UJ DUNGEONS

Dokumentacja implementacyjna

Opis

Dokument skupia się na implementacji ładowania i obsługi widoku oraz tworzenia map.

Spis treści

Wprowadzenie

Dokument skupia się na wybranym segmencie aplikacji „UJ Dungeons” – ładowaniu oraz obsłudze widoków, a także tworzeniu map. Jest to jeden z głównych punktów aplikacji.

Opierajmy się na wzorcu architektonicznym – MVC (Model-View-Controller), który służy do organizowania struktury aplikacji posiadającej interfejs graficzny.

Ładowanie lokacji

Wybieramy lokację, którą należy załadować – jest to określone w pliku Game.java i zależne jest od wyglądu mapy.

Do samego ładowania wykorzystujemy metody zawarte w klasie View.java. Komnaty (czyli sceny zawierające się w naszym głównym oknie aplikacji) są wyświetlane poprzez załadowanie loadera FXML, który jest zwracany w metodzie loadRoom (String path) z klasy SceneController.java:

```
public FXMLLoader loadRoom(String path)  {
    FXMLLoader loader = new FXMLLoader();

    Parent root = null;
    Scene scene;
    loader.setLocation(getClass().getResource(path));
    try {
        root = loader.load();
        mainPane.getChildren().setAll(root);
    } catch (IOException e) {
        e.printStackTrace();
    }

    return loader;
}
```

Tworzymy Loader. Następnie, jeśli ścieżka jest poprawnie określona to ładujemy naszą scenę do określonego przez nas miejsca na scenie głównej:

```
public void loadMonsterChamber(Game game, Chamber chamber){
    FXMLLoader myLoader = sceneController.loadRoom("/monsterRoom.fxml");
    MonsterChamberController myController = myLoader.getController();
    myController.setMediator(game);
    myController.setChamber(chamber);
}
```

Kolejnym krokiem, po uzyskaniu Loadera jest ładowanie kontrolera, a następnie ustawienie mediatora i komnaty, która ma być przepisana do tego kontrolera. Ma to na celu połączenie kontrolera z komnatą, dla której będzie obsługiwał zdarzenia.

Scena startowa, scena do wyboru profesji i scena główna są ładowane przez specjalną metodę w klasie View. Omawianą metodą jest loadMenuScene (String path):

```
public FXMLLoader loadMenuScene(String path){
    FXMLLoader loader = new FXMLLoader();

    Parent root = null;
    Scene scene;
    loader.setLocation(getClass().getResource(path));
    try {
        root = loader.load();
    } catch (IOException e) {
        e.printStackTrace();
    }

    scene = new Scene(root);
    this.stage.setScene(scene);
    this.stage.show();

    return loader;
}
```

Wyżej omawiana I pokazana metoda jest podobna do SceneControler.java z tą różnicą, że w tej ładujemy wszystko jako główną scenę.

Model

Model związany z obsługą komnaty z potworem:

```
public class MonsterChamber extends Chamber {

    Hero myHero;
    public MonsterChamber(Game game){
        super(game);
        myHero = game.getHero();
    }

    @Override
    public void selectOption2() {
        option2 = new DiedChamber(game);
        option2.loadChamber();
    }

    @Override
    public void loadChamber() {
        game.getView().loadMonsterChamber(game, this);
    }

    public String fight(){
        String fightStatus = "";
        Monster normal_monster = new Monster(myHero.getLevel(), false);
        Character character = (Character)normal_monster;
        fightStatus = fightStatus + myHero.fight(normal_monster,
fightStatus);
        if (myHero.getCurrentHealth() > 0)
        {
            myHero.levelup();
        }
        return fightStatus;
    }

    public Hero getHero(){
        return myHero;
    }
}
```

Najważniejszymi metodami w kontekście MVC są: loadChamber, selectOption2(), fight().

- loadChamber() – ładuje naszą komnatę z poziomu zarządzania mapą (o tym w sekcji na temat konstruowania mapy)

- selectOption2() – nadpisujemy tutaj metodę dla własnych potrzeb. W tym przypadku załadowania informacji o przegraniu.

- fight() – metoda do przeprowadzenia walki z potworem. Metoda zwraca nam opis przebiegu walki.

Kontroler

Przykładowe metody z klasy ChamberController. Są one wykorzystywane z poziomu zarządzania mapą.

```
@Override
public void setChamber(Chamber chamber) {
    super.setChamber(chamber);
    currentChamber = (MonsterChamber)chamber;
}

@Override
public void onOption1Clicked() {
    super.onOption1Clicked();
}

@Override
public void onOption2Clicked() {
    super.onOption2Clicked();
}
```

Tutaj wybieramy mediatora, którym jest klasa Game

```
@Override
public void setMediator(Game mediator) {
    super.setMediator(mediator);
}
```

Obsługę przycisków zawdzięczamy rozwiązaniom dostarczonym przez JavaFX. Wystarczy z pliku fxml wpisać nazwę naszego przycisku.

Przykład obsługi przycisku. Najpierw go wyłączamy, aby nie móc drugi raz uruchomić walki. Następnie wywołujemy walkę i prezentujemy rezultat.

```
@FXML
private Button buttonRun;
```

```
public void FightAction(){
    buttonFight.setDisable(true);
    buttonRun.setDisable(true);

    String result = currentChamber.fight();
    info_fight.setText(result);
    buttonFinish.setDisable( false );
}
```

Przykład tego jak wygląda obsługa z poziomu FXML. OnAction mówi, która funkcja ma być uruchomiona po kliknięciu. W pliku FXML jest wskazany kontroler.

```
<Button id="button__fight" fx:id="buttonFight" layoutX="46.0" layoutY="89.0" mnemonicParsing="false" onAction="#FightAction"
```

Tak samo jest to zrobione z wybieraniem opcji 1 lub 2. Wystarczy przypisanie onAction to tych samych funkcji, czyli selectOptionClicker1 lub selectOption2Clicked – kontrolery dziedziczą po głównym ChamberController.

Mapa

MapGenerator tworzy pierwsze instancje, jeśli chcemy użyć dwóch instancji, np.: żeby były dwie walki z potworem, to trzeba będzie stworzyć kolejną instancję.

```
WinChamber win; // no exit, please do not set option1 and option2
DiedChamber died; // no exit, please do not set option1 and option2
FoodFountainChamber food; // one exit
LeftRight leftRight;
MonsterChamber monsterChamber; // one exit
TraderChamber traderChamber; // one exit
Boss boss; // no exit, please do not set option1 and option2
EmptyRoom empty;    /// one exit
TrapChamber trapChamber;  //one exit
ChestAfterOpen chestAfterOpen;  // one exit

Chamber[][] maps = new Chamber[5][20];

public MapGenerator(Game game){
    this.game = game;
     win = new WinChamber(game);
     died = new DiedChamber(game);
     food = new FoodFountainChamber(game);
     empty = new EmptyRoom(game);
     leftRight = new LeftRight(game);
     monsterChamber = new MonsterChamber(game);
     traderChamber = new TraderChamber(game);
     trapChamber = new TrapChamber(game);
     chestAfterOpen = new ChestAfterOpen(game);
     boss = new Boss(game);
}
```

Metoda prepareMap() tworzy nam zestaw map. Jak widać wystarczy tylko określić co jest w jakim miejscu i jakie powiązania są między nimi (na zasadzie: co ma nastąpić po sobie, gdy wybierzemy opcje pierwszą lub drugą).