KOKOU CEDRIC
BTS SIO SLAM 2EME ANNEE

ATELIER DE PROFESSIONNALISATION 1

URBAN MARGINAL

SOMMAIRE

2.5. MISSION 5 : DEPLACEMENTS DES JOUEURS, COMBATS (THREAD) ET GESTION DES SONS

- Contexte
- Détails Techniques
    - Déplacements des joueurs
    - Combats (threads)
    - Gestion des sons

2.6. MISSION 6 : CORRECTION D'ERREURS, TESTS UNITAIRES ET DEPLOIEMENT

- Contexte
- Détails Techniques
    - Correction d'erreurs
    - Tests unitaires
    - Déploiement

3. SITUATION PROFESSIONNELLE

3.1. PRESENTATION DU CONTEXTE

- Description de l'entreprise
- Objectifs de la mission
- Encadrement et support

3.2. PRESENTATION DE L'APPLICATION

- Description de l'application de jeu de combat 2D
- Fonctionnalités principales et objectifs techniques

3.3. MA TACHE

- Détail des étapes de développement
    - Étape 1 : Utilisation des outils de connexion
    - Étape 2 : Création des fonctionnalités de la fenêtre de choix du joueur
    - Étape 3 : Création des jeux serveurs et clients, et enregistrement du client
- Plan de tests pour chaque étape

1. INTRODUCTION

Contexte général du projet "Urban Marginal"

"Urban Marginal" est un jeu 2D développé en Java avec l'IDE Eclipse. Ce jeu simule des défis sociaux et économiques dans un environnement urbain. Le projet utilise une architecture MVC (Modèle-Vue-Contrôleur) pour séparer les préoccupations et faciliter la maintenance.

Objectifs et utilité pour les utilisateurs

L'objectif du jeu est de fournir une simulation interactive où les utilisateurs peuvent naviguer et résoudre des problèmes urbains. Les utilisateurs peuvent bénéficier d'une expérience ludique tout en apprenant les dynamiques sociales et économiques des environnements urbains.

2. DOCUMENTATION TECHNIQUE

2.1. MISSION 1 : CONSTRUCTION DE LA STRUCTURE D'UNE APPLICATION EN MVC ET DEPOT DISTANT

Contexte : La première étape de développement de "Urban Marginal" consiste à construire la structure de l'application en utilisant le modèle MVC. Cette architecture permet de séparer la logique métier, l'interface utilisateur et le contrôle des flux d'application. Le dépôt distant sur GitHub assure la gestion des versions et la collaboration.

Détails Techniques

1. **Architecture MVC :**
   - **Modèle (Model)** : Gère les données et la logique métier.
   - **Vue (View)** : Gère l'affichage et l'interface utilisateur.
   - **Contrôleur (Controller)** : Gère les entrées de l'utilisateur et met à jour le modèle et la vue.
2. **Création du dépôt distant :**
   - Initialiser un dépôt Git :

     ```
     bash
     Copier le code
     git init
     ```

   - Ajouter des fichiers et faire le premier commit :

     ```
     bash
     Copier le code
     git add .
     git commit -m "Initial commit"
     ```

   - Lier le dépôt distant :

     ```
     bash
     Copier le code
     git remote add origin <URL_du_dépôt>
     git push -u origin master
     ```

3. **Configuration du projet dans Eclipse :**
   - Créer un nouveau projet Java :
     - Fichier > Nouveau > Projet Java
   - Configurer le dépôt Git dans Eclipse :
     - Clic droit sur le projet > Equipe > Partager le projet > Git > Utiliser ou créer un dépôt existant

2.2. MISSION 2 : INTERFACES GRAPHIQUES ET PARCOURS UTILISATEUR

Contexte : Cette mission consiste à développer les interfaces graphiques du jeu et à définir le parcours utilisateur. L'objectif est de créer une interface intuitive et engageante qui guide les joueurs à travers les différentes fonctionnalités du jeu.

Détails Techniques

1. **Utilisation de Java Swing pour l'interface graphique :**
   - Création de fenêtres et de panneaux :

```
java
Copier le code
import javax.swing.*;

public class GameView {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Urban Marginal");
        JPanel panel = new JPanel();
        frame.add(panel);
        frame.setSize(800, 600);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

2. **Définir le parcours utilisateur :**
   - Écran d'accueil, menu principal, options de jeu, etc.
   - Implémenter les actions des boutons et les transitions entre les écrans.

2.3. MISSION 3 : CHOIX DU JOUEUR ET UTILISATION DE CLASSES TECHNIQUES DE CONNEXION (MULTIJOUEURS)

Contexte Les joueurs peuvent choisir leur personnage et se connecter pour jouer en mode multijoueur. Cette mission implique la création de classes pour gérer la connexion et la communication entre les joueurs.

Détails Techniques

1. **Choix du joueur :**
   - Interface de sélection de personnage :

```
java
Copier le code
JComboBox<String> characterSelect = new JComboBox<>(new
String[]{"Personnage 1", "Personnage 2"});
panel.add(characterSelect);
```

2. **Classes de connexion :**
   - Utilisation de sockets pour la communication :

```
java
Copier le code
import java.net.*;

public class Server {
    public static void main(String[] args) throws IOException {
        ServerSocket serverSocket = new ServerSocket(12345);
        Socket clientSocket = serverSocket.accept();
        // Logique de gestion de la connexion
    }
}
```

2.4. MISSION 4 : AJOUT DES MURS ET DES JOUEURS DANS L'ARENE, GESTION DU TCHAT

Contexte Cette mission consiste à ajouter des éléments de jeu tels que les murs et les joueurs dans l'arène, ainsi qu'à implémenter une fonctionnalité de tchat pour la communication en jeu.

Détails Techniques

1. **Ajout des murs et des joueurs :**
   - Création des classes `Wall` et `Player` et ajout de ces éléments dans l'arène :

```
java
Copier le code
public class Wall {
    int x, y;
    public Wall(int x, int y) {
        this.x = x;
        this.y = y;
    }
    // Méthodes
}
public class Player {
    int x, y;
    public Player(int x, int y) {
        this.x = x;
        this.y = y;
    }
    // Méthodes
}
```

2. **Gestion du tchat :**
    - Interface de tchat :

```
java
Copier le code
JTextArea chatArea = new JTextArea();
JTextField chatInput = new JTextField();
panel.add(chatArea);
panel.add(chatInput);
```

2.5. MISSION 5 : DEPLACEMENTS DES JOUEURS, COMBATS (THREAD) ET GESTION DES SONS

Contexte Le déplacement des joueurs, la gestion des combats et l'ajout de sons sont essentiels pour rendre le jeu interactif et immersif. Cette mission utilise des threads pour gérer les actions en parallèle.

Détails Techniques

1. **Déplacements des joueurs :**
    - Gestion des événements clavier :

```
java
Copier le code
frame.addKeyListener(new KeyAdapter() {
    public void keyPressed(KeyEvent e) {
        // Logique de déplacement
    }
});
```

2. **Combats (threads) :**
    - Utilisation de threads pour les combats :

```
java
Copier le code
Thread combatThread = new Thread(new Runnable() {
    public void run() {
        // Logique de combat
    }
});
combatThread.start();
```

3. **Gestion des sons :**
    - Utilisation de la bibliothèque `javax.sound.sampled` :

```
java
Copier le code
import javax.sound.sampled.*;

public class SoundPlayer {
    public void playSound(String soundFile) throws Exception {
        AudioInputStream audioStream =
AudioSystem.getAudioInputStream(new File(soundFile));
        Clip clip = AudioSystem.getClip();
        clip.open(audioStream);
        clip.start();
    }
}
```

2.6. MISSION 6 : CORRECTION D'ERREURS, TESTS UNITAIRES ET DEPLOIEMENT

Contexte : La dernière mission consiste à corriger les erreurs, à implémenter des tests unitaires pour assurer la qualité du code, et à déployer l'application.

Détails Techniques

1. **Correction d'erreurs :**
    - Débogage et correction des bugs identifiés.
2. **Tests unitaires :**
    - Utilisation de JUnit pour les tests :

```
java
Copier le code
import static org.junit.Assert.*;
import org.junit.Test;

public class PlayerTest {
    @Test
    public void testPlayerMovement() {
        Player player = new Player();
        player.move(Direction.UP);
        assertEquals(expectedPosition, player.getPosition());
    }
}
```

3. **Déploiement :**
    - Création d'un fichier JAR exécutable dans Eclipse :
        - Fichier > Exporter > Java > Fichier JAR > Suivre les instructions pour créer le fichier JAR.

3. SITUATION PROFESSIONNELLE

3.1. PRESENTATION DU CONTEXTE

Description de l'entreprise : L'entreprise pour laquelle j'ai travaillé s'est spécialisé dans le développement de jeux sur divers supports. Grâce à une croissance rapide, l'entreprise a récemment recruté une équipe de développeurs juniors, dont je fais partie, pour participer à une phase de tests sur plusieurs petits projets.

Objectifs de la mission : Cette mission a représenter une occasion idéale pour évaluer mes compétences. J'ai été encadré par un responsable qui m'a fourni les documents et productions nécessaires et m'a également guider dans mes tâches.

3.2. PRESENTATION DE L'APPLICATION

Description de l'application de jeu de combat 2D L'application que je dois créer est un jeu de combat 2D destiné aux ordinateurs de bureau. Elle doit permettre de lancer un serveur de jeu sur un ordinateur qui servira de console pour visualiser l'arène de jeu. De plus, elle doit permettre de lancer un ou plusieurs clients, lesquels se connecteront au serveur pour entrer dans l'arène et jouer.

Fonctionnalités principales et objectifs techniques

- Chaque joueur peut se déplacer dans une arène contenant des murs infranchissables et tirer des boules de feu pour toucher les adversaires.
- Gestion des points de vie et une zone de tchat pour la discussion entre les joueurs.
- Développement en Java.

3.3. MA TACHE

Détail des étapes de développement

1. **Étape 1 :** Utilisation des outils de connexion pour créer le serveur, s'y connecter et ouvrir les fenêtres.
2. **Étape 2 :** Création des fonctionnalités de la fenêtre de choix du joueur.
3. **Étape 3 :** Création des jeux serveurs et clients, et enregistrement du client.

Plan de tests pour chaque étape

- Définir des tests unitaires pour vérifier la fonctionnalité de chaque composant.
- Utiliser des scénarios de test pour valider les interactions entre les composants.