Computer simulation I

is a very important topic

- Considering a problem complicated enough to get imagined, we should get an opinion of it.
- We may simulate many topics (e.g., injury – namely its healing, how the alcohol gets spreaded through the organism, how the elevators are serving people...)
- Computer simulation is a simulation where for modelling a computer-program is used.
- The aim is to decide how the simulated objects interact, i.e., how the whole simulated system works.
- Simulation should not optimize the processes!
- Results may differ, basic information is time of end (of a simulation).

Computer simulation II

- Continuous VS Discrete event simulation,
- problem of continuous simulation usually needs some (differential) equation to get solved,
- we restrict our attention to discrete event simulation,
- continuous simulation can be approximated making small steps and recalculating very often (which needs so called stability of the system).

Cars transporting a sand

very typical problem on our lectures

- We want to move a sand-heap to a building yard [site].
- We have a given number of workers and a given number of cars,
- along the path, critical sections may turn up (at the building year and at the heap, too).
- Number of workers is bounded, road may be narrow (for one car only driven by traffic-light or there are no overtaking zones).
- How to schedule workers and cars to move the heap as soon as possible?
- Discrete simulation simulates a given schedule (workers and cars) and determins when the heap will be moved completely.
- Usually we will consider one narrow zone (for one car only).

Elevators

- D. Marx (earlier leader of MERL) claimed that the industrial norm expects lifts to arrive within 30 seconds.
- Tests on humans are expensive (customers are running out of patience),
- thus an environment simulating people calling the lift may be useful.
- Interesting: Distribution of waiting-times for individual people.

Department store

- How to distribute goods in a department store (supermarket) to make average customer pass through the whole shop (to convince them to buy something they do not need but they see it)?
- How many customers get angry at too long queues thus how many of them finally buy elsewhere (and how many baskets we'll have to tidy away?
- Customers are passing through the store looking for items on their lists,
- with the amount of shopped items the patience of a customer grows.

How to solve general discrete simulation problem

in the object environment

- Note that time when a process is running is uninteresting,
- interesting is just start/end of a process. Thus we focus only on (so called) events.
- Usually, we implement classes that represent individual participants (using attributes and methods).
- We also implement an event-calendar (which tells us when what happens),
- yet we implement a simulating kernel able to handle individual events...
- and that's all.

Event calendar

- Contains list of expected events (including the time when they take place).
- It must be able to tell us next event.
- We must be able to modify the calendar (add, remove or re-schedule the events).
- Simulating kernel picks the first event (given by the calendar, i.e., that one that takes place as the first one).
- While serving this event, kernel may modify the calendar.
- Calendar is usually able to measure time (determine the end of simulation).
- Simulation ends after handling such an event which keeps the calendar empty (with no further expected events).

Process-states

- Each (simulated) process is (always) in a particular state.
- Typical states are:
- Running (active) – process is right being served (some its event).
- Scheduled – process is waiting until a given time.
- Waiting (passive) – waits until some (other) process wakes it up.
- Terminated – process ended and incurs no further events.

Situations and their solutions I/II

- When two cars meet at a critical section, one of them has to wait. How to do that?
- Either the waiting car enters state `waiting`, or it calculates the time when the car in the critical section leaves it (and schedules itself for that time).
- In the former case, someone must wake the waiting car up,
- in the latter case it is necessary to ensure that the critical section is free.
- Race condition – why how and when may the cars crash?

Situations and their solutions II

- What happens if the program is represented by more processes and between the moment when one car checks whether critical section is free and (before) enters it, some other process asks about the same?
- This problem is generally called "race-condition".
- What hapens when a car realizes that the critical section is busy (occupied by someone else) but before it enqueues into the waiting-queue, the car leaves critical section?
- This problem is kind of deadlock.
- These problems (deadlock, race-condition and starving) is explained in course of operating systems (spin-locks, semaphores, with active and passive waiting...).
- At the moment, we try to avoid these problem by simulating without paralelism and preemptions.

Discrete simulation

back on trees – i.e., back to cars transporting the sand

Implementation I:

- The event-calendar may be implemented by bidirectional circular linked-list.
- We are scheduling individual processes (type of the event is reflected by the process and its attributes).
- Waiting in queues (e.g., a supermarket): Process that is quitting the queue activates the next one (from the queue).
- So we use passive-waiting. Advantages, disadvantages???:
- Passive waiting (compared to the active one, called busy-waiting) does not waste processor,
- Busy waiting: Process is continuously asking whether it already can start, thus there is lower risk of process being forgotten (before it gets enqueued, the only its ancestor finishes).

Event-description

enumerating data-types

- `enum typename{constants,separated,by,commas};`
- or `{some_constant=its_value,...}`
- Example: `enum state{waiting,runs,loading};`
- `enum state{waiting=0,runs=1,loading=3};`
- We may increment individual variables (++, −−).
- Still nobody checks whether we overflow (or underflow)!

Lists

- Lists (linked) and basic data-structures were lectured in Pascal,
- as in C# they are already implemented:
- `System.Collections.ArrayList` is a universal list.
- Instances of this class are equipped with several methods, e.g.:
  - `Add` – adds an element (into the list).
  - `Remove` – removes an element (one occurence).
  - `Sort` – sorts the list (by default integers),
  - `IndexOf` – looks up and element, when found, it returns non-negative value, when not found, returns -1.

Example

```
using System.Collections;
ArrayList AL = new ArrayList();
AL.Add("First");
AL.Add(222);
AL.Add(100);
AL.Add(1);
AL.Add(null);
AL.Remove("First");
```

Example, cont...

We can output all elements using `foreach`:

```
Console.WriteLine("Number (of el's): 0", AL.Count );
Console.WriteLine("First element: 0", AL[0]);

AL.Sort();

System.Console.Write("All elements: ");
foreach (object obj in AL)
System.Console.Write("0", obj);
System.Console.WriteLine();
```

Typed list and generics

- Often we need some structure for different data-types,
- but we know that each list shall be homogeneous (all elements of the same type).
- Now we may employ **generics** (in C++ called *templates*. They can be recognized by the argument in "acute-brackets":
- `List<int> numbers=new List<int>();`
- Once defined, we work with generic as with a normal variable:
- `numbers.Add(10);`
- Today we show how to use the generics.

Generic List

- It is a successor of `ArrayListu` since C# 2.0 thus it is used in a similar way:
-
```
List<int> integers=new List<int>();
integers.Add(5);
integers.Add(3);
integers.Add(1);
foreach(int i in integers)
     Console.WriteLine(i);
Console.WriteLine("In total 0",integers.Count);
```

Complex numbers

... there is a problem that integers sometimes do not suffice

```
class Compl
{    public double Re,Im;
     public Compl(double Re,double Im)
     { this.Re=Re; this.Im=Im;}
}
List<Compl> s=newList<Compl>();
s.Add(new Compl(1,0));
s.Add(new Compl(0,1));
```

Generic class `List`

- contained in `System.Collections.Generic`,
- contains many methods, e.g.:
- `Add, Contains, Sort, BinarySearch`
- Example:

```
List<string> s1 = new List<string>();
s1.Add("abcd");
s1.Add("efgh");
if(s1.Contains("abcd"))
     Console.WriteLine("It is included!");
```

2nd possibility

how to implement discrete event simulation

- Calendar is a list of events `List<OurEvent>`,
- individual processes are operated by event-handlers,
- thus when implementing a queue for waiting cars, the queues are not necessary,
- waiting can be implemented by scheduling the event at the first possible time (when it can occur),
- we have to pay attention at race-conditions.