BME - Intelligens rendszerfelügyelet (BMEVMIA370)

IRF Házi feladat III.

3B JavaPongClock with log4j dokumentáció

Kovács Roland György

2011.04.28.

1 Bevezető

Kerettörténet

Cégünk menedzsmentje hangulatjavító intézkedésként le kívánja cserélni a faliórákat a pihenőszobákban és az előtérben a Java Pong Clock-ot1 mutató nagyméretű kijelzőkre.

Mivel az alkalmazás ilyen speciális környezetben fog futni, ezért bármilyen hiba esetén csak úgy van esély a hiba okának megtalálására, ha az alkalmazás képes részletesen naplózni. A probléma megoldására a legalkalmasabb megközelítés az alkalmazás kiegészítése úgy, hogy a log4j3 keretrendszer segítségével naplózzon. (Mivel az alkalmazás a GPL v2 licenszet használja, a forráskódját belső céljainkra szabadon módosíthatjuk.)

1.1 A házi feladat célja (hivatalos kiírás)

1.1.1 Ismerkedés az alkalmazással

Töltse le az alkalmazást, ismerkedjen meg a forráskóddal és röviden foglalja össze, hogy erőforráshasználat-minimalizálás szempontjából milyen változtatásokat tartana azon mindenképp szükségesnek! Ezen változtatásokat hajtsa is végre!

- *Segítség*: a feladat megoldásának szempontjából legfőképp a „Pong” osztály érdemel figyelmet, itt található egy olyan ciklusszervezési megoldás, ami felesleges CPU-használatot generál.

1.1.2 Modellezés

Hogy jobban megértsük az alkalmazás működését, a forráskód tanulmányozásával készítsen egy UML modellt (tipikusan osztálydiagramokkal), ami ábrázolja a rendszer főbb részeit és azok kapcsolatait. Készítse el ezen kívül a Pong osztály viselkedését leíró dinamikus modellt is UML állapotgépek formában.

Figyelem: nem egy reverse engineering eszközzel előállított részletes osztálydiagramot kérünk, hanem egy saját modellt, ami csak az alkalmazás fontosabb részeit ábrázolja!

1.1.3 Felügyeleti modell elkészítése

Megfelelő érveléssel alátámasztva sorolja fel, hogy a kiszolgáló komponensnek milyen, a működésével kapcsolatos eseményeket kellene rendszerfelügyeleti célokra elérhetővé tennie! Az eseményekhez legalább a következő adatokat adja meg:

- esemény azonosítója (integer),
- esemény súlyossága,
- esemény szövege,
- az esemény definíciója (annyira formálisan és részletesen, hogy az további pontosítás nélkül implementálható legyen).

Legalább hat esemény definiálását várjuk el, ezek között kell lennie Error vagy Warning, Information és Verbose súlyosságú eseménynek is.

1.1.4 Naplózás implementálása

Log4j instrumentáció hozzáadásával és a szükséges egyéb módosításokkal érje el, hogy a kiszolgáló fent definiált eseményei naplózva legyenek! Felhívjuk rá a figyelmét, hogy a kiadott kódban a hibakezelés szándékoltan részleges és elnagyolt, annak rendbetétele a feladat részét képezi.

A feladat megoldásának része az a mérnöki érvelés, hogy hány különböző helyre és milyen logika

szerint naplóz az alkalmazás.

1.1.5 Tudnivalók

- Az alkalmazás publikus elemeihez kötelező Javadoc típusú kommenteket készíteni.
- A tesztelés tartalmazza az összes esemény kiváltásának módját és azok naplózódásának bemutatását.
- Felhívjuk a figyelmét, hogy a dokumentáció mellett a forráskód, .jar-ként a lefordított kód és egy indítószkript is leadandó. (A hangállományok nem; az ellenőrzőkörnyezetben a jar állomány könyvtárában megtalálható az eredeti projekt „sound” könyvtára.)
- Ügyeljen továbbá arra, hogy a módosított, illetve saját állományok fejlécében jelezze az eredeti projektet, a kiegészítés/módosítás tényét-idejét-elkövetőjét és azt, hogy az így létrejött megoldásra továbbra is a GPL v2 vonatkozik.

2 Feladatok Megoldása

2.1 Ismerkedés az alkalmazással - teljesítmény javítás

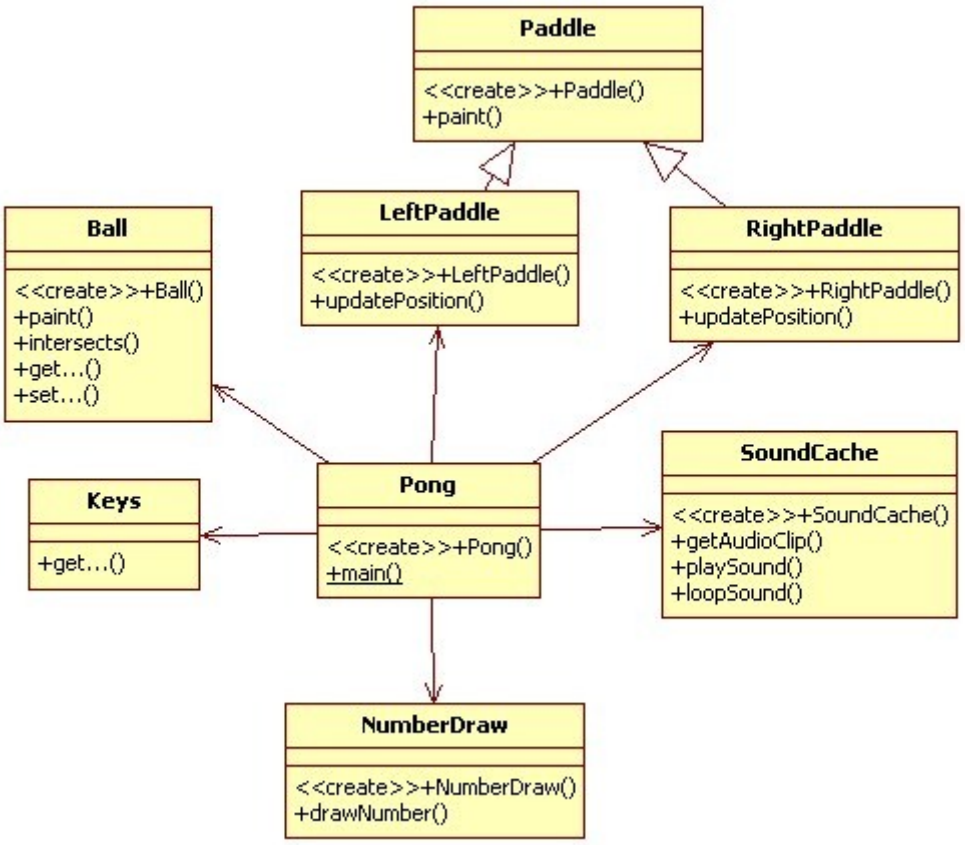
A Pong osztályban található while loop (gameLoop függvény) miatt gyakorlatilag a program CPU használata 100%. Ahhoz, hogy az alkalmazás igénye ne legyen ekkora, és jelentősen növekedjen a teljesítmény, ezért a kérdéses ciklust kiegészítettem az alábbival:

```
try
{
	Thread.sleep(1000/60);
}
catch (InterruptedException e)
{
	System.err.print("Threading error: " + e);
}
```

Azaz a kérdéses szálat pontosan annyi időre tesszük inaktívvá, amit a program eredeti készítője megadott időközként az új adatok lekérdezésére. A CPU használat így processzortól függően körülbelül 1-5%-ra csökken le.

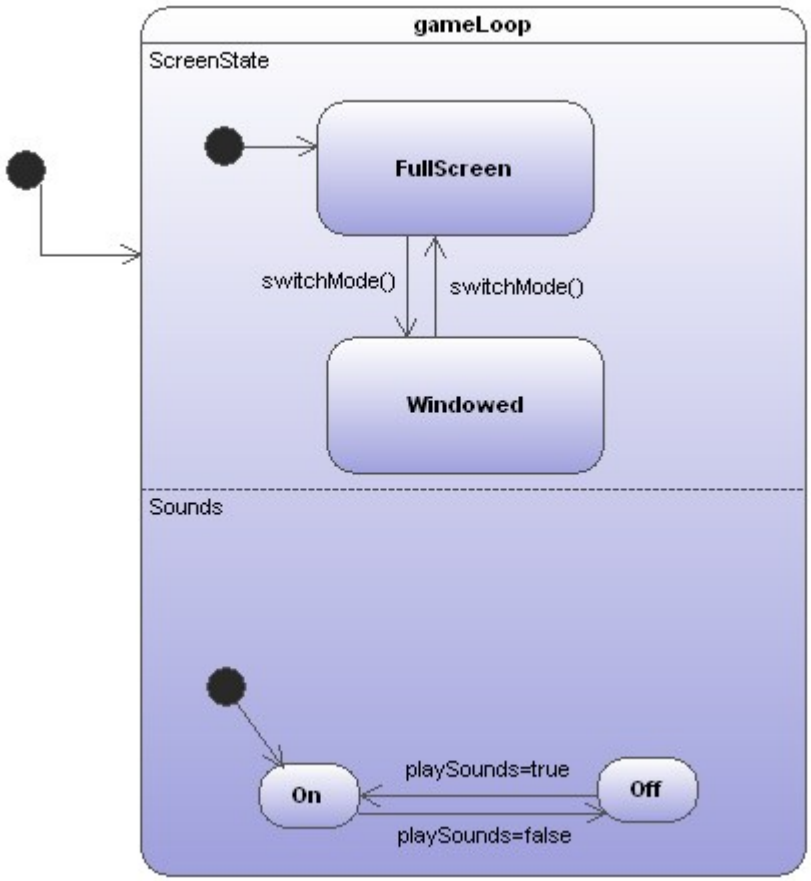
2.2 Modellezés

2.2.1 Osztálydiagram (StarUML segítségével készült)



A diagramon feltüntettem az összes osztályt, hisz mindegyikük meglehetősen fontos a működés szempontjából, valamint köztük a kapcsolatokat. Jól látható, hogy a programot vezérlő Pong osztály minden másikkal asszociációban van. A <<create>> előtaggal az osztályok konstruktorát jelöltem. Az osztályok metódusai közül csak a publikusakat és a kiemelten fontosak vannak feltűntetve. A jobb átláthatóság érdekében a függvények szignatúrái rejtve vannak. A get-set függvényekből több van, ezeket egyként jelenítettem meg.

2.2.2 Állapotgép (Altova Umodel, a StarUML ehhez nem volt elég komplex)



A program inicializálás után végig a gameLoop-ban van, ezalatt meghatározott időközönként lekérdezéseket végez, hogy frissítse a kiírást, valamint rajzol (frissíti a labda és ütők helyzetét, valamint a pontokat írja ki). Alaphelyzetben teljes képernyős módban és bekapcsolt hangokkal kezd az alkalmazás. Gombnyomásra váltogathatunk az állapotok között.

2.3 Felügyeleti modell elkészítése

A kijelzőként alkalmazott program legfontosabb eseményei az alábbi táblázatban találhatóak:

ID	Súlyosság	Szöveg / Definíció
101	**INFO**	**"JavaPongClock initialized and started successfully."** *Az alkalmazás sikeresen befejezte a bemeneti paraméterek feldolgozását, létrehozta a szükséges objektumokat, és megkezdte működését.*
102	**INFO**	**"JavaPongClock entered into fullscreen mode."** *Teljesképernyős megjelenítési módra váltás sikeres.*
103	**INFO**	**"JavaPongClock entered into windowed mode."** *Ablakos megjelenítési módra váltás sikeres.*
104	**WARN**	**"Invalid or out-of-bounds parameters, resetting to default screen size."** *Az alkalmazás bemeneti paramétere irreális, vagy nem megfelelően formázott.*
105	**DEBUG**	**"Minute ticked successfully"** *A percszámláló sikeresen váltott.*

106	**DEBUG**	**"Hour ticked successfully"** *Az óraszámláló sikeresen váltott.*
201	**FATAL**	**"JavaPongClock initialization unsuccessful! JavaPongClock will now exit."** *Az inicializáció sikertelen, a program így kilép.*
202	**ERROR**	**"JavaPongClock failed to enter into fullscreen mode."** *Az alkalmazás teljesképernyős megjelenítési módjára váltás sikertelen volt.*
203	**ERROR**	**"JavaPongClock failed to enter into windowed mode."** *Az alkalmazás ablakos megjelenítési módjára váltás sikertelen volt.*

Kiegészítés: a feladat további megoldása közben derült ki, hogy a log4j nem tartalmaz ID-k beállítására lehetőséget. Több ötlet merült fel, pl hibaosztály létrehozása**, int ID**, **string ErrorMsg** tulajdonsággal, a ToString() [ID + " - " + ErrorMsg] formátumban kerülne bele a log()-ba, azonban végül ezt nem tartottam jó megközelítésnek, így a **végleges implementáció nem tartalmazza az ID-ket.**

- **"JavaPongClock initialized and started successfully."**
  Fontosnak tartom kinaplózni, hiszen így bizonyosodhatunk meg arról, hogy megfelelően elindult az alkalmazás, létrejöttek az osztálypéldányok
- **"JavaPongClock entered into fullscreen mode."**
  Ez alapvetően egy állapotváltás, ezért fontos kinaplózni. Valamint ha az alkalamazást óraként használjuk kijelzőkön, fontos tudni a jelenlegi állapotot, hogy ha valamiért megváltozna rögtön tudjunk rá reagálni.
- **"JavaPongClock entered into windowed mode."**
  Lásd fullscreen
- **"Invalid or out-of-bounds parameters"**
  Nem megfelelő, vagy irreális paraméterezés esetén fontos erről is tudnunk, azonban az alkalmazás ezt lekezeli, és default méretekkel indul el, ezért a WARN kategória.
- **"Minute ticked successfully":**
  Debug üzenet, minden percváltáskor kiíródik (ha az sikeres volt), tudjuk vele ellenőrizni, hogy tényleg szinkronban van-e JavaPongClock órája a futtató számítógépével.
- **"Hour ticked successfully"**
  Debug üzenet, minden óraváltáskor kiíródik (ha az sikeres volt), tudjuk vele ellenőrizni, hogy tényleg szinkronban van-e JavaPongClock órája a futtató számítógépével.
- **"JavaPongClock failed to enter into fullscreen mode."**
  Ha hiba történik a képernyőmód váltások között, erről fontos tudni, hiszen ha óraként használjuk a programot megjelenítőkön, akkor jó ha teljes képernyőn vagyunk és nem látszik az alkalmazás mögött az operációs rendszer.
- **"JavaPongClock failed to enter into windowed mode."**
  Lásd fullscreen

2.4 Naplózás implementálása

2.4.1 Előkészítés log4j-re

A Log4j project honlapjáról letöltött zip fileból a **log4j-1.2.16.jar**-ra lesz szükség. A project mappájában létrehoztam egy újat, **lib** néven, ebben helyeztem el a jar filet. A .classpath -hoz hozzáadtam a jar elérési útját, majd a **Pong** osztályhoz a következőket adtam hozzá:

```
/*
 * log4j import
 */
import org.apache.log4j.*;
…
public class Pong {

	/*
	 * log4j logger added
	 */
	public static Logger PongLog = Logger.getLogger(Pong.class);
```

A **log4j** loggerei többféleképpen **konfigurálhatóak**. Lehetőségünk van **XML konfigurációs file** vagy **.properties file** létrehozására, de akár **kódból** is konfigurálhatunk. Az **újrahasználhatóság** érdekében külön file-t hoztam létre, amely a java **kulcs-érték módszerével operál (.properties).** Ennek a segítségével **a program újrafordítása nélkül** bármikor állíthatjuk a naplózási beállításokat.

2.4.2 Naplózott események a kódban

2.4.2.1 101 - INFO - Inicializálás

Az egész inicializációs folyamat **try-catch** blokkba került. Sikeres inicializálás esetén, **gameLoop** indítása előtt kapjuk ezt az eseményt.

```
/*
 * Logging successful initialization
 */
PongLog.info("JavaPongClock initialized and started successfully.");
```

2.4.2.2 201 - ERROR - Inicializálás hiba

Az egész inicializációs folyamat **try-catch** blokkba került. Sikertelen inicializálás esetén kapjuk ezt az eseményt.

```
/*
 * if error, catch & exit
 */
catch(Exception e)
{
	PongLog.fatal("JavaPongClock initization unsuccessful! JavaPongClock
will now exit.");
	System.exit(0);
}
```

2.4.2.3 102 - INFO - Fullscreen-re váltás (két helyen szerepel a kódban, egyszer a Pong konstruktorban, másodszor pedig a switchMode() metódusban)

```
/*
 * Logging successful mode switch
 */
PongLog.info("JavaPongClock entered into fullscreen mode.");
```

2.4.2.4 202 - ERROR - Fullscreen-re váltás hiba (két helyen szerepel a kódban, egyszer a Pong konstruktorban, másodszor pedig a switchMode() metódusban)

```
/*
 * Logging unsuccessful mode switch
 */
PongLog.error("JavaPongClock failed to enter into fullscreen mode.");
```

2.4.2.5 103 - INFO - Windowed-re váltás

```
/*
 * Logging successful mode switch
 */
PongLog.info("JavaPongClock entered into windowed mode.");
```

2.4.2.6 203 - ERROR - Windowed-re váltás hiba

```
/*
 * Logging unsuccessful mode switch
 */
PongLog.error("JavaPongClock failed to enter into windowed mode.");
```

2.4.2.7 104 - WARN - Nem megfelelő bemeneti paraméterek

```
                /*
                 * Logging parameter-related warning
                 */
                PongLog.warn("Invalid or out-of-bounds parameters, resetting to default
screen size.");
                new Pong(800,600);
```

2.4.2.8 105 - DEBUG - Perc váltás

```
                /*
                 * Logging parameter-related warning
                 */
                PongLog.warn("Invalid or out-of-bounds parameters, resetting to default
screen size.");
                new Pong(800,600);
```

2.4.2.9 106 - DEBUG - Óra váltás

```
/*
 * Hour ticked
 */
PongLog.debug("Hour ticked.");
```

2.4.3 Naplózás helyei és logika

A szoftver több helyre naplóz. A naplózási beállítások a konfigurációs fileban találhatóak (**log4j.properties**) a project gyökérmappájában. A naplózási logika kitalálása során arra jutottam, hogy ha kijelzőkön akarjuk megjeleníteni a programot, és óraként használni alapvetően kétféle módot követhetünk. Ha minden kijelzőhöz külön számítógép tartozik, akkor mindenképpen jó lenne az ezeken futtatott szoftverek eseményeit egy közös helyre, távoli szerverre logolni. Ha pedig egyetlen gép képét küldjük ki az összes kijelzőre, akkor is feltétlenül szükséges a naplózás, ezért a program mappájába, file-ba is mentünk, valamint a konzolon megjelenítünk minden információt (ez pl. továbbfejlesztéskor és debugkor jól jöhet).

2.4.4 log4j.properties file tartalma és a naplózási helyek magyarázata

2.4.4.1 Fejléc (a file létrejöttének körülményei, licensz)

```
#log4j properties file
#Created by Roland Kovacs on April 28, 2011
#As the rest of the software this file also falls under GNU General Public
License v2
```

2.4.4.2 ROOT LOGGER

```
# ROOT LOGGER
log4j.rootLogger=DEBUG, Console, Report, LogFile, Remote
```

Logoláshoz a root loggert használjuk. Debug és annál súlyosabb eseményeket képes logolni. A hozzátartozó appenderek a Console, Report, LogFile és Remote (melyekről alább lesz szó):

2.4.4.3 Console Appender

```
#Console - logs INFO+ events
            log4j.appender.Console=org.apache.log4j.ConsoleAppender
            log4j.appender.Console.Threshold=DEBUG
            log4j.appender.Console.layout=org.apache.log4j.PatternLayout
            log4j.appender.Console.layout.ConversionPattern=%d{yyyy.MM.dd HH:mm}
[%t] %-5p %c %x - %m%n
```

Ez az appender a **konzolra** továbbítja a **DEBUG** és annál súlyosabb eseményekről készült naplóbejegyzéseket (pl. képernyő-mód váltás, inicializáció, hibák). Ez fontos lehet akkor, ha a program előtt ülve debuggolunk illetve tesztelünk (ne kelljen a naplófájlokat nyitogatni, mindig csak az legyen ott, ami most keletkezett). Természetesen a konfigurációs file használata biztosítja, hogy ezek a naplózások bármikor **kikapcsolhatóak** legyenek.

2.4.4.4 Report Appender

```
#Report - logs ALL events
             log4j.appender.Report=org.apache.log4j.RollingFileAppender
             log4j.appender.Report.File=JavaPongReport.log
             log4j.appender.Report.Append=true
             log4j.appender.Report.File=JavaPongReport.log
             log4j.appender.Report.layout=org.apache.log4j.PatternLayout
             log4j.appender.Report.layout.ConversionPattern=%d{yyyy.MM.dd HH:mm}
[%t] %-5p %c %x - %m%n
```

A **JavaPongReport.log** fileba **minden** eseményt naplózunk (gyakorlatilag a konzol appender file-változata), hogy az visszakereshető legyen, valamint ha utólag tekintenénk rá a program futására, és debug információkra szeretnénk szűrni, akkor ez nyilván megfelelő lesz erre a célra. A file maximális mérete 150KB lehet (persze ez változtatható). Ha eléri a megadott méretet, akkor egy **backup** file jön létre **JavaPongReport.log.1** néven, az eredeti file kiürül, és folytatódik bele a naplózás.

2.4.4.5 Error Appender

```
#Error - logs events with WARN or higher severity to a separate file so it will
be easier to check for errors
              log4j.appender.LogFile=org.apache.log4j.FileAppender
              log4j.appender.LogFile.Threshold = WARN
              log4j.appender.LogFile.File=JavaPongError.log
              log4j.appender.LogFile.layout=org.apache.log4j.PatternLayout
              log4j.appender.LogFile.layout.ConversionPattern=%d{yyyy.MM.dd HH:mm}
[%t] %-5p %c %x - %m%n
```

A **JavaPongError.log** file-ba menti a **WARN** és annál súlyosabb eseményeket. Fontos lehet ezeket a lényeges eseményeket mindenféle szűrés nélkül azonnal látni.

2.4.4.6 Remote Appender (alapból kikommentezve lásd Indítás pont)

```
#Remote - logs to a remote server
              log4j.appender.Remote=org.apache.log4j.net.SocketAppender
              log4j.appender.Remote.Threshold=WARN
              #RemoteHost: address and port for the remote server which runs a
service with SocketReceiver
              # such as Chainsaw
              #Modify as needed
              log4j.appender.Remote.RemoteHost=[REMOTE ADDRESS]
              log4j.appender.Remote.Port=4445
              log4j.appender.Remote.LocationInfo=false
```

A Remote Appender egy távoli szerverre naplózza a **WARN** és annál súlyosabb eseményeket, amelyekről mindenképpen fontos értesülnünk a rendszer üzemeltetése során. Ha minden kijelzőhöz külön számítógépet használunk eme funkció használata elengedhetetlen.

Megjegyzés: Ha a távoli számítógépen nem fut szolgáltatás, amely fogadhatná a naplózást, de az appender aktív, akkor a hálózati kapcsolódás mikéntjéből adódóan sok időbe telik mire a program elindul, ez normális működés, továbbá ilyenkor a konzolra kapunk egy hibaüzenetet, hogy nem sikerült kapcsolódni a távoli szerverhez, és később újrapróbálkozik a szoftver.

2.4.4.7 Események formátuma

```
2011.04.28 20:29 [main] INFO  PongClock.Pong  - JavaPongClock initialized and started successfully.
```

3 Indítás és környezet

3.1 Hozzáadott fájlok listája

2011.04.28.	20:58	<DIR>	**lib**	*külső jar(ok) mappája*
2010.03.30.	23:16	481˙534	**/lib/log4j-1.2.16.jar**	*log4j jar*
2011.04.30.	20:32	1˙640	**log4j.properties**	*log4j konfigurációs file*
2011.04.30.	19:31	56	**run.bat**	*Windows indítóscript*
2011.04.30.	22:41	107	**run.sh**	*Linux indítóscript*

és a [DOC] mappa, melyben jelen dokumentáció található.

3.2 Indítás

Windows rendszeren **run.bat**, unix-like rendszereken **run.sh,** a program eredeti paraméterezése ([width] [height]) továbbra is megmarad az indítófileok paramétereként. Mindkét esetben szükség lehet a „PATH" környezeti változó beállítására, hogy a java parancs lefusson. Ehhez segítség pl. itt található:
http://www.java.com/en/download/help/path.xml . Az eredeti Eclipse projectet fejlesztettem tovább, így természetesen Eclipse-ben megnyitva és a futtatva a programot ugyanazt az eredményt kell kapjuk.

A **remote, SocketAppender** típusú naplózás bekapcsolásához az alábbiakat kell végrehajtanunk:

- Távoli gépen a figyelő alkalmazás elindítása és konfigurációja (pl. Chainsaw)
- A **log4j.properties** fileból a megfelelő részt átírni az alábbi módon:

```
# ROOT LOGGER
log4j.rootLogger=DEBUG, Console, Report, LogFile
#, Remote
HELYETT
# ROOT LOGGER
log4j.rootLogger=DEBUG, Console, Report, LogFile, Remote
```

- A Remote appender megfelelő tulajdonságait beállítani a távoli gép címe alapján **(address és port)**

Azért szükséges ezt külön bekapcsolni, mert ha nincs szolgáltatás a távoli gépen amelynek a címe be van írva az Address-hez és nincs kikommentezve az appender akkor zavaróan lassan fog elindulni a program (a hálózati próbálkozások miatt), ráadásul hibát is dob az elérhetetlen kiszolgálóról. Jobbnak láttam így továbbadni a programot, és erre felhívni a figyelmet.

3.3 Fejlesztési környezet

Windows 7 SP1 számítógépen Eclipse IDE felhasználásával. Java verzió: 1.6

3.4 Tesztelési környezet

- CentOS 5.6 IRF (kiadott virtuális gép) - minden, kivéve Remote
- Windows 7 SP1 - minden, kivéve Remote
- Windows XP SP3 (Remote appender teszteléséhez a Chainsaw futtatási helye)

Megjegyzés: a kiadott **CentOS** gépen az **eredeti, módosítatlan project** windowed módba váltása minden esetben sikertelen volt, java.lang.IllegalArgumentException: Invalid display mode hibaüzenettel. Egy másik, Ubuntu VM-en is kipróbáltam ott pedig másik problémába ütköztem: a módváltás sikeres volt (hibaüzenet nélkül), de egy, a 2. ábrához hasonló hibás működés volt az eredménye. Windows rendszereken nem tapasztaltam a hibát. A hibajelenség a saját programomban is fennáll.

A megjelölt programsornál történt a hiba:

```
private boolean switchMode() {
        //switches between fullscreen and windowed mode, returns true if successful
        if(fullscreen) {
            try {
                graphicsDevice.setFullScreenWindow(null);
                fullscreen = false;
```

A hiba lehetséges oka: A **Linux** rendszeres virtuális gépeken (VMWare Workstation) a képernyő frissítési frekvencia **0Hz** (Windowsokon 60Hz). Valószínű, hogy a **java.awt.GraphicsDevice** tartalmaz egy nem megfelelően lekezelt kódsorozatot az eszközök lekérésénél (nincs ellenőrizve 0Hz-re és nem lesz érvényes a jelenlegi eszköz) és beállításánál (setFullScreenWindow(**null**)). Ezért nem tudja visszarakni ablakos módba a programot, illetve ezt megfelelően kezelni (az Ubuntu esetében a program ugyanúgy fullscreen maradt, viszont a játékteret leszűkítette a windowed mód méreteire és így is rajzolt, de a catch nem fogta meg a hibát).

4 Tesztelés

A tesztesetek során a naplózandó események minden helyre bekerültek, ahova szükséges volt. A jobb áttekinthetőség jegyében a **Console** nevű appender kimenete látható a teszteseteknél.

4.1 A program indítása (inicializáció és fullscreen), majd kilépés

```
d:\stuff\egyetem\irf\3\javapong-irf\JavaPong>run.bat
d:\stuff\egyetem\irf\3\javapong-irf\JavaPong>java -cp "lib/log4j-1.2.16.jar;bin"
PongClock.Pong
2011.04.30 19:25 [main] WARN  root  - Invalid or out-of-bounds parameters, resetting to
default screen size.
2011.04.30 19:25 [main] INFO  root  - JavaPongClock entered into fullscreen mode.
2011.04.30 19:25 [main] INFO  root  - JavaPongClock initialized and started successfully.

d:\stuff\egyetem\irf\3\javapong-irf\JavaPong>
```

Naplózott események:

101	**INFO**	**"JavaPongClock initialized and started successfully."** *Az alkalmazás sikeresen befejezte a bemeneti paraméterek feldolgozását, létrehozta a szükséges objektumokat, és megkezdte működését.*
102	**INFO**	**"JavaPongClock entered into fullscreen mode."** *Teljesképernyős megjelenítési módra váltás sikeres.*
104	**WARN**	**"Invalid or out-of-bounds parameters, resetting to default screen size."** *Az alkalmazás bemeneti paramétere irreális, vagy nem megfelelően formázott.*

4.2 Képernyőváltás (lenyomott gombok: z-z-z)

```
2011.04.30 19:38 [main] INFO  root  - JavaPongClock entered into windowed mode.
2011.04.30 19:38 [main] INFO  root  - JavaPongClock entered into fullscreen mode.
2011.04.30 19:38 [main] INFO  root  - JavaPongClock entered into windowed mode.
```

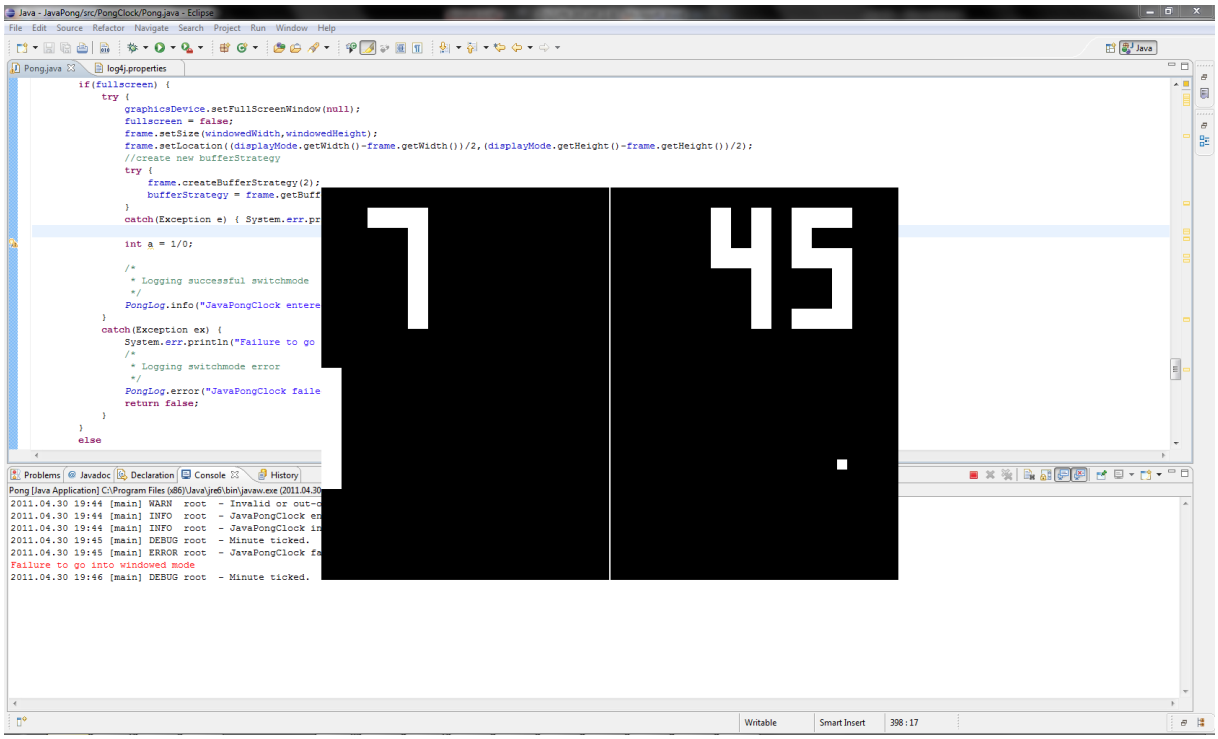
102	**INFO**	**"JavaPongClock entered into fullscreen mode."** *Teljesképernyős megjelenítési módra váltás sikeres.*
103	**INFO**	**"JavaPongClock entered into windowed mode."** *Ablakos megjelenítési módra váltás sikeres.*

4.3 Windowed-be váltás error (kiváltási módja: Try blokkba int a = 1/0)

```
2011.04.30 19:44 [main] WARN  root  - Invalid or out-of-bounds parameters, resetting to
default screen size.
2011.04.30 19:44 [main] INFO  root  - JavaPongClock entered into fullscreen mode.
2011.04.30 19:44 [main] INFO  root  - JavaPongClock initialized and started successfully.
2011.04.30 19:45 [main] DEBUG root  - Minute ticked.
2011.04.30 19:45 [main] ERROR root  - JavaPongClock failed to enter into windowed mode.
2011.04.30 19:46 [main] DEBUG root  - Minute ticked.
```

```
2011.04.30 19:47 [main] DEBUG root  - Minute ticked.
```

Ahogy az az 1. ábrán látható, ilyenkor windowed mode-ba kapcsoláskor hibaüzenetet kapunk. A program tovább fut, de a képernyőn csak a „játéktér” kis része látható, valamint a labda is megbolondul, viszont ha mégegyszer módot váltunk a Z gomb segítségével, a fullscreen megint tökéletesen működik.



1. ábra Windowed mód hibás

A naplózott hibabejegyzések:

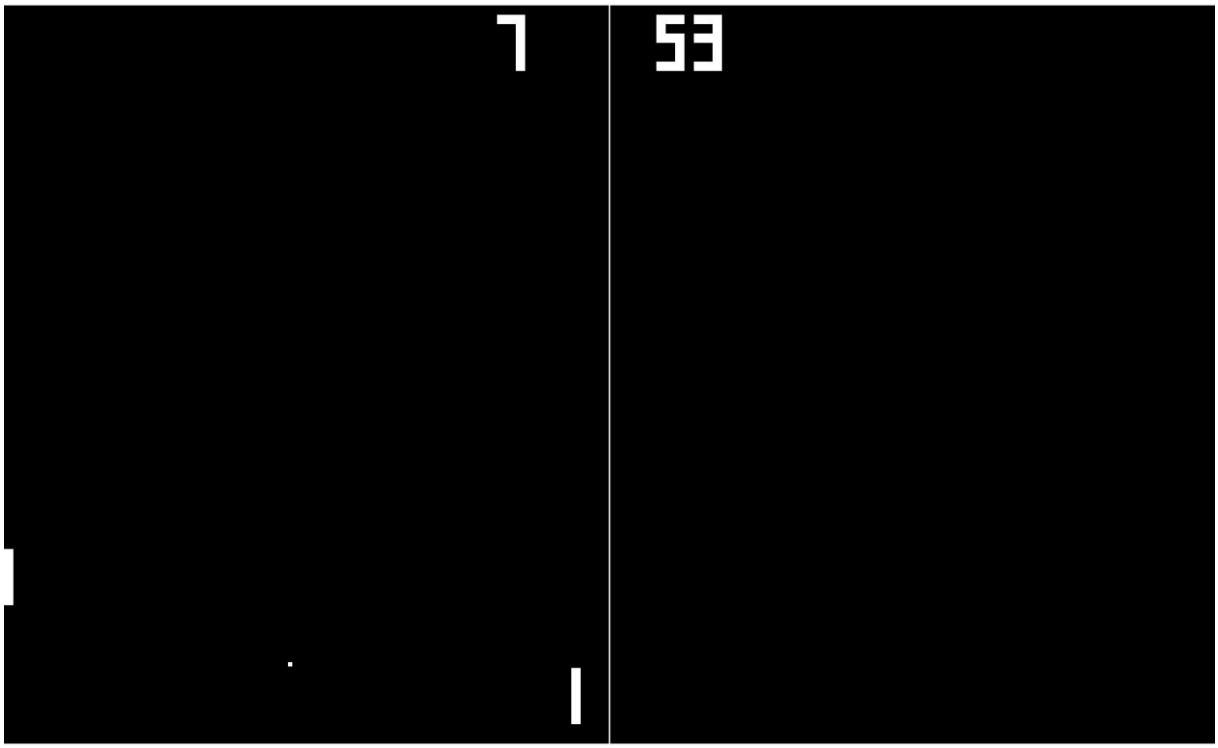
ID	Súlyosság	Szöveg / Definíció
105	**DEBUG**	**"Minute ticked successfully"** *A percszámláló sikeresen váltott.*
203	**ERROR**	**"JavaPongClock failed to enter into windowed mode."** *Az alkalmazás ablakos megjelenítési módjára váltás sikertelen volt.*

4.4 Fullscreenre váltás error (kiváltási módja: Try blokkba int a = 1/0)

```
2011.04.30 19:53 [main] INFO  root  - JavaPongClock initialized and started successfully.
2011.04.30 19:53 [main] INFO  root  - JavaPongClock entered into windowed mode.
2011.04.30 19:53 [main] ERROR root  - JavaPongClock failed to enter into fullscreen mode.
```

Fullscreenbe kapcsoláskor hibaüzenetet naplózunk, elvárt kimenet (továbbá ahogy a 2. ábrán látszik a fullscreen módban futtatott program tényleg furán működik).

ID	Súlyosság	Szöveg / Definíció
202	**ERROR**	**"JavaPongClock failed to enter into fullscreen mode."** *Az alkalmazás teljesképernyős megjelenítési módjára váltás sikertelen volt.*



2. ábra Fullscreen hiba

4.5 Inicializáció sikertelen (kiváltási módja: Try blokkba int a = 1/0)

```
2011.04.30 19:57 [main] WARN  root  - Invalid or out-of-bounds parameters, resetting to
default screen size.
2011.04.30 19:57 [main] INFO  root  - JavaPongClock entered into fullscreen mode.
2011.04.30 19:57 [main] FATAL root  - JavaPongClock initization unsuccessful!
JavaPongClock will now exit.
```

ID	Súlyosság	Szöveg / Definíció
201	FATAL	**"JavaPongClock initization unsuccessful! JavaPongClock will now exit."** *Az inicializáció sikertelen, a program így kilép.*

4.6 Óra váltás (kiváltási mód: automatikus)

```
2011.04.30 20:00 [main] DEBUG root  - Minute ticked.
2011.04.30 20:00 [main] DEBUG root  - Hour ticked.
```

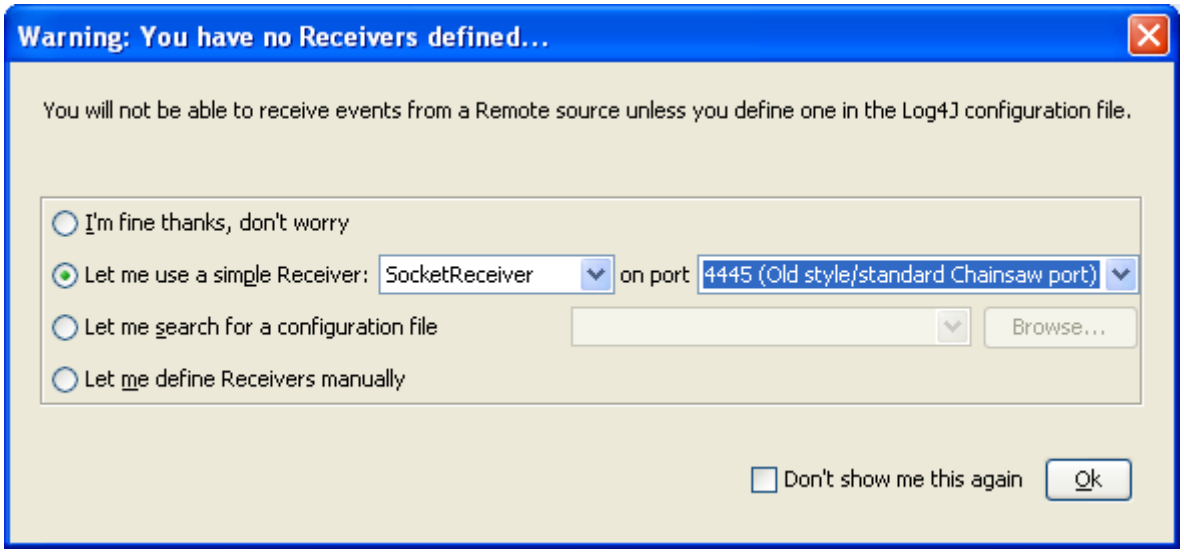
ID	Súlyosság	Szöveg / Definíció
105	DEBUG	**"Minute ticked."** *A percszámláló sikeresen váltott.*
106	DEBUG	**"Hour ticked. "** *A percszámláló sikeresen váltott.*

4.7 Remote Appender tesztelése és beállításai

A remote appendert figyelő szolgáltatást egy Windows XP SP3 virtuális gépen futtattam. A program neve Chainsaw, szintén Apache „termék”, egy GUI-val rendelkező log böngésző, amely rendelkezik távoli figyelő szolgáltatással. Az automatikus figyelő szolgáltatást használtam, és a JavaPongClock

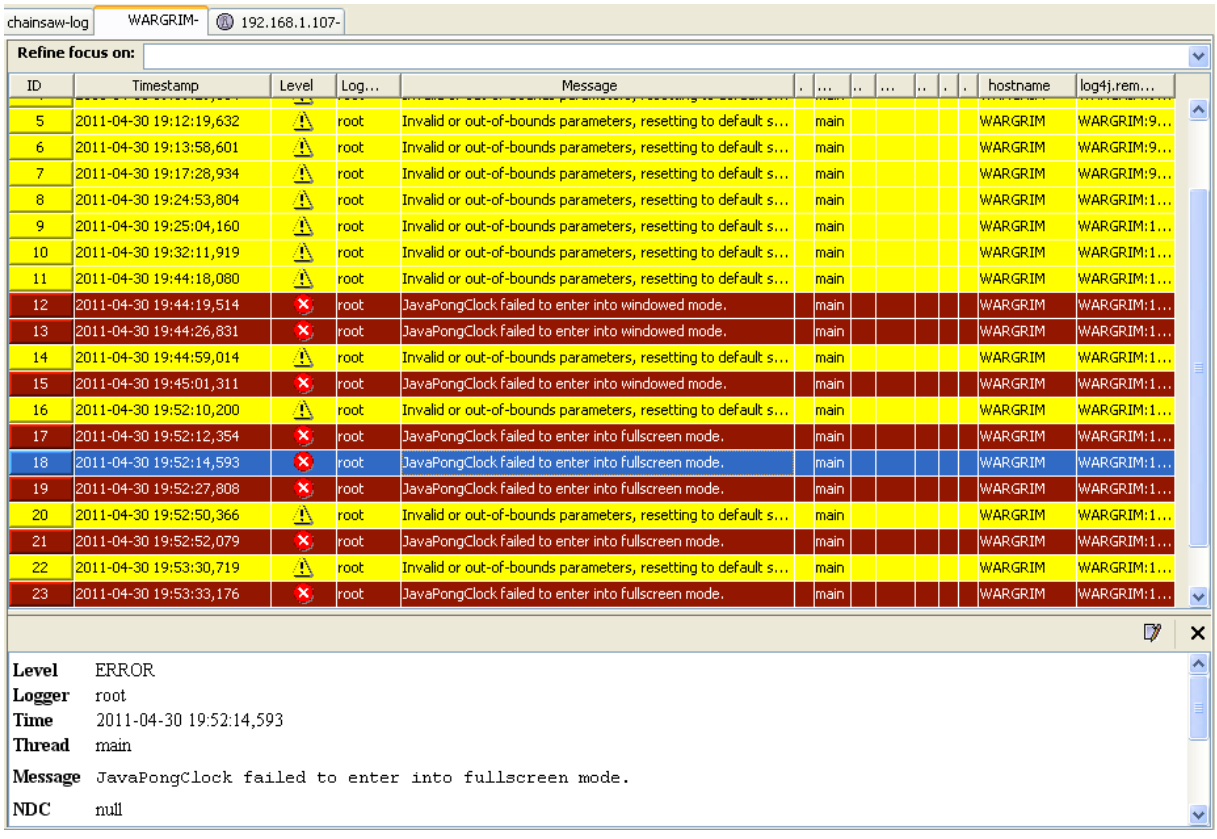
alkalmazáshoz tartozó log4j.properties file-ban a SocketAppender címe és portja a Windows XP SP3 számítógépnek megfelelően volt beállítva.

Chainsaw beállítása (3. ábra):



3. ábra Chainsaw-ban szolgáltatás indítása

Miközben a fenti teszteket futtattam, a szolgáltatás végig működésben volt, és meg is jelentek a fenti hibaüzenetek és figyelmeztetések a számítógépnek megfelelő fülön (4. ábra):



4. ábra Chainsaw figyelőben megjelenő események

Tehát a távoli appender megfelelően működik, a teszt sikeres volt.

5 Hivatkozások

[1] BME MIT, *Hogyan készítsünk házi feladat dokumentációt és mérési jegyzőkönyvet*, elérhető online: http://www.inf.mit.bme.hu/edu/dokumentacio

[2] Log4j letöltése és leírása: http://logging.apache.org/log4j/index.html és http://www.google.com

[3] Chainsaw letöltése és leírása http://logging.apache.org/chainsaw/index.html