

Simulation using heaps

You need to be alert to (usually minor) changes that may be made to the assignment statement or to the guidelines after the assignment is first put up. Refresh this frame and re-read the assignment carefully before you make your final submission.

Assignment statement

You are required to write a program to simulate the collision of some balls on a 2D planar region bounded by straight walls. The simulation is to be done efficiently using the application of the priority queue data structure, which is the objective of this experiment.

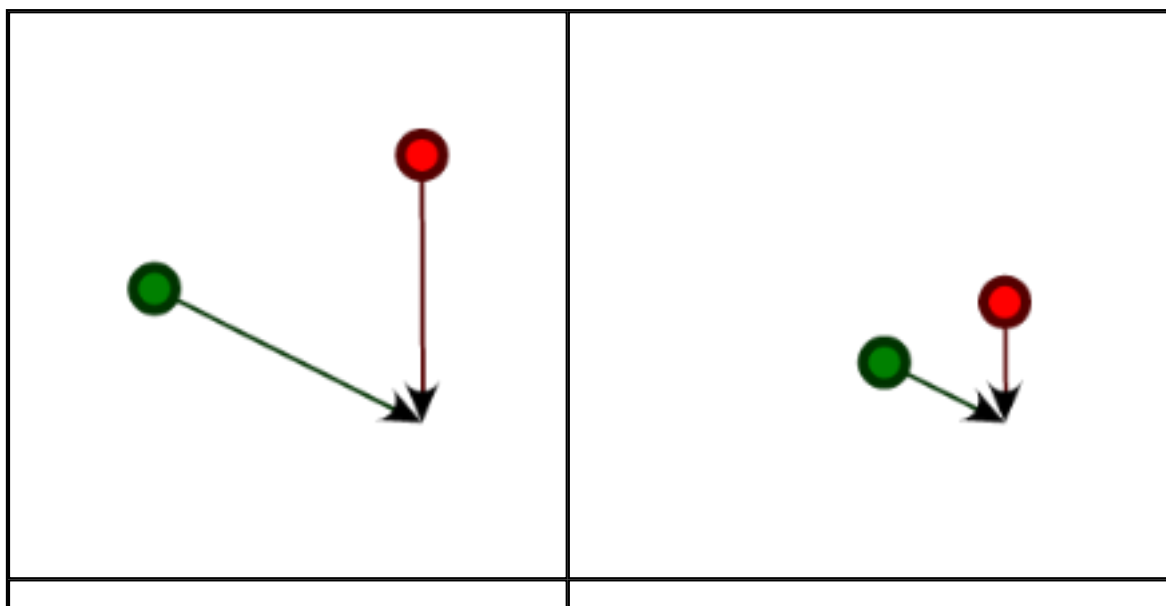
The very basics of carrying out simulation are first explained. Next, optimised simulation using priority queues is explained. Details of carrying out the experiment and writing the report follow thereafter.



Time and event driven simulation schemes

You are provided a set of objects and a set of rules governing some dynamical properties of the objects. Your program keeps on updating the properties/states by executing the rules after a specific time stamp. For example, there are a set of spherical particles moving on a 2-d plane with some predefined constant speed. You solve standard equations of motion to compute the position and velocity of each particle after every 10 milli-second (say). The idea discussed this far is the basic concept behind "Time-driven simulation" of particle dynamics for which the steps are as follows.

- Discretize time in quanta of size δt .
- Update the position of each particle after every δt units of time and check for overlaps.
- If overlap, roll back the clock to the time of the collision, update the velocities of the colliding particles, and continue the simulation.

Consider the elaboration as provided in the figure. However, the approach has the following issues.



At time t	At time $t + \delta t$
	
Collision detected at time $t + 2\delta t$	Roll back simulation and execute collision

- For N particles, N^2 overlap checks per time quantum.
- May miss collisions if δt is too large and colliding particles fail to overlap when we are looking.
- Simulation is too slow if δt is very small.

An alternative strategy is "Event-driven simulation" which essentially asks for doing something only when something other than usual (an event) is supposed to occur. In such a strategy, we change state instantaneously only when an event (collision) happens. Such a simulation strategy will focus only on time-points when collisions occur resulting in a sudden change of state of the system. For each particle, you have to initialize the following information fields: radius, position co-ordinates, velocity along each of x and y axes. For each particle, compute the time instants when it may (possibly) collide with every other particle and every wall. In that way, you get a set of pending events $\langle \text{particle}_1/\text{wall}, \text{particle}_2/\text{wall}, \text{time}, \text{valid}/\text{invalid} \rangle$. Initially, all of these are valid events. Insert the pending events into a priority queue. Your main loop iterates as follows.

1. Extract and delete the event which is immediately pending (with time-stamp t say) from the queue.
2. If the event is invalid, ignore it.
3. Advance all particles to time t on a straight line trajectory.
4. If the extracted event is valid,
 - Update the velocities of colliding particle(s).
 - Every future event involving the particle(s) should be recognised as invalid.
 - Predict future particle-wall and particle-particle collisions involving the colliding particle(s) and insert these new events onto the queue.

Optimised simulation of events

It is a good idea to have the entire state of the system maintained in a single data structure (let us call STATE). For every particle, store its 'color', radius, last computed position (p_x, p_y), velocity (v_x, v_y) and time of update. **While forming the dynamical equations for collision detection, keep in mind the radius of the particles and do not treat them as points without dimensions.**

We need an efficient routine which should mark some future events as invalid after

processing a collision event. We want to avoid a linear search (which shall be slow in case of large no of particles). Hence, we shall have a linked list for each particle in the structure STATE (which stores the information for every particle as discussed earlier). Thus for any particle (i say), every node in the linked list contains a pointer which shall point to a collision event stored in the heap in which particle i is involved. Thus, for marking events which involve i , you simply traverse the linked list for i inside STATE, get directly to the requisite positions in the heap and mark those as invalid. You should deallocate the list once all the events are marked invalid. Again, whenever you compute future collisions involving i , you need to create the nodes in the list for i .

The list implementation adds one overhead which you need to be careful about. Whenever the elements in the heap change their positions, the pointers in the list need to be carefully updated.

Input and output specification

Initialize your simulation with a pre-defined area (length and breadth specification), a set of five particles with initial velocities (as per your convenience) and radius. The user may provide a time horizon (100 sec say). Your output should be a plot exhibiting the trajectories of the five particles (in different colors) up to 100 sec. Also, in a separate text file, you should log the velocity and position vectors of each particle at the moment of each collision.

Generating plot data

Store the trajectories for each particle in separate log files. Suppose particle i suffers two collisions at instants t_1 followed by t_2 . You should compute intermediate position coordinates for the particle at time steps $t_1 + k \times \frac{(t_2 - t_1)}{10}$ for $k = 0, 1, \dots, 10$ and store this in the file for particle i . With each location information, you definitely need to store the time when the particle was in that location. Use this data which you have generated for each particle to obtain the plots.

Report writing

In the report (in latex), first list describe the implementation of the simulation mechanism in pseudocode. Next, identify the key operations used in the pseudocode and indicate the data structure you have used to support those operations. Having done that, indicate the time complexities of those operations wrt to the chosen data structure. Thereby, derive the time complexity of each simulation run.

Marking guidelines

Assignment marking is to be done only **after** the deadline expires, as submissions gets blocked after the assignment is marked.

Overall implementation of the simulation loop	10
Optimal sequencing of events using a priority queue	10
Specification of heap datatype and implementation of heap operations	4×5=20
Efficient implementation of scheme for marking invalid collisions	5
Documentation of test run	5

Report with details of pseudocode, choice of datatypes and their definitions and complexity analysis	10
<i>Total Marks</i>	60
<i>Bonus Marks</i> animated display of results	5

Assignment submission

Use electronic submission via the [WBCM link](#)

You should keep submitting your incomplete assignment from time to time after making some progress, as you can submit any number of times before the deadline expires.

Warning

Cases of copying will be dealt with seriously and severely, with recommendation to the Dean to de-register the student from the course.