Nassiri Amir
Marouan Hazim-Rayan
INFO-1C

SAÉ 2.01 et 2.02
L'Apprenti Ordonnateur

IUT de Vélizy
BUT INFORMATIQUE *2023/2024*

Sommaire

1- Introduction

Dans le cadre de la SAÉ 2.01 et 2.02, nous avons développé une application pour aider à réaligner les cristaux magiques dans le royaume imaginaire de Krystalia. Ce royaume est peuplé de sorciers, magiciens et enchanteurs, dont les pouvoirs sont alimentés par des cristaux de pouvoir situés dans différents temples, chacun dédié à une couleur spécifique. Après une tempête magique, les cristaux se déplacent de façon aléatoire, et il devient crucial de les réaligner pour rétablir l'équilibre magique.

Notre application place l'utilisateur dans le rôle de l'apprenti ordonnateur, dont la tâche est de réaligner ces cristaux dans leurs temples respectifs. L'apprenti commence son périple à la position initiale (0,0) (au centre de la map) en étant représenté par un ovale beige et peut se déplacer dans les quatre directions cardinales. Il ne peut transporter qu'un seul cristal à la fois, et doit collecter, échanger et déposer les cristaux pour les replacer dans les temples correspondants.

Fonctionnalités de l'application

1. Chargement de scénarios:
- L'application permet de charger différents scénarios à partir de fichiers texte. Chaque scénario décrit la position des temples, leur couleur et la couleur des cristaux qu'ils contiennent après la tempête magique.

2. Interface graphique:
- Une interface utilisateur permet de visualiser les positions et les couleurs des temples, les couleurs des cristaux, la position de l'apprenti ainsi que le cristal actuellement porté. L'utilisateur peut cliquer sur des cases pour déplacer l'apprenti pas à pas jusqu'à la destination souhaitée. La manipulation des cristaux se fait à l'aide de boutons lorsque l'apprenti entre dans un temple. Le lancement de l'algorithme de tri ou heuristique se fait également à l'aide de boutons visibles sur l'écran.

3. Affichage des informations de la partie :
- Un compteur affiche le nombre total de pas effectués par l'apprenti, permettant à l'utilisateur de suivre son efficacité. Cela permet également de comparer les résultats entre différentes tentatives et approches. L'affichage des informations des temples est également disponible en dessous du compteur de nombre de pas. Le cristal actuellement porté par l'apprenti est également affiché en dessous des deux informations citées juste avant.

Approches algorithmiques

1.Algorithme de tri:
-L'algorithme de tri de l'application utilise un tri par sélection pour réaligner les cristaux. Pour chaque couleur de cristal, l'apprenti se rend au temple qui contient ce cristal, le prend, et le dépose dans le temple correspondant à cette couleur. Si le temple cible contient déjà un cristal, l'apprenti échange les cristaux : il dépose celui qu'il transporte et prend celui du temple. Il répète ce processus pour chaque couleur, jusqu'à ce que tous les cristaux soient placés dans les bons temples. L'algorithme assure que chaque action est valide pour éviter les erreurs.

2. Algorithme heuristique:

- L'algorithme heuristique de l'application aide à réaligner les cristaux plus efficacement en prenant des décisions intelligentes. D'abord, il vérifie si tous les cristaux sont déjà bien placés. Ensuite, il cherche les temples avec des cristaux qui ne sont pas à leur place et se déplace vers le temple le plus proche avec un cristal mal placé. L'apprenti prend ce cristal et va directement au temple de la couleur correspondante. Si ce temple contient déjà un cristal, l'apprenti échange les cristaux : il dépose le cristal qu'il transporte et prend celui qui se trouve dans le temple. Il répète ce processus jusqu'à ce que tous les cristaux soient bien placés dans les bons temples. Cela permet de réduire les déplacements inutiles et de réaligner les cristaux plus rapidement.

Ces algorithmes se lancent en cliquant sur les boutons portant le nom de l'algorithme voulu.

Objectifs et performance

L'objectif principal de l'application est de réaligner les cristaux en minimisant le nombre de déplacements. La performance de chaque algorithme est évaluée et comparée, ce qui permet de mieux comprendre leur efficacité. Les résultats montrent que si l'algorithme heuristique n'est pas toujours optimal, il peut trouver des solutions optimales dans de nombreux cas et ainsi améliorer l'efficacité par rapport à des méthodes de tri plus simples.

En conclusion, cette application explore de manière ludique et pédagogique différentes approches algorithmiques pour résoudre un problème complexe de réalignement de cristaux. Elle fournit un outil interactif pour visualiser et comparer les performances de différentes méthodes.

2-La qualité de développement

2.1- L’organisation du travail

Cycle de Vie de L'Apprenti Ordonnateur

1. Initiation

Définition du projet

Le projet "L'Apprenti Ordonnateur" a pour but de développer une application permettant de réaligner manuellement des cristaux de différentes couleurs dans leurs temples respectifs. L'objectif principal est aussi de créer un algorithme capable de résoudre ce problème de manière efficace, ainsi qu'une interface graphique pour visualiser les résultats.

Étude de faisabilité

Réalisation:
Ce projet a été réalisé par Nassiri Amir et Marouan Hazim-Rayan pour le 5 juin 2024.
Évaluation technique :
Nous avons évalué nos compétences pour développer ce projet en utilisant Java pour la programmation, JavaFX pour créer l'interface graphique, et des algorithmes de tri et heuristiques pour la logique du jeu tout en respectant la structure MVC (Modele-Vue-Controleur).

2. Planification

Élaboration du plan de projet

Tâches:
- Comprendre le sujet.
- Répartir les tâches.
- Développer la partie logique.
- Adapter la logique à l'interface graphique en respectant la structure MVC.
- Effectuer des commits réguliers sur Bitbucket.
- Implémenter les algorithmes de tri et heuristique.
- Réaliser les tests unitaires des classes du modèle.
- Rédiger la documentation.
- Finaliser et tester l'application.
- Rédiger le rapport et rendre les livrables avant le 5 juin à 10h sur Ecampus.

Liste des livrables:
- Code source de l'application.
- Documentation détaillée.
- Rapport final du projet.
- Tests unitaires.

Gestion des risques
- Problèmes d'intégration entre la logique et l'interface graphique.
- Bugs non détectés lors des tests unitaires.

Stratégies de mitigation :
- Communication régulière et suivi des tâches.
- Tests fréquents après chaque changement.

3. Exécution

Réalisation des tâches
- Outils :
Utilisation d'IntelliJ IDEA sur ordinateur pour le développement.

Gestion de l'équipe
- Coordination :
Nassiri Amir a principalement développé le code, tandis que Marouan Hazim-Rayan a résolu de nombreux problèmes d'implémentation graphique.
- Communication: Réunions pendant les séances de SAE et appels pour avancer le projet.

4. Suivi et contrôle

Suivi du projet
- Évaluations périodiques :
Contrôle régulier pour s'assurer du bon fonctionnement de l'application après chaque modification.

Assurance qualité
- Tests réalisés :
Tests unitaires sur les classes du modèle.
- Vérification des critères de qualité :
Validation des fonctionnalités et correction des bugs.

5. Clôture

Finalisation des livrables
- Livrables finalisés :
  - Application fonctionnelle.
  - Documentation complète.
  - Rapport détaillé du projet.
  - Tests unitaires.

Évaluation du projet

- Analyse des résultats :
Le projet a été évalué comme réussi, avec une bonne intégration entre la logique et l'interface graphique ainsi que des algorithmes de tri et heuristiques performants.

- Retour d'expérience :
Les difficultés rencontrées lors du projet ont été surmontées tels que le lien entre la logique et le graphique avec notamment l'implémentation d'un controleur fonctionnel correct ou encore l'implémentation d'un algorithme heuristique.

Leçons apprises
- Enseignements tirés :
Importance de la communication régulière et de la validation fréquente des fonctionnalités.
- Suggestions pour les futurs projets :
Maintenir une bonne documentation et un suivi régulier des tâches pour éviter les problèmes d'intégration.

Répartition des tâches

Nassiri Amir a principalement développé la majorité du code, créé la logique de l'application, implémenté les algorithmes de tri et heuristiques, et rédigé les tests unitaires. Il a aussi contribué à la documentation du projet. Hazim-Rayan Marouan s'est chargé de l'implémentation graphique, en résolvant les problèmes liés aux déplacements de l'apprenti, au débogage de l'affichage des temples, et à la mise à jour du canvas. Il a également aidé à adapter la logique à l'interface graphique et assuré des tests réguliers après chaque modification. Les deux ont été impliqués, communiquant régulièrement pour s'assurer de l'avancement cohérent du projet.

2.2- Outils utilisés

Pour le développement du projet, nous avons utilisé plusieurs outils. Git a été notre système de contrôle de version, permettant de suivre les modifications et de mieux collaborer. Les tests unitaires ont été réalisés avec JUnit, garantissant la fiabilité et la robustesse de notre code. Bitbucket a été utilisé pour héberger notre dépôt Git, facilitant la gestion de notre code en ligne et la collaboration à distance. Enfin, nous avons utilisé IntelliJ IDEA comme environnement de développement, offrant des fonctionnalités pour coder, tester et déboguer notre application Java avec JavaFX. Ces outils ont été cruciaux pour le succès et l'organisation de notre projet.

2.3- Analyse de la méthode de développement

Pour le projet "L'Apprenti Ordonnateur", nous avons adopté une méthode de développement structurée et collaborative. Tout d'abord, nous avons pris le temps de bien comprendre les exigences du projet, puis nous avons réparti les tâches en fonction des compétences de chaque membre de l'équipe. La partie logique du jeu a été développée en premier, en s'assurant que toutes les fonctionnalités essentielles fonctionnaient bien avant de les intégrer à l'interface graphique, en suivant la structure MVC pour une meilleure organisation du code.

Pour le contrôle de version et la collaboration, nous avons utilisé Git et Bitbucket. Ces outils ont permis à chaque membre de travailler sur différentes parties du projet en parallèle sans conflits majeurs. Nous avons également intégré des tests unitaires avec JUnit pour assurer la qualité du code. Ces tests ont permis de vérifier des aspects essentiels du jeu, comme la prise, l'échange et le dépôt des cristaux, ainsi que la détection correcte des conditions de fin de jeu.

IntelliJ IDEA a été notre environnement de développement principal, offrant des fonctionnalités avancées qui ont facilité notre travail. Nous avons maintenu des réunions régulières pour suivre l'avancement du projet, discuter des difficultés rencontrées et trouver des solutions rapidement. Des tests fréquents étaient réalisés après chaque modification pour s'assurer que le jeu fonctionnait comme prévu.

En résumé, notre méthode de développement a été efficace grâce à une planification rigoureuse, une répartition claire des tâches et l'utilisation d'outils robustes pour la collaboration et les tests. Cela nous a permis de développer un jeu fonctionnel et de haute qualité tout en respectant les délais.

3- La conception générale

3.1- Diagramme de classes

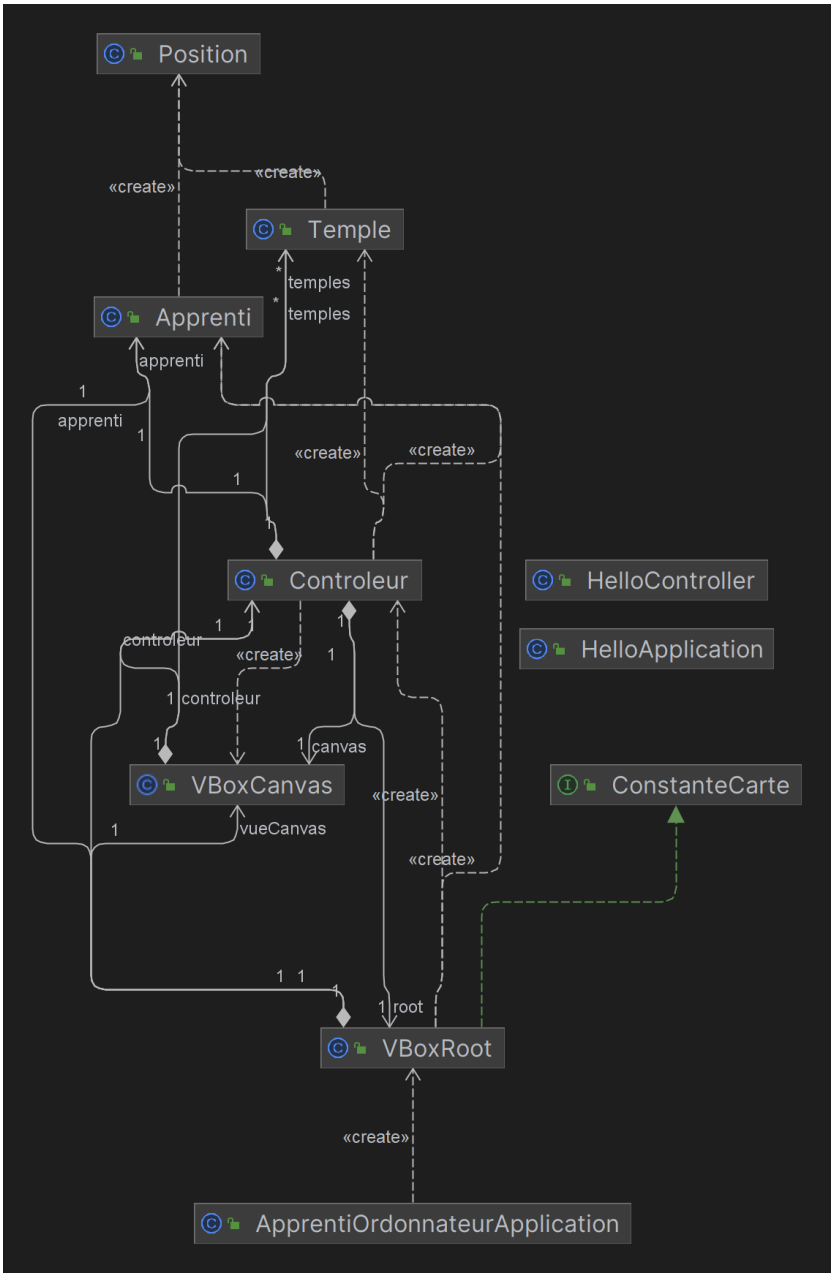
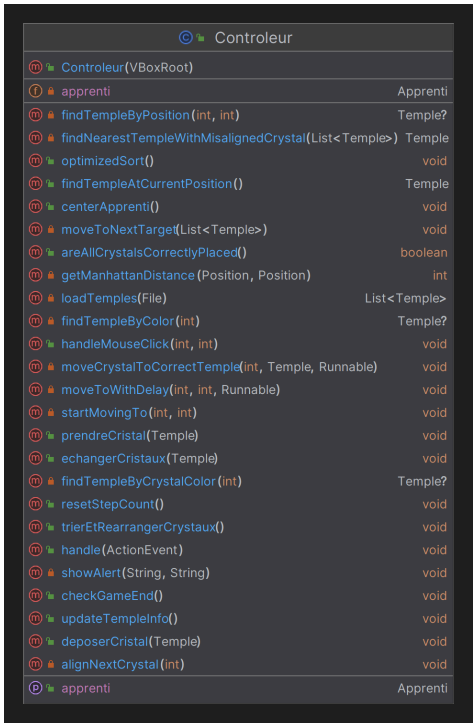
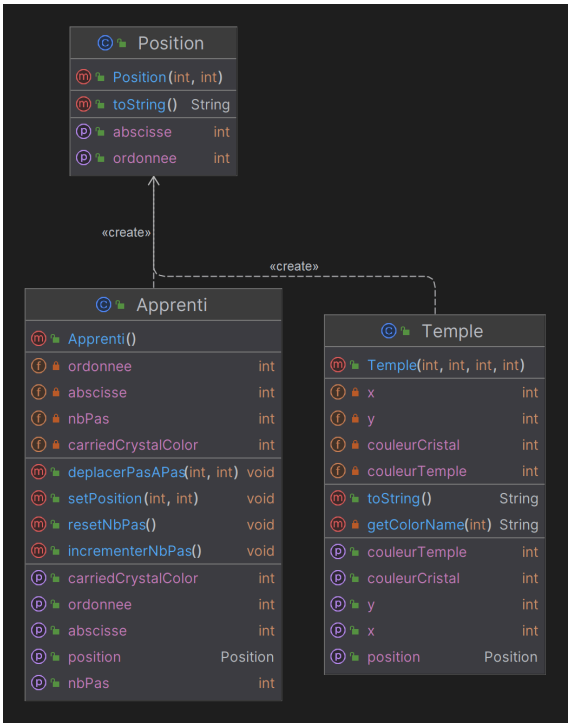


Diagramme de classes de haut niveau

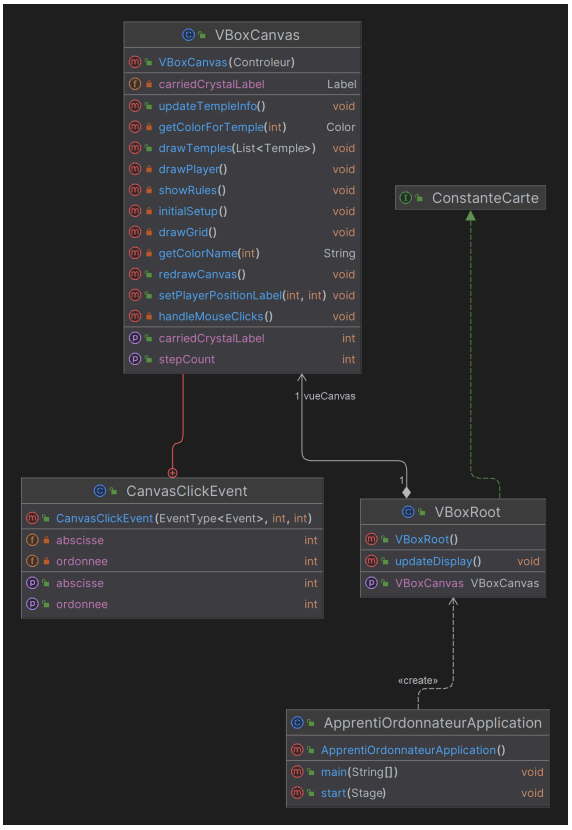
3.2- Diagramme de classes détaillés



Le diagramme des classes détaillées du controleur



Le diagramme des classes détaillées du modèle



Le diagramme des classes détaillées de la vue

3.3- Structures de données

Pour le projet de l'Apprenti Ordonnateur, nous avons utilisé des structures de données simples mais suffisamment efficaces, ainsi que des stratégies algorithmiques adaptées aux exigences du jeu. Les classes principales sont les suivantes: la classe **Apprenti**, qui représente l'apprenti avec ses coordonnées et la couleur du cristal qu'il transporte, la classe **Temple**, qui représente un temple avec ses coordonnées, la couleur du temple et celle du cristal qu'il contient, et enfin la classe **Position**, qui représente une position avec des coordonnées x et y. Nous avons utilisé une collection *List<Temple>* pour stocker la liste des temples, ce qui permet un accès rapide et direct pour des recherches fréquentes et des mises à jour des temples.

En plus des classes du modèle, nous avons des classes de la vue telles que **VBoxCanvas** et **VBoxRoot** qui gèrent l'affichage graphique et l'interaction utilisateur. **VBoxCanvas** s'occupe du dessin des temples, des cristaux, de l'apprenti sur le canvas, de l'affichage des informations des temples et de la partie en cours. Tandis que **VBoxRoot** gère la structure principale de l'interface utilisateur, y compris les menus et les boutons de commande. Nous avons également utilisé une interface **ConstanteCarte** afin de stocker des constantes. Ces classes permettent de visualiser facilement l'état du jeu et de suivre les déplacements de l'apprenti.

Enfin, pour assurer la robustesse du code, des tests unitaires ont été inclus pour vérifier le bon fonctionnement de chaque composant de manière isolée. Ils testent, par exemple, la prise, l'échange et le dépôt des cristaux, ainsi que la détection des conditions de fin de jeu. Ces tests permettent de détecter et corriger les erreurs tôt dans le développement, garantissant ainsi la qualité du code.

3.4- Stratégies algorithmiques

En termes de stratégies algorithmiques, nous avons implémenté un algorithme de tri par sélection pour réaligner les cristaux en les échangeant entre les temples. L'apprenti se déplace vers chaque temple dans l'ordre des couleurs de cristaux. Pour chaque couleur, il trouve le temple qui contient le cristal correspondant, le prend, et se rend au temple de la couleur correspondante. Si le temple cible est vide, il y dépose le cristal. Si le temple cible contient déjà un cristal, un échange est effectué. Ce processus est répété pour chaque couleur jusqu'à ce que tous les cristaux soient placés correctement, tout en évitant les actions impossibles. Nous avons également développé un algorithme heuristique pour minimiser les déplacements de l'apprenti. D'abord, ce dernier vérifie si tous les cristaux sont déjà bien placés. Ensuite, il cherche les temples avec des cristaux qui ne sont pas à leur place et se déplace vers le temple le plus proche avec un cristal mal placé. L'apprenti prend ce cristal et va directement au temple de la couleur correspondante. Si ce temple contient déjà un cristal, l'apprenti échange les cristaux : il dépose le cristal qu'il transporte et prend celui qui se trouve dans le temple. Il répète ce processus jusqu'à ce que tous les cristaux soient bien placés dans les bons temples. Nous avons vérifié et confirmé son efficacité à l'aide d'un programme Python permettant de mesurer l'efficacité, l'erreur relative et absolue moyenne.

Valeurs obtenues par les algorithmes:

Valeur obtenues par l'algorithme de tri:
Scénario 0: 44 pas
Scénario 1: 84 pas
Scénario 2: 158 pas
Scénario 3: 166 pas
Scénario 4: 199 pas
Scénario 5: 272 pas

Valeur obtenues par l'algorithme heuristique:
Scénario 0: 26 pas
Scénario 1: 44 pas
Scénario 2: 88 pas
Scénario 3: 121 pas
Scénario 4: 86 pas
Scénario 5: 116 pas

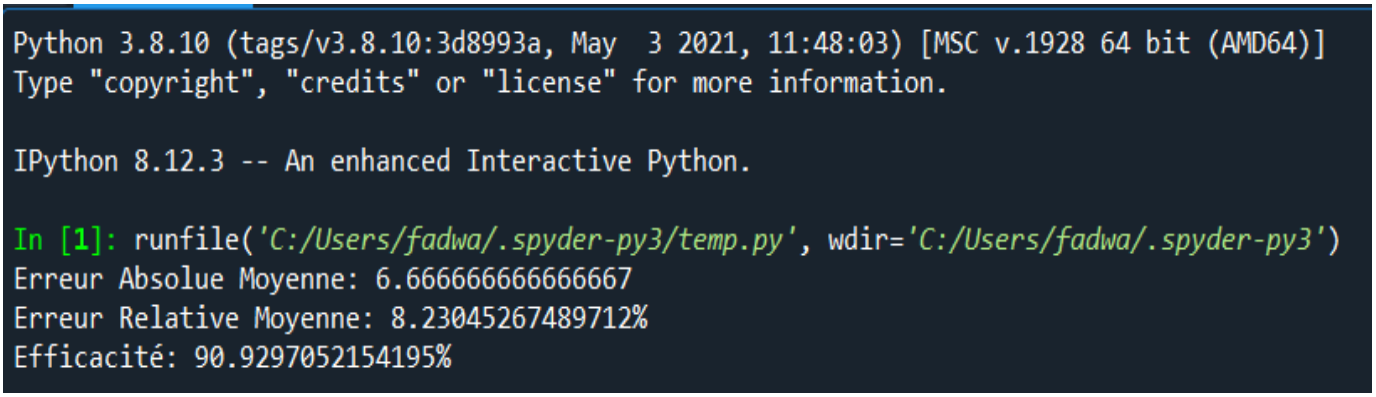
Vérification de l'efficacité de l'algorithme heuristique:

Des recherches avaient été entamées dans l'objectif d'affirmer que notre algorithme heuristique était performant. Lors de nos recherches, nous sommes tombés sur un programme Python nous permettant de savoir si notre algorithme heuristique est efficace.

Le programme Python ci-dessous (provenant de la documentation officielle de la bibliothèque NumPy) va comparer les résultats obtenus par l'algorithme heuristique avec les valeurs optimales et va retourner trois valeurs qui pourront témoigner de l'efficacité de l'algorithme heuristique.

```python
import numpy as np

#Valeurs
valeurs_optimales = [26,44,88,81,86,116]
valeurs_obtenues = [26,44,88,121,86,116]

# Calcul de l'erreur absolue moyenne
eam = np.mean(np.abs(np.array(valeurs_optimales) - np.array(valeurs_obtenues)))

# Calcul de l'erreur relative moyenne
erm = np.mean(np.abs((np.array(valeurs_optimales) - np.array(valeurs_obtenues)) / np.array(valeurs_optimales)))

# Calcul de l'efficacité
efficacite = (1 - np.sum(np.abs(np.array(valeurs_optimales) - np.array(valeurs_obtenues))) / np.sum(valeurs_optimales)) * 100

# Affichage des résultats
print(f"Erreur Absolue Moyenne: {eam}")
print(f"Erreur Relative Moyenne: {erm * 100}%")
print(f"Efficacité: {efficacite}%")
```

Programme Python de vérification exécuté sur le logiciel Spyder



Résultat de l'algorithme Python exécuté

Analyse des Résultats de l'Algorithme

Les résultats obtenus lors de l'évaluation de l'algorithme sont les suivants :

1. Erreur Absolue Moyenne (EAM) : 6.67
2. Erreur Relative Moyenne (ERM) : 8.23%
3. Efficacité : 90.93%

Interprétation des Résultats

- Erreur Absolue Moyenne (EAM) : L'algorithme a une EAM de 6.67, ce qui signifie que, en moyenne, les résultats obtenus sont à 6.67 unités des valeurs optimales. Cela montre un écart vraiment pas conséquent entre les résultats attendus et obtenus.

- Erreur Relative Moyenne (ERM) : L'ERM est de 8.23%, ce qui indique que les résultats obtenus sont en moyenne à 8.23% des valeurs optimales. Cela donne une idée de l'écart par rapport aux valeurs optimales.

- Efficacité : L'efficacité de 90.93% montre que l'algorithme fonctionne très bien mais n'est pas parfait (ce qui est normal). Il atteint 90.93% de la performance idéale, laissant une marge de 9.07% pour des améliorations afin d'atteindre les valeurs optimales.

Conclusion des performances de l'algorithme heuristique:

Ces résultats montrent que l'algorithme est performant. L'écart observé est principalement dû aux résultats obtenus dans le scénario 3, où une valeur plus élevée a été trouvée (121 pas au lieu de 81, soit un écart de 30 pas). Le principe de l'algorithme heuristique n'est pas de trouver la solution optimale dans tous les cas, mais il doit tout de même pouvoir donner la solution optimale dans la plupart des scénarios. Ce qui est le cas de notre algorithme qui retourne les valeurs optimales dans 5 scénarios sur 6.
En résumé, après analyse et démonstration, notre algorithme heuristique est performant.

4- Conclusion

Le projet de l'Apprenti Ordonnateur nous a permis de développer une application interactive ayant pour objectif de réaligner des cristaux dans leurs temples respectifs après une tempête dans le royaume de Krystalia. Ce projet nous a offert l'opportunité d'explorer diverses approches algorithmiques, de créer une interface Homme-machine intuitive et de renforcer nos compétences en développement.

Bilan du Travail Réalisé:

Nous avons réussi à mettre en place les fonctionnalités essentielles de l'application, incluant le chargement de scénarios, la visualisation graphique de la carte et des déplacements de l'apprenti, ainsi que l'implémentation de deux algorithmes principaux : un algorithme de tri par sélection et un algorithme heuristique. Ces algorithmes permettent de réaligner les cristaux de manière efficace, avec des performances comparées et évaluées à l'aide d'un programme Python. De plus, nous avons ajouté quelques fonctionnalités supplémentaires. Nous avons ajouté un menu *quitter* pour sortir du jeu proprement. Nous avons aussi ajouté un menu *aide* qui affiche les règles du jeu quand on clique dessus afin de les avoir à disposition. Nous avons également implémenté des alertes servant à informer le joueur de différentes situations ou actions dans le jeu, assurant ainsi une expérience interactive et intuitive. Lorsque l'utilisateur tente de prendre un cristal d'un temple sans qu'il y ait de cristal disponible, ou lorsque l'apprenti porte déjà un cristal, une alerte s'affiche avec le message : *"Impossible de prendre. Aucun cristal disponible pour prendre."* De même, une alerte d'échange de cristaux est déclenchée si le joueur essaie d'échanger des cristaux entre l'apprenti et un temple sans que cela soit possible, avec le message : *"Impossible d'échanger. Aucun échange possible ici."* Une autre alerte informe le joueur lorsqu'il tente de déposer un cristal dans un temple sans succès, affichant : *"Impossible de déposer. Vous ne pouvez pas déposer de cristal ici."* Enfin, une alerte de fin de jeu apparaît lorsque tous les cristaux sont correctement alignés avec leurs temples respectifs, signalant la victoire de l'utilisateur avec le message : *"Partie Terminée. Félicitations ! Tous les cristaux sont alignés !"* Ces alertes utilisent des boîtes de dialogue JavaFX pour fournir des messages clairs et interactifs, aidant le joueur à comprendre les actions possibles de manière intuitive. Enfin nous avons aussi ajouté une *scrollbar* sur le côté de la map si jamais la taille de l'écran est réduite. Du style CSS (à partir du fichier styleApplication.css) a été appliqué afin de rendre l'application plus captivante.

Difficultées rencontrées:

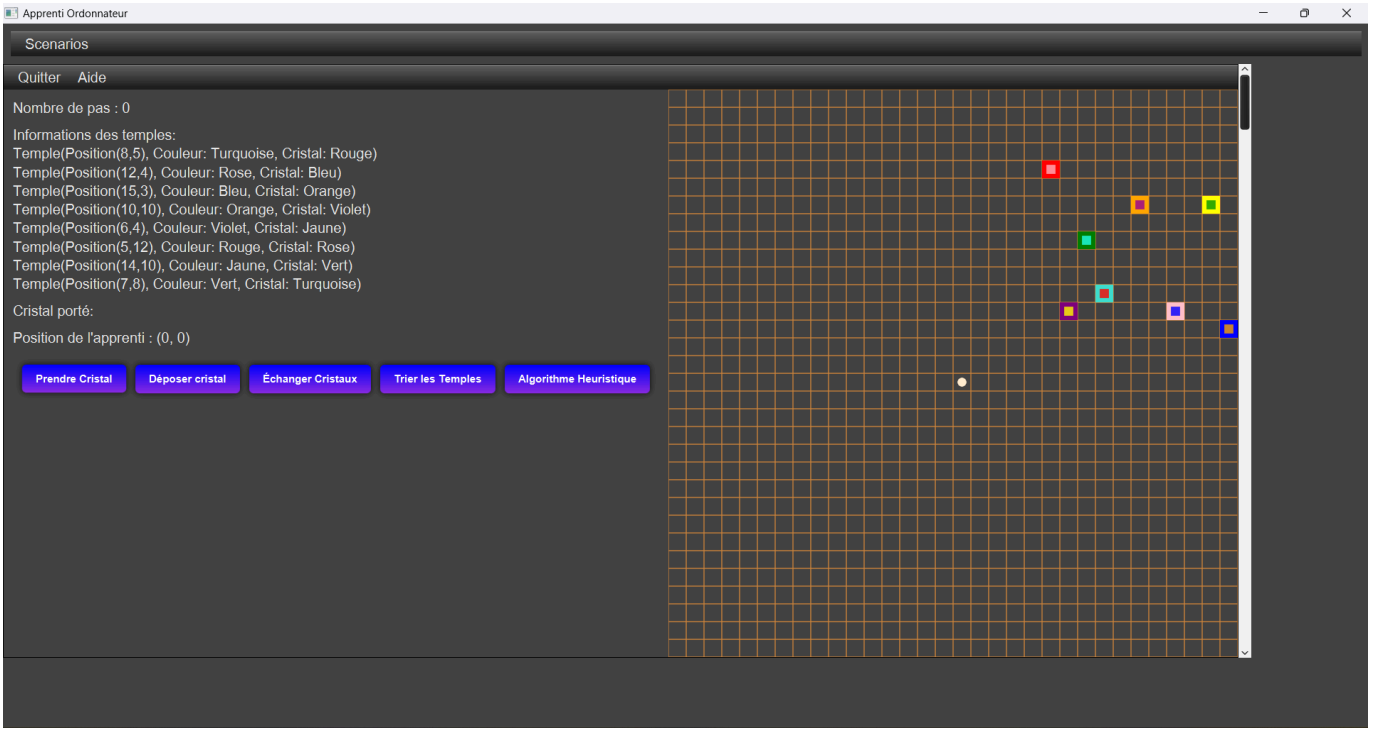
Nous avons rencontré de nombreuses difficultés au cours de ce projet. En effet, allier la partie logique et graphique fut difficile au début au vu de la grande quantité d'éléments à devoir lier. De plus, après avoir réussi à lier la partie logique et graphique, nous avons eu du mal à bien respecter la structure MVC tout au long même si au final cette dernière a été respectée comme il le faut. Enfin, l'implémentation de l'algorithme heuristique a été dans l'ensemble d'une grande difficulté en général.

Tâches Non Réalisées:
Néanmoins, certaines fonctionnalités n'ont pas pu être implémentées dans le cadre de ce projet. Notamment, le niveau 3 du projet qui consiste à implémenter un algorithme utilisant les graphes et l'algorithme de Dijkstra pour trouver la solution optimale n'a pas été réalisé. Cette tâche aurait permis d'améliorer encore davantage l'efficacité et de garantir des solutions optimales dans tous les scénarios.

Les perspectives d'Évolution:
Pour les futures évolutions de l'application, nous pourrions envisager d'intégrer l'algorithme de Dijkstra afin de trouver les chemins les plus courts pour l'apprenti et ainsi rendre parfait l'algorithme heuristique pour trouver la solution optimale. Cela permettrait non seulement de garantir des solutions optimales, mais aussi de renforcer notre compréhension et notre maîtrise des algorithmes de graphes.
De plus, l'ajout de fonctionnalités avancées, telles que la sauvegarde et le chargement des états de jeu, la génération aléatoire de scénarios et l'amélioration de l'interface Homme-machine avec des animations plus fluides, pourrait rendre l'application encore plus interactive et ludique.

En conclusion, ce projet nous a permis de développer des compétences solides en développement, en algorithmes et en conception d'interfaces graphiques. Nous avons appris à surmonter des défis techniques et à collaborer efficacement pour atteindre nos objectifs. Les perspectives d'évolution nous motivent à continuer à améliorer notre application et à explorer de nouvelles solutions algorithmiques pour des problèmes complexes.

Aperçu de l'application développé:



Interface graphique d'une partie de l'Apprenti Ordonnateur (scénario 2)

5- Annexes

5.1- Lien vers le dépôt GIT

Lien:

https://Amir_Nassiri@bitbucket.org/qualite-dev1/appordo.git