Université d'Angers

Faculté des Sciences

Master 1 Mathématiques et Applications

Parcours Data Science

Année Universitaire 2025/2026

RAPPORT DE PROJET

IMPLEMENTATION ET SIMULATION DU JEU "DEEP SEA ADVENTURE"

Présenté par :

DANIELE ISABELLE Nana Fotzeu

GEORGES Jean Constantin Bernard

KHAMASSI Mehdi

WADE Abdou

Sous la direction de : **DAVOT-GRANGE Tom**

Dans le cadre du module : **PROGRAMMATION ORIENTÉE OBJET**

Préface

Ce rapport présente le travail complet réalisé pour l'implémentation du jeu *Deep Sea Adventure* dans le cadre du cours de programmation orientée objet. Ce projet représente l'aboutissement de plusieurs semaines de développement collaboratif et illustre notre capacité à concevoir une application Python complexe en respectant les principes d'architecture logicielle modernes.

Remerciements

Nous remercions notre enseignant Tom Davot-Grange pour son accompagnement et ses conseils précieux. Nos remerciements vont également à nos familles pour leur soutien et à nos camarades pour la collaboration fructueuse.

Résumé

Ce projet consistait à implémenter le jeu *Deep Sea Adventure* en Python avec une architecture orientée objet. Nous avons développé un moteur de jeu complet, deux interfaces (CLI et GUI), et trois types d'intelligence artificielle. Le code est modulaire, documenté et testé.

Table des matières

1. Introduction

Le présent rapport décrit la conception et l'implémentation du jeu *Deep Sea Adventure* en Python, dans un cadre d'apprentissage centré sur la programmation orientée objet. L'objectif était de reproduire l'intégralité des mécaniques du jeu réel tout en développant une architecture modulaire permettant deux interfaces distinctes : une interface ASCII (CLI) et une interface graphique (GUI) basée sur PySide6.

1.1 Contexte et objectifs

Ce projet visait à implémenter le jeu *Deep Sea Adventure* en Python en appliquant les concepts de POO. Les objectifs étaient : créer un moteur de jeu fidèle, développer deux interfaces, implémenter une IA variée, et produire un code modulaire et testé.

1.2 Présentation du jeu

Deep Sea Adventure est un jeu de plongée où 2 à 6 joueurs partagent une réserve d'oxygène. Le but est de collecter des trésors et de revenir au sous-marin avant l'épuisement de l'air. Le jeu se déroule en trois manches.

2. Fonctionnalités implémentées

2.1 Logique du jeu

Toutes les règles principales sont implémentées : déroulement complet d'une partie (3 manches), mouvement des joueurs basé sur deux dés à 3 faces, pénalité de déplacement selon le poids des trésors, gestion complète de l'air partagé, système de ramassage et dépôt de trésors, compression du chemin et descente des trésors noyés, calcul des scores sécurisés en fin de manche et fin de partie.

2.2 Gestion des joueurs

Le système inclut des joueurs humains (*Player*) et des joueurs IA (*AIPlayer*) avec une logique d'exploration simple mais fonctionnelle. Chaque joueur gère ses trésors transportés, sécurisés, sa direction (descente/remontée) et son état (revenu ou non).

2.3 Limitations identifiées

L'intelligence artificielle suit des règles de décision simples et ne s'appuie pas sur des algorithmes complexes. L'interface graphique peut présenter des ralentissements sur machines anciennes. Le système de sauvegarde de parties n'a pas été implémenté, ce qui empêche de reprendre une partie interrompue. Le mode réseau multijoueur n'est pas disponible.

3. Interfaces utilisateurs

3.1 Interface en Ligne de Commande (CLI)

Le fichier *cli/cli_game.py* implémente une interface textuelle (ASCII) permettant de jouer au jeu sans interface graphique. Cette interface est entièrement séparée de la logique métier : elle se contente d'afficher l'état du jeu et de relayer les décisions des joueurs humains.

Caractéristiques techniques :

- Utilisation de méthodes dédiées du moteur (*get_board_ascii()* et *get_players_status_ascii()*) pour représenter le plateau et les joueurs sous forme de texte
- Déroulement strict des règles du jeu : affichage du numéro de manche et de l'air restant, affichage du plateau ASCII en "ligne", gestion complète des tours
- Choix interactifs pour les joueurs humains : direction (descendre/remonter), actions (ramasser/poser trésors)
- Support intégré des IA avec affichage de leurs décisions
- Fin de manche avec calcul et affichage des scores
- Fin de partie avec annonce des vainqueurs

Objectifs :

- Offrir une version minimale, légère et facilement testable du jeu
- Permettre le débogage de la logique métier indépendamment de l'interface graphique

- Fournir une alternative fonctionnelle sans dépendances graphiques

3.2 Interface Graphique (GUI)

L'interface graphique, située dans le dossier *gui/*, constitue une version moderne et interactive du jeu. Développée avec PySide6, elle permet de construire des interfaces complexes avec widgets, animations et styles visuels personnalisés.

Architecture générale :

- ***SetupWindow*** : Configuration d'une nouvelle partie (choix du nombre de joueurs, noms, humain/IA)
- ***GameWindow*** : Fenêtre principale du jeu (plateau, état du jeu, actions des joueurs)
- ***BoardWidget*** : Représentation graphique du plateau sous forme de grille en "serpent"
- ***SpaceWidget*** : Représentation visuelle d'une case du plateau (vide, trésor, sous-marin)
- ***EndGameDialog*** : Fenêtre affichée à la fin de la partie (scores et vainqueurs)

Structure MVC naturelle :

- **Modèle** : La logique métier (déplacements, air, trésors, règles) est entièrement gérée dans *src/game.py* et les classes associées
- **Vue** : L'interface graphique ne fait qu'afficher l'état actuel du jeu
- **Contrôleur** : Les événements de la GUI relaient les actions du joueur au moteur du jeu

Composants détaillés :

- **Fenêtre principale (*GameWindow*)** : Divisée en trois zones : bandeau supérieur (titre, manche, air), partie gauche (plateau et résumé des joueurs), partie droite (contrôles du joueur)
- **Interactions** : Le bouton "Jouer le tour" déclenche l'appel au moteur de jeu (*begin_turn()*) avec les paramètres sélectionnés, suivie d'une animation de dé et du rafraîchissement de l'interface
- **Représentation visuelle** : Couleurs par niveau de profondeur, affichage des valeurs de trésors, indicateurs visuels des joueurs
- **Gestion des IA** : Désactivation automatique des contrôles pour les joueurs IA qui décident automatiquement

Points forts :

- Interface moderne grâce à un style QSS personnalisé
- Organisation claire par classes et widgets spécialisés
- Forte séparation interface/logique de jeu
- Plateau dynamique et intuitif
- Animation du dé pour une meilleure immersion

- Interface facilement extensible

4. Architecture et conception

4.1 Modélisation de classes

L'architecture du jeu repose sur une hiérarchie de classes soigneusement conçue, organisée selon les principes de la programmation orientée objet. Les composants principaux et leurs responsabilités sont les suivants :

- ***Game* (Contrôleur principal)** : Classe centrale qui orchestre le déroulement d'une partie, gère les tours de jeu et les interactions entre les autres composants.
- ***Board* (Gestion du plateau)** : Représente le plateau de jeu. Elle est composée d'une collection d'objets *Space* et gère leur disposition.
- ***Space* (Case du plateau)** : Modélise une case individuelle sur le plateau. Elle peut contenir une collection d'objets *RuinTile*.
- ***RuinTile* (Jeton trésor)** : Représente un trésor (jeton de ruine) avec ses attributs, tels que sa valeur et son niveau.
- **Système de joueurs polymorphique** : La gestion des joueurs utilise une abstraction pour permettre des comportements variés :
    - ***Player*** : Classe de base ou interface définissant le contrat commun.
    - ***HumanPlayer*** : Implémentation pour un joueur humain, gérant les interactions via l'interface.
    - ***AIPlayer*** : Implémentation de base pour une intelligence artificielle, structurée selon le pattern Strategy pour encapsuler les algorithmes de décision. Trois sous-classes concrètes en dérivent, chacune correspondant à une stratégie spécifique.

4.2 Organisation modulaire

Le projet est structuré en packages modulaires assurant une séparation claire des responsabilités :

- ***Package src/* (Cœur du jeu)** :
    - ***game.py*** : Centralise la logique principale et le flux du jeu.
    - ***board.py*** et ***space.py*** : Gèrent la structure et les composants du plateau.
    - ***player.py*** et ***ai_player.py*** : Définissent les comportements des joueurs humains et intelligences artificielles.
    - ***ruin_tile.py*** : Modélise les trésors (jetons de ruines).
    - ***dice.py*** : Contrôle les mécaniques aléatoires des dés.
- ***Package gui/*** : Isole l'interface graphique dans cinq modules spécialisés.
- ***Package cli/*** : Contient une interface en ligne de commande autonome.

4.3 Principes de conception appliqués

Notre architecture applique de manière systématique les principes SOLID et des patrons de conception reconnus :

- **Responsabilité unique (S)** : Chaque classe a une fonction précise et limitée.
- **Ouvert/fermé (O)** : La hiérarchie d'IA est extensible sans modification du code existant.
- **Pattern Strategy** : Structure le système d'intelligence artificielle pour une interchangeabilité aisée des comportements.
- **Séparation des préoccupations** : La logique métier (*src/*) est strictement indépendante des interfaces utilisateur (*gui/, cli/*), garantissant maintenabilité et évolutivité.

5. Analyse critique

5.1 Points forts

Notre implémentation présente plusieurs atouts structurels et fonctionnels significatifs qui ont contribué à la qualité du projet.

- **Architecture modulaire** : La séparation claire des composants (logique métier, interface graphique, interface ligne de commande) facilite la compréhension, la maintenance et l'extension du code.
- **Fidélité aux règles** : Le respect scrupuleux des règles du jeu original a été validé par une série de tests unitaires approfondis.
- **Qualité du code** : Le projet se caractérise par une documentation complète (docstrings, README), un nommage explicite des variables et fonctions, et une adhésion aux conventions de style.
- **Interfaces multiples** : La présence de deux interfaces distinctes (graphique avec PySide6 et texte) répond à des cas d'usage variés et démontre la flexibilité de l'architecture.
- **Extensibilité prouvée** : Le système d'intelligence artificielle, basé sur le pattern Strategy, a permis l'ajout aisé de nouvelles stratégies de jeu sans modifier le code existant.

5.2 Points faibles

Malgré ses qualités, le projet présente certaines limitations identifiées lors du développement et de la phase de test.

- **Intelligence artificielle simpliste** : Les stratégies d'IA reposent sur des algorithme relativement simples et ne simulent pas une réflexion profonde ou une anticipation des actions des autres joueurs.

- **Couverture de tests incomplète** : Bien que les tests unitaires couvrent les fonctions principales, l'absence de tests d'intégration et de tests système limite la validation des interactions complexes.
- **Performances de l'interface graphique** : Certains rendus ou animations pourraient être optimisés pour une expérience encore plus fluide, notamment avec un grand nombre d'éléments.
- **Absence de système de sauvegarde** : Cette limitation fonctionnelle majeure empêche les joueurs d'interrompre une partie et de la reprendre ultérieurement. L'ajout de cette fonctionnalité aurait nécessité la conception d'une interface dédiée et d'un système de sérialisation des états du jeu.

5.3 Rétrospective constructive

Avec le recul du développement, plusieurs pistes d'amélioration architecturales et fonctionnelles se dégagent pour une version future.

- **Système de sauvegarde/chargement** : Implémenter une sérialisation de l'état du jeu aurait considérablement enrichi l'expérience utilisateur.
- **Système à base d'événements (Event-Driven)** : Une architecture basée sur des événements (pub/sub) aurait permis un découplage encore plus poussé entre la logique métier et les interfaces, favorisant la modularité.
- **Fichier de configuration externalisé** : Un tel système (en JSON ou YAML) aurait offert aux utilisateurs la possibilité de personnaliser des paramètres (niveau de l'IA, apparence) sans toucher au code.
- **Journalisation (Logging) structurée** : Mettre en place un système de logging avec différents niveaux de sévérité aurait grandement facilité le débogage et le suivi du comportement de l'application pendant le développement.

6. Extensibilité et perspectives

6.1 Améliorations prioritaires

La mise en place d'un système de sauvegarde constitue la priorité. Cette fonctionnalité permettrait aux joueurs d'interrompre une partie et de la reprendre ultérieurement. L'implémentation pourrait inclure des sauvegardes automatiques et manuelles avec gestion de plusieurs slots.

Un mode réseau multijoueur élargirait considérablement l'audience du jeu. Un éditeur de niveaux intégré autoriserait la création de plateaux personnalisés. L'ajout de statistiques détaillées répondrait aux joueurs compétitifs.

6.2 Améliorations techniques

L'intelligence artificielle pourrait évoluer vers des algorithmes plus sophistiqués comme MinMax. L'interface graphique gagnerait à supporter des thèmes visuels interchangeables.

L'expérience immersive serait enrichie par des bandes sonores et des animations supplémentaires.

Le système de sauvegarde pourrait utiliser la sérialisation JSON existante et l'enrichir d'une interface utilisateur dédiée. La persistance des données pourrait être améliorée avec un système de gestion de profils joueurs.

6.3 Perspectives à long terme

À plus long terme, le développement d'une application web permettrait une diffusion plus large. Une version mobile toucherait un public différent. Une API REST ouvrirait des possibilités d'intégration avec d'autres systèmes.

L'infrastructure pourrait évoluer vers une architecture microservices pour le multijoueur en ligne. Un système cloud de sauvegarde permettrait aux joueurs d'accéder à leurs parties depuis différents appareils.

7. Organisation du travail

7.1 Méthodologie et outils

Notre méthodologie a combiné des pratiques agiles adaptées. Nous avons travaillé par sprints de deux semaines avec des objectifs clairement définis. Les revues de code systématiques ont assuré une qualité homogène.

Git avec GitHub a géré le contrôle de version et la collaboration. Discord a permis la communication en temps réel. Les documents partagés ont centralisé la documentation et la planification.

7.2 Répartition des responsabilités

La première phase a mobilisé l'ensemble de l'équipe pour définir l'architecture commune. La phase de développement a été répartie en binômes spécialisés. Le développement des interfaces a suivi une approche similaire.

La phase finale de tests a impliqué tous les membres dans des sessions de test croisé. Cette organisation a permis de maintenir une avancée régulière tout en garantissant la cohérence globale.

7.3 Défis techniques surmontés

La complexité de certaines règles a constitué un défi significatif. La synchronisation entre le moteur de jeu et l'interface graphique a représenté un autre défi technique majeur.

L'absence initiale de système de sauvegarde a été compensée par une architecture permettant une implémentation ultérieure. Les méthodes de sérialisation existantes facilitent l'ajout de cette fonctionnalité.

8. Conclusion

En définitive, ce projet de développement nous a permis de concrétiser un jeu fonctionnel tout en consolidant des compétences techniques et collaboratives essentielles. La synthèse qui suit dresse un bilan de cette réalisation, en présente les prolongements possibles et propose une réflexion synthétique sur cette expérience formatrice.

8.1 Bilan technique et pédagogique

Ce projet représente une réussite sur les plans technique et pédagogique. Techniquement, nous avons produit un moteur de jeu robuste, deux interfaces fonctionnelles, et un système d'IA varié. L'architecture modulaire constitue une base solide.

Pédagogiquement, cette expérience nous a permis d'appliquer concrètement des concepts théoriques. La collaboration a développé nos compétences en communication technique et en résolution collective de problèmes.

8.2 Recommandations pour la continuité

L'implémentation d'un système de sauvegarde constitue la recommandation prioritaire. Cette fonctionnalité manquante limiterait l'adoption du jeu par des joueurs occasionnels. Son ajout améliorerait significativement l'expérience utilisateur.

L'enrichissement de l'intelligence artificielle avec des algorithmes plus sophistiqués augmenterait la profondeur stratégique. L'optimisation des performances de l'interface graphique garantirait une expérience fluide.

8.3 Réflexions finales

La réalisation de ce projet nous a confrontés aux réalités du développement logiciel. L'équilibre entre perfectionnisme technique et pragmatisme des délais a été une leçon formatrice.

La satisfaction de voir un produit abouti constitue la récompense la plus significative de nos efforts. Cette expérience renforce notre confiance dans notre capacité à mener à bien des projets logiciels complexes.

Postface

Ce projet *Deep Sea Adventure* a été une expérience formatrice qui nous a confrontés aux réalités du développement logiciel. Au-delà des compétences techniques acquises en Programmation Orientée Objet et architecture, nous retenons l'importance du travail d'équipe, de la planification et des compromis nécessaires entre perfectionnisme et livraison dans les délais.

Les limites identifiées, notamment l'absence de système de sauvegarde, représentent des pistes d'amélioration concrètes pour l'avenir. Cette expérience renforce notre confiance dans notre capacité à mener à bien des projets complexes et constitue une base solide pour nos futures réalisations professionnelles.