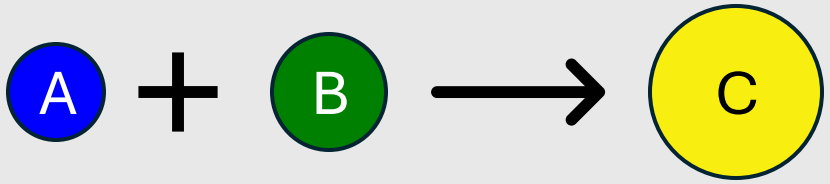
Simulating Continuous Systems

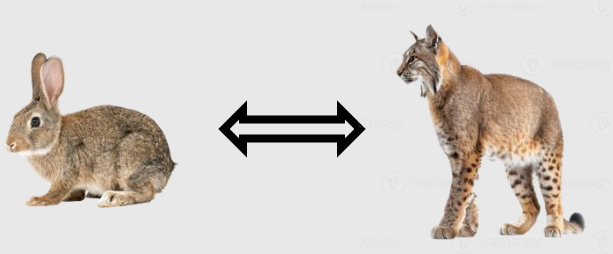
CMSC 326 Simulations

Simulating Continuous Systems

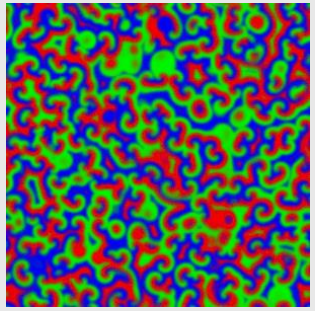
Simple chemical reaction



Rabbit–Lynx “reaction”

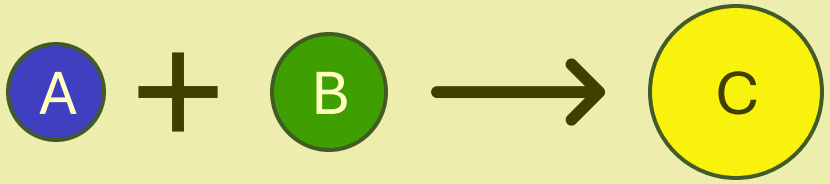


B-Z reaction

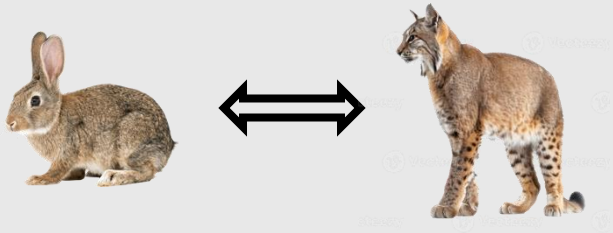


Simulating Continuous Systems

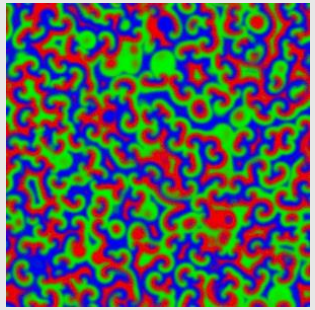
Simple chemical reaction



Rabbit–Lynx “reaction”



B-Z reaction

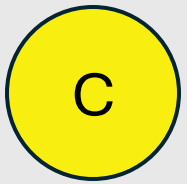


Simulation: Simple Chemical Reaction

Three molecules:

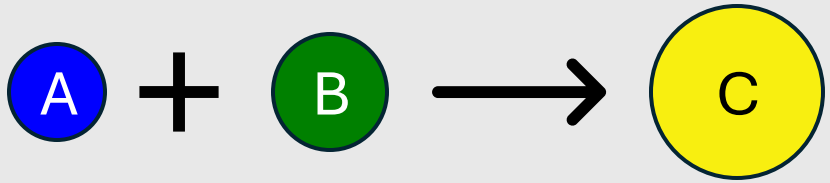
Substance 

Substance 

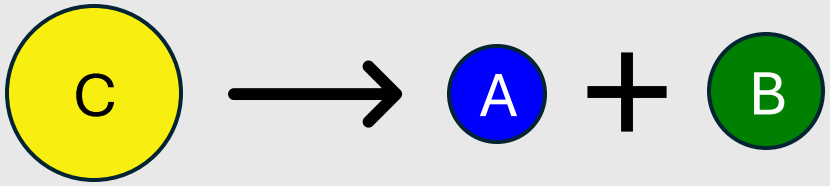
Substance 

Simulation: Simple Chemical Reaction

Reaction 1:
One molecule of **A** *reacts*
with one molecule of **B**
to produce one molecule of **C**

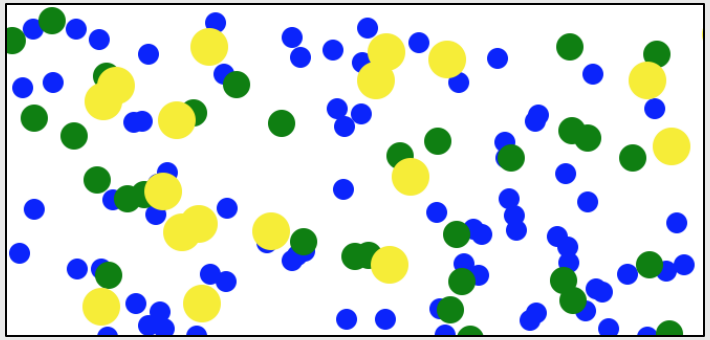


Reaction 2:
A molecule of **C** can *breakdown*
to produce one of each **A** and **B**

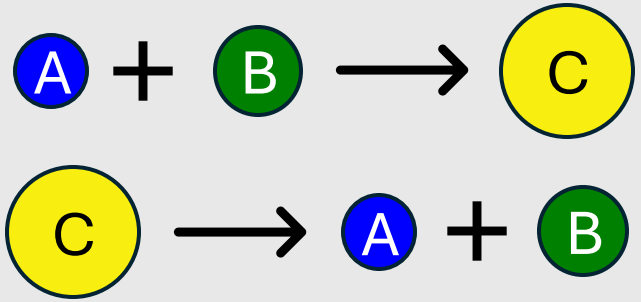


Simulate molecules in two dimensions

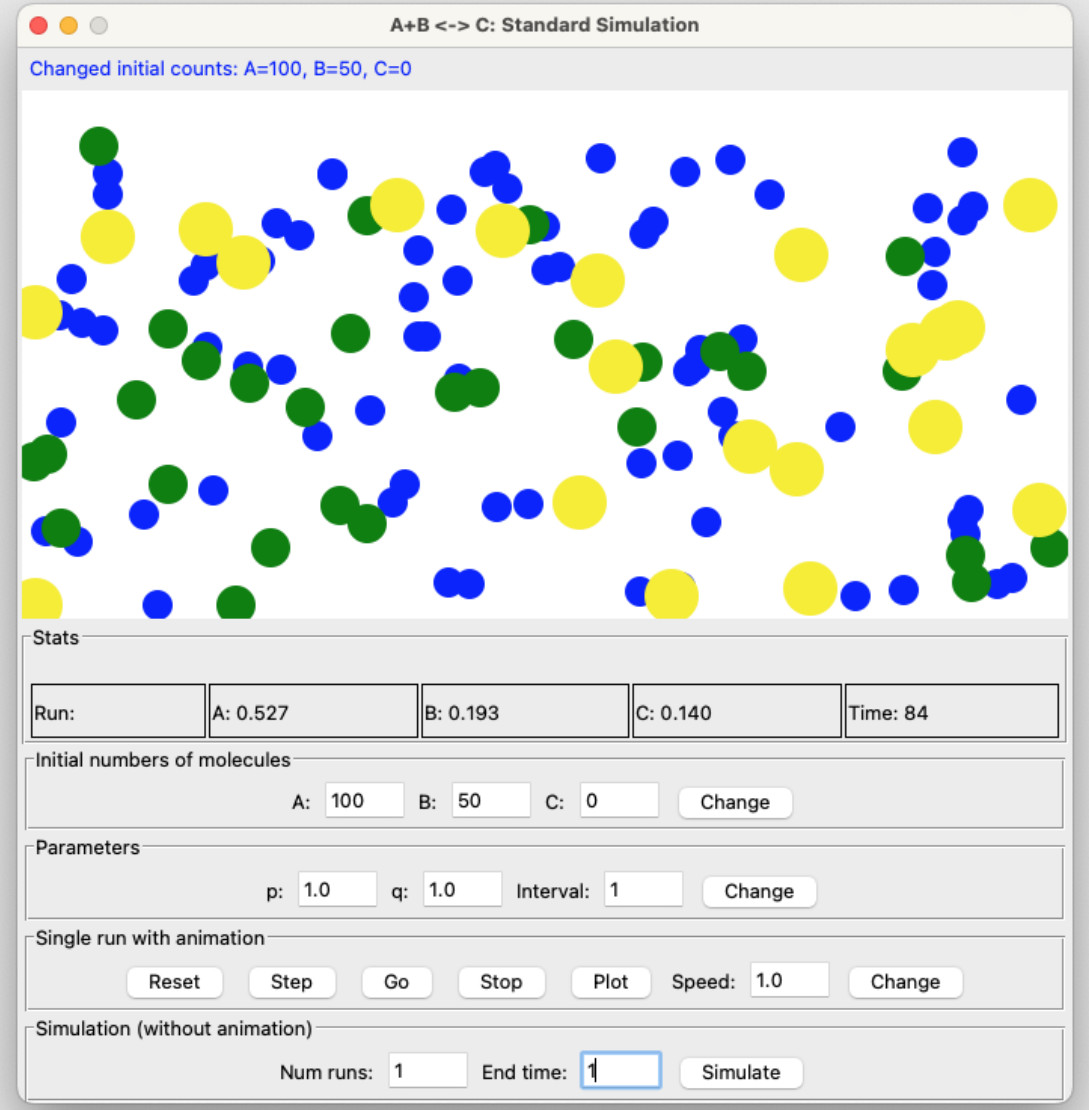
Molecules move about randomly in a solution or a gaseous state



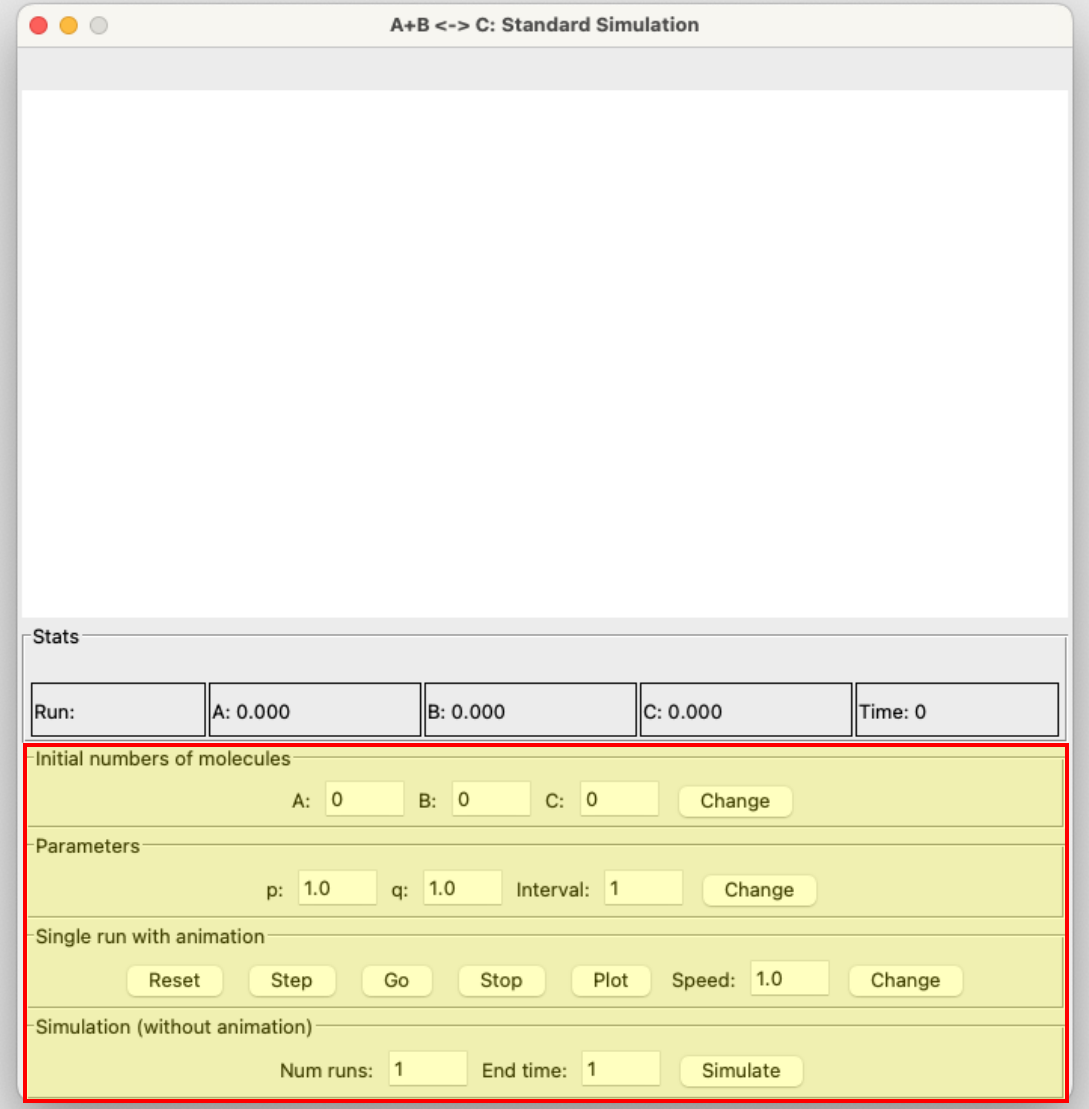
At any given instant, exactly one of the two reactions takes place



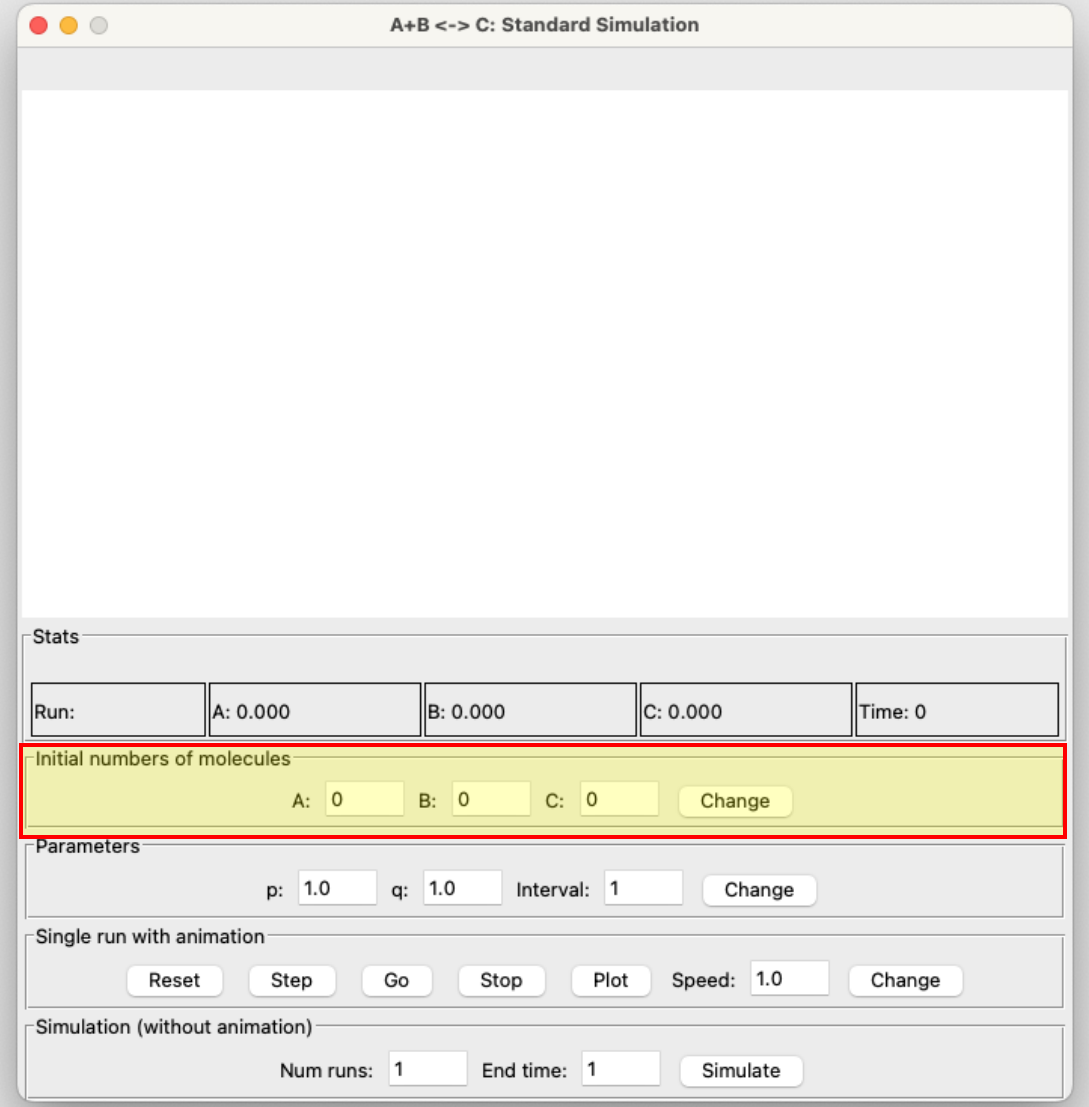
Demo: Simple Chemical Reaction



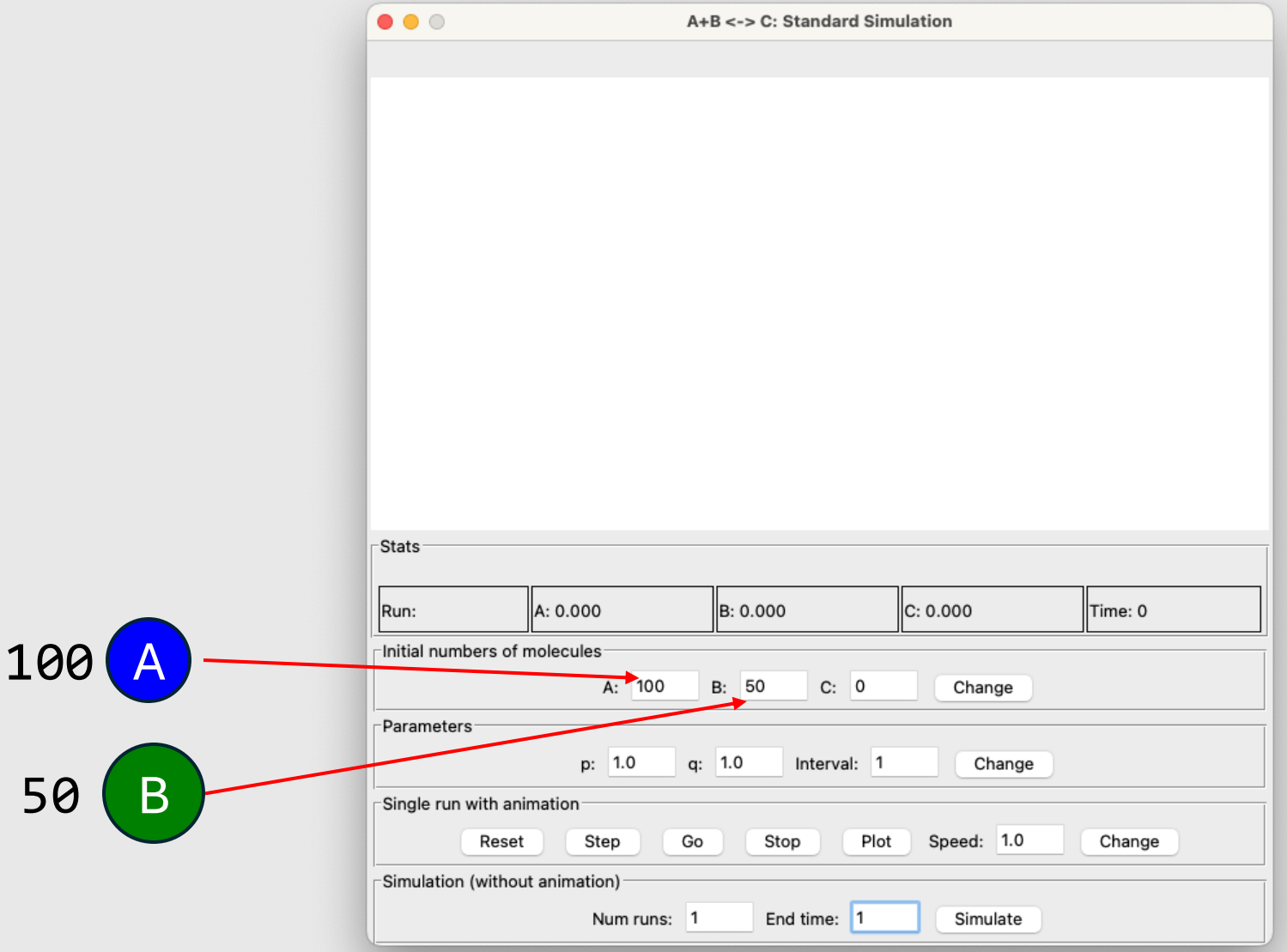
Setup the Simulation



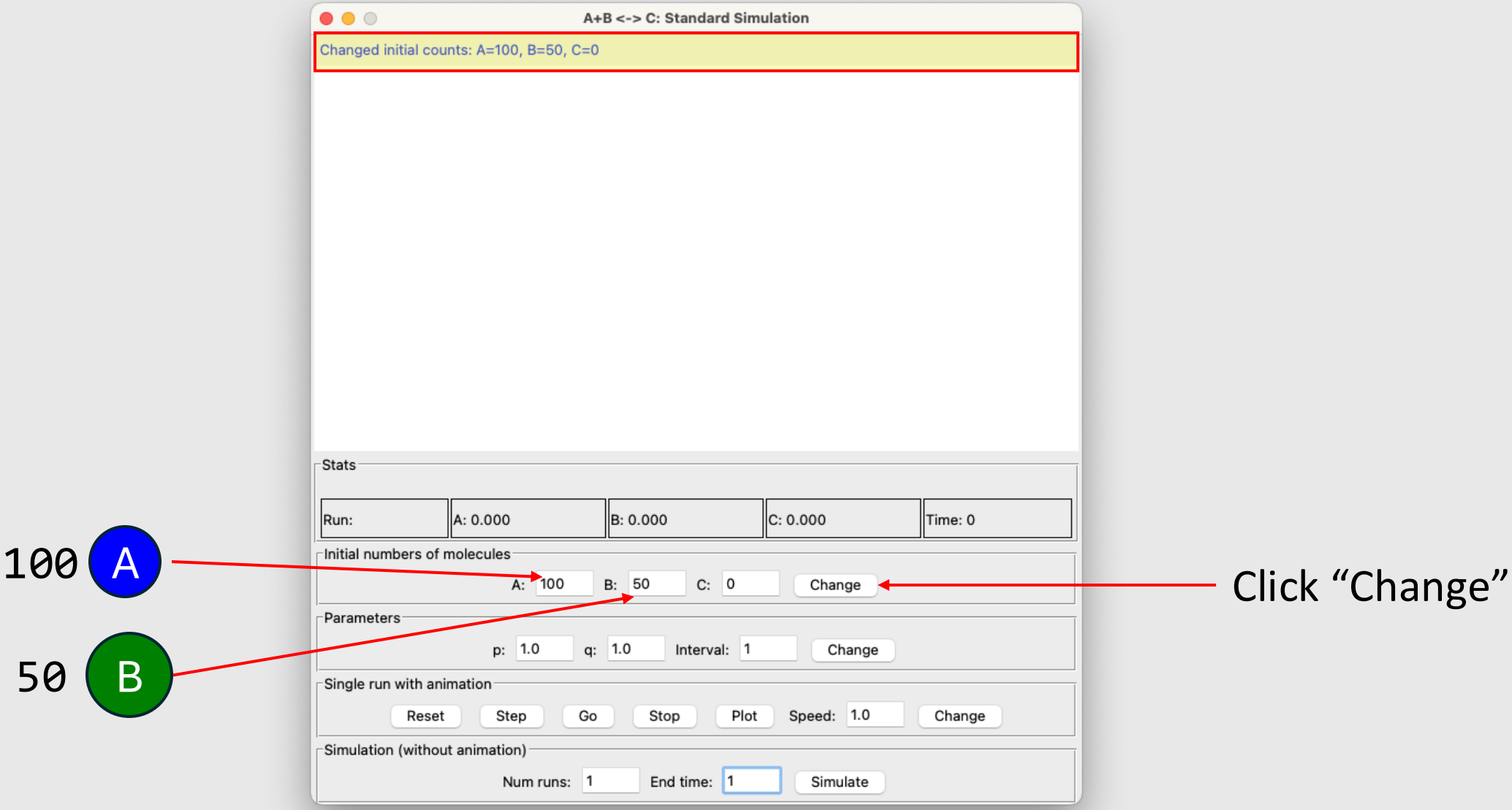
Setup the Simulation



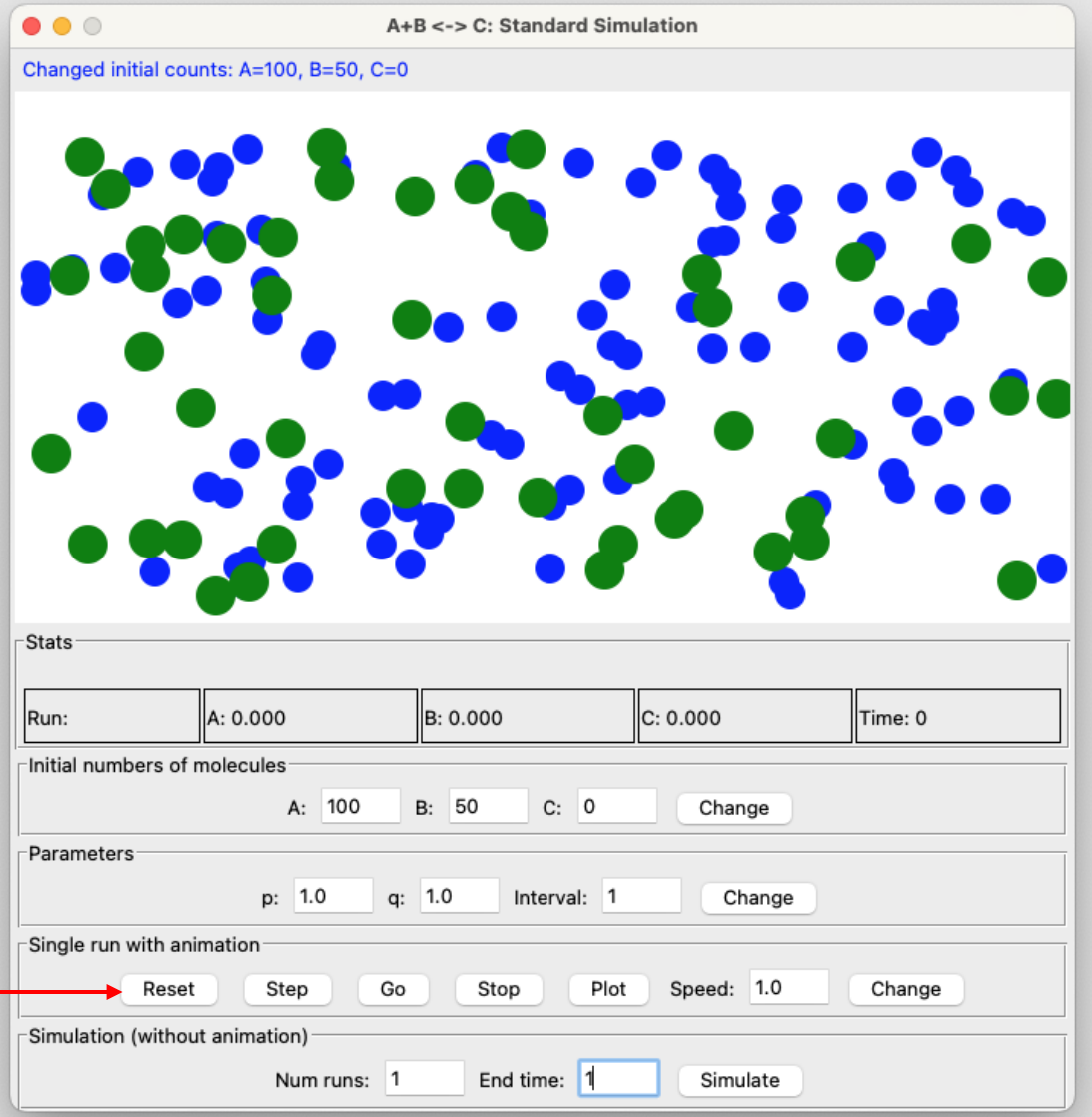
Setup the Simulation



Setup the Simulation

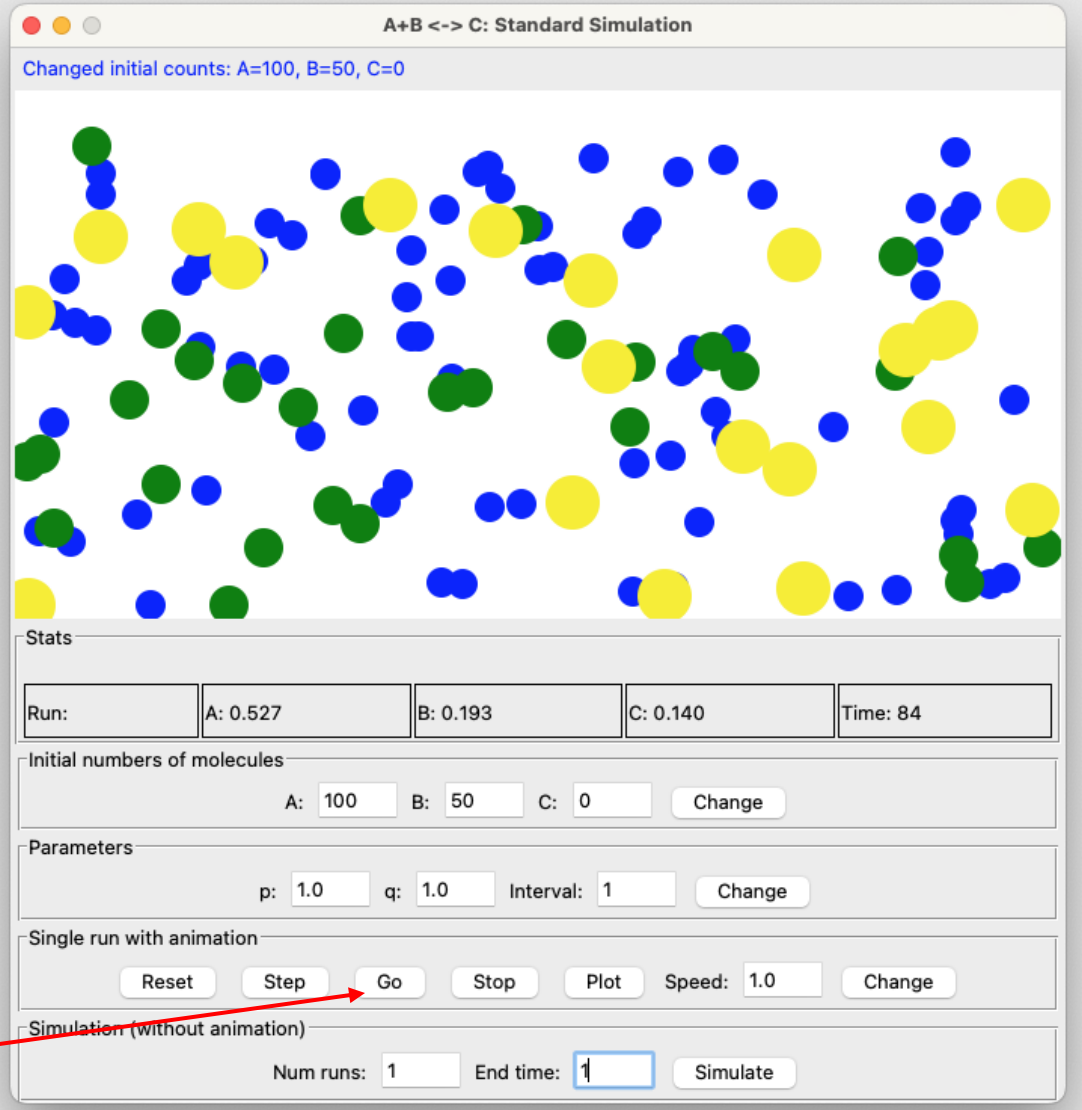


Setup the Simulation



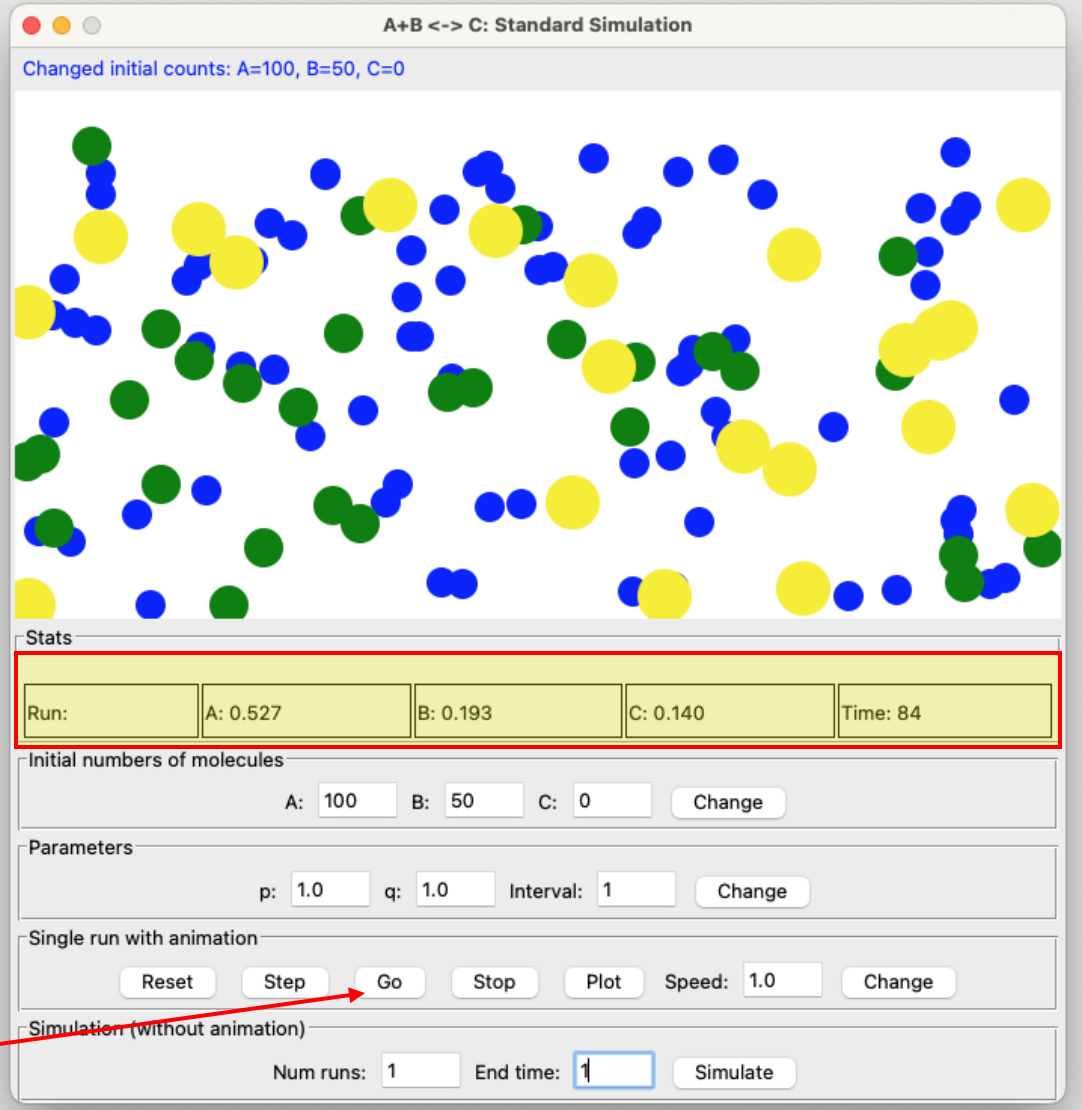
Click “Reset”

Run the Simulation



Click “Go”

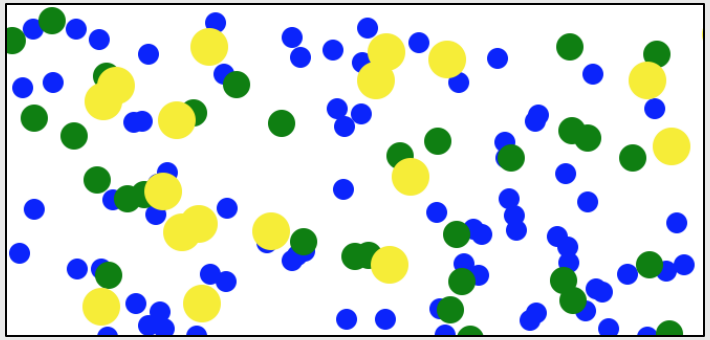
Run the Simulation



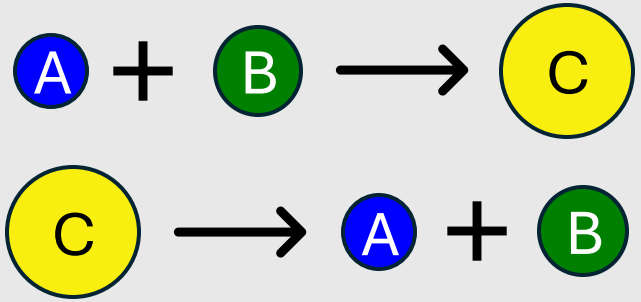
Click “Go”

Simulate molecules in two dimensions

Molecules move about randomly in a solution or a gaseous state



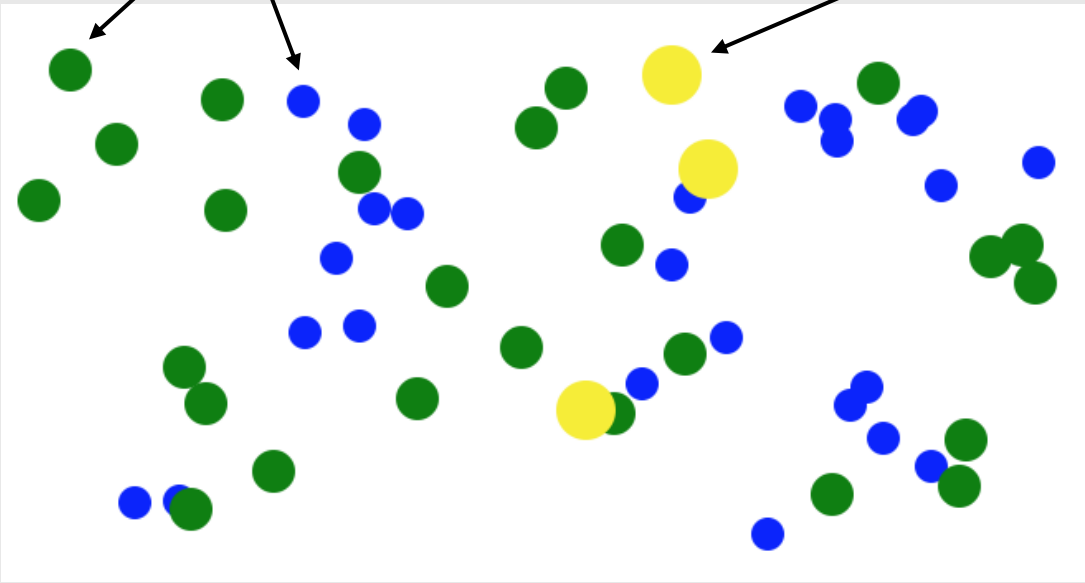
At any given instant, exactly one of the two reactions takes place



How do we decide which reaction?

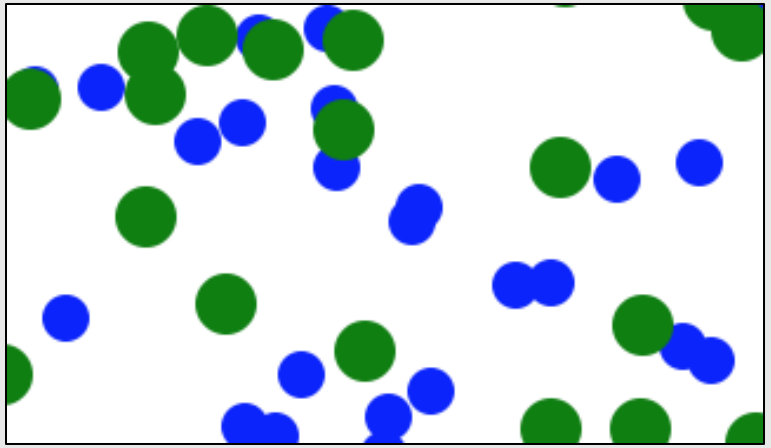
Many more **A** (green) and **B** (blue) molecules
A + B → C more likely

Fewer **C** (yellow) molecules
C → A + B less likely

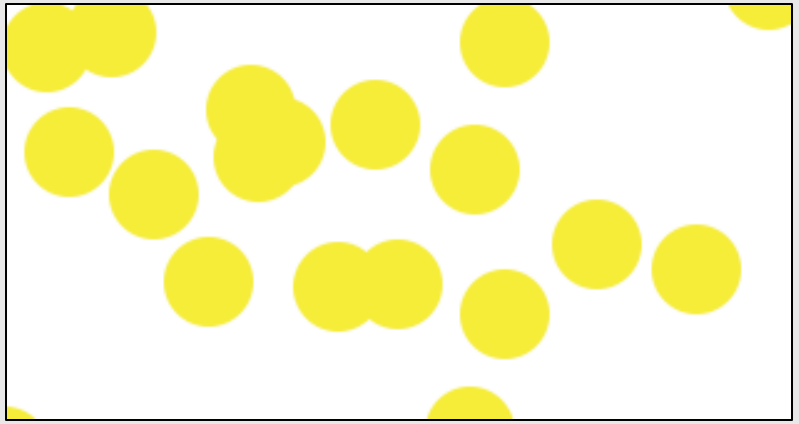


How do we decide which reaction?

The *likelihood* of the first reaction is **proportional** to the number of **A** molecules and the number of **B** molecules



The more of **C**, the *more likely* we'll have the second reaction



Reaction Model

N_A = number of A molecules

N_B = number of B molecules

N_C = number of C molecules

Reaction Model

E_A = concentration of A

E_B = concentration of B

E_C = concentration of C

Reaction Model

To get the concentration (mass per unit volume), we'll approximate the actual physical volume as:

$$V_{total} = N_A + N_B + 2N_C$$

Reaction Model

$$E_A = \frac{N_A}{V_{total}}$$

$$E_B = \frac{N_B}{V_{total}}$$

$$E_C = \frac{N_C}{V_{total}}$$

Reaction Model

Concentration of A

$$E_A = \frac{N_A}{V_{total}}$$

Concentration of B

$$E_B = \frac{N_B}{V_{total}}$$

Concentration of C

$$E_C = \frac{N_C}{V_{total}}$$

Reaction Model

R_1 = Probability that first reaction takes place

R_2 = Probability that second reaction takes place

Reaction Model

The chances of the first reaction taking place are *proportional* to both the number of **A** molecules (N_A)
and the number of **B** molecules (N_B)

Thus,

$$R_1 \propto N_A N_B$$

$$R_2 \propto N_C$$

Reaction Model

Because the numbers of molecules N_A, N_B, N_C are proportional to concentrations E_A, E_B, E_C, we can write:

$$R_1 \propto E_A E_B$$

$$R_2 \propto E_C$$

Reaction Model

The constant of proportionality may be different

To model this, we'll use constants $\mathbf{p}$ and $\mathbf{q}$

$$R_1 = pE_A E_B$$

$$R_2 = qE_C$$

Reaction Model

What happens at each step of the simulation is this:

1. Randomly choose between doing the first or second reaction, based on the relative proportion of $\mathbf{R_1}$ and $\mathbf{R_2}$

2. If it's the first, then remove one each of an **A** and a **B** molecule and create a new **C** molecule

3. If it's the second, we remove one **C** molecule and create a new **A** and a new **B** molecule

Reaction Model

```
for step in number_of_steps:

    # Compute concentrations
    V = NA + NB + (2 * NC)
    EA = NA / V
    EB = NB / V
    EC = NC / V

    # Calculate reaction probabilities
    R1 = p * EA * EB
    R2 = q * EC

    # Randomly choose one reaction
    if random.uniform(0, 1) < R1 / (R1 + R2):
        # A + B → C
        # Remove one A and one B, make one C
        ...
    else:
        # C → A + B
        # Remove one C, make one A and one B
        ...
```

Reaction Model

```
for step in number_of_steps:

    # Compute concentrations
    V = NA + NB + (2 * NC)
    EA = NA / V
    EB = NB / V
    EC = NC / V

    # Calculate reaction probabilities
    R1 = p * EA * EB
    R2 = q * EC

    # Randomly choose one reaction
    if random.uniform(0, 1) < R1 / (R1 + R2):
        # A + B → C
        # Remove one A and one B, make one C
        ...
    else:
        # C → A + B
        # Remove one C, make one A and one B
        ...
```

Reaction Model

```
for step in number_of_steps:

    # Compute concentrations
    V = NA + NB + (2 * NC)
    EA = NA / V
    EB = NB / V
    EC = NC / V

    # Calculate reaction probabilities
    R1 = p * EA * EB
    R2 = q * EC

    # Randomly choose one reaction
    if random.uniform(0, 1) < R1 / (R1 + R2):
        # A + B → C
        # Remove one A and one B, make one C
        ...
    else:
        # C → A + B
        # Remove one C, make one A and one B
        ...
```

Reaction Model

```
for step in number_of_steps:

    # Compute concentrations
    V = NA + NB + (2 * NC)
    EA = NA / V
    EB = NB / V
    EC = NC / V

    # Calculate reaction probabilities
    R1 = p * EA * EB
    R2 = q * EC

    # Randomly choose one reaction
    if random.uniform(0, 1) < R1 / (R1 + R2):
        # A + B → C
        # Remove one A and one B, make one C
        ...
    else:
        # C → A + B
        # Remove one C, make one A and one B
        ...
```

$$N_A = 60$$
$$N_B = 30$$
$$N_C = 10$$

$$V_{total} = N_A + N_B + 2N_C$$
$$V_{total} = 60 + 30 + (2 \times 10) = 110$$

$$E_A = \frac{N_A}{V_{total}} = \frac{60}{110} = 0.5454 \ldots$$

$$E_B = \frac{N_B}{V_{total}} = \frac{30}{110} = 0.2727 \ldots$$

$$E_C = \frac{N_C}{V_{total}} = \frac{10}{110} = 0.0909 \ldots$$

Reaction Model

```
for step in number_of_steps:

    # Compute concentrations
    V = NA + NB + (2 * NC)
    EA = NA / V
    EB = NB / V
    EC = NC / V

    # Calculate reaction probabilities
    R1 = p * EA * EB
    R2 = q * EC

    # Randomly choose one reaction
    if random.uniform(0, 1) < R1 / (R1 + R2):
        # A + B → C
        # Remove one A and one B, make one C
        ...
    else:
        # C → A + B
        # Remove one C, make one A and one B
        ...
```

Reaction Model

```
for step in number_of_steps:

    # Compute concentrations
    V = NA + NB + (2 * NC)
    EA = NA / V
    EB = NB / V
    EC = NC / V

    # Calculate reaction probabilities
    R1 = p * EA * EB
    R2 = q * EC

    # Randomly choose one reaction
    if random.uniform(0, 1) < R1 / (R1 + R2):
        # A + B → C
        # Remove one A and one B, make one C
        ...
    else:
        # C → A + B
        # Remove one C, make one A and one B
        ...
```

$$E_A = 0.54$$
$$E_B = 0.27$$
$$E_C = 0.09$$

$$p = 1.0$$
$$q = 1.0$$

$$R_1 = pE_AE_B = 1.0 \times 0.54 \times 0.27 = 0.1458$$

$$R_2 = qE_C = 1.0 \times 0.09 = 0.09$$

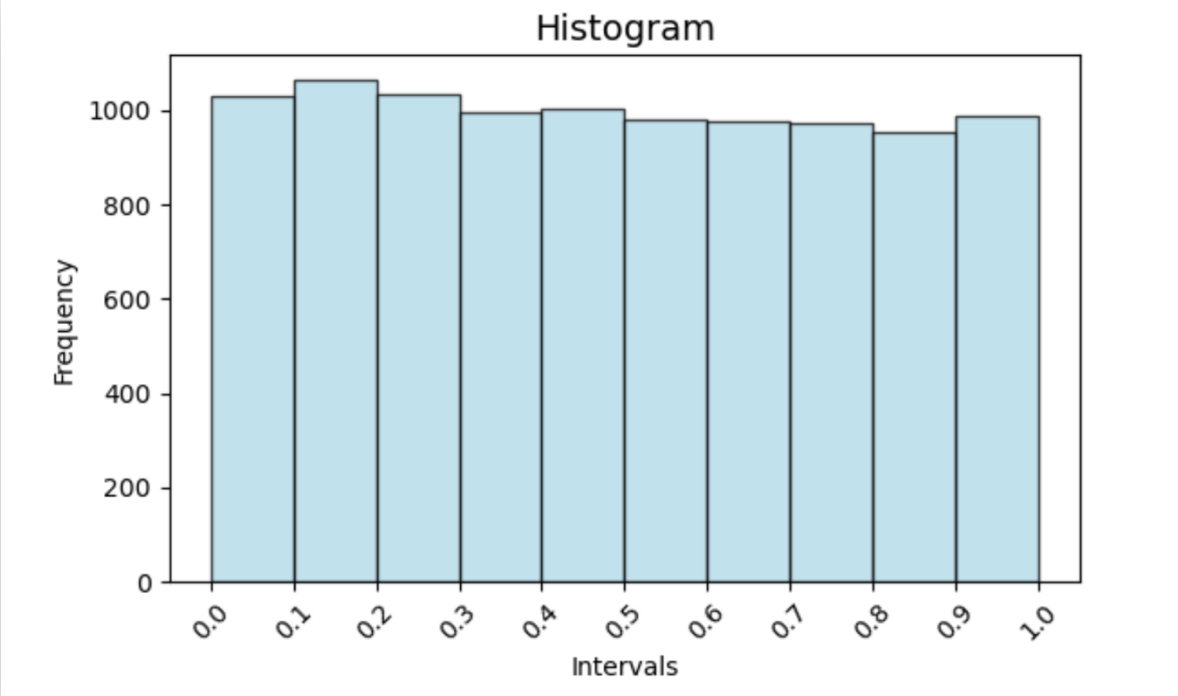
Reaction Model

```
for step in number_of_steps:

    # Compute concentrations
    V = NA + NB + (2 * NC)
    EA = NA / V
    EB = NB / V
    EC = NC / V

    # Calculate reaction probabilities
    R1 = p * EA * EB
    R2 = q * EC

    # Randomly choose one reaction
    if random.uniform(0, 1) < R1 / (R1 + R2):
        # A + B → C
        # Remove one A and one B, make one C
        ...
    else:
        # C → A + B
        # Remove one C, make one A and one B
        ...
```

Reaction Model

```python
for step in number_of_steps:

    # Compute concentrations
    V = NA + NB + (2 * NC)
    EA = NA / V
    EB = NB / V
    EC = NC / V

    # Calculate reaction probabilities
    R1 = p * EA * EB
    R2 = q * EC

    # Randomly choose one reaction
    if random.uniform(0, 1) < R1 / (R1 + R2):
        # A + B → C
        # Remove one A and one B, make one C
        ...
    else:
        # C → A + B
        # Remove one C, make one A and one B
        ...
```

```python
random.uniform(0, 1)
```



Reaction Model

```
for step in number_of_steps:

    # Compute concentrations
    V = NA + NB + (2 * NC)
    EA = NA / V
    EB = NB / V
    EC = NC / V

    # Calculate reaction probabilities
    R1 = p * EA * EB
    R2 = q * EC

    # Randomly choose one reaction
    if random.uniform(0, 1) < R1 / (R1 + R2):
        # A + B → C
        # Remove one A and one B, make one C
        ...
    else:
        # C → A + B
        # Remove one C, make one A and one B
        ...
```

$$R_1 = 0.1458$$
$$R_2 = 0.09$$

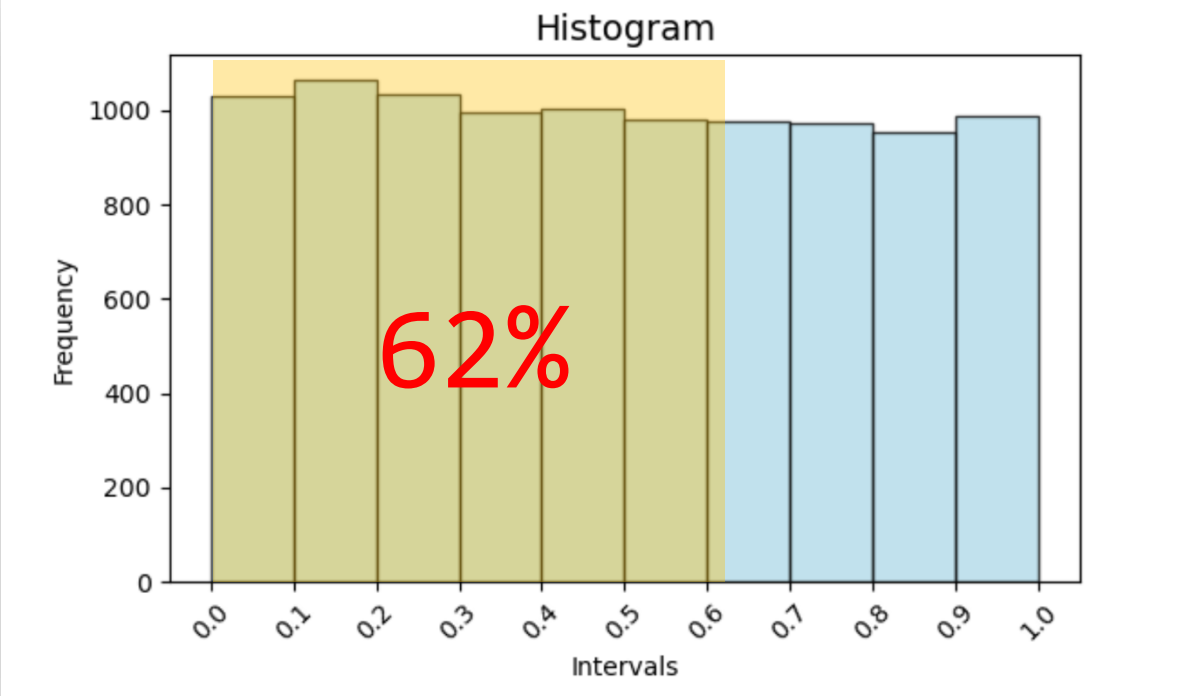
Compute the ratio of $\mathbf{R_1}$ reaction probability relative to the total reaction probability $\mathbf{R_1} + \mathbf{R_2}$

$$\frac{R_1}{R_1 + R_2} = \frac{0.1458}{0.1458 + 0.09} = \frac{0.1458}{0.2358} \cong 0.6183$$

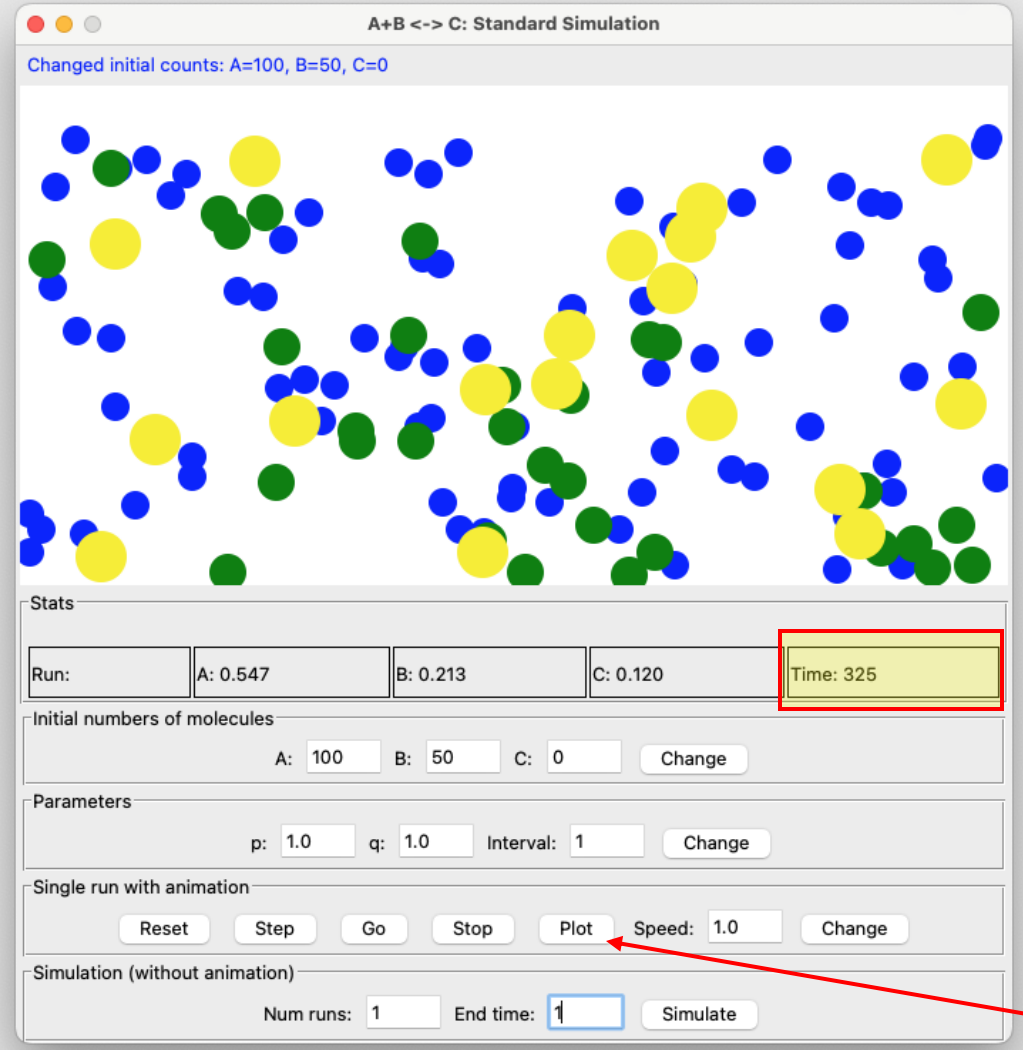
Reaction Model

```python
for step in number_of_steps:

    # Compute concentrations
    V = NA + NB + (2 * NC)
    EA = NA / V
    EB = NB / V
    EC = NC / V

    # Calculate reaction probabilities
    R1 = p * EA * EB
    R2 = q * EC

    # Randomly choose one reaction
    if random.uniform(0, 1) < R1 / (R1 + R2):
        # A + B → C
        # Remove one A and one B, make one C
        ...
    else:
        # C → A + B
        # Remove one C, make one A and one B
        ...
```

```python
if random.uniform(0, 1) < 0.6183:
```

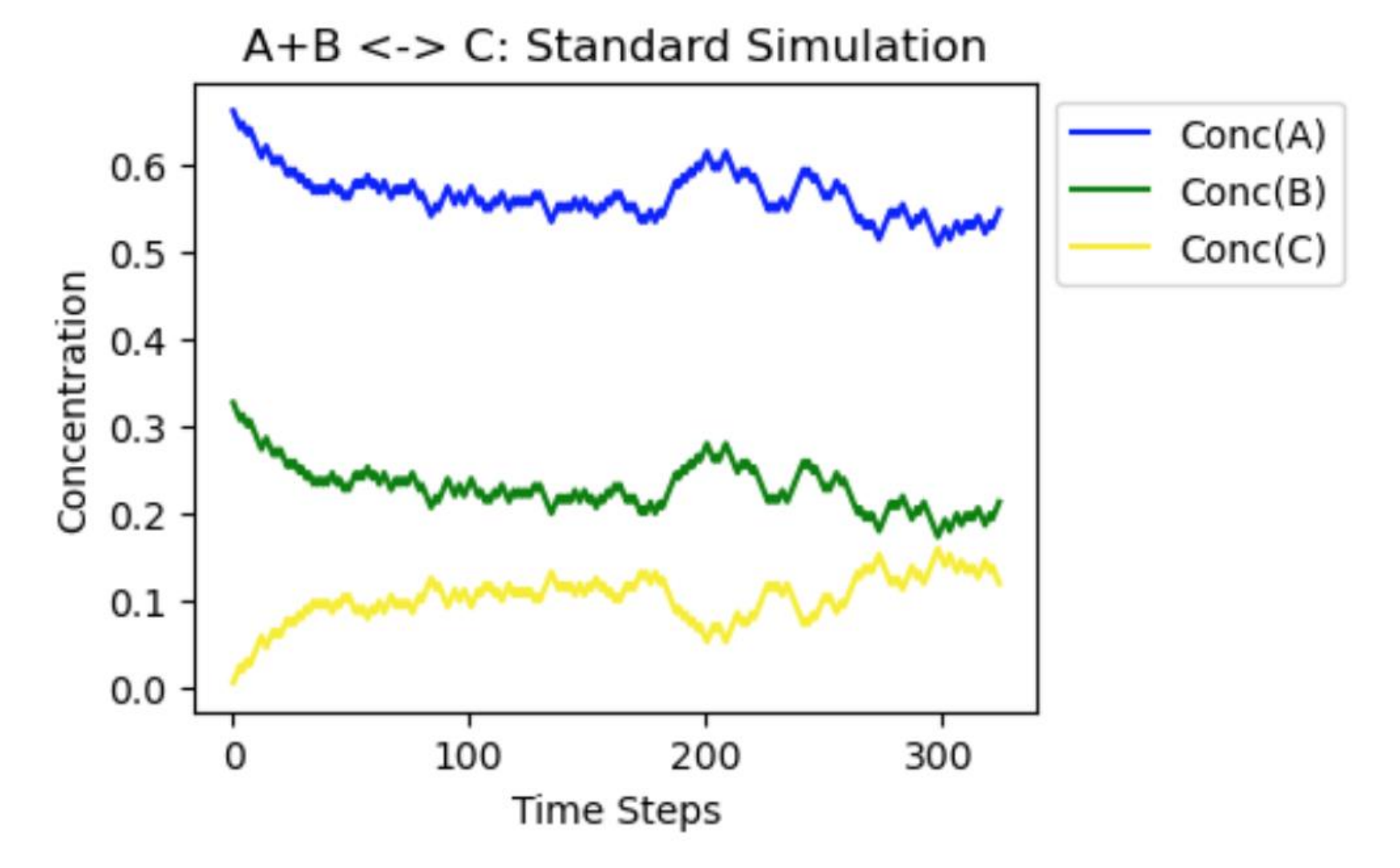


Long-Term Behavior



Click “Plot”

Long-Term Behavior



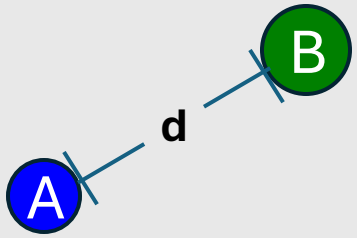
A more detailed model

How closely does our model correspond to reality?

Molecules react when they bump into each other, but we did **not** consider *spatial* closeness at all

Model with Spatial Detail

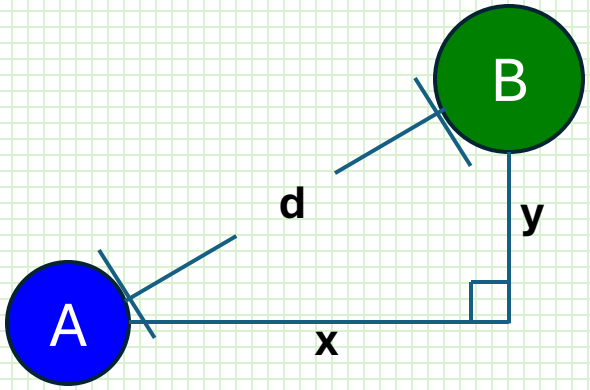
Molecules A and B will
react only if close enough



We'll use a *distance* parameter that we can vary to decide what's "close enough"

Two Ways to Calculate Distance

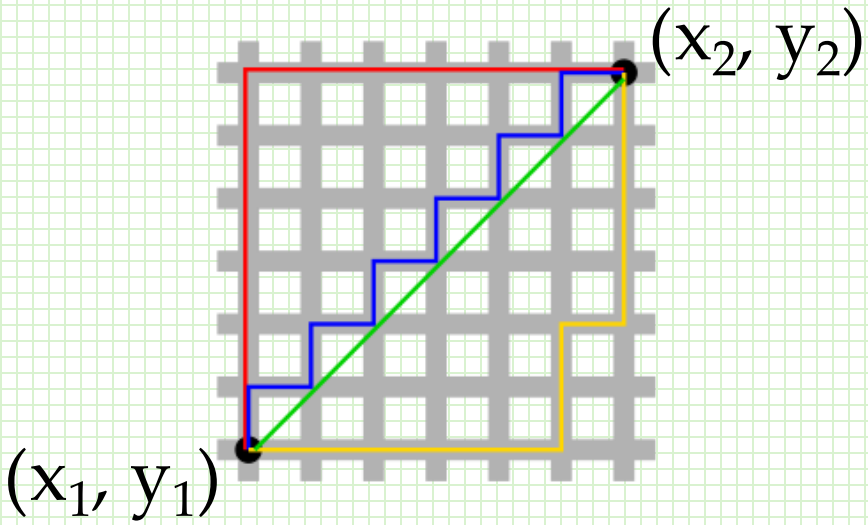
Euclidian Distance



$$distance = \sqrt{x^2 + y^2}$$

Two Ways to Calculate Distance

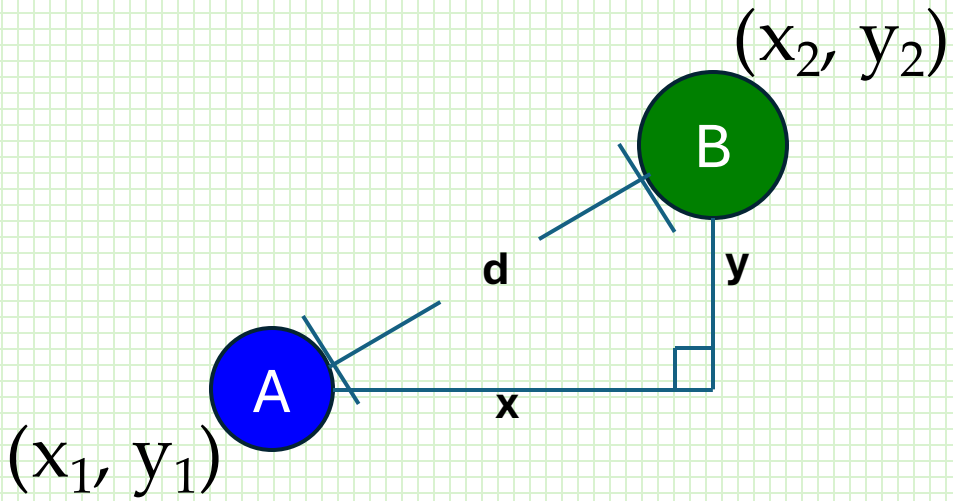
Manhattan Distance



$$distance = |x_2 - x_1| + |y_2 - y_1|$$

Squared Distance

Euclidian Distance



$$distance = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

$$distance^2 = (x_2 - x_1)^2 + (y_2 - y_1)^2$$

Using Squared Distance

```
# For each molecule A find the closest molecule B
for molecule_A in molecules_type_A:

    # Save the closest molecule B squared distance found so far
    closest_molecule_B_squared_distance = float("inf")

    for molecule_B in molecules_type_B:

        # Use the squared distance to save time/computation.
        squared_distance = (molecule_A.x - molecule_B.x)**2 + (molecule_A.y - molecule_B.y)**2

        # If the squared distance of this molecule B
        # is less than the closest molecule B found so far
        if squared_distance < closest_molecule_B_squared_distance:
            closest_molecule_B_squared_distance = squared_distance
```

Using Squared Distance

```
# For each molecule A find the closest molecule B
for molecule_A in molecules_type_A:

    # Save the closest molecule B squared distance found so far
    closest_molecule_B_squared_distance = float("inf")

    for molecule_B in molecules_type_B:

        # Use the squared distance to save time/computation.
        squared_distance = (molecule_A.x - molecule_B.x)**2 + (molecule_A.y - molecule_B.y)**2

        # If the squared distance of this molecule B
        # is less than the closest molecule B found so far
        if squared_distance < closest_molecule_B_squared_distance:
            closest_molecule_B_squared_distance = squared_distance
```

Reaction Occurs if Close Enough

```
# If the A molecule is close enough to a B molecule then a reaction occurs
if closest_molecule_B_distance < closeness_distance:
    # Select a uniform random number between 0 and 1
    # If the random number is less than the constant p
    # then the first reaction A + B -> C occurs
    if random.uniform(0, 1) < p:
        to_remove_A.append(molecule_A)
        to_remove_B.append(molecule_B)
        # Form a new C at midpoint
        xC = (molecule_A.x + molecule_B.x) / 2
        yC = (molecule_A.y + molecule_B.y) / 2
        to_add_C.append(Molecule(xC, yC))
```

Reaction Occurs if Close Enough

```
# If the A molecule is close enough to a B molecule then a reaction occurs
if closest_molecule_B_distance < closeness_distance:
    # Select a uniform random number between 0 and 1
    # If the random number is less than the constant p
    # then the first reaction A + B -> C occurs
    if random.uniform(0, 1) < p:
        to_remove_A.append(molecule_A)
        to_remove_B.append(molecule_B)
        # Form a new C at midpoint
        xC = (molecule_A.x + molecule_B.x) / 2
        yC = (molecule_A.y + molecule_B.y) / 2
        to_add_C.append(Molecule(xC, yC))
```

Reaction Occurs if Close Enough

```
# If the A molecule is close enough to a B molecule then a reaction occurs
if closest_molecule_B_distance < closeness_distance:
    # Select a uniform random number between 0 and 1
    # If the random number is less than the constant p
    # then the first reaction A + B -> C occurs
    if random.uniform(0, 1) < p:
        to_remove_A.append(molecule_A)
        to_remove_B.append(molecule_B)
        # Form a new C at midpoint
        xC = (molecule_A.x + molecule_B.x) / 2
        yC = (molecule_A.y + molecule_B.y) / 2
        to_add_C.append(Molecule(xC, yC))
```

Random Movement of Molecules

The source of randomness in the **standard model** is which of two reactions to choose at each step

What happens in the **spatial model** depends on the random movement of the molecules

More "*chanciness*" and hence more fluctuation

Random Movement of Molecules

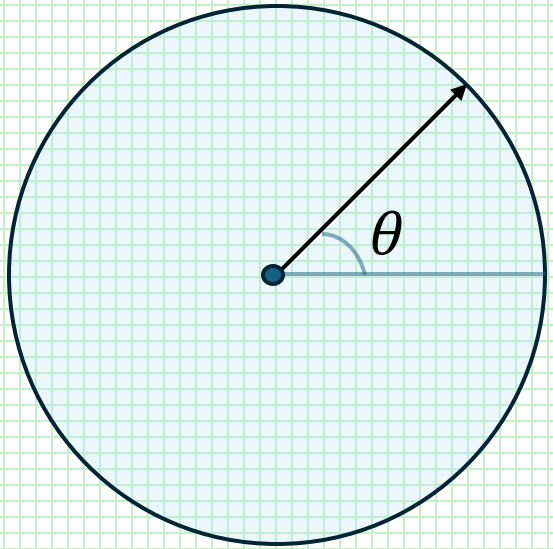
```
def move_molecules(molecules_list):

    # Randomly move each molecule
    for molecule in molecules_list:

        # Random chance to change direction
        if random.uniform(0, 1) < theta_change_probability:
            molecule.theta = random.uniform(0, 2 * math.pi)

        updated_x_position = molecule.x + move_length * math.cos(molecule.theta)
        updated_y_position = molecule.y + move_length * math.sin(molecule.theta)
```

Random Movement of Molecules

```
def move_molecules(molecules_list):

    # Randomly move each molecule
    for molecule in molecules_list:

        # Random chance to change direction
        if random.uniform(0, 1) < theta_change_probability:
            molecule.theta = random.uniform(0, 2 * math.pi)

        updated_x_position = molecule.x + move_length * math.cos(molecule.theta)
        updated_y_position = molecule.y + move_length * math.sin(molecule.theta)
```

Random Movement of Molecules

```
# Random chance to change direction
if random.uniform(0, 1) < theta_change_probability:
    molecule.theta = random.uniform(0, 2 * math.pi)
```



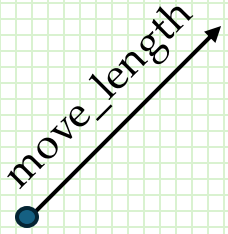
Random angle θ
in radians between 0 and 2π

Random Movement of Molecules

```
def move_molecules(molecules_list):

    # Randomly move each molecule
    for molecule in molecules_list:

        # Random chance to change direction
        if random.uniform(0, 1) < theta_change_probability:
            molecule.theta = random.uniform(0, 2 * math.pi)

        updated_x_position = molecule.x + move_length * math.cos(molecule.theta)
        updated_y_position = molecule.y + move_length * math.sin(molecule.theta)
```

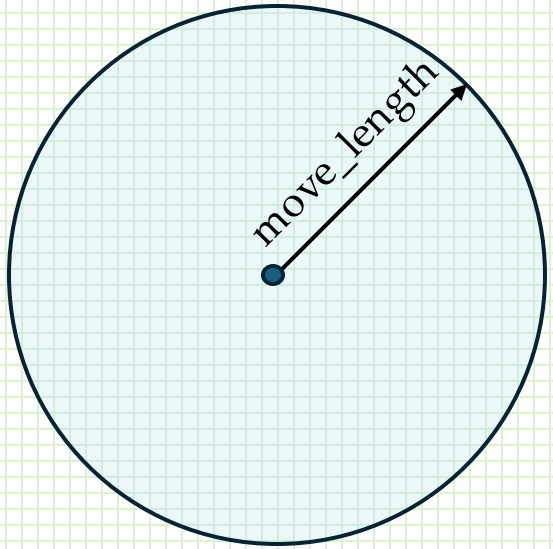
Random Movement of Molecules

```
updated_x_position = molecule.x + move_length * math.cos(molecule.theta)
updated_y_position = molecule.y + move_length * math.sin(molecule.theta)
```



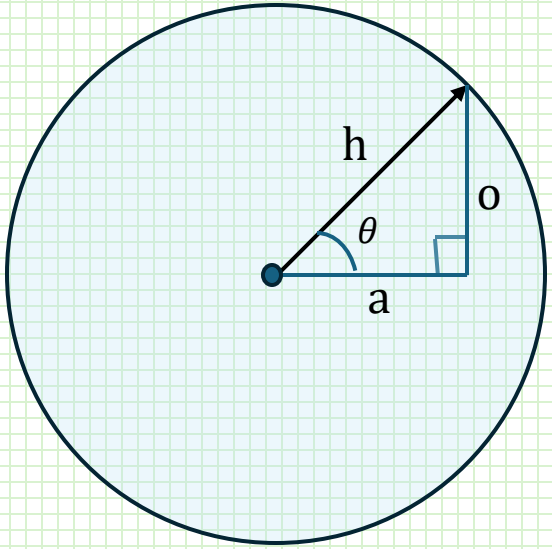
Random Movement of Molecules

```
updated_x_position = molecule.x + move_length * math.cos(molecule.theta)
updated_y_position = molecule.y + move_length * math.sin(molecule.theta)
```



Random Movement of Molecules

```
updated_x_position = molecule.x + move_length * math.cos(molecule.theta)
updated_y_position = molecule.y + move_length * math.sin(molecule.theta)
```

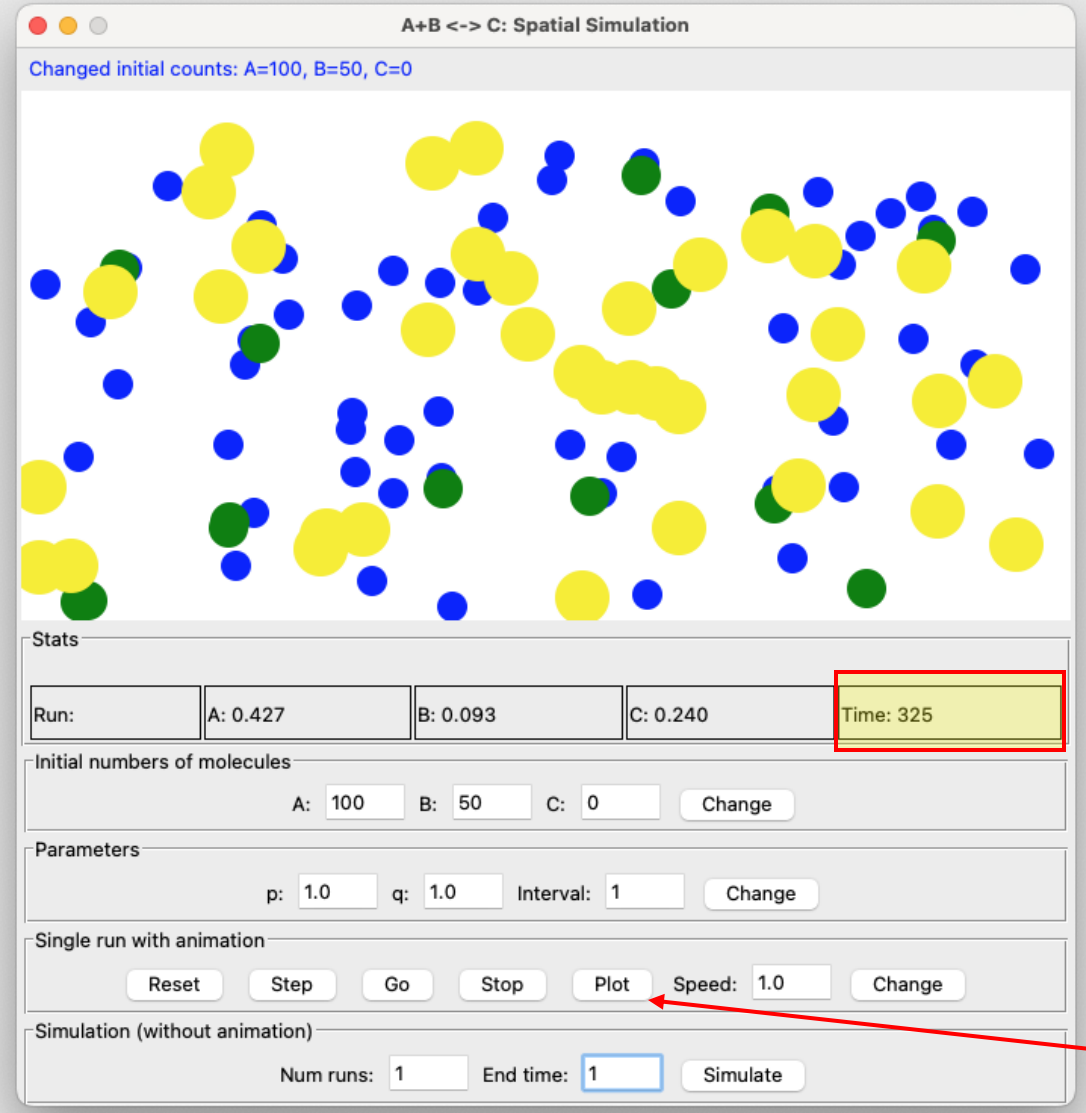


$$\boldsymbol{c}os(\theta) = \frac{\boldsymbol{a}djacent}{\boldsymbol{h}ypotenuse} \qquad \mathbf{s}\mathrm{in}(\theta) = \frac{\boldsymbol{o}pposite}{\boldsymbol{h}ypotenuse}$$

x component: $hypotenuse \times cos(\theta) = adjacent$

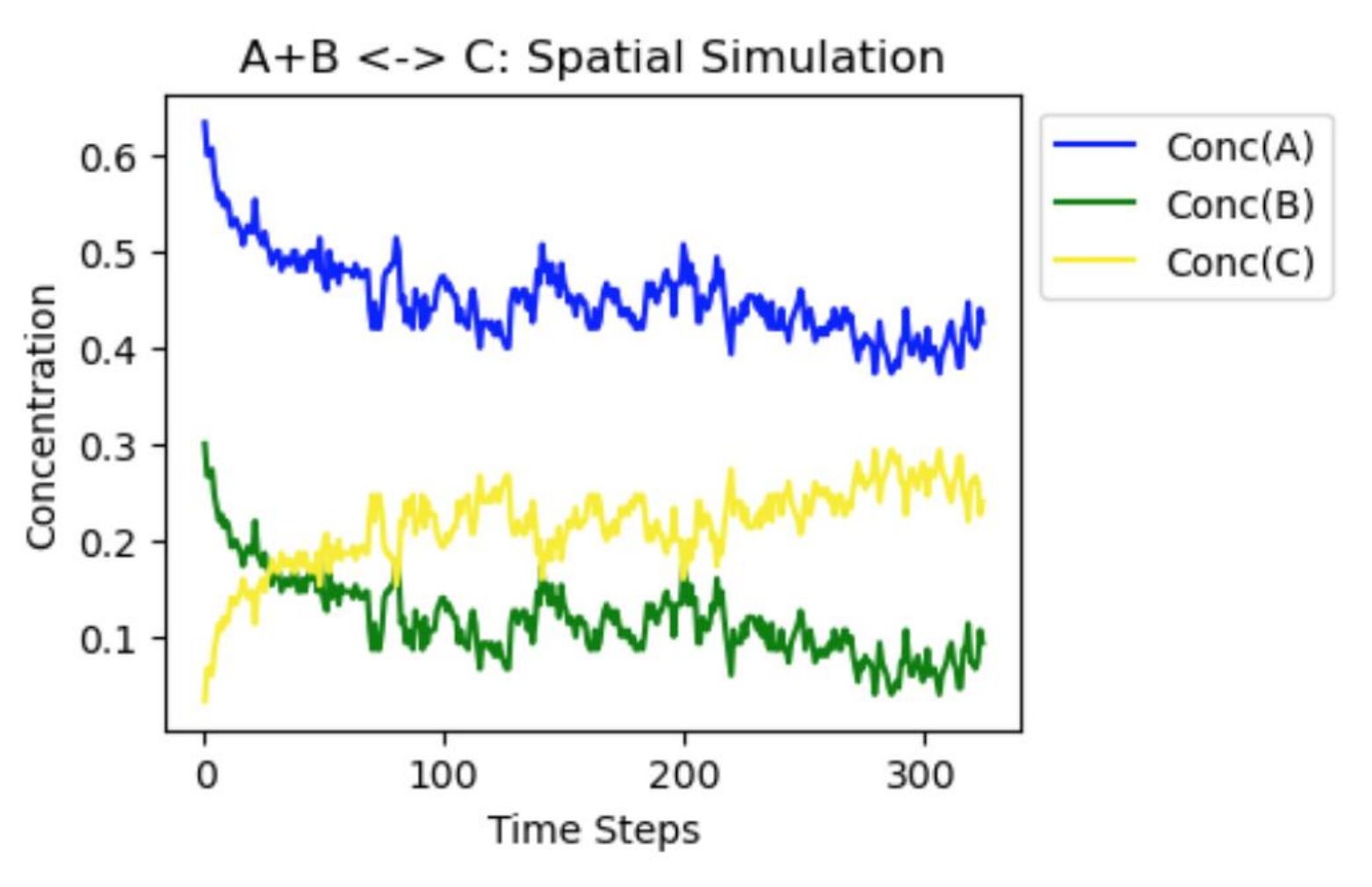
y component: $hypotenuse \times \sin(\theta) = opposite$

Long-Term Behavior

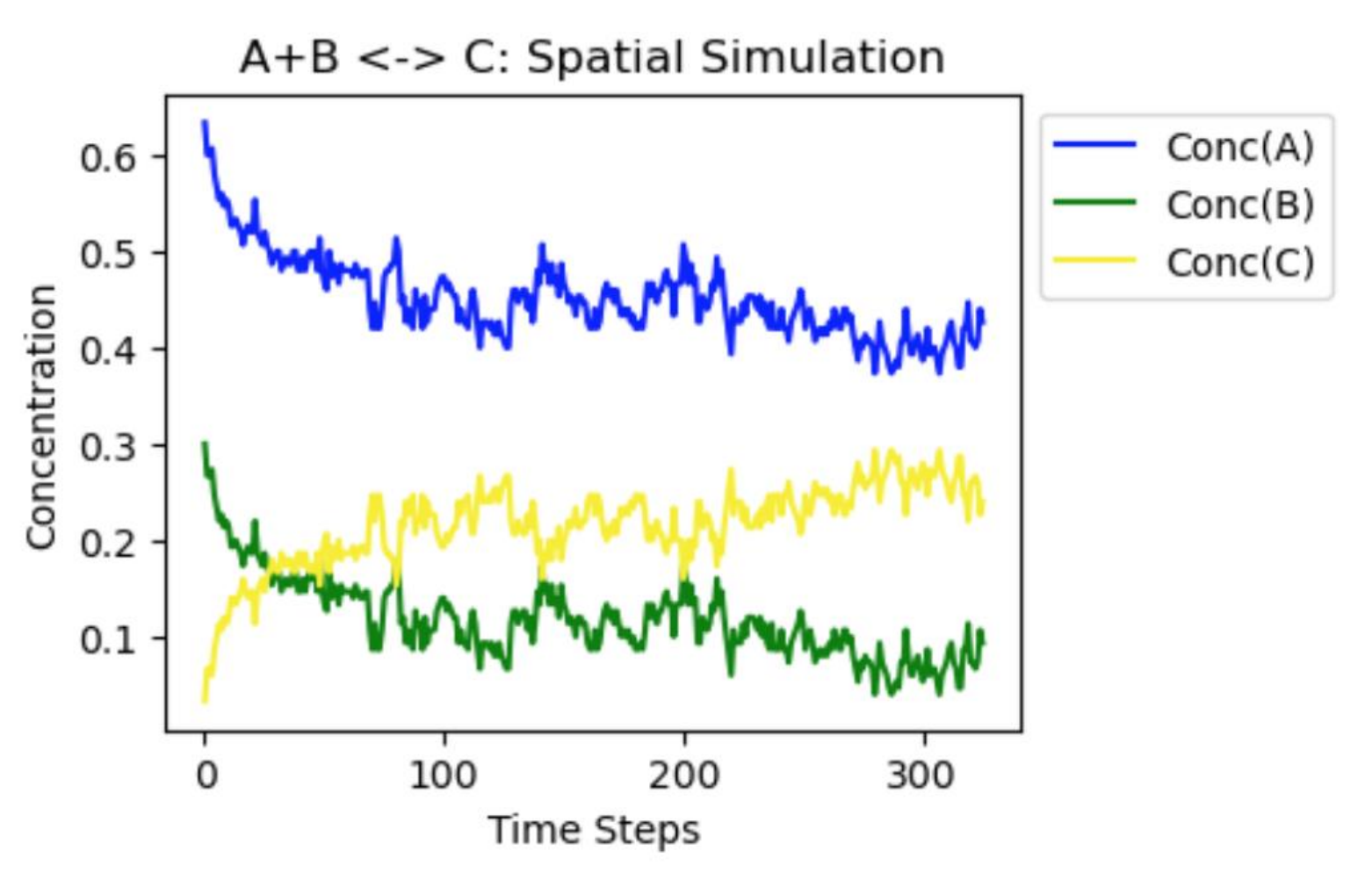


Click “Plot”

Long-Term Behavior

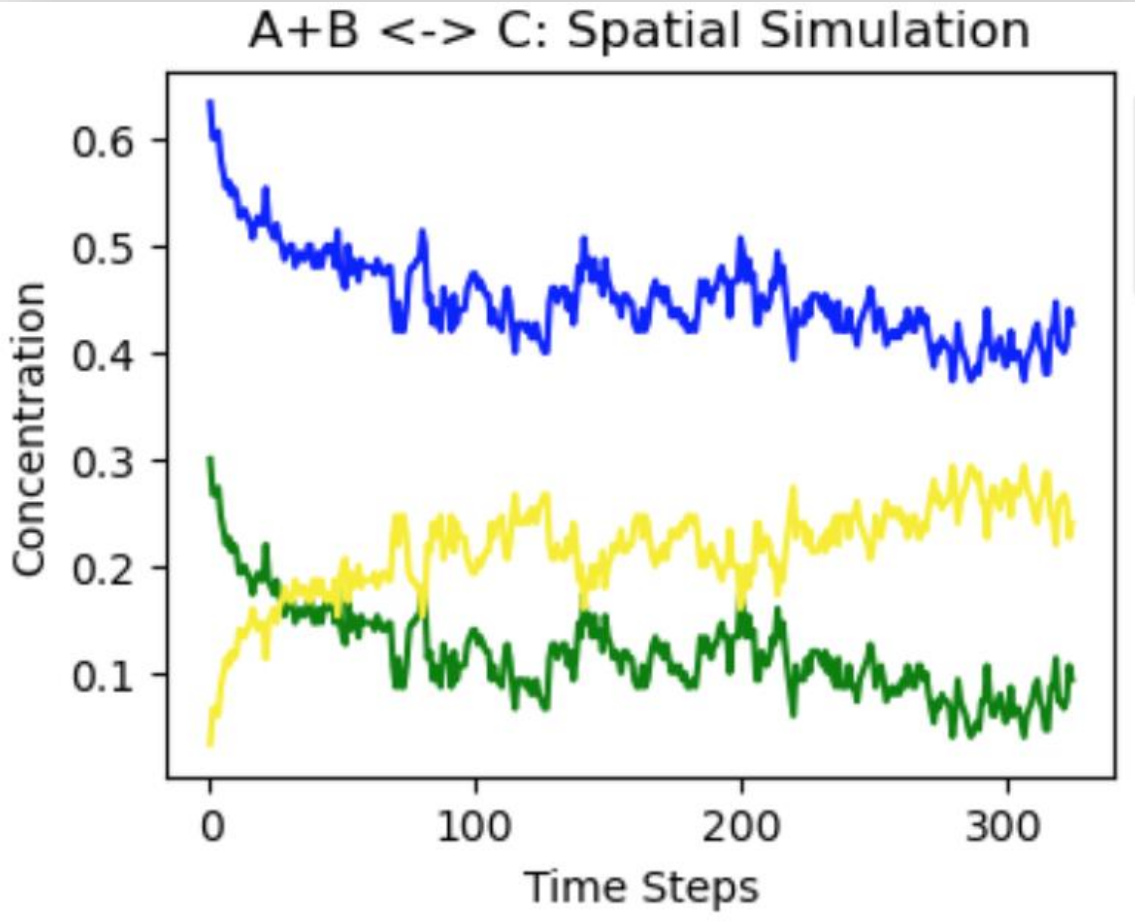


Long-Term Behavior

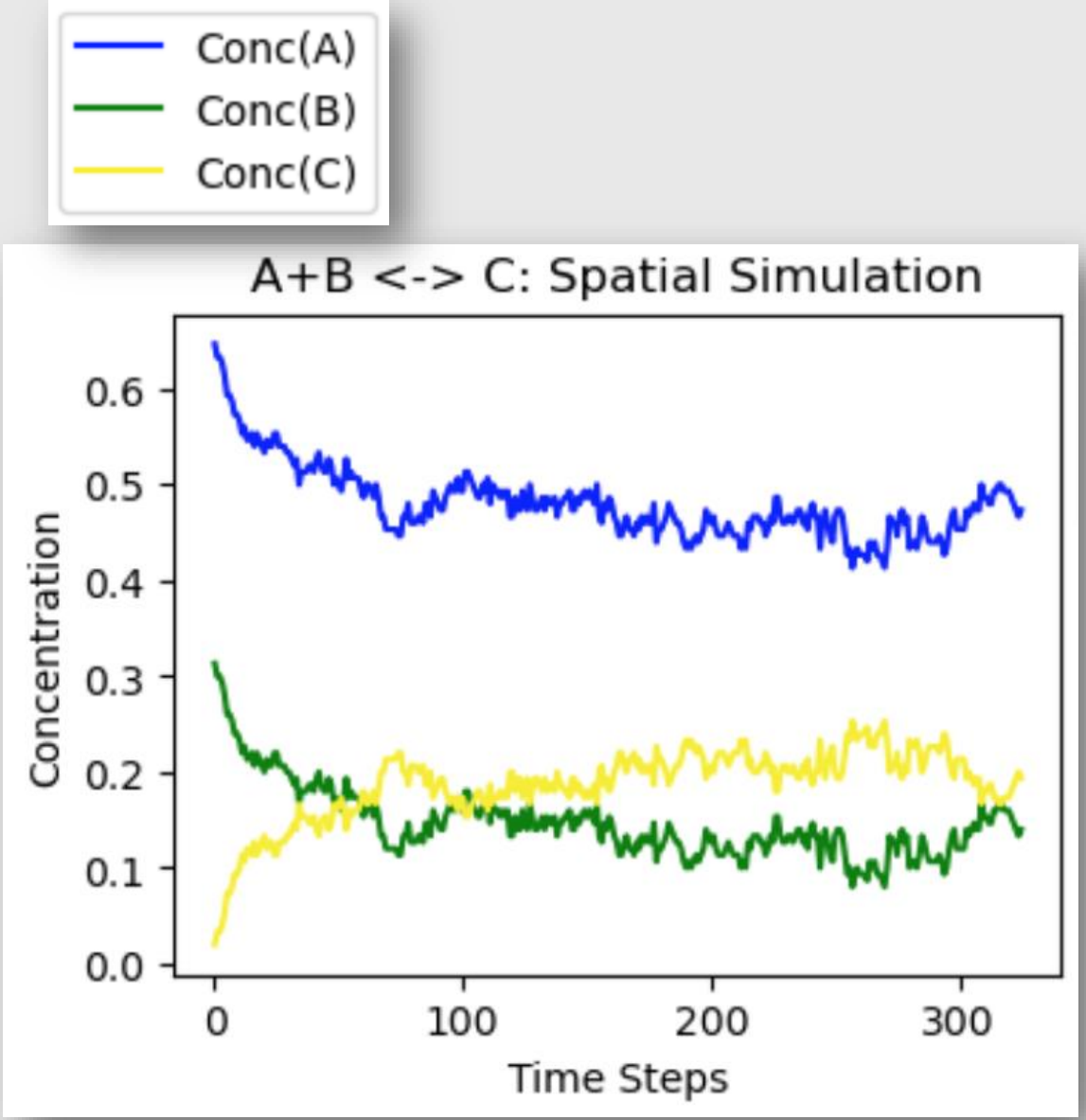



```
closeness_distance = 100
```

Long-Term Behavior

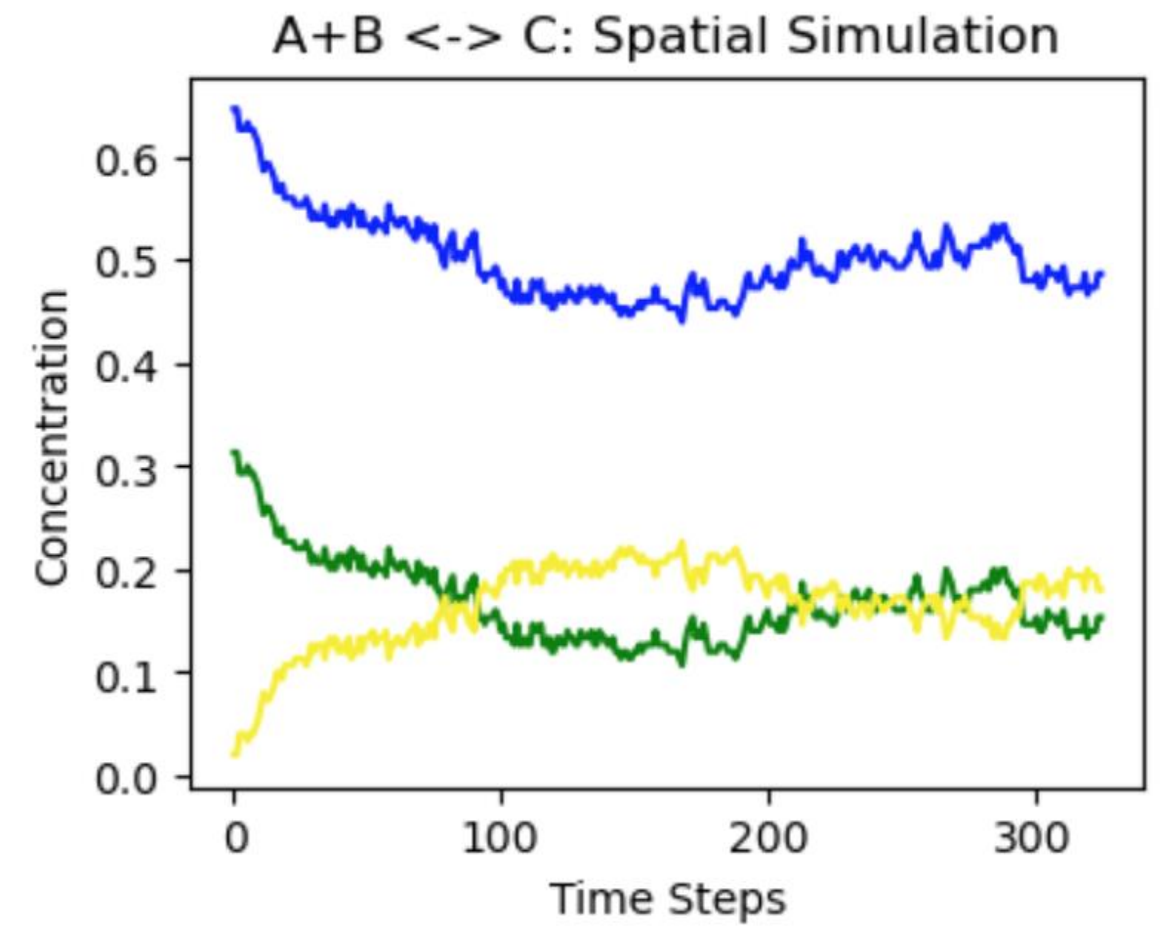



```
closeness_distance = 100
```

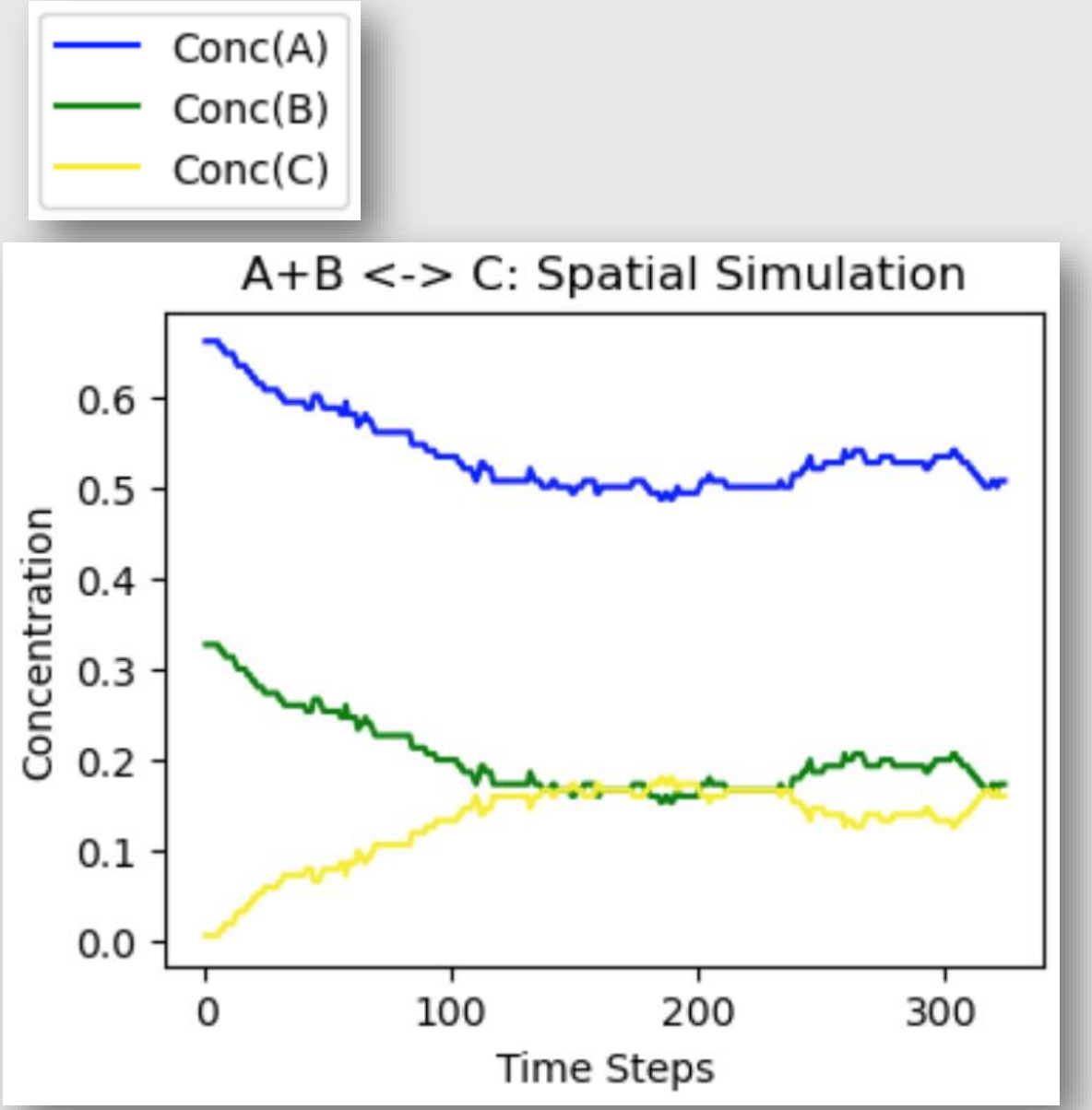



```
closeness_distance = 50
```

Long-Term Behavior




```
closeness_distance = 25
```




```
closeness_distance = 5
```