



Modellek programozott feldolgozása

Rendszertervezés laboratórium 1

Jegyzőkönyv

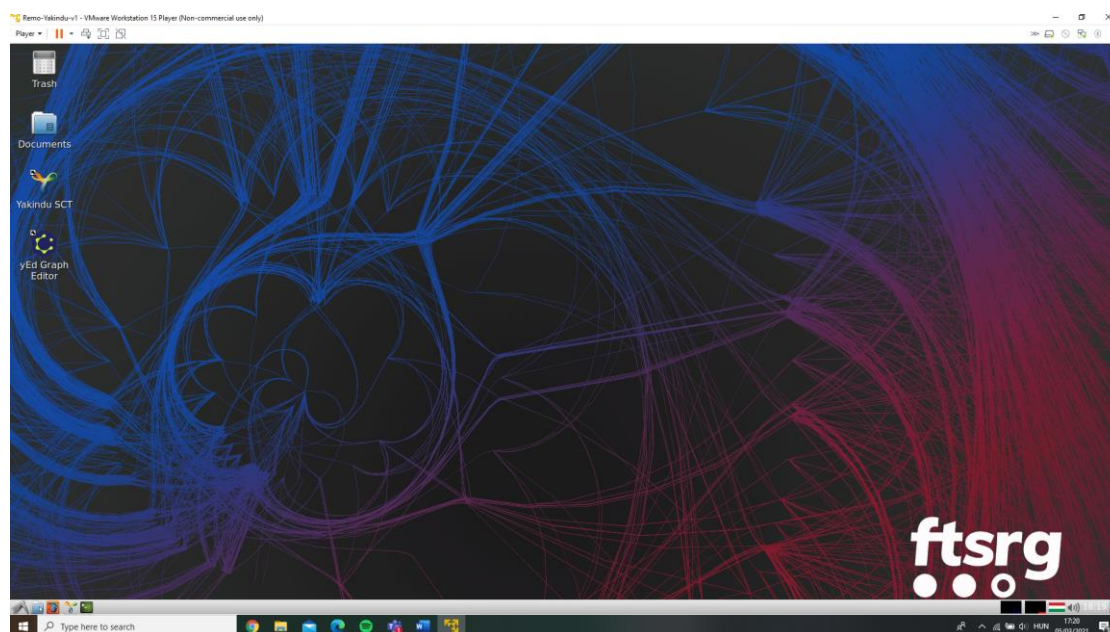
2021. tavasz

Készítette	Somogyi Bence (Q79IBL)
Dátum	2021. március 05.
GitHub hivatkozás	https://github.com/Bence56/RETElab1MIT2
Bónusz feladat	elkészült

1 Bevezetés

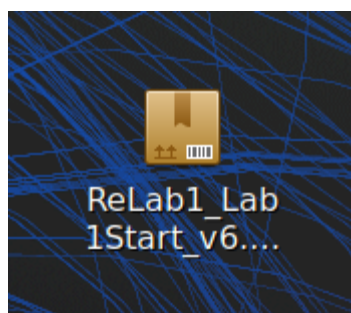
1.1 Virtuális gép elérése

Letöltöttem a virtuális gép fileját és VMWare Player 15-öt használok a feladatok megoldására?

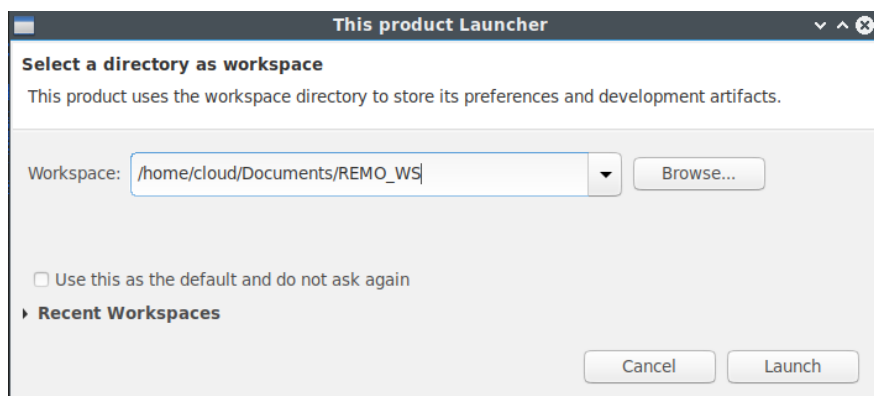


1.2 Kiindulási fájlok letöltése

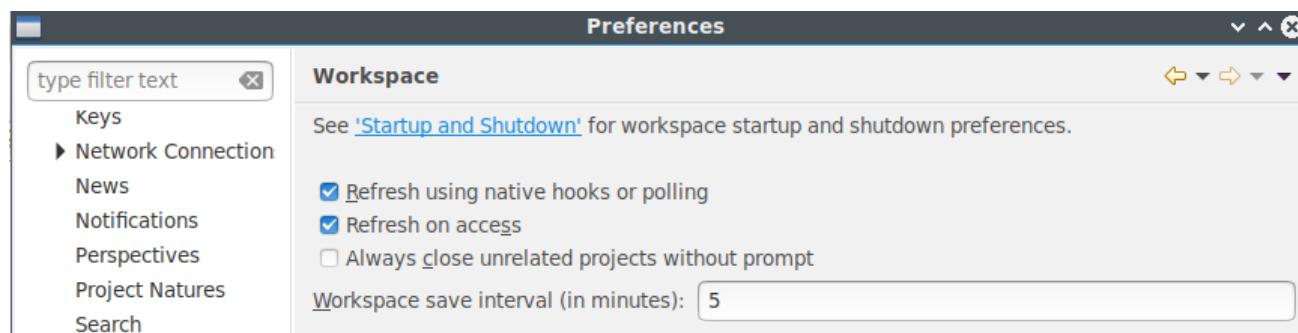
Volt telepítve egy tool már a VMWare playeremhez, ami megengedi, hogy a host gépről CTRL+C CTRL+V kombinációkkal fileokat másoljak a host gépről a guest gépre. Nem tudom miért, de a guest gép nem érte el az internetet, ezért így volt a legegyszerűbb átmásolni az állományokat.



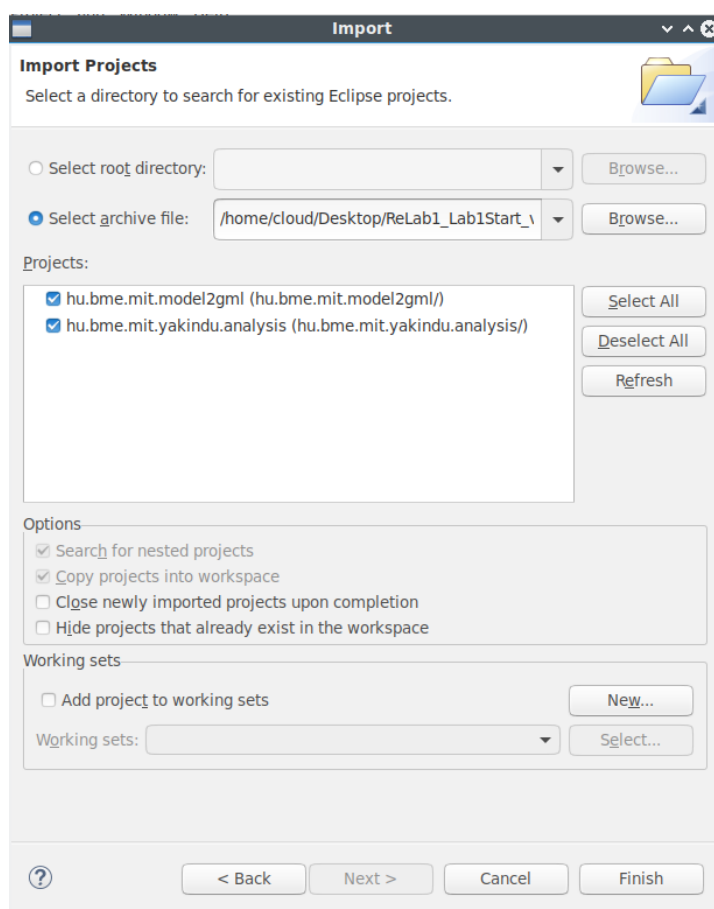
1.3 Yakindu SCT megnyitása az alapértelmezett workspace-szel



1.4 Tipp alkalmazása, az automatikus frissítésre



1.5 Projekt importálása

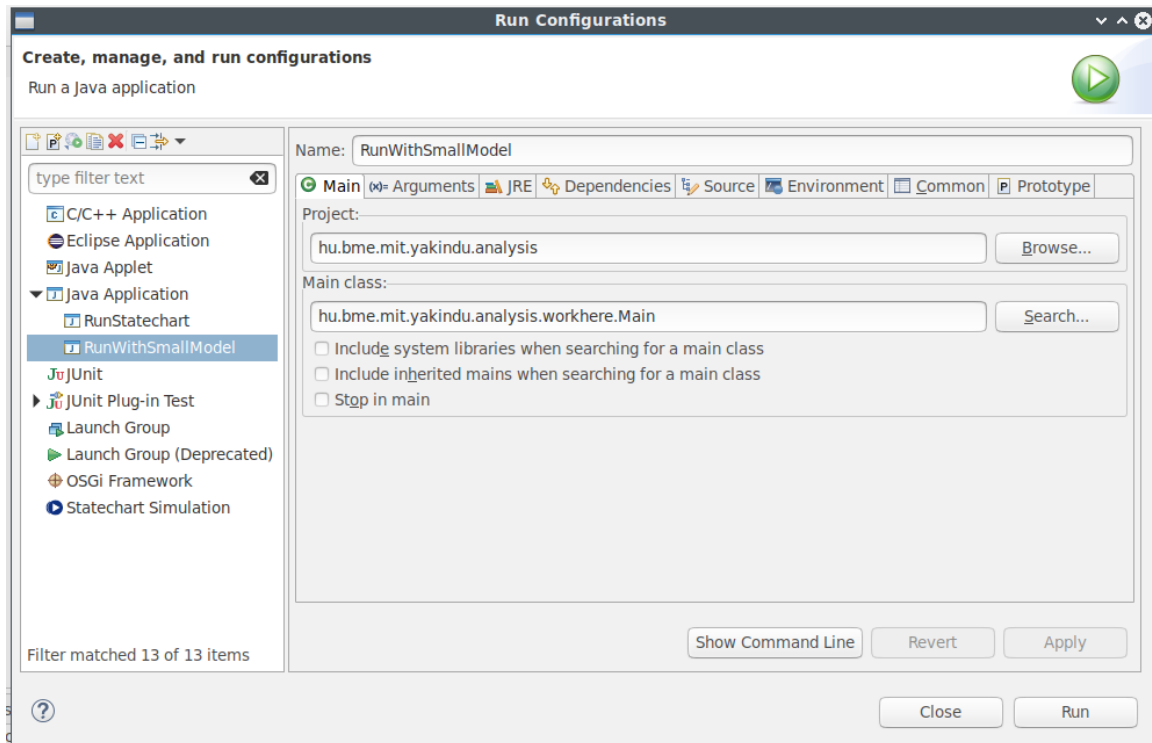


1.6 Projekt átnézése

Eloolvastam, megnéztem a leírtakat.

2 Modell bejárása

2.1 Futtatás RunWithSmallModel konfigurációval

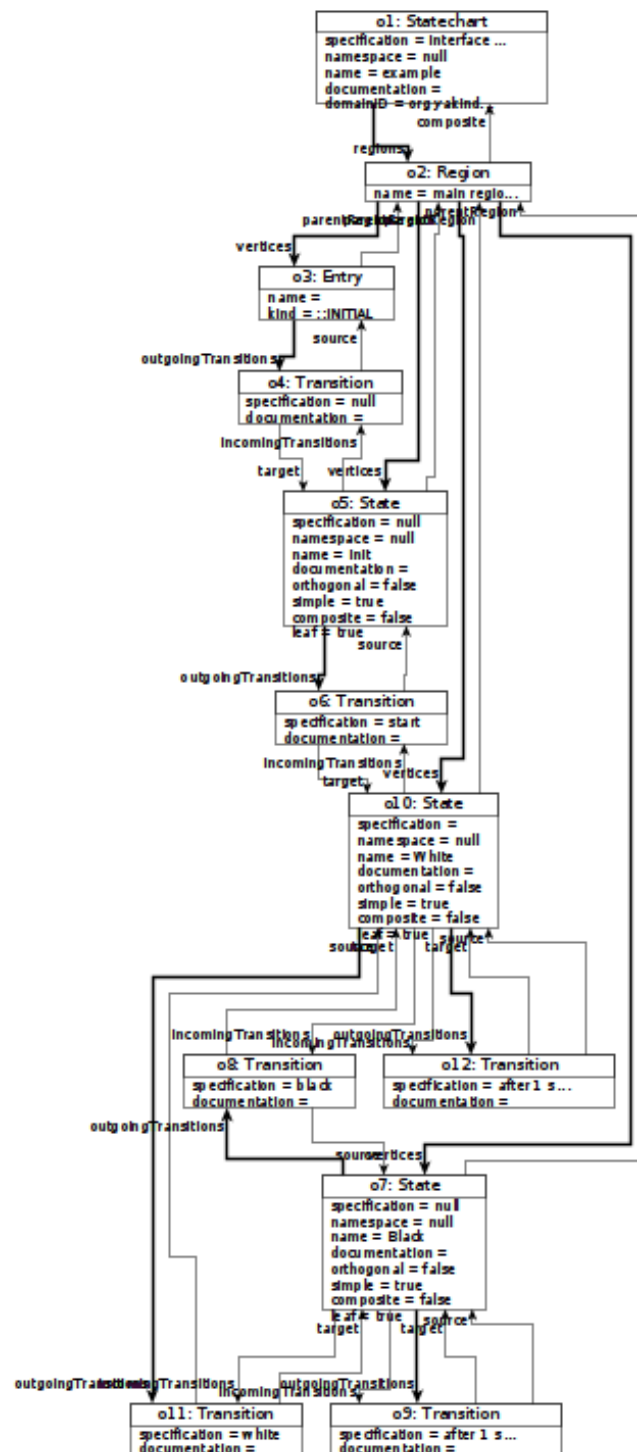


A kimenet a következő:

```
Tasks Console
<terminated> RunWithSmallModel [Java Application] /usr/lib/jvm/java-11-openjdk-amd64/bin/java (Mar 5, 2021, 9:17:12 PM)
Init
Black
White
```

2.2 Generált vizuális fájl megtekintése

A fájl yEd alkalmazással való megnyitása után, alkalmaztam a Layout → Hierarchical → Ok műveletsort. Sajnos a kép nem a legszebb a felbontás miatt. A vizuális modell a következő:



2.3 Tranzíciók kiírása

A feladatot kiegészíteném annyival, hogy a kezdőállapot előtt nincs állapot (Example állapotgép esetén Init állapot). Ezért az elé a következőt írom: „Kezdőállapot :”. A kiegészített forráskód a következő képen látható.

```
// Reading model
Statechart s = (Statechart) root;
TreeIterator<EObject> iterator = s.eAllContents();
int i = 0;
State elozostate=null;
while (iterator.hasNext()) {
    EObject content = iterator.next();

    if(content instanceof State) {
        State state = (State) content;
        if(i==0) {
            System.out.print("Kezdőállapot : ");
            i++;
        }
        else {
            System.out.print(elozostate.getName() + " -> ");
        }
        System.out.println(state.getName());
        elozostate = state;
        //i++;
    }
}

// Transforming the model into a graph representation
String content = model2gml.transform(root);
// and saving it
manager.saveFile("model_output/graph.gml", content);
}
```

Tasks Console

<terminated> RunWithSmallModel [Java Application] /usr/lib/jvm/java-11-openjdk-amd64/bin/java (Mar 5, 2021, 9:45:55 PM)

Kezdőállapot ---->Init
Init -> Black
Black -> White

2.4 Csapdaállapotok kiírása

A kód amivel kiegészíthetem a következő képen látható. Sajnos nem fér el az egész képernyőre és csak a fontos, és új részeket szeretném mutatni. Ezen kívül egy listát csináltam a csapdaállapotoknak. Ez a kódban feljebb látható. Ennek a kódja: `ArrayList<State> csapdaallapotok = new ArrayList<State>();`

```

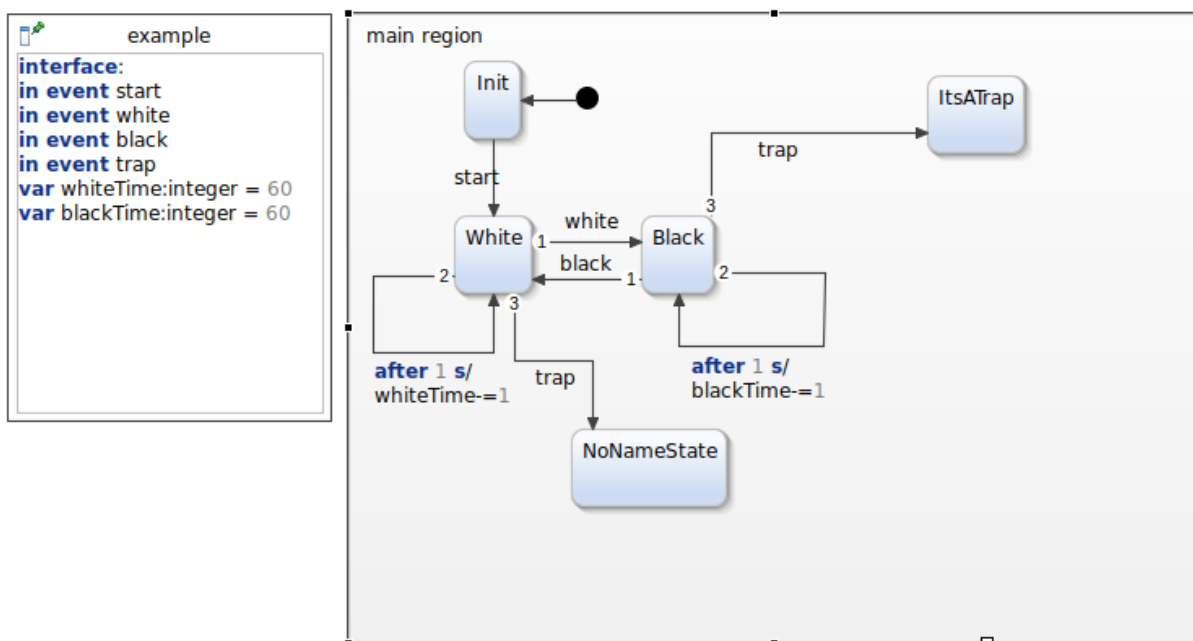
while (iterator.hasNext()) {
   EObject content = iterator.next();

    if(content instanceof State) {
        State state = (State) content;
        if(i==0) {
            System.out.print("Kezdőállapot : ");
            i++;
        }
        else {
            System.out.print(elozostate.getName() + " -> ");
        }
        System.out.println(state.getName());
        elozostate = state;
        //i++;

        if(state.getOutgoingTransitions().isEmpty()) {
            csapdaallapotok.add(state);
        }
    }
}
if(csapdaallapotok.size() == 0) {
    System.out.println("Nincs csapda állapot");
}
else {
    System.out.println("Csapdaállapotok: ");
    for(int j=0; j<csapdaallapotok.size();j++) {
        System.out.println(csapdaallapotok.get(j).getName());
    }
}

```

A módosított modell:



A kimenet a módosított modellre:

```
Kezdőállapot : Init
Init -> Black
Black -> White
White -> ItsATrap
ItsATrap -> NoNameState
Csapdaállapotok:
ItsATrap
NoNameState
```

Mint az látható, nem az igazi tranzíció kiírás...

Tudom, hogy fa (gráf) bejárási algoritmust kellene implementálni, de úgy gondolom nem ez a labor célja. Idő hiányában nem oldom meg helyesen a feladatot.

2.5 Névgenerálás névnélküli állapotoknak

Készítettem egy egyszerű függvényt, ami konkatenál egy állandó stringet és egy stringesített, statikus, állandóan változó(növekvő) számot. Elsőre, nem tűnik túl beszédesnek az előtag, de szerintem kellően felhívja a figyelmet a felhasználó számára, hogy nevezze át az állapotot.

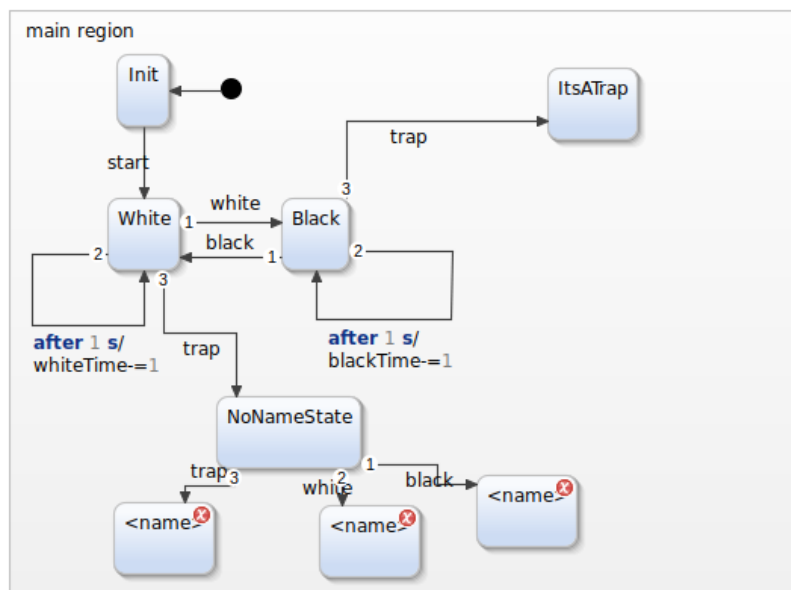
```
public static String nevgenerator() {
    String elotag = "NAMELESSSTATE_";
    int kozvetkezoszam=szam;
    szam++;
    String ret =elotag;
    String num=String.valueOf(kozvetkezoszam);
    ret.concat(num);
    return ret;
}
```

2.6 Ellenőrzés, esetleges névegyezésre.

A következő képen látható a névegyezés ellenőrzése, és az átnevezés is. Ez igazából az előző feladathoz tartozik, de így több, releváns kód látható egy képen.

```
System.out.println("Nevnelkuli állapotok száma: "+nevnelkuliallapotok.size());
for(int j=0;j < nevnelkuliallapotok.size();j++) {
    String nev=nevgenerator();
    boolean jonev=false;
    while(!jonev) {
        int cnt=0;
        for(int k=0;k < nevnelkuliallapotok.size();k++) {
            if(mindenallapot.get(k).getName()==nev) {
                cnt++;
            }
        }
        if(cnt>0) {
            jonev=true;
        }
        else {
            jonev=false;
            nev=nevgenerator();
        }
    }
    System.out.println("Ezt a nevet javaslom az állapotnak: "+nev);
    nevnelkuliallapotok.get(j).setName(nev);
}
```


Módosítottam a modellt:



Azért a trap átmenetet használtam, mert nem akartam jobban komplikálni a modellt. A lényeges részek látszódnak a részfeladat szempontjából. A lefutás a modellre:

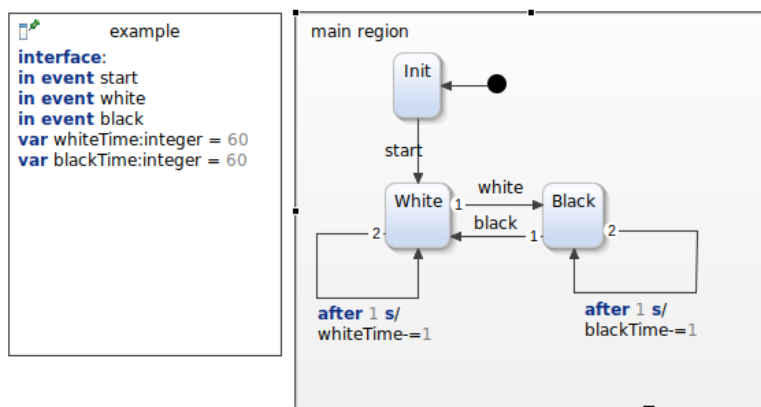
```
Kezdőállapot : Init
Init -> Black
Black -> White
White -> ItsATrap
ItsATrap -> NoNameState
NoNameState -> null
null -> null
null -> null
Csapdaállapotok:
ItsATrap
null
null
null
Nevnelkuli állapotok szama: 3
Ezt a nevet javaslom az állapotnak: NAMELESSSTATE_0
Ezt a nevet javaslom az állapotnak: NAMELESSSTATE_1
Ezt a nevet javaslom az állapotnak: NAMELESSSTATE_2
```

Sajnos mégjobban kijönnek a tranzíciós rész hibái ☹

3 Yakindu kódgenerátor használata

3.1 Modell visszaállítása

A modellt visszaállítottam:



3.2 Kódgenerátor kiegészítése

A kód bemásolása után a következőket tudtam meg: A *GeneralFeatures* megengedi, hogy konfiguráljunk különböző szolgáltatásokat, amik az állapotgépen kívül különböző dolgokat generálnak. Alapvetően az összes érték *false*, azaz nem generál semmit. Opcionális funkciói:

- *InterfaceObserverSupport*: Engedélyezi, vagy tiltja (nem engedélyezi) az állapotgép *ListenerInterface*-ek generálását.
- *RuntimeService*: Engedélyezi, vagy tiltja (nem engedélyezi) a futásidejű szolgáltatások generálását, ami *triggereli* az életciklus eseményeket az életciklus-alapú állapotgépekben.
- *TimerService*: Engedélyezi, vagy tiltja (nem engedélyezi) az időzítéssel kapcsolatos események generálását, ami a *java.util.Timer* könyvtárat használja.

3.3 RunStateChar.java file áttekintése

Miután átnéztem a kódot bemásoltam az importokat, és kikommenteztem a kódot.

3.4 Futtatás RunStatechart.launch konfigurációval

A futtatás kimenete:

```
Tasks Console
<terminated> RunStatechart [Java Application] /usr/lib/jvm/java-11-openjdk-amd64/bin/java (Mar 6, 2021, 3:00:35 PM)
W = 60
B = 60
I am allowed to write anything here
W = 37
B = 60
```

3.5 Alkalmazás készítése

Az alkalmazást a *hu.bme.mit.yakindu.analysis* projektben, azon belül a *hu/bme/mit/yakindu/analysis/workhere* packageben készítettem el *ExcerciseToWriteAnApp* néven. A *main* függvényben inicializálom az állapotgépet, a kezdeti változókat, belépek a ciklusba. A tranzíció beolvasása után meghívom a *transition* függvényt, paraméterei az állapotgép és a beolvasot sor.

```
public static void main(String[] args) throws IOException {
    // TODO Auto-generated method stub
    ExampleStatemachine stm = new ExampleStatemachine();
    stm.setTimer(new TimerService());
    RuntimeService.getInstance().registerStatemachine(stm, 200);
    stm.init();
    stm.enter();
    stm.runCycle();
    boolean exit=false;
    BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
    while(exit == false) {
        System.out.print("Write a transition: ");

        String line = reader.readLine();

        exit = transition(line,stm);
        if(exit==false)
            exit = isEnoughTimeLeft(stm);
    }

    System.out.println("End of game!");
    System.exit(0);
}
```

A függvény ellenőrzi, hogy melyik tranzíciót hívta tüzelte a felhasználó, azt tüzeli és meghívja a *runCycle()* függvényt. Ellenőrzi, hogy ki akar-e lépni a felhasználó az *exit* paranccsal. Ha ki akar lépni, akkor igaz ellenkező esetben hamis értékkel tér vissza. Ha nem talál ilyen parancsot a függvény, akkor hibaüzenetet küld, de nem dob kivételt.

```
private static boolean transition(String line,ExampleStatemachine stm) {
    if(line.equals("start")) {
        stm.raiseStart();
        stm.runCycle();
        print(stm);
    }
    else if(line.equals("black")) {
        stm.raiseBlack();
        stm.runCycle();
        print(stm);
    }
    else if(line.equals("white")) {
        stm.raiseWhite();
        stm.runCycle();
        print(stm);
    }
    else if(line.equals("exit")) {
        return true;
    }
    else {
        System.out.println("No transition like this");
    }
    return false;
}
```

A függvény amivel kiírom a játékosok hátralevő idejét a korábban tanulmányozott programból másoltam ki.

Végül fontosnak tartottam ellenőrizni, hogy vége van-e a játéknak. Nincs értelme bekérni tranzíciókat, ha valamelyik játékosnak elfogyott az ideje. A képen a kiírást végző függvény is látható.

```
public static boolean isEnoughTimeLeft(ExampleStatemachine stm) {
    if(stm.getSCInterface().getBlackTime()<=0 || stm.getSCInterface().getWhiteTime()<=0) {
        return true;
    }
    else {
        return false;
    }
}

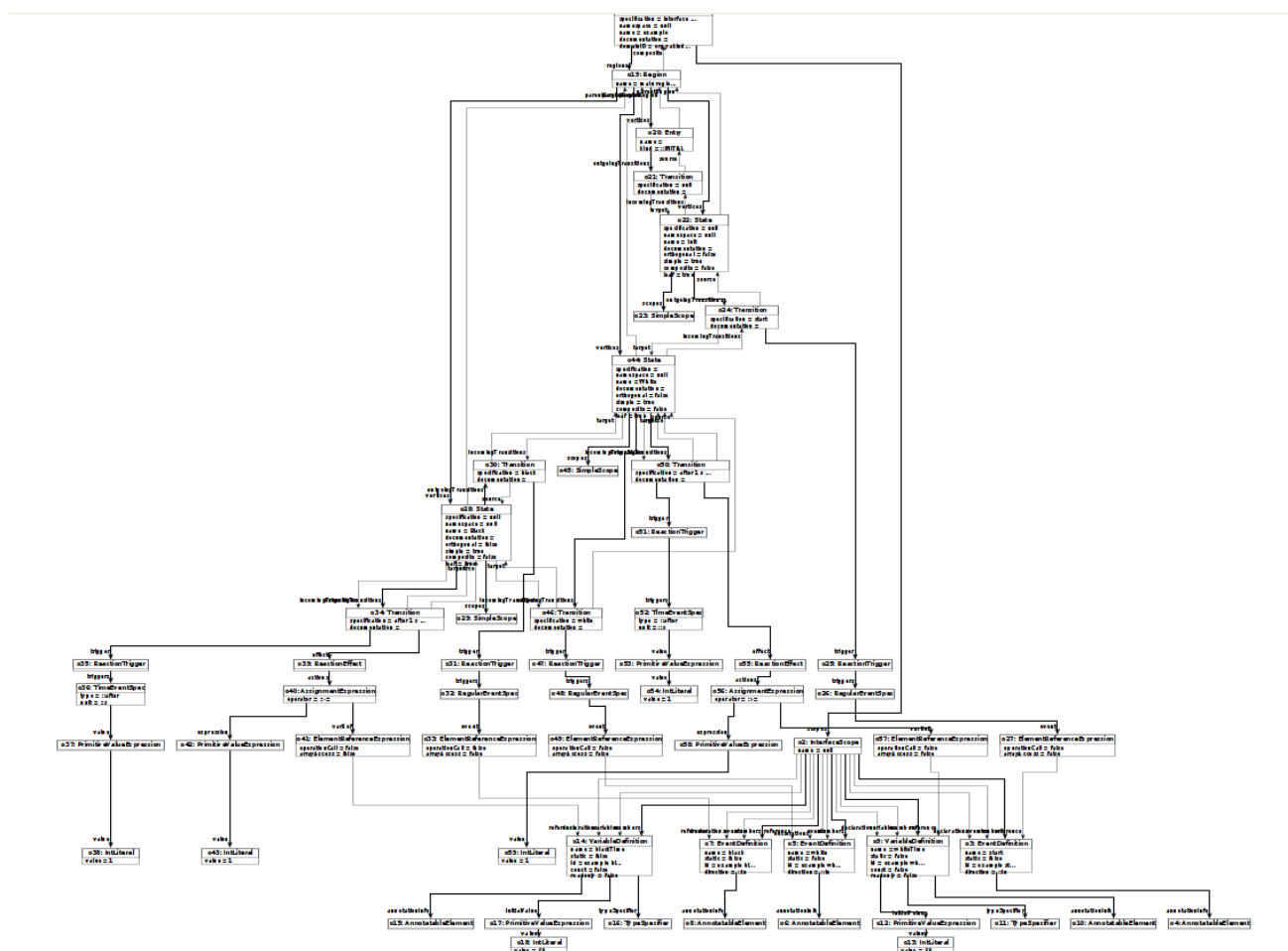
public static void print(IEExampleStatemachine stm) {

    System.out.println("\nW = " + stm.getSCInterface().getWhiteTime());
    System.out.println("B = " + stm.getSCInterface().getBlackTime());
}
```

4 Saját kódgenerátor készítése

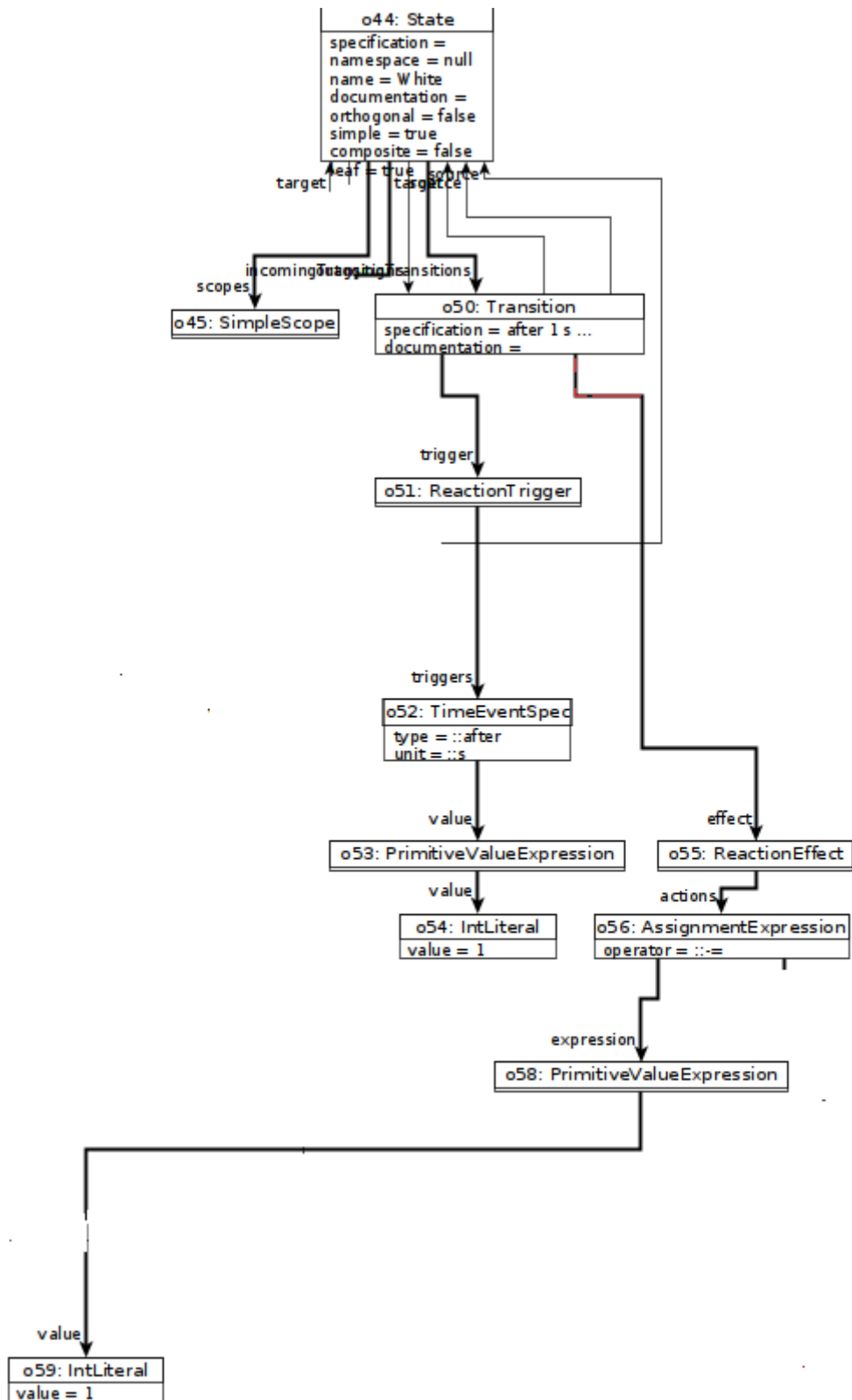
4.1 Generált gráf áttekintése

A launch file első sorából kiderül, hogy ez egy JUnitos konfiguráció. A generált gráf a következő képen látható, sokkal bővebb mint az előzőleg generált gráf.



4.2 whiteTime -= 1 az új gráfon

Az átláthatóság kedvéért próbáltam kitörölni az irreleváns részeket, nagyjából csak azt megtartani, ami a feladat megoldása szempontjából fontos.



4.3 Belső változók és bemenő események kiírása

Megtrükközött a running configuration, de sikerült:

```
System.out.println("Események: ");
iterator = s.eAllContents();
while(iterator.hasNext()) {
    EObject content = iterator.next();

    if(content instanceof Event) {
        Event ev= (Event) content;
        System.out.println("\t"+ev.getName());
    }
}
System.out.println("Változók: ");
iterator = s.eAllContents();
while(iterator.hasNext()) {
    EObject content = iterator.next();

    if(content instanceof VariableDefinition) {
        VariableDefinition ev= (VariableDefinition) content;
        System.out.println("\t"+ev.getName());
    }
}

// Transforming the model into a graph representation
```

Console [JUnit]

<terminated> CodeGenerator [JUnit Plug-in Test] /usr/lib/jvm/java-11-openjdk-amd64/bin/java (Mar 11, 2021, 3:19:26 PM)

Események:
start
white
black

Változók:
whiteTime
blackTime

4.4 Programkód kiírása

A kiegészített kód sajnos nem fér rá egy kijelzőre, de a *CodeGenerator.java* fileban megtalálható.

4.5 Kód kiírása

Az alábbi két képen a konzolról készített képernyőképek láthatóak:

```
public static void main(String[] args) throws IOException {
    // TODO Auto-generated method stub
    ExampleStatemachine stm = new ExampleStatemachine();
    stm.setTimer(new TimerService());
    RuntimeService.getInstance().registerStatemachine(stm, 200);
    stm.init();
    stm.enter();
    stm.runCycle();
    boolean exit=false;
    BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
    while(exit == false) {
        System.out.print("Write a transition: ");

        String line = reader.readLine();

        exit = transition(line,stm);
        if(exit==false)
            exit = isEnoughTimeLeft(stm);
    }

    System.out.println("End of game!");
    System.exit(0);
}

private static boolean transition(String line,ExampleStatemachine stm) {

    if(line.equals(Start)) {
        stm.raiseStart();
        stm.runCycle();
        print(stm);
    }

    else if(line.equals(White)) {
        stm.raiseWhite();
        stm.runCycle();
        print(stm);
    }

    else if(line.equals(Black)) {
        stm.raiseBlack();
        stm.runCycle();
        print(stm);
    }

    else {
        System.out.println("No transition like this");
    }
    return false;
}

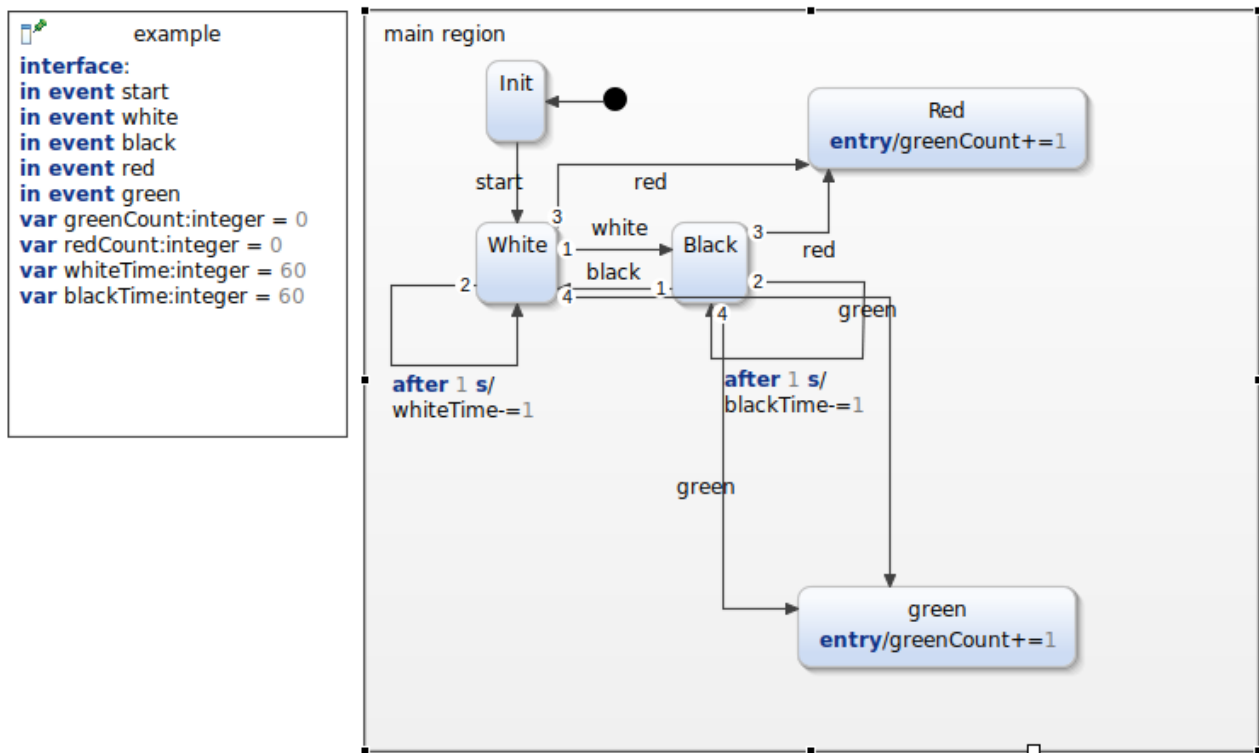
public static boolean isEnoughTimeLeft(ExampleStatemachine stm) {
    if(stm.getSCInterface().getBlackTime()<=0 || stm.getSCInterface().getWhiteTime()<=0) {
        return true;
    }
    else {
        return false;
    }
}

public static void print(IExampleStatemachine stm) {

    System.out.println("W = " + s.getSCInterface().getWhiteTime());
    System.out.println("B = " + s.getSCInterface().getBlackTime());
}
}
```

4.6 Modell módosítása, majd kód futtatása

A módosított modell:



Releváns kódrészletek arról, hogy a generator nem beégetett:

```
        else if(line.equals(red)) {
            stm.raiseRed();
            stm.runCycle();
            print(stm);
        }

        else if(line.equals(green)) {
            stm.raiseGreen();
            stm.runCycle();
            print(stm);
        }

        else {
            System.out.println("No transition like this");
        }
        return false;
    }

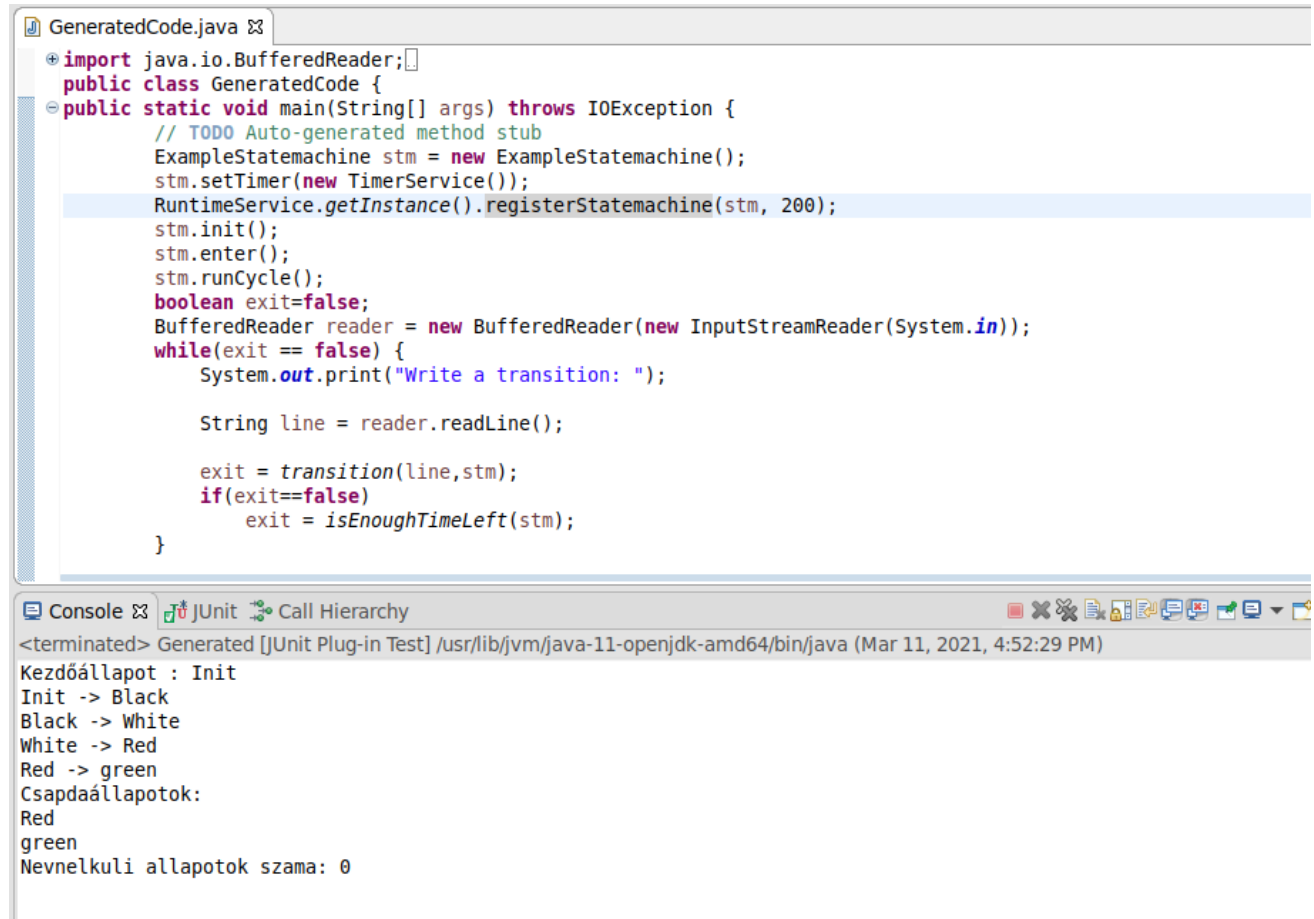
    public static boolean isEnoughTimeLeft(ExampleStatemachine stm) {
        if(stm.getSCInterface().getBlackTime()<=0 || stm.getSCInterface().getWhiteTime()<=0) {
            return true;
        }
        else {
            return false;
        }
    }

    public static void print(IEExampleStatemachine stm) {

        System.out.println("G = " + s.getSCInterface().getGreenCount());
        System.out.println("R = " + s.getSCInterface().getRedCount());
        System.out.println("W = " + s.getSCInterface().getWhiteTime());
        System.out.println("B = " + s.getSCInterface().getBlackTime());
    }
}
```


4.7 Futtatható kód készítése

Készítettem egy filebaírót, ami beírja a *GeneratedCode.java* fileba ami a workspace gyökér könyvtárában, ahol vannak a launch fileok. A következő kép már ennek a file-nak a futtatásának az eredménye. A konfiguráció amivel futtattam a BigModel konfiguráció duplikálásával készült. Az indentálás nem szép, de ez egy nagyon jó feladat volt! Különösen izgalmas és élvezetes volt számomra.



The screenshot shows an IDE with two panels. The top panel displays the `GeneratedCode.java` file, and the bottom panel shows the console output.

```
GeneratedCode.java
import java.io.BufferedReader;
public class GeneratedCode {
    public static void main(String[] args) throws IOException {
        // TODO Auto-generated method stub
        ExampleStatemachine stm = new ExampleStatemachine();
        stm.setTimer(new TimerService());
        RuntimeService.getInstance().registerStatemachine(stm, 200);
        stm.init();
        stm.enter();
        stm.runCycle();
        boolean exit=false;
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
        while(exit == false) {
            System.out.print("Write a transition: ");

            String line = reader.readLine();

            exit = transition(line,stm);
            if(exit==false)
                exit = isEnoughTimeLeft(stm);
        }
    }
}
```

Console

```
<terminated> Generated [JUnit Plug-in Test] /usr/lib/jvm/java-11-openjdk-amd64/bin/java (Mar 11, 2021, 4:52:29 PM)
Kezdőállapot : Init
Init -> Black
Black -> White
White -> Red
Red -> green
Csapdaállapotok:
Red
green
Nevnelkuli állapotok szama: 0
```