Rapport de Tests Unitaires

Introduction

Ce document présente les tests unitaires développés pour les classes identifiées dans le projet. Les tests couvrent les principales fonctionnalités des classes, notamment celles utilisant des design patterns (Observer, MVC, Singleton, Factory, Decorator). Chaque section détaille les tests effectués, leurs résultats attendus, et le framework utilisé pour leur mise en œuvre.

Configuration des Tests

- **Framework de Test** : Google Test (gTest)
- **Langage** : C++
- **Plateforme** : Linux

Classe : MobileElementView

Méthode : update

Description : Cette méthode met à jour la position de l'élément mobile en fonction d'une structure de données contenant des informations sur les nouvelles positions ("x", "y").

Cas de Test :

1. **Mise à jour correcte avec des valeurs valides**
    a. **Entrée** : { "x": 20, "y": 30 }
    b. **Attendu** : Position mise à jour à (20, 30).
2. **Mise à jour partielle**
    a. **Entrée** : { "x": 50 }
    b. **Attendu** : Position mise à jour à (50, y) (valeur actuelle de y).
3. **Gestion des exceptions pour des clés manquantes**
    a. **Entrée** : { "z": 40 }
    b. **Attendu** : Aucune modification de position.
4. **Gestion des exceptions pour un type incorrect**
    a. **Entrée** : { "x": "invalid" }

b. **Attendu** : Affichage d'une erreur sans modification de position.

Exemple de Test :

```
TEST(MobileElementViewTest, Update) {
  MobileElementView view;
  std::unordered_map<std::string, std::any> data = { {"x", 20}, {"y", 30} };
  view.update(data);
  EXPECT_EQ(view.getX(), 20);
  EXPECT_EQ(view.getY(), 30);
}
```

Méthode : setElementPosition

Description : Met à jour la position de l'élément graphique dans la scène.

Cas de Test :

1. **Mise à jour correcte des coordonnées**
   a. **Entrée** : newX = 15, newY = 25
   b. **Attendu** : Position mise à jour dans l'interface graphique.
2. **Vérification des appels systèmes**
   a. **Entrée** : newX = 0, newY = 0
   b. **Attendu** : Appel de la méthode setPos avec les nouvelles coordonnées.

Exemple de Test :

```
TEST(MobileElementViewTest, SetElementPosition) {
  MobileElementView view;
  view.setElementPosition(15, 25);
  EXPECT_EQ(view.getX(), 15);
  EXPECT_EQ(view.getY(), 25);
}
```

Classe : Factory

Classe : clientGroupFactory

Méthode : create

Description : Crée un groupe de clients selon un paramètre d'entrée.

Cas de Test :

1. **Création d'un groupe de clients valide**
    a. **Entrée** : "Family"
    b. **Attendu** : Groupe de type Family retourné.
2. **Type inconnu**
    a. **Entrée** : "UnknownGroup"
    b. **Attendu** : Retour d'une exception ou nullptr.

Exemple de Test :

```
TEST(ClientGroupFactoryTest, Create) {
  clientGroupFactory factory;
  ClientGroup* group = factory.create("Family");
  EXPECT_NE(group, nullptr);
  EXPECT_EQ(group->getType(), "Family");
}
```

Classe : TableFactory

Méthode : create

Description : Crée une table spécifique selon un paramètre d'entrée.

Cas de Test :

1. **Création d'une table valide**
    a. **Entrée** : "Round"

b. **Attendu** : Table de type Round retournée.

2. **Type inconnu**
    a. **Entrée** : "UnknownTable"
    b. **Attendu** : Retour d'une exception ou nullptr.

Exemple de Test :

```
TEST(TableFactoryTest, Create) {
  TableFactory factory;
  Table* table = factory.create("Round");
  EXPECT_NE(table, nullptr);
  EXPECT_EQ(table->getType(), "Round");
}
```

Classe : Factory

Méthode : create

Description : Crée un objet d'un type spécifique selon un paramètre d'entrée.

Cas de Test :

1. **Création d'un objet valide**
    a. **Entrée** : "TypeA"
    b. **Attendu** : Objet de type TypeA retourné.
2. **Type inconnu**
    a. **Entrée** : "UnknownType"
    b. **Attendu** : Retour d'une exception ou nullptr.

Exemple de Test :

```
TEST(FactoryTest, Create) {
  Factory factory;
  BaseType* obj = factory.create("TypeA");
  EXPECT_NE(obj, nullptr);
  EXPECT_EQ(obj->getType(), "TypeA");
```

}

Classe : Decorator

Méthode : operate

Description : Ajoute des fonctionnalités à un objet de base tout en conservant son interface.

Cas de Test :

1. **Ajout d'une fonctionnalité**
    a. **Entrée** : Objet de base + décorateur
    b. **Attendu** : Comportement modifié tout en maintenant les fonctionnalités existantes.

Exemple de Test :

```
TEST(DecoratorTest, Operate) {
  BaseComponent* component = new ConcreteComponent();
  BaseComponent* decorated = new ConcreteDecorator(component);
  EXPECT_EQ(decorated->operate(), "Decorated Operation");
}
```

Conclusion

Les tests unitaires ont permis de vérifier le bon fonctionnement des méthodes critiques des classes identifiées. Les frameworks comme Google Test facilitent la validation et la maintenance des fonctionnalités du projet. Les prochaines étapes incluent l'augmentation de la couverture des tests et l'automatisation de leur exécution dans une intégration continue.