qui se créé très facilement pour pouvoir facilement avoir une multitude de boutons à l'écran sans que cela soit compliqué à gérer.
Une fois que les joueurs ont choisi le niveau, le jeu charge le niveau et crée le nombre d'instances nécessaires en fonction du nombre de joueurs.

Musique et sons:

Les sons et la musique du fond du jeu original ont été repris dans cette version.
La musique joue en boucle en arrière-plan dès le lancement du launcher, dans les menus, et en pleine partie. Ils peuvent être désactivés/activés séparément l'un de l'autre à tout moment dans le jeu.

Les bruitages sont sélectionnés aléatoirement parmi un ensemble d'effets sonores similaires de la même catégorie. L'intégralité du son du jeu se gère dans le fichier Sound.java
.

Le Contrôleur

Le contrôle du jeu est géré par la classe `PlayerController.java` qui implémente les interfaces `MouseListener` et `MouseMotionListener` . Il se fait donc exclusivement par la souris. C'est dans cet objet aussi que se trouve le undoManager mentionné ci-dessus.

En cliquant sur une icône d'une action de l'une des pièces sur la carte, on parcourt la chaîne de décorateurs de cette pièce jusqu'à en trouver le bon. ie: Celui qui a la même position que le point du click. Une fois trouvé, on appelle la méthode decorator.action(position de souris fournie dans les paramètres).

En particulier, si une pièce est dotée d'une translation, le joueur peut alors la glisser avec la souris vers la direction souhaitée. Quand on effectue un glissement avec la souris, on calcule la direction de ce dernier. Puis, selon les translations autorisées de la pièce, on exécute ce coup ou on l'ignore.

Conclusion

En conclusion, ce projet de programmation a été une expérience enrichissante et formatrice qui nous a permis de consolider nos compétences en développement logiciel. À travers la réalisation de ce projet, nous avons non seulement appliqué les concepts théoriques appris en cours, mais également développé des compétences pratiques essentielles telles que la gestion de projet, la résolution de problèmes complexes et le travail en équipe.

Nous avons rencontré et surmonté plusieurs défis techniques, notamment l'implémentation des patterns , l'optimisation des performances et la gestion des erreurs. Ces obstacles nous ont permis d'affiner notre compréhension des bonnes pratiques de programmation et de la conception logicielle.

En perspective, nous pouvons envisager plusieurs améliorations et extensions pour ce projet, telles que l'ajout de nouvelles fonctionnalités, un algorithme de recherche de solution, l'amélioration de l'interface utilisateur et l'optimisation des performances. Ces perspectives ouvrent la voie à de futurs développements et à une exploration plus approfondie des technologies utilisées.

En somme, ce projet de programmation a été une opportunité précieuse pour appliquer nos connaissances théoriques dans un contexte pratique, nous préparant ainsi à des projets plus complexes et à une carrière professionnelle dans le domaine de l'informatique.

Nous tenons à exprimer notre profonde gratitude envers l'ensemble de l'équipe encadrant ce projet. Nous adressons des remerciements particuliers à M. Y. Jurski pour ses conseils pertinents et son accompagnement précieux tout au long du développement de notre projet.

- Beales Archie 22201677
- Chèze Rémi 22207793
- Jin Christophe 22202010
- Benmeziane Yacine 22215212

Licence 2 Informatique Générale, groupe 4, YJ2-C

Annexe