Simulation Update Loop

From Simspark

SimSpark implements a simple internal event model that immediately executes every action received from an agent. It does not try to compensate any network latency or compensate for different computing resources available to the connected agents.

A consequence is that SimSpark currently does not guarantee that events are reproducible. This means repeated simulations may have a different outcome, depending on network delays or load variations on the machines hosting the agents and the server.

A benefit of the simple structure however are speed gains that make it interesting for machine learning tasks as in these setups an often large number of different agent and simulation configurations are repeatedly tested.

Further the SimSpark main loop is highly customizable as it is entirely build upon plugins we call simcontrol nodes. Simcontrol nodes are registered to the simulation server. They act in response to control events. The simulation server repeatedly generates these as it executes an abstracted the main loop.

The event types are an 'init' event once when the simulation server starts and a 'done' event on shutdown. The main then loop cycles repeatedly through the 'start cycle', 'sense agent, 'act agent' and 'end cycle' events.

Apart from generating control events the simulation server advances the simulation time passed in the last cycle. Depending on its configuration it either does this in discrete time quanta or in one single step.

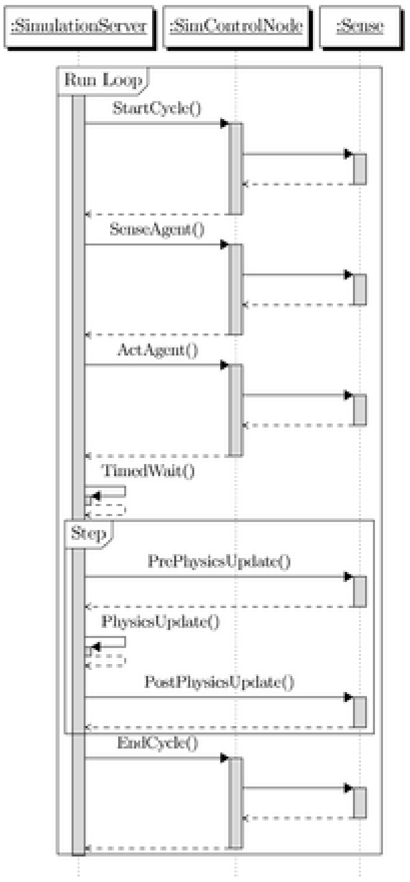
A simcontrol node further can take responsibility for the time measurement, for example to synchronize the simulation time with the real time used to render the scene. Otherwise the simulation is stepped a fixed time step as often as possible.

In this way all management tasks are implemented as plugins to the simulation server. This involves the agent management, monitor management, rendering, mouse and keyboard input and network code.

This setup allows us to configure the simulation at runtime as either a monolithic application that does both simulation and rendering or as a dedicated simulation server that defers rendering to a remote monitor application.

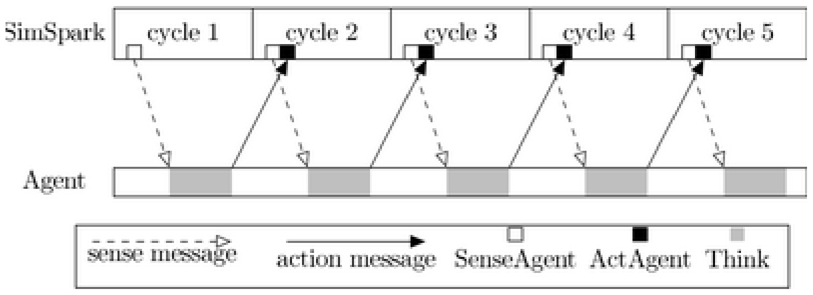
Single-threaded Timer

In the singled-threaded mode, the main loop cycles repeatedly through the 'start cycle', 'sense agent', 'act agent' and 'end cycle' events.



There are two noticeable details:

- Each cycle duration is 20ms, if the simulation is fast than real time, it will wait; otherwise, if the simulation is very slow, it will run many physics updates once without interaction with agents. If the simulation is very slow, it will give up to catch up the real time and print warning. So you may have problem while the computer is not fast enough.
- The 'act agent' event is followed after 'sense agent', the action which the agent send according to nth cycle will be realized in the (n + 1)th cycle, i.e. the action has been delayed one cycle, as shown in the following diagram.

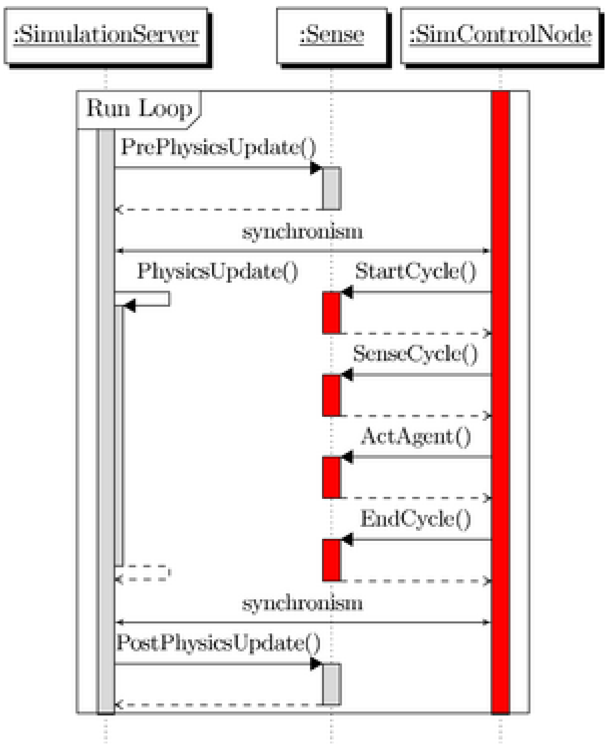


Multi-threaded Timer

In modern time, computers have more than one CPU or dual cores in one CPU. This improve the performance greatly, but only the multi-threaded program can benefit. SimSpark has an experimental multi-threaded running loop, it can be switched on simply by change the `simulationServer.setMultiThreads(false)` to `simulationServer.setMultiThreads(true)` in the `spark.rb` file.

The implementation of multi-threaded loop is based on two conditions. First, every SimControlNode response for different parts of the simulation, they perform one by one in the singled-threaded mode, but they can run in parallel. Second, there is a active scene which stores the whole simulation data in a tree. The physics engine and SimControlNode interact through the active scene. As we know, the physics computation is the most time-consuming, and the physics engine does not need to access the active scene during physics computation.

So the physics computation and SimControlNodes can run in parallel. At last, we get the multithreaded simulation loop as shown in the following UML diagram. Note that the agent’s action are also delayed one cycle in the multi-threaded loop.


:SimulationServer
:Sense
:SimControlNode
Run Loop
PrePhysicsUpdate()
synchronism
PhysicsUpdate()
StartCycle()
SenseCycle()
ActAgent()
EndCycle()
synchronism
PostPhysicsUpdate()