SORBONNE UNIVERSITY

Faculté des Sciences et Ingénierie

Parcours Science et Technologie du Logiciel (STL)

Conception et Pratique de l'Algorithmique (CPA) : Refonte d'une application de jeu vidéo - Frères Marius (Refonte de Mario Bros. 1983 - Level 1-1)

Étudiants :
Abdelkader Boumessaoud, N°21218306, abdelkader.boumessaoud@etu.sorbonne-universite.fr

Encadrants :
Binh-Minh Bui-Xuan, buixuan@lip6.fr
Arthur Escriou

7 mai 2024

Table des matières

1 Introduction

1.1 Objectif du projet

Ce projet vise à développer un clone du jeu *Super Mario Bros* (1983), en mettant l'accent sur une implémentation moderne tout en préservant les mécaniques et l'esthétique qui ont établi sa renommée. En particulier, le projet se concentre sur la reproduction fidèle des personnages, des éléments interactifs, et des environnements du jeu original.

Architecture logicielle : Le code est organisé en plusieurs répertoires reflétant différentes composantes du jeu :

- **Personnages :** Gestion des personnages via les classes `Character`, `Player`, et `Enemy`, permettant de modéliser les interactions entre le joueur et les adversaires.
- **Images et Items :** Stockage et gestion des ressources visuelles et des objets de jeu, assurant une fidélité graphique et une interaction immersive.
- **Carte et Mécaniques :** Définition des niveaux et implémentation des règles de jeu, garantissant une expérience qui respecte l'esprit du titre original.

L'objectif est de démontrer une compréhension approfondie des principes de développement de jeux vidéo par le biais d'une application concrète, en utilisant Java pour la programmation. Ce choix permet d'exploiter un langage performant et versatile, adapté aux exigences du développement de jeux.

1.2 Règles du jeu

Le jeu est un platformer de type Mario, où le joueur utilise les touches fléchées gauche et droite pour naviguer dans les niveaux et la touche espace pour sauter. Voici les règles principales du jeu :

- **Contrôles :** Utiliser les touches fléchées gauche et droite pour déplacer le personnage et la touche espace pour sauter.
- **Ennemis :** Deux types d'ennemis sont présents dans le jeu :
  - Turtles et Mushrooms - peuvent être éliminés ou évités par un saut dessus, une technique communément appelée "pogo jump".
- **Objectifs :** Le joueur doit ramasser 15 pièces dispersées à travers les différentes phases de plateformes du niveau.
- **Condition de victoire :** Pour gagner, le joueur doit collecter toutes les pièces et atteindre la fin du niveau avant l'expiration du temps imparti.

1.3 Analyse fonctionnelle détaillant les cas d'utilisation

1.3.1 Use Cases

Acteurs	Joueur
Préconditions	Le jeu est lancé, et le level commence.
Déroulement	1. Le jeu charge la map du monde level. 2. Le joueur navigue Mario à travers le niveau, en évitant les obstacles, collectant les pièces et en combattant ou esquivant les ennemis. 3. Le niveau se termine quand Mario atteint le château ou meurt.
Postconditions	Affichage du message de fin de partie, victoire ou échec.

1.3.2 User Stories

En tant que joueur, j'aimerais voir des animations fluides et des réponses immédiates aux commandes pour une expérience de jeu optimale.

En tant que joueur, j'aimerais voir mes scores et le temps restant de la partie.

En tant que joueur, je souhaite que le jeu soit équitable et que les mécaniques de jeu soient claires et cohérentes.

1.3.3 Maquette

FIGURE 1 – Maquette du jeu

1.4 Illustrations

Début du Niveau

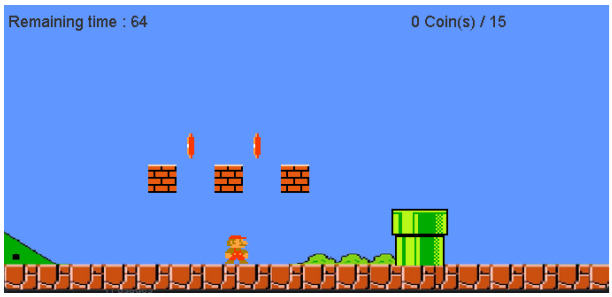


FIGURE 2 – Illustration du début du premier niveau

Cette illustration montre le début du premier niveau, où Marius apparaît à l'entrée du monde. Le décor est vibrant et accueillant, typique des premiers niveaux pour initier le joueur. L'image capture Marius prêt à entamer son aventure, avec quelques blocs de base et des pièces à collecter visibles. Cela établit le ton du jeu et présente une interface utilisateur montrant le score actuel et le temps restant.

Milieu du Niveau

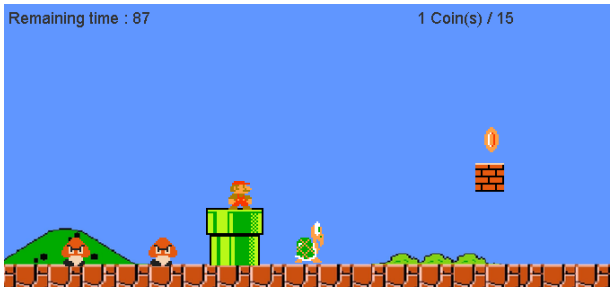


FIGURE 3 – Illustration du milieu du niveau

Cette image dépeint une scène plus complexe située au milieu du niveau. Elle inclut tous les éléments de gameplay : divers ennemis, obstacles et plateformes.

Écran d'Échec

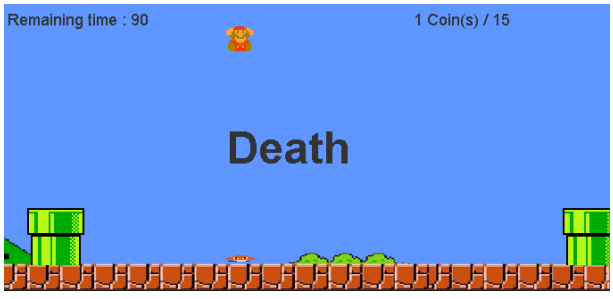


FIGURE 4 – Illustration de l'écran d'échec

L'écran de défaite présente une illustration de Mario en situation d'échec après avoir été touché par un ennemi, accompagnée d'un message affichant "Death".

Fin de Niveau et Victoire

FIGURE 5 – Illustration de la fin de niveau et de la victoire

Cette dernière illustration montre Mario atteignant le chateau à la fin du niveau, souvent une scène emblématique dans les jeux Mario, avec un message de victoire "Win" s'affichant, conditionné a l'obtention de toutes les pièces.

1.5 Technologies utilisées avec justification par un tableau comparatif

1.5.1 Java AWT (Abstract Window Toolkit)

Java AWT est une des bibliothèques GUI originales de Java, utilisée pour implémenter des interfaces utilisateur qui s'intègrent directement avec les composants GUI du système d'exploitation. Bien qu'AWT fournisse une interaction proche du système natif, ses composants sont considérés comme lourds et peuvent ne pas offrir la flexibilité requise pour des applications modernes, interactives et graphiquement intensives comme les jeux.

1.5.2 Java Swing

Par contraste, Java Swing est une bibliothèque GUI plus moderne et robuste, conçue pour fournir une suite complète de composants d'interface utilisateur légers qui ne dépendent pas directement du système d'exploitation sous-jacent. Cela permet une plus grande portabilité et une personnalisation approfondie de l'interface utilisateur, des aspects essentiels pour le développement de jeux sur différentes plateformes.

1.5.3 Tableau Comparatif des Technologies

Technologie	**Utilisation dans le Projet**	**Justification**
Java AWT	Potentiellement pour des outils de support	Adaptée pour les outils qui bénéficient de l'adhérence à l'interface native du système, tels que des éditeurs de niveau ou des outils de débogage spécifiques.
Java Swing	Gestion des interfaces du jeu	Privilégiée pour sa flexibilité, sa portabilité et ses capacités de personnalisation poussée, Swing permet de créer des interfaces utilisateurs riches et interactives, indispensables pour les éléments de contrôle de jeu, menus, et configurations dynamiques.

1.5.4 Conclusion

En conclusion, la sélection de Java Swing pour le développement des interfaces utilisateur du jeu est justifiée par sa supériorité en termes de personnalisation et de portabilité par rapport à Java AWT. Swing offre également une architecture plus moderne qui favorise la séparation des préoccupations via son modèle MVC, essentiel pour maintenir la modularité et l'évolutivité du code dans les projets de développement de jeux. Java AWT, bien qu'utile pour certaines fonctionnalités spécifiques nécessitant une intégration système étroite, est généralement réservé pour des cas d'utilisation plus spécialisés au sein de ce projet.

1.6 Bilan de ressources déployées pour le projet

1.6.1 Ressources Humaines

Développeur de logiciels : En tant que seul développeur, la nécessité de naviguer entre différentes responsabilités et exigences technologiques a exigé de la polyvalence.

1.6.2 Ressources Technologiques

Environnement de développement : IntelliJ IDEA a été choisi comme IDE principal en raison de ses capacités avancées pour le développement Java, incluant la gestion des dépendances et l'intégration avec les systèmes de gestion de version. Cela a permis de maintenir une efficacité dans le codage et la gestion du projet malgré les changements d'architecture et les interruptions.

Changement initial de technologie : Le projet avait initialement envisagé l'utilisation de JavaFX pour ses fonctionnalités étendues en matière de GUI et de graphiques riches. Cependant, des ajustements d'architecture ont conduit à l'adoption de Java Swing, adapté aux exigences spécifiques et à la complexité du projet.

Système de gestion de versions : GitHub a joué un rôle crucial, facilitant la gestion des différentes versions du projet, surtout lors des transitions entre les architectures et les interruptions dues à d'autres projets universitaires.

1.6.3 Ressources Temporelles

Interruptions et gestion du temps : Les interruptions fréquentes pour des raisons académiques et les changements d'architecture ont significativement affecté le calendrier du projet. La gestion du temps a été particulièrement critique, nécessitant une adaptation constante pour balancer le développement du jeu avec d'autres engagements.

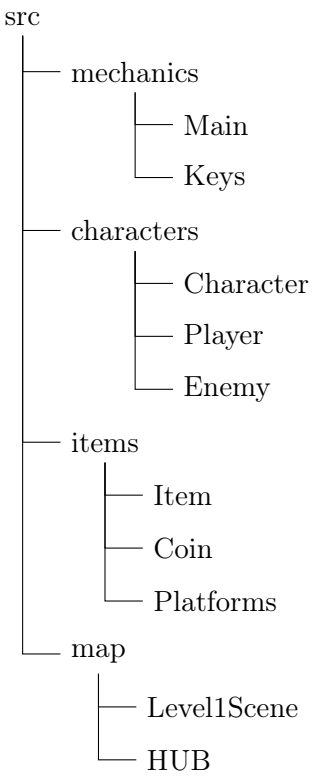
1.6.4 Scrum Backlog

Liste des fonctionnalités, tâches, et corrections à apporter, classées par priorité.

Backlog Item	**Notes et Priorité**
Implémentation de la gestion des entrées clavier via KeyListener	**Priorité :** Haute. Essentiel pour une interaction fluide dans le jeu.
Développement des algorithmes de déplacement et collision	**Priorité :** Haute. Crucial pour les interactions entre les sprites et la gestion des collisions.
Optimisation des performances pour réduire les drops de frames	**Priorité :** Moyenne. Nécessaire pour améliorer la fluidité du jeu, surtout dans les niveaux chargés.
Amélioration de la fidélité graphique et fonctionnelle	**Priorité :** Moyenne. Balancer innovation et respect de la nostalgie du jeu original.
Création de nouveaux niveaux et amélioration de l'interface utilisateur	**Priorité :** Basse. Pour étendre le contenu du jeu et améliorer l'expérience utilisateur.
Révision et test des mécanismes de jeu pour garantir l'équité	**Priorité :** Moyenne. Assurer que les mécaniques de jeu sont claires et cohérentes.
Sprint Planifié	**Objectifs**
Sprint 1	Focalisation sur les entrées utilisateur et les mécanismes de base du jeu.
Sprint 2	Développement des fonctionnalités de collision et de déplacement avancé.
Sprint 3	Optimisation des performances et développement des niveaux supplémentaires.

2 Partie technique

2.1 Architecture générale de l'application



Composants et Structures de Données

Package mechanics **Main :** Sert de point d'entrée de l'application. Cette classe configure le JFrame principal, définissant des propriétés telles que la taille, la visibilité, et les paramètres

de fermeture. Le cœur de cette classe est la création et l'initialisation de Level1Scene, qui est ajoutée au JFrame comme contenu principal.

Keys : Implémente l'interface KeyListener, permettant de gérer les entrées clavier. Les méthodes keyPressed, keyReleased, et keyTyped sont utilisées pour modifier les attributs de déplacement du Player basés sur les interactions de l'utilisateur, affectant directement la logique de déplacement dans Level1Scene.

Package characters **Character :** Classe abstraite définissant les attributs communs et les méthodes de base pour tous les personnages, tels que les coordonnées x, y, les dimensions width, height, et les états walk, alive. Des méthodes comme walk() et movement() fournissent des implémentations génériques pour la gestion des animations et des déplacements.

Player : Étend Character. Gère les états spécifiques du joueur comme jump, et inclut des méthodes pour les animations spécifiques et la gestion des interactions avec d'autres objets (contact()). Utilise une gestion d'état pour contrôler les animations et les réponses aux interactions du jeu.

Enemy : Étend Character et implémente Runnable pour gérer les déplacements autonomes via un thread séparé. Cela permet aux ennemis de se déplacer et d'interagir avec l'environnement de manière asynchrone par rapport à la boucle principale du jeu.

Package items **Item :** Définit la base pour des objets interactifs statiques et mobiles dans le jeu. Chaque Item possède une position, des dimensions, et une image représentative. Les méthodes comme movement() sont utilisées pour ajuster la position relative des objets non-joueur en fonction des actions du joueur.

Coin : Spécialisation de Item, inclut une animation simple gérée par un changement d'image pour simuler le clignotement.

Platforms : Représente les plates-formes statiques sur lesquelles le joueur peut se déplacer.

Package map **Level1Scene :** Agit comme un conteneur pour tous les objets du jeu. Cette classe hérite de JPanel et intègre des éléments tels que Player, Enemy, et Item. Gère le rendu, les déplacements de fond, et les interactions entre les objets. Utilise des listes (ArrayList) pour gérer dynamiquement les collections d'ennemis, de pièces, et d'autres items.

Mécanismes de Programmation et Patterns de Conception

Modularité et encapsulation : Chaque classe a des responsabilités clairement définies avec une interaction minimaliste entre elles, permettant une maintenance et une évolutivité facilitées.

Gestion des événements et des entrées : L'utilisation du pattern Observer via l'implémentation de KeyListener permet de réagir aux actions de l'utilisateur de manière efficace et isolée.

Multithreading pour les ennemis : L'utilisation de threads pour gérer le comportement des ennemis (Enemy) permet de simplifier la logique de mouvement sans bloquer la boucle principale de rendu.

Animation et gestion du temps : Techniques de buffering d'images et de gestion du temps pour créer des animations fluides et réactives.

Gestion des Ressources Graphiques : Les ressources graphiques sont gérées à travers des instances de ImageIcon, converties en Image pour un affichage optimisé dans le contexte du Graphics de Java Swing. Cela permet de maintenir une haute performance lors des redessinements fréquents du JPanel.

2.2 Description de la couche client

Architecture de la Vue

Le jeu *Frères Marius* est structuré autour d'une application Java utilisant Swing pour gérer l'interface utilisateur et le rendu graphique. La couche client est principalement incarnée par la classe `Level1Scene`, qui est un composant Swing (`JPanel`) où tous les éléments du jeu sont dessinés.

Gestion des Fenêtres

Le jeu utilise une seule fenêtre principale, créée et configurée dans la classe `Main` :

- **Dimensions :** La fenêtre est définie avec des dimensions statiques de 700x360 pixels.
- **Comportement :** Elle est configurée pour se fermer lorsque l'utilisateur envoie l'ordre de fermeture, ne peut pas être redimensionnée, est positionnée au centre de l'écran, et reste toujours au-dessus des autres fenêtres.

Rendu Graphique

La méthode `paintComponent` de `Level1Scene` est le cœur du rendu graphique dans le jeu. Elle est appelée automatiquement par le système de peinture Swing à chaque fois que le composant doit être redessiné. Voici les principales fonctionnalités de cette méthode :

- **Fond et Décor :** Images de fond qui défilent pour créer un effet de mouvement (scrolling) à mesure que le joueur avance dans le niveau. Les coordonnées de ces images sont ajustées en fonction de la position du joueur pour simuler le déplacement dans l'environnement du jeu.
- **Objets Interactifs :** Les blocs (`Platforms`), les pièces (`Coin`), et les ennemis (`Enemy`) sont dessinés sur le panel. Leur position est également ajustée dynamiquement pour correspondre au déplacement du fond.
- **Personnage Principal et Ennemis :** Les animations du personnage joueur et des ennemis sont gérées par des modifications des images en fonction de leur état (marche, saut, etc.). Les interactions telles que les collisions avec les objets ou les ennemis sont vérifiées et traitées pour refléter les réactions appropriées comme l'arrêt ou la mort du personnage.

Interaction et Événements

La classe `Keys` implémente l'interface `KeyListener` pour gérer les entrées au clavier, permettant au joueur de contrôler le personnage principal à travers des commandes simples (mouvement à droite ou à gauche et saut).

2.3 Description de la couche data

Gestion des Données en Mémoire

Le jeu utilise principalement une gestion en mémoire des états de jeu, sans recourir à des bases de données externes ou des systèmes de fichiers persistants pour stocker les informations relatives à l'état du jeu entre les sessions. Les principales données gérées incluent :

- **Hitboxes :** Chaque objet dans le jeu, y compris les personnages et les éléments de décor tels que les blocs et les tuyaux, possède une hitbox définie par les attributs *width*, *height*, x, et y. Ces hitboxes sont essentielles pour la détection de collision et sont recalculées en temps réel à chaque cycle de jeu.
- **Nombre de pièces collectées (Coins) :** Le nombre de pièces collectées est stocké dans l'attribut *nbrCoins* de l'instance de la classe HUB. Ce compteur est incrémenté à chaque fois qu'un personnage entre en collision avec une pièce et la collecte.
- **Temps restant :** Le compte à rebours du temps restant est géré par un thread dans la classe HUB, où *countTime* est décrémenté à intervalles réguliers. Ce temps est affiché à l'écran et mis à jour pour refléter le temps restant jusqu'à la fin du jeu ou la perte de la partie.

Absence de Persistance Externe
Aucune technologie de stockage externe n'est utilisée pour la persistance des données du jeu. Toutes les informations sont perdues lorsque l'application est fermée, car elles sont uniquement conservées en mémoire vive pendant l'exécution du jeu. Cette approche simplifie la structure du programme mais limite les fonctionnalités liées à la sauvegarde et à la récupération des états de jeu pour les sessions futures.

2.4 Explication de la couche serveur

Architecture et Conception
Le jeu *Frères Marius* est architecturé autour d'un modèle de programmation orienté événements et de manipulation d'objets dans un environnement 2D. Bien que structuré principalement comme un jeu client, certains concepts de "serveur" dans un sens logique peuvent être identifiés dans la gestion des données et des règles de jeu.

Gestion des Événements
La logique centrale repose sur des événements de clavier capturés par la classe `Keys` qui modifie l'état du jeu selon les entrées de l'utilisateur. Par exemple, la gestion des déplacements du personnage (à droite, à gauche et sauter) est implémentée ici. Ces interactions sont ensuite reflétées dans l'interface graphique par la mise à jour des positions des objets et des personnages.

Algorithmes de Déplacement et de Collision
La classe `Character` et ses dérivés (`Player`, `Enemy`) contiennent des méthodes pour le déplacement (*movement*) et la gestion des collisions (*contactAhead*, *contactBehind*, *contactDown*, *contactUp*). Ces méthodes utilisent des calculs de rectangles englobants pour déterminer les interactions entre les personnages et les obstacles ou autres personnages.

- **Déplacement :** La position du personnage est ajustée en fonction de la direction et de la présence d'obstacles. Le déplacement est conditionné par la vérification des limites de la scène et les positions relatives des objets interactifs.
- **Collision :** Les collisions sont déterminées par la superposition des zones occupées par les objets et personnages. Les réponses aux collisions incluent des ajustements de position, des changements d'état comme arrêter le mouvement, initier un saut, ou déclencher des animations de mort.

Gestion de l'État du Jeu
Le `Level1Scene` agit comme le contrôleur principal où tous les objets du jeu sont initialisés, et où la logique de mise à jour de l'état du jeu est exécutée. Cette classe gère également les éléments suivants :

- **Rendu Graphique :** Dessin de tous les composants visuels à chaque rafraîchissement du jeu.
- **Chronométrage et Score :** Suivi du temps restant et des pièces collectées à travers la classe `HUB`, qui exécute une boucle dans un thread séparé pour décrémenter le temps et mettre à jour l'affichage.

Optimisations et Performances
La performance du jeu est assurée par l'utilisation judicieuse des threads pour les animations et la mise à jour de l'état du jeu, permettant une expérience de jeu fluide sans bloquer l'interface utilisateur principale.

3 Conclusion et retour d'expérience

Le projet *Frères Marius*, une réinterprétation moderne de *Mario Bros.* de 1983, a été une formidable occasion de démontrer la puissance et la flexibilité de Java dans le développement de jeux vidéo. En utilisant Java AWT et Swing, j'ai pu recréer fidèlement l'esthétique et le gameplay

qui ont fait le succès du jeu original, tout en les adaptant aux normes modernes de développement logiciel.

3.1 Apprentissages Techniques

— **Gestion des Interactions et Événements :** L'implémentation de l'écoute des entrées du clavier via `KeyListener` a permis une gestion fluide des commandes, essentielle pour une expérience de jeu interactive.

— **Algorithmes de Déplacement et Collision :** Les techniques de collision et de mouvement ont été cruciales. Elles ont nécessité une compréhension précise des mathématiques derrière les interactions entre les sprites, permettant une simulation réaliste des mouvements et des réactions des personnages et objets.

3.2 Défis Rencontrés

— **Optimisation des Performances :** Assurer que le jeu reste fluide et réactif malgré la complexité des animations et la gestion simultanée de multiples objets a été un défi. L'utilisation de threads pour les ennemis et la gestion fine du rafraîchissement de l'interface graphique ont été des solutions clés.

— **Résolution des Drops de Frames :** Les chutes de fréquence d'images devenaient particulièrement notables vers la fin du niveau lorsque de nombreux éléments étaient présents à l'écran. Des ajustements dans le moteur de rendu et une optimisation des ressources graphiques ont été nécessaires pour minimiser ces incidents.

— **Fidélité Graphique et Fonctionnelle :** Recréer un jeu aussi iconique tout en restant fidèle à l'original a nécessité un équilibre délicat entre innovation et respect de la nostalgie et des mécanismes classiques.

3.3 Perspectives Futures

Cette expérience m'ouvre plusieurs pistes d'amélioration et d'extension du jeu, notamment l'ajout de nouveaux niveaux et l'amélioration de l'interface utilisateur pour rendre les menus et les écrans de jeu plus interactifs et visuellement attractifs.

Références

[1] *«Refonte d'une application de jeu vidéo»*. BM. Bui-Xuan et A. Escriou. https://www-npa.lip6.fr/~buixuan/files/cpa2023/projetCPA.pdf.

[2] *«Qualité d'un simulateur»*. BM. Bui-Xuan. https://www-npa.lip6.fr/~buixuan/files/cpa2023/cpa2023_tme6.pdf.

[3] *«javax.swing (Java Platform SE 7).»*. Oracle. https://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.htmlf.

[4] *«java.awt (Java Platform SE 7).»*. Oracle. https://docs.oracle.com/javase%2F7%2Fdocs%2Fapi%2F%2F/java/awt/package-summary.html.

[5] *«Super Mario Bros. Sprites»*. https://www.videogamesprites.net/SuperMarioBros1/.