Stochastic Modelling of Chemical Reactions

An overview by Conor Johnson

OverReaction uses the [Gillespie Algorithm](https://en.wikipedia.org/wiki/Gillespie_algorithm) to simulate chemical reactions in the system.

The logic for updating chemical counts in OverReaction is mainly handled in the Simulation.cs script. The important local variables for this simulation are:

* The two-dimensional integer array *updateMatrix* keeps track of how the ith reaction updates each species if the reaction occurs.
  + The ith row of the array corresponds to the ith reaction
  + The jth row of the array corresponds to the jth chemical and how the ith reaction updates that chemical in *liveSpecies*
  + If the [i,j] element of the array is positive, the reaction adds that many to the system, if negative it removes that much. Tentatively, positive elements mean the jth chemical is a product, if it is negative it means it is a reactant.
* The one-dimensional integer array *liveSpecies* keeps track of the number of each chemical in the system.
* The list of **ChemicalReaction** objects *reactions* keeps track of what reactions are active in the game.
* The time since the last reaction occurred *simTime*

In the Start() method, initSimulation() is called to initialize the local variables needed to track and update OverReaction’s simulation. The list *reactions* is initialized to an empty list; *liveSpecies* is initialized to the initial counts for each chemical; *updateMatrix* is initialized to all zeros.

Each frame, Update() is called and calls StepSimulate() and UpdateSimulate() which calculate the change in *liveSpecies* and then updates the information tied to the reactions in the simulation. To see if a reaction should occur, NextReactionOccurs() is true (false if otherwise).

So how do we calculate if a reaction should happen at a given time and which reaction to occur?

It is important to note a few important variables while using the Gillespie Algorithm.

* Rate for a given reaction =
  + Thus each reaction’s rate is directly proportional to how many reactants are in the system.
  + Note: if a reaction uses more than 1 of the same type of chemical, each subsequent use of the same reaction subtract 1 from the previous number of that reactant.
* Total rate of all reactions =
* Probability that reaction i occurs = (Rate of Reaction i) / Total Rate
* The probability that a reaction occurs given t units of time has passed =

Let us walk through exactly how an update works for simulation:

* Update() is called by Unity
* Some UI stuff occurs
* If the player is the host or playing offline it checks if NextReactionOccurs()
* In next ReactionOccurs(), calculate using *simTime* in place of t, then a random number between 0 and 1 is calculated. If that random number is less than this number, then a reaction will occur, otherwise a reaction does not occur and Update() continues. We use a TIME\_SCALAR to slow down how fast reactions occur.
* *liveSpecies* has added and removed the necessary counts of each chemical to the system. Thus, total rate has changed and must be updated as well as if any new reactions were added they should be updated.
* If a reaction occurs, then StepSimulate() is called.
* In StepSimulate() a weighted random number, based on the probability that each reaction occurs, is calculated. This is done in WeightedRandom() which generates a number between 0 and the number of reactions - 1.
  + WeightedRandom() generates a random number between 0 and 1
  + Then loops through each reaction. During the ith loop it adds the probability that a given reaction occurs to a total number (*choice*). If the random number is less than the running total, that means the ith reaction occurs as it was randomly selected to occur based on the probabilities calculated.
* Knowing that the *next* reaction is the one occurring means we can update the counts of each chemical based on how the given reaction changes those. Each element in *liveSpecies* is updated by adding the *next* row and the ith column from *updateMatrix*.
* *simTime* is then set to the current time so as to be used in the future.
* This is repeated as long as the game is running.

Whenever the rate of a given reaction is needed, the RateExpression() method in ChemicalReaction.cs is called.