Magdalena Osiewicz, IO, gr 2.

Rozproszona sztuczna inteligencja.

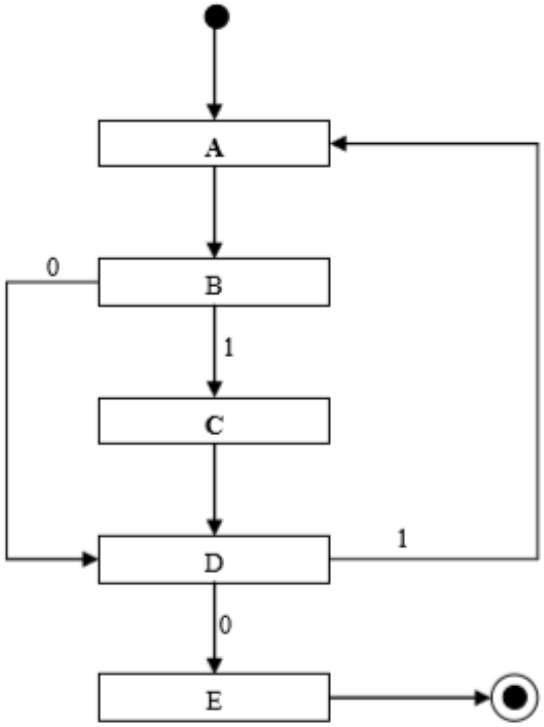
Sprawozdanie nr 8.

Temat: Zachowania – część 2.

W ramach zajęć zrealizowałam następujące kroki:

1. Utworzenie klasy agenta o nazwie Klasa_1_2.

Agent ten wykonuje zachowanie odwzorowujące następującą maszynę skończenie stanową:



Stany A, C i E polegają na wypisaniu nazwy stanu. Przejścia z tych stanów następują bezwarunkowo dalej. W stanach B i D również następuje wypisanie nazwy stanu, ale oprócz tego losowana jest liczba ze zbioru 0 i 1, która jest zwracana w chwili kończenia się zachowań związanych ze stanami.

W tym celu należało wykorzystać bardziej skomplikowane zachowanie jakim jest FSMBehaviour (ang. Finite-State-Machine Behaviour), które działa właśnie jak skończona maszyna stanów.
W tej klasie definiujemy zachowania reprezentujące różne stany systemu oraz przejścia pomiędzy poszczególnymi stanami zdeterminowanymi przez wydarzenie kończące poprzedni stan.

```
//rejestracja pojedynczego zachowania jako poczatkowe
 fsm.registerFirstState(new NamePrinter(),  STATE_A);
 //rejestracja jako stany posrednie
 fsm.registerState(new RandomGenerator( max: 1),  STATE_B);
 fsm.registerState(new NamePrinter(),  STATE_C);
 fsm.registerState(new RandomGenerator( max: 1),  STATE_D);
 //rejestracja jako stan koncowy
 fsm.registerLastState(new NamePrinter(),  STATE_E);

 //przejscie ze stanu do stanu, niezaleznie od zdarzenia zakonczenia stanu
 fsm.registerDefaultTransition( STATE_A,  STATE_B);
 fsm.registerTransition( STATE_B,  STATE_C,  event: 1);
 fsm.registerTransition( STATE_B,  STATE_D,  event: 0);
 fsm.registerDefaultTransition( STATE_C,  STATE_D);
 fsm.registerTransition( STATE_D, STATE_E,  event: 0);
 fsm.registerTransition( STATE_D,  STATE_A,  event: 1);

 addBehaviour(fsm);
```

Metody ***RegisterFirstState()*** i ***registerLastState()*** pozwalają nam rejestrować zachowania jako początkowe i końcowe.
RegisterState() używa jest do dodawania stanu do instancji FSMBehaviour.
Przyjmuje dwa argumenty:
- nazwa zarejestrowanego stanu
- zachowanie, które będzie w nim wykonane.

RegisterTransition() z kolei używana jest do dodawania zmiany stanu do intancji FSMBehaviour. Akceptuje trzy argumenty: stan początkowy, docelowy stan oraz wartość typu int definiującą etykietę oznaczającą przejście.

Wyniki działania agenta klasy Klasa1_2 w zależności od wylosowanych na poszczególnych etapach wartości:

```
INFO: ----------------------------------------
Agent container Main-Container@192.168.0.52 is ready.
--------------------------------------------
Stan: A
Stan: B
       Wylosowano: 1
Stan: C
Stan: D
       Wylosowano: 1
Stan: A
Stan: B
       Wylosowano: 0
Stan: D
       Wylosowano: 1
Stan: A
Stan: B
       Wylosowano: 1
Stan: C
Stan: D
       Wylosowano: 1
Stan: A
Stan: B
       Wylosowano: 1
Stan: C
Stan: D
       Wylosowano: 0
Stan: E
```

```
INFO: ----------------------------------------
Agent container Main-Container@192.168.0.52 is ready.
--------------------------------------------
Stan: A
Stan: B
       Wylosowano: 1
Stan: C
Stan: D
       Wylosowano: 0
Stan: E
```

```
INFO: ----------------------------------------
Agent container Main-Container@192.168.0.52 is ready.
--------------------------------------------
Stan: A
Stan: B
       Wylosowano: 0
Stan: D
       Wylosowano: 0
Stan: E
```

2. Utworzenie zachowania, które będzie polegało na równoległym wykonywaniu trzech zachowań „generycznych” z zadania poprzedniego (z pliku Klasa_4.java).

```
protected void setup() {

    System.out.println("startuje");
    final ParallelBehaviour par = new ParallelBehaviour();

    par.addSubBehaviour( new OneShotBehaviour()
    {
        public void action() { System.out.println( "pierwszy krok!" ); }
    });

    par.addSubBehaviour( new OneShotBehaviour()
    {
        public void action() { System.out.println( "drugi krok!" ); }
    });

    par.addSubBehaviour( new OneShotBehaviour()
    {
        public void action() {
            System.out.println( "trzeci krok!" );
            removeBehaviour(par);
            System.out.println( "usuwam!" );
        }
    });
    addBehaviour( par );

}
```

Klasa ParallelBehaviour wykonuje grupę zachowań równolegle.

Oznacza to, że za każdym razem, gdy metoda *action()* równoległego zachowania jest realizowana, to odwołuje się do metody *action()* obecnego „dziecka” a następnie posuwa do przodu do następnego zachowania niezależnie od tego czy poprzednie zostało zrealizowane.

(bez wywłaszczeń)

Zachowanie zostało dodane do agenta, którego klasa nosi nazwę Klasa_2_3.

```
INFO: MTP addresses:
http://192.168.0.52:7778/acc
startuje
cze 16, 2018 12:11:04 PM jade.core.AgentContainerImpl joinPlatform
INFO: ----------------------------------------
Agent container Main-Container@192.168.0.52 is ready.
--------------------------------------------
pierwszy krok!
trzeci krok!
usuwam!
zaraz sie usune!
```

3. Utworzenie zachowania, które będzie polegało na sekwencyjnym wykonywaniu trzech zachowań „generycznych” z zadania drugiego (z Klasa_4.java).

```
final SequentialBehaviour threeStepBehaviour = new SequentialBehaviour();
threeStepBehaviour.addSubBehaviour( new OneShotBehaviour()
{
    public void action() { System.out.println( "pierwszy krok!" ); }
});
threeStepBehaviour.addSubBehaviour( new OneShotBehaviour()
{
    public void action() { System.out.println( "drugi krok !" ); }
});
threeStepBehaviour.addSubBehaviour( new OneShotBehaviour()
{
    public void action() {
        System.out.println( "trzeci krok!" );
        removeBehaviour(threeStepBehaviour);
        System.out.println( "usuwam!" );
    }
});
addBehaviour(threeStepBehaviour);
```

Zachowanie zostało dodane do agenta, którego klasa nosi nazwę Klasa_2_4.

Klasa *SequentialBehaviour* zarządza grupą podległych jej zachowań sekwencyjnie.

Klasa ta realizuje złożone zachowanie, które planuje zachowanie jego podzachowań w następujący sposób:

- zaczyna od pierwszego dziecka; gdy to jest skończone (t.j. jego metoda done () zwrocila true) to przechodzi do drugiego i tak dalej.

- gdy ostatnie podzachowanie jest zrealizowane, całe zachowanie wygasa

```
INFO: MTP addresses:
http://192.168.0.52:7778/acc
startuje
pierwszy krok!
drugi krok !cze 16, 2018 12:47:29 PM jade.core.AgentContainerImpl joinPlatform
INFO: ----------------------------------------
Agent container Main-Container@192.168.0.52 is ready.
--------------------------------------------

trzeci krok!
usuwam!
zaraz sie usune!
```

4. Utworzenie agenta, który będzie wykonywał dwa zachowania cykliczne (wypisujące odpowiednio „cyclic 1” oraz „cyclic 2”) w dwóch osobnych wątkach.

```
ParallelBehaviour par = new ParallelBehaviour();
par.addSubBehaviour( new CyclicBehaviour()
{
    public void action() {System.out.println( "cyclic_1" ); }
});

par.addSubBehaviour( new CyclicBehaviour()
{
    public void action() { System.out.println( "cyclic_2" ); }
});
addBehaviour(par);
```

```
cyclic_2
cyclic_1
cyclic_1
cyclic_1
cyclic_1
cyclic_1
cyclic_1
cyclic_1
cyclic_1
cyclic_1
cyclic_2
cyclic_2
cyclic_1
cyclic_1
cyclic_2
```