ARCHITEKTÚRÁLIS RÉSZLETEK

(munkaközi, 2014.11.09.)

1. Modulok

Ebben a pontban bemutatásra kerülnek a program logikai komponensei, más néven a modulok. A program három modulból áll, ezek a `menu` modul, a `game` modul és a `dao` modul. A modulok szerinti felosztás a fejlesztői szerepekhez próbál igazodni. Elvileg minden modult más fejlesztő készít el és a modulok csak (a lehetőségek szerint minimalizált) interfészeken keresztül érintkeznek.

A `menu` modul jeleníti meg a bejelentkező képernyőt, innen lehet a különböző almenükbe navigálni, például a beállítások megadására vagy egy elmentett játék betöltésére szolgáló menük érhetők el ebben a modulban. Itt tekinthetők meg a legmagassabb pontszámok is. Amikor a felhasználó elindítja a játékot, akkor a `menu` modul `JFrame` -jén belül megjelenik a játékfelület panelje.

A `game` modul biztosítja a játékfelületet, azaz itt lehet játszani a játékkal, és innen lehet állásokat elmenteni. A `game` modul számára a `menu` modul továbbítja az ablak-eseményeket, például a főablak minimalizálásakor (task-ba leküldésekor) a `menu` modul fő `JFrame` −je jelzi a `game` modulnak, hogy a játék szüneteljen. A `game` modul jelzi a `menu` -nek ha egy játszma befejeződött, és a játékos visszalép a főmenübe.

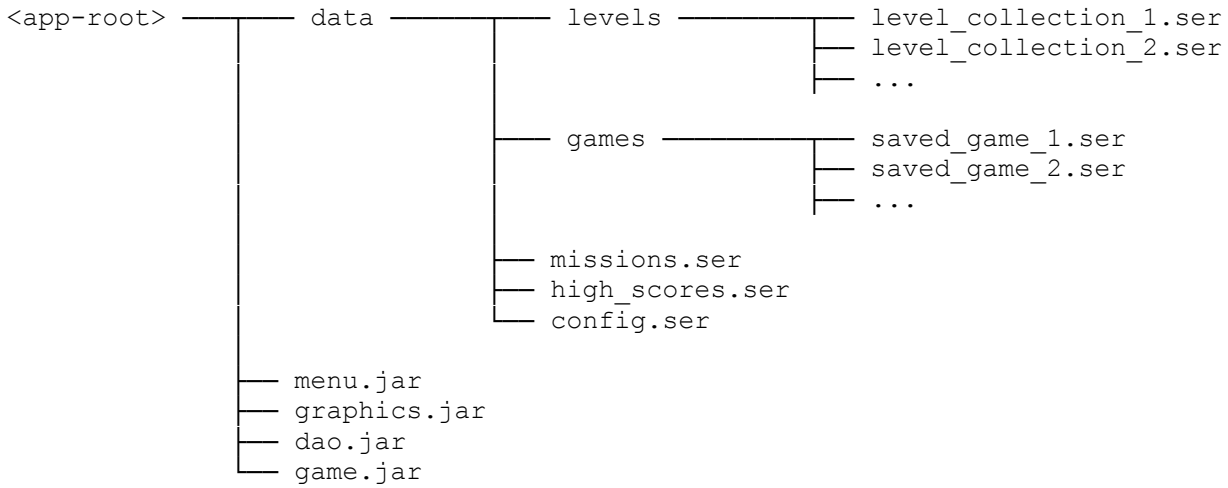
A `dao` (Data Access Object) modul biztosítja a perzisztenciát, mint például a pályák vagy játékosok adatbázisát. Ezt a modult mindkét másik modul használja.

A kommunikáció a `game` és a `menu` között kétirányú, a `game` és a `dao` között egyirányú, valamint a `menu` és a `dao` között is egyirányú (ami azt jelenti, hogy csak a `dao` -t hívják, de ő nem hív vissza).

Fontos megjegyezni, hogy a program az MVC tervezési mintát követi. A modulrendszer kiosztásában ez a következőképpen valósul meg: a `dao` modul valósítja meg a Model-t, amelyet a másik két modul használ. A `menu` és a `game` modulok Controller és View komponensekre lettek felbontva, ezek a részletek azonban csak a belső struktúra elemzésekor láthatók, amelyet a ... fejezet tárgyal.

2. Fizikai architektúra

Ebben a pontban bemutatásra kerülnek a program fizikai komponensei, azaz a programot alkotó fájlok:



1 .ábra - A program részeit képző fájlok

Az *1. ábra* az `<app-root>` gyökérkönyvtár tartalmát mutatja be, ahol

- Az `<app-root>` könyvtár az alkalmazás gyökérkönyvtára, a program csak ebbe a könyvtárba ─ és az alkönyvtárakba ─ ír illetve csak innen olvas ki adatokat. E könyvtár neve lehet például `untrusted_0.1` (ahol a 0.1 a verziószám). Tehát a program Windows-on nem használja a Registry-t és a Linuxokon sem a különböző csomagkezelő vagy rendszerleíró adatbázisokat.

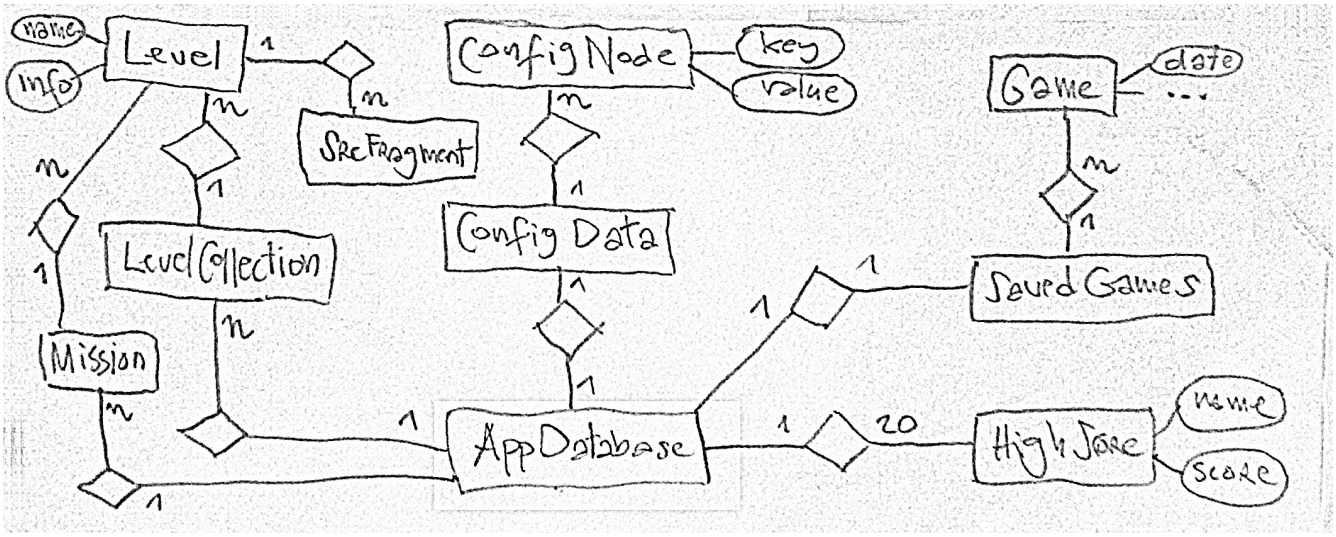
- Az `<app-root>/data` könyvtárba kerülnek a perzisztens adatok, mint például az elmentett játszmák, pályák stb. Ezt a mappát illetve a perzisztenciát biztosító `dao` modult a *3. Perzisztencia* című fejezet mutatja be.
- Az `<app-root>/game.jar` fájl a `game` modul.
- Az `<app-root>/dao.jar` fájl a perzisztenciáért felelős `dao` modul.
- Az `<app-root>/graphics.jar` fájl a grafikai és egyéb segédosztályokat tartalmazza, például a betűkészletek és a képek kezelését végző Helper-eket. Ezt egyaránt használhatja a `menu` és a `game` modul.
- Az `<app-root>/menu.jar` fájl a főprogram, ez a `menu` modul. Ennek a fájlnak a Manifest-jében szerepel a MainClass bejegyzés, vagyis egy futtatható Java alkalmazásról van szó.

A különböző `.jar` fájlokban elhelyezett osztályok a Java platform Service Discovery mechanizmusát használva érik el egymást, azaz interfészek és azok implementációin keresztül.

3. Perzisztencia

Ebben a pontban a program által használt adatbázis modellje és implementációja kerül bemutatásra.

3.1. Adatbázismodell



2 .ábra - A program adatbázismodelljének ER diagramja

A *2. ábra* az adatbázismodellt mutatja be, ahol
- A `ConfigData` a konfigurációs adatbázis. (Egyelőre nem világos, hogy mi kerül ide, de később ez tárolhatná az anyanyelvi beállítást, a játék nehézségi szintjét stb.) Ez egy "dummy" bejegyzés most még.
- A `ConfigNode` egy kulcs/érték pár a konfigurációs bejegyzésekben.
- A `Level` egy pálya a játékban, amelyet a programozó készít el. A `Level` JavaScript forráskódot tartalmaz, elkülönítve a játékos által szerkeszthető, nem szerkeszthető valamint a nem látható állományt. Minden pályán szerepel egy `Exit`. Ezek a pálya teljesítését jelentő kijáratokat reprezentálják. A `Level` nem tudja, hogy az `Exit` után milyen következő pálya, vagy éppen győzelem következik.
- A `LevelCollection` pályák egy gyüjteménye, például pálya-fejlesztők szerint.
- A `Mission` egy küldetés, ami `Level` -ek egy sorozatát jelenti. A játékos küldetést választ ki és azt játssza le.
- A `HighScores` a legmagasabb elért pontszámokat tartalmazó, (mondjuk) 20 elemű adatbázis. (játékos neve, pontszám) párokat tartalmaz. A `menu` olvassa, a `game` írja ezeket a bejegyzéseket. A játékosokról külön adatbázis nem készül.
- A `SavedGames` az elmentett játékok adatbázisa.
- A `Game` egy bejegyzés az elmentett játékokat tartalmazó adatbázisban. Ezeket a játékosok mentik el a mentési pontoknál illetve töltik be a játék indításakor.

3.2. Interfészek

Ezeken az interfészeken keresztül kommunikálnak az adatbázissal a kliensek.

```
package org.szoftverfolyamat.osz.Menu;

/*
A  dao  es a  menu  modul kozti interface.
*/
public interface DaoMenu {
    /*
    A konfigurációs beállítások lekérdezése és megadása.
    */
    String getConfigValue(String key);

    void setConfigProperty(String key, String value);

    /*
    Az elerheto Mission-ok listazasa.
    */
    Mission[] getMissions();

    /*
    Legmagasabb pontszamokat listazza ki.
    */
    String[][] getHighScores();

    /*
    Mentett jatszmak listajanak lekerese; ha a  player==null, akkor az osszeset
    kilistazza, kulonben csak az adott nevu jatekoset.
    */
    GameStub[] getSavedGames(String player);

    /*
    Adott azonositoju jatszma betoltese.
    */
    Game loadGame(long gameId);

    /*
    IO error, 0 ha OK.
    */
    int errno();
}
```

```
package org.szoftverfolyamat.osz.game;

/*
A  dao  es a  game  modul kozti interface.
*/
public interface DaoGame {
    /*
    Folyamatban levo jatszma elmentese.
    */
    void saveGame(Game game);

    /*
    Jatek vegen a pontszam es a jatekos nevenek mentese.
    */
    void saveScore(int score, String player);

    /*
    IO error, 0 ha OK.
    */
    int errno();
}
```

```
package org.szoftverfolyamat.osz.menu;

/*
Egy elmentett jatek bejegyzese.  Eloszor csak ezek lesznek betoltve, a jatekos
ezek kozul valaszt, aztan az  id  alapjan betolti a tenyleges  Game -et.
*/
public interface GameStub {
    /*
    Az elmento jatekos neve.
    */
    String getPlayerName();

    /*
    Az elmento jatekos aktualis pontszama.
    */
    int getScore();

    /*
    Opcionalis megjegyzesek.
    */
    String getNote();

    /*
    A mentes datuma.
    */
    String getDate();

    /*
    Az elmento jatekos neve.
    */
    long getId();
}
```

```
package org.szoftverfolyamat.osz.game;

/*
Forraskod toredek. Minden palya forrasa ilyen toredekekbol all.
*/
public interface SourceFragment {
    /*
    Ha rejtett, akkor a jatekos nem latja.
    */
    boolean isVisible();

    /*
    Irasvedett-e.
    */
    boolean isReadonly();

    /*
    Maga a kod szovege.
    */
    String getCode();
}
```

```
package org.szoftverfolyamat.osz.game;

/*
Egy palyat reprezentalo interface; magat a palyat a palya-programozo kesziti el.
*/
public interface Level {
    /*
    A palya neve.
    */
    String getName();

    /*
    A palya adatai.
    */
    String getInfo();

    /*
    A forraskod-toredekek sorozata.
    */
    Iterator<SourceFragment> getFragments();
}
```

```
package org.szoftverfolyamat.osz.game;

/*
Egy elmentett es betoltheto jatszma.
*/
public interface Game {
    /*
    Az hatralevo resze a kuldetesnek.
    */
    Mission getMission();

    /*
    Az aktualis Level szerializalva.
    */
    Object getCurrentLevel();
}
```

```
package org.szoftverfolyamat.osz.menu;

/*
Egy betoltheto kuldetes, amelyben az  Iterator  felsorolja az egymas utan
kovetkezo palyakat.
*/
public interface Mission {
    /*
    A kuldetes neve.
    */
    String getName();

    /*
    A kuldetes adatai.
    */
    String getInfo();

    /*
    Felsorolja az egymas utan kovetkezo palyakat. Egyik palya teljesitese
    utan jon a kovetkezo.
    */
    Iterator<Level> getLevels();
}
```

3.3. Fizikai megvalósítás

Az `<app-root>/data` könyvtárba kerülnek a perzisztens adatok, a köveetkezők szerint:
- Az `<app-root>/data/levels` könyvtárba kerülnek a pályák gyüjteményeit tartalmazó fájlok. Ezek lehetnek xml fájlok, ahol a forráskód-részeket tartalmazó tag-ek jelölhetik, hogy az adott kódrészlet írható illetve csak olvasható. A pályák forráskódja pedig CDATA szekciókban tárolt.
- Az `<app-root>/data/missions.ser` fájlba kerülnek a küldetések, amelyek lényegében pályák sorozatai. Itt nyílván csak a pályákra mutató hivatkozások vannak. Ez is lehet egy xml fájl.
- Az `<app-root>/data/high_scores.ser` fájlba kerülnek a legmagasabb pontszamok (string, integer) párként. Ez lehet egy Java properties fájl.
- Az `<app-root>/data/config.ser` fájlba kerülnek a konfigurációs bállítások (string, string) kulcs/érték párként. Ez is lehet egy Java properties fájl.
- Az `<app-root>/data/games` fájlba kerülnek az elmentett játszmák szerializált Java objektumokként.