Carte des chapitres

Figure 1: Carte des chapitres

L'Épée du Samouraï

1. Analyse du projet

1.1 Structure du projet

Le jeu "L'Épée du Samouraï" est une adaptation numérique d'un livre-jeu, développée en Java selon l'architecture MVC (Modèle-Vue-Contrôleur) :

- **Modèle** : Représente les données du jeu (chapitres, personnage, ennemis)
- **Vue** : Interface utilisateur (console ou graphique)
- **Contrôleur** : Gestion de la logique du jeu

1.2 Composants principaux

Modèle

- `Chapitre` : Représente un chapitre du livre avec son texte et ses choix
- `Personnage` : Gère les statistiques et l'inventaire du joueur
- `Enemy` : Définit les caractéristiques des ennemis
- `Scenario` : Organise l'ensemble des chapitres du jeu

Vue

- `SamuraiSwingUI` : Interface graphique principale avec style samouraï
- `CombatUI` : Interface de combat
- `ChapterMapUI` : Carte des chapitres montrant la progression

Contrôleur

- `GameController` : Gère la progression du joueur dans l'histoire
- `ScenarioLoader` : Charge les scénarios de jeu

1.3 Fonctionnalités implémentées

- **Navigation narrative** : Progression à travers différents chapitres
- **Système de combat** basé sur les règles du livre original
- **Gestion d'inventaire** et utilisation d'objets
- **Carte des chapitres** pour visualiser la progression
- **Système de statistiques** (habileté, endurance, etc.)
- **Sauvegarde/Chargement** de partie

1.4 Captures d'écran

La carte des chapitres (ci-dessus) permet aux joueurs de visualiser leur progression. Le cercle rouge indique le chapitre actuel.

2. Présentation du jeu

2.1 L'histoire

"L'Épée du Samouraï" est un jeu d'aventure textuel inspiré des livres-jeux. Vous incarnez un samouraï en quête de l'épée légendaire de votre ancêtre, connue sous le nom de "Mort Joyeuse".

Le Shogun Kihei Hasekawa vous a confié la mission de retrouver cette épée qui a été dérobée par Ikiru, le Maître des Ombres. Sans cette épée, le Shogun perd peu à peu son autorité, et des seigneurs félons commencent à se rallier à Ikiru.

Votre aventure vous conduira à travers des forêts mystérieuses, des villages abandonnés et des montagnes dangereuses, jusqu'au repaire d'Ikiru dans le Gouffre des Démons.

2.2 Mécaniques de jeu

- **Système de chapitres** : L'histoire se déroule à travers plusieurs chapitres avec des choix multiples
- **Combats tactiques** : Affrontez des ennemis en utilisant votre habileté et votre endurance
- **Objets et inventaire** : Collectez et utilisez des objets pour vous aider dans votre quête
- **Statistiques variables** : Vos caractéristiques évoluent selon vos choix et les combats
- **Plusieurs fins possibles** : Vos décisions influencent le déroulement et la conclusion de l'histoire

2.3 Interface utilisateur

Le jeu propose deux interfaces : - Une interface textuelle pour les puristes - Une interface graphique élaborée dans le style samouraï pour une expérience immersive

3. Installation et exécution

3.1 Prérequis

- Java Development Kit (JDK) version 21
- Un environnement capable d'exécuter des applications Java Swing

3.2 Installation

1. Téléchargez le code source depuis le dépôt GitHub ou utilisez la version compilée
2. Si vous utilisez le code source, assurez-vous que la structure des dossiers est préservée

3.3 Exécution

Pour lancer le jeu :

1. Ouvrez le fichier `src/Main.java` dans votre IDE ou éditeur de texte.

Capture d'écran d'interface

Figure 2: Capture d'écran d'interface

2. Exécutez la fonction `main` de la classe `Main`.

Vous pouvez également utiliser la ligne de commande :

```
cd /home/abdulmalek/Documents/Projects/Java-Mini-Project
javac src/Main.java
java -cp src Main
```

Assurez-vous d'avoir Java installé (JDK 21 recommandé).

3.4 Configuration requise

- **Système d'exploitation** : Windows 10/11, macOS 10.15+, ou Linux récent
- **Processeur** : Dual Core 2GHz ou plus
- **Mémoire** : 4 Go RAM minimum
- **Espace disque** : 100 Mo disponibles
- **Affichage** : Résolution minimum de 1024x768
- **Java Runtime Environment (JRE)** : Version 21 ou ultérieure

5. Développement du projet

5.1 Prompts utilisés pour le développement avec l'IA

Le développement de ce projet a été assisté par l'IA en utilisant les prompts suivants:

1. **Architecture du projet** :

   ```
   Cree l'archi de ce projet, le nom de livre est l'épée de samouraï
   ```

2. **Interface graphique inspirée de Ghost of Tsushima** :

   ```
   add a menu when chosing the graphcal interface based of the screenshot that i gave an
   ```

3. **Animation de fond d'écran** :

   ```
   i have an animated wallpaper its a mp4 video called wandering-samurai-no-way-home-moe
   ```

4. **Dialogue de sélection d'interface** :

   ```
   instead of giving the choice in the terminal you make a small popup using the same th
   ```

5. **Système de combat** :

   ```
   implement a combat system based on the photo i gave you for the ui, in the first chap
   ```

5.2 Approche de développement "Vibe Coding"

Le projet a été développé en utilisant l'approche "vibe coding", qui consiste à:

1. **Conception itérative** basée sur des screenshots et références visuelles
2. **Développement orienté ambiance** pour créer une atmosphère immersive de samouraï

3. **Utilisation de prompts** pour guider le développement avec l'aide de l'IA
4. **Focus sur l'expérience utilisateur** en privilégiant le style visuel et l'ergonomie

Cette approche a permis de créer une interface cohérente avec l'univers du jeu et d'offrir une expérience utilisateur immersive dans l'esprit des jeux de samouraï comme Ghost of Tsushima.

6. Comment continuer le développement du jeu

Ce jeu est actuellement en cours de développement avec certains chapitres non finalisés. Si vous souhaitez contribuer à son achèvement, cette section détaille la méthodologie à suivre pour intégrer harmonieusement votre contribution au projet existant.

6.1 Structure narrative détaillée

Le jeu "L'Épée du Samouraï" suit fidèlement la structure du livre-jeu original, avec une architecture narrative adaptée au format interactif :

1. **Organisation des chapitres** :
   - Chaque section du livre correspond à un objet `Chapitre` dans le code
   - Les chapitres sont numérotés selon la même logique que le livre original
   - Les choix du joueur déterminent le flux narratif entre les chapitres
2. **Éléments à reproduire** :
   - Texte narratif (descriptions, dialogues)
   - Options de choix et leurs conséquences
   - Tests d'attributs (Habileté, Endurance, Chance)
   - Rencontres avec des ennemis et mécaniques de combat
   - Acquisition et utilisation d'objets
3. **Adaptation au format numérique** :
   - Ajout d'illustrations et d'éléments visuels correspondant au texte
   - Effets sonores appropriés aux situations (combats, découvertes)
   - Indicateurs visuels pour les statistiques du personnage

6.2 Guide complet pour ajouter un nouveau chapitre

Pour intégrer un nouveau chapitre au jeu, suivez ce processus détaillé :

Étape 1 : Analyse du livre-jeu

1. Identifiez la section du livre que vous souhaitez implémenter
2. Notez tous les choix possibles et leurs destinations
3. Identifiez les combats, tests d'attributs et objets spéciaux
4. Repérez les connexions avec les chapitres déjà implémentés

Étape 2 : Préparation du code

1. Ouvrez le fichier `controller/ScenarioLoader.java`

2. Examinez les méthodes existantes (`creerScenarioChapitre1()`, `creerScenarioChapitre2()`) pour comprendre le style et la structure
3. Créez une nouvelle méthode pour votre chapitre (exemple : `creerScenarioChapitreX()`)

Étape 3 : Implémentation du chapitre Suivez ce modèle détaillé pour l'implémentation :

```
/**
 * Crée un scénario pour le chapitre X
 *
 * @return Un nouveau scénario contenant les sections du chapitre X
 */
public static Scenario creerScenarioChapitreX() {
    try {
        System.out.println("Création du scénario du chapitre X...");

        Scenario scenario = new Scenario("L'Épée du Samouraï - Chapitre X: [Titre]",
                "[Description détaillée du chapitre avec contexte narratif]");

        // Section 1 - Début du chapitre
        Chapitre chapitre1 = new Chapitre(1, "[Titre évocateur]",
                "[Texte narratif détaillé avec descriptions, dialogues et ambiance]\n\n" +
                "[Suite du texte avec mise en situation et éléments importants]\n\n" +
                "[Présentation des choix possibles ou de la situation à résoudre]", false)

        // Section 2 - Suite de l'aventure après un choix
        Chapitre chapitre2 = new Chapitre(2, "[Titre de la section]",
                "[Conséquences du choix précédent et nouvelle situation]\n\n" +
                "[Développement de l'intrigue avec nouveaux éléments narratifs]", false);

        // Section finale ou importante - Potentiellement une fin (victoire ou défaite)
        Chapitre chapitreFinal = new Chapitre(10, "Fin du parcours",
                "[Description de la conclusion de ce chapitre et transition vers le suivan

        // Définition des choix pour chaque section
        chapitre1.ajouterChoix(new Choix("[Description détaillée du premier choix]", 2, fa
        chapitre1.ajouterChoix(new Choix("[Description détaillée du second choix]", 3, fal
        chapitre1.ajouterChoix(new Choix("[Option cachée ou disponible sous condition]", 4

        // Définition des ennemis (avec statistiques équilibrées)
        Enemy ennemi = new Enemy(
            "[Nom caractéristique]",          // Nom
            7,                                // Habileté (entre 5-12 selon difficulté)
            10,                               // Endurance (entre 6-15 selon importance)
            "[Description physique détaillée et comportement au combat]"  // Description
        );
        chapitre2.setEnemy(ennemi);

        // Configuration des chapitres de défaite
```

```
        Chapitre chapitreDefaite = new Chapitre(99, "Défaite",
                "[Description de la défaite contre l'ennemi avec conséquences narratives]"
        chapitre2.setChapitreDefaite(chapitreDefaite);

        // Ajout de tous les chapitres au scénario
        scenario.ajouterChapitre(chapitre1);
        scenario.ajouterChapitre(chapitre2);
        // [Ajouter tous les autres chapitres]
        scenario.ajouterChapitre(chapitreFinal);
        scenario.ajouterChapitre(chapitreDefaite);

        return scenario;
    } catch (Exception e) {
        System.err.println("Erreur lors de la création du scénario du chapitre X: " + e.ge
        e.printStackTrace();
        return null;
    }
}
```

Étape 4 : Intégration au jeu

1. Ouvrez la classe `Main.java`
2. Localisez la méthode `preloadScenarios()`
3. Ajoutez l'appel à votre nouvelle méthode :

```
private static void preloadScenarios() {
    // Scénarios existants
    ScenarioLoader.creerScenarioChapitre1();
    ScenarioLoader.creerScenarioChapitre2();
    // Votre nouveau scénario
    ScenarioLoader.creerScenarioChapitreX();
}
```

Étape 5 : Création d'éléments visuels associés

1. Sélectionnez ou créez des images d'ambiance correspondant à votre chapitre
2. Placez-les dans `src/Assets/Photos/`
3. Référencez-les dans l'interface utilisateur lors de l'affichage du chapitre

6.3 Lier les chapitres existants avec précision

Pour créer une progression narrative fluide entre les chapitres existants et nouveaux :

1. **Analyse des points de connexion** :
   - Ouvrez les fichiers de chapitres existants dans `controller/ScenarioLoader.java`
   - Identifiez les chapitres de fin ou points de transition (par exemple, `chapitre6` dans le premier scénario)
   - Repérez les attributs `estFin` mis à `true` dans les chapitres existants
2. **Modification des chapitres de transition** :

```
// Exemple de modification d'un chapitre de fin existant pour créer une transition
Chapitre chapitreTransition = scenario.getChapitre(6); // Récupérer le chapitre de tr
chapitreTransition.setEstFin(false); // Le marquer comme non final
chapitreTransition.ajouterChoix(new Choix(
    "Continuer votre quête vers les montagnes de l'est",
    1, // ID du premier chapitre de votre nouveau scénario
    false
));
```

3. **Création d'un système de transition entre scénarios** :

```
// Dans GameController.java - Ajouter une méthode pour gérer les transitions
public void transitionVersScenario(String nomScenario, int chapitreId) {
    // Charger le nouveau scénario
    Scenario nouveauScenario = ScenarioLoader.chargerScenario(nomScenario);
    // Définir le nouveau scénario comme scénario actuel
    this.scenario = nouveauScenario;
    // Naviguer vers le chapitre spécifié
    naviguerVersChapitre(chapitreId);
    // Mettre à jour l'interface
    notifierObservateurs();
}
```

6.4 Enrichissement approfondi du gameplay

Pour améliorer l'expérience de jeu au-delà de la simple narration :

6.4.1 Système de combat avancé Implémentez les techniques martiales mentionnées dans le livre :

1. **Kyujutsu (Art de l'arc)** :

```
// Dans la classe Combat.java
public void utiliserKyujutsu(Personnage joueur, Enemy ennemi) {
    // Le joueur attaque à distance avant le combat rapproché
    int degats = calculerDegatsKyujutsu(joueur);
    ennemi.reduireEndurance(degats);

    // Notification visuelle et sonore
    notifierObservateurs("kyujutsu", degats);
}

private int calculerDegatsKyujutsu(Personnage joueur) {
    // Dégâts basés sur l'habileté et le niveau de Kyujutsu du joueur
    return Math.min(2, joueur.getNiveauKyujutsu()) +
           lancerDes(2); // 2D6 pour l'élément aléatoire
}
```

2. **Iaijutsu (Art du dégainage rapide)** :

```
// Implémentation de l'initiative au combat basée sur l'Iaijutsu
public boolean obtenirInitiative(Personnage joueur, Enemy ennemi) {
    int seuilInitiative = 7 - joueur.getNiveauIaijutsu(); // Seuil plus bas = plus fa
```

```
        return lancerDes(2) >= seuilInitiative; // 2D6 >= seuil
    }
```

3. **Karumijutsu (Agilité)** :

```
    // Possibilité d'esquiver les attaques ennemies
    public boolean tenterEsquive(Personnage joueur) {
        int seuilEsquive = 8 - joueur.getNiveauKarumijutsu();
        return lancerDes(2) >= seuilEsquive;
    }
```

4. **Ni-to-Kenjutsu (Combat à deux sabres)** :

```
    // Attaques multiples avec deux sabres
    public void attaqueDoubleSabre(Personnage joueur, Enemy ennemi) {
        if (joueur.possedeDualKatana()) {
            // Première attaque
            resoudreTourCombat(joueur, ennemi);

            // Seconde attaque si niveau suffisant
            if (joueur.getNiveauNiToKenjutsu() >= 2 && ennemi.estVivant()) {
                resoudreTourCombat(joueur, ennemi);
            }
        }
    }
```

6.4.2 Système d'inventaire enrichi Ajoutez des objets spéciaux du livre avec des effets uniques :

```
// Dans la classe Personnage.java
public void ajouterObjetSpecial(String nom, String description, Map<String, Integer> effet
    Objet nouvelObjet = new Objet(nom, description, effets);
    inventaire.add(nouvelObjet);

    // Appliquer effets permanents si nécessaire
    if (effets.containsKey("habiletePermanente")) {
        this.modifierHabilete(effets.get("habiletePermanente"));
    }
}

// Exemples d'objets spéciaux à implémenter
public static Objet creerKatanaDeQualite() {
    Map<String, Integer> effets = new HashMap<>();
    effets.put("bonusCombat", 1);  // +1 en combat
    return new Objet(
        "Katana de qualité",
        "Une lame parfaitement équilibrée forgée par un maître. Confère un bonus de +1 en
        effets
    );
}

public static Objet creerParcheminDuVent() {
```

```
    Map<String, Integer> effets = new HashMap<>();
    effets.put("esquive", 2);  // +2 aux jets d'esquive
    return new Objet(
        "Parchemin du Vent",
        "Un ancien parchemin contenant les secrets de l'art de l'esquive.",
        effets
    );
}
```

6.4.3 Carte des chapitres interactive Améliorez la visualisation de la progression du joueur :

```
// Dans ChapterMapUI.java
public void initialiserCarteComplete() {
    // Configuration de base de la carte
    setLayout(new AbsoluteLayout());

    // Créer un graphe des chapitres avec coordonnées X,Y pour chaque nœud
    for (Integer chapitreId : scenario.getChapitres().keySet()) {
        // Calculer position basée sur l'arborescence des choix
        Point position = calculerPositionChapitre(chapitreId);

        // Créer un nœud visuel pour ce chapitre
        JLabel nodeChapitre = creerNodeChapitre(chapitreId, position);
        add(nodeChapitre);

        // Créer des connexions visuelles vers les chapitres liés
        for (Choix choix : scenario.getChapitre(chapitreId).getChoixPossibles()) {
            creerConnexion(position, calculerPositionChapitre(choix.getDestinationId()));
        }
    }

    // Marquer le chapitre actuel
    marquerChapitreActuel(gameController.getChapitreActuel().getId());
}
```

6.5 Tests et validation approfondis

Pour garantir la qualité de votre contribution, suivez ces étapes de test détaillées :

6.5.1 Tests narratifs

1. **Couverture des choix** :
   - Parcourez chaque branche narrative pour vérifier que tous les choix mènent aux chapitres attendus
   - Testez les conditions spéciales (objets requis, tests d'attributs)
2. **Cohérence de l'histoire** :
   - Vérifiez que les transitions entre chapitres existants et nouveaux sont logiques
   - Assurez-vous que les références à des événements passés sont correctes

6.5.2 Tests techniques

1. **Équilibrage des combats** :
   - Testez chaque combat avec différentes valeurs d'Habileté et d'Endurance
   - Vérifiez que la difficulté est appropriée à la progression du joueur
2. **Performances** :
   - Mesurez l'impact de vos ajouts sur le temps de chargement
   - Optimisez les ressources (images, sons) pour maintenir la fluidité
3. **Compatibilité d'interface** :

```
# Exécutez le jeu avec différentes résolutions pour tester l'adaptabilité
java --enable-preview -jar EpeeDuSamourai.jar
```

6.5.3 Documentation du test Créez un document de test décrivant vos scénarios de test et résultats :

```
# Rapport de test - Chapitre X

## Tests narratifs
- Choix 1 → Chapitre Y: OK
- Choix 2 → Chapitre Z: OK
- Condition spéciale (possession de l'amulette) → Chapitre W: OK

## Tests de combat
- Combat contre Ninja (Habileté joueur = 8):
  * Victoire à 80% (attendu ~75%)
  * Perte moyenne d'Endurance: 4 points

## Problèmes identifiés
- La transition vers le chapitre Z manque de contexte narratif
- Le combat contre le seigneur démon est trop difficile
```

6.6 Ressources détaillées pour le développement

Pour continuer efficacement le développement, ces ressources sont essentielles :

6.6.1 Ressources littéraires

- **Livre original** : "L'Épée du Samouraï" par Mark Smith et Jamie Thomson (ISBN: 2-07-056648-0)
- **Documentation des règles** : Consultez les pages 9-20 du livre pour les mécaniques de jeu
- **Structure narrative** : Cartographiez les sections du livre pour comprendre les connexions

6.6.2 Ressources techniques

- **Documentation JavaDoc** : Générez une documentation JavaDoc pour le code existant :

```
cd /home/malek/Documents/Projects/Java-Mini-Project
javadoc -d docs -sourcepath src -subpackages controller:model:view
```

- **Outils de développement recommandés** :
  - IDE: IntelliJ IDEA ou Eclipse avec plugins Java Swing
  - Versionning: Git avec branches par chapitre
  - Tests: JUnit pour les tests unitaires de la logique de jeu

6.6.3 Ressources artistiques

- **Banques d'images** :
  - Artstation pour des concepts arts de style samouraï
  - Wikimedia Commons pour des images historiques du Japon féodal
- **Outils de création graphique** :
  - GIMP ou Photoshop pour l'édition d'images
  - Inkscape pour les éléments d'interface vectoriels

Les contributeurs sont vivement encouragés à respecter l'esprit du projet tout en apportant leur créativité. L'objectif est de créer une adaptation fidèle mais enrichie du livre-jeu original, offrant aux joueurs une expérience immersive dans l'univers du samouraï.

4. Crédits

Ce jeu est une adaptation numérique du livre-jeu "L'Épée du Samouraï" écrit par Mark Smith et Jamie Thomson, et traduit en français par Pascale Jusforgues.