CELL

an exploration of artificial life

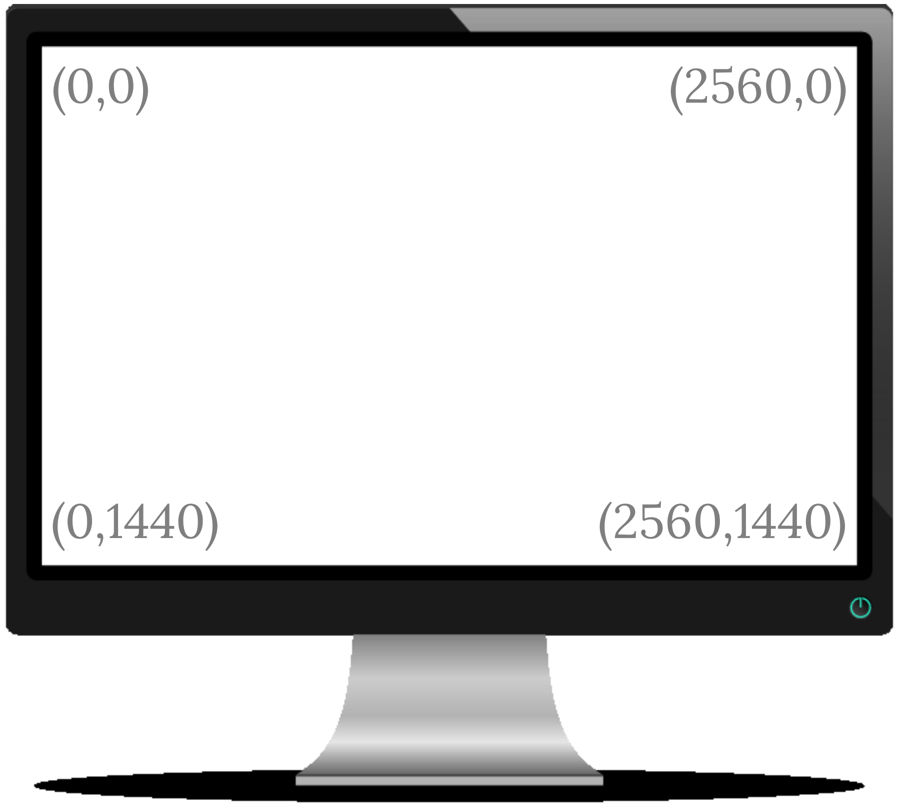
Owen Martin – Quarter One Evidence Packet

PROGRAM MATRIX

Fundamentally, the CELL program is a constantly painted and repainted applet generated by Java code in which objects interact according to the laws of physics.

The applet screen is set to 2560 x 1440 pixels by default. This creates a two-dimensional spacial matrix within which the particles of CELL can operate; their positions are individually determined as pixel coordinates along the x and y axes.

The computer does not recognize the traditional representation of Euclidean space. Instead, the y axis is designated as increasing positively in the downward direction, still perpendicular to the x axis.



PARTICLE ATTRIBUTES

Every particle in CELL has physical attributes constantly passed into equations that govern their movements. These attributes include net force, mass, acceleration, velocity, and position.

force and motion equations

$$\vec{a} = \frac{\Sigma \vec{F}}{m}$$

$$x = x_0 + v_{x0}t + \frac{1}{2}a_x t^2$$

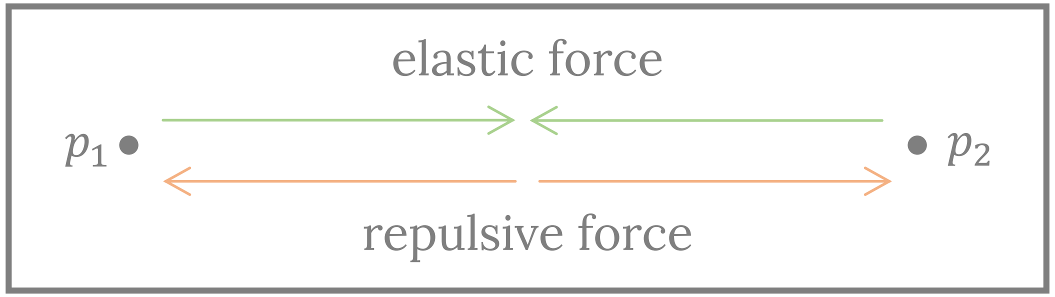
$$v_x = v_{x0} + a_x t$$

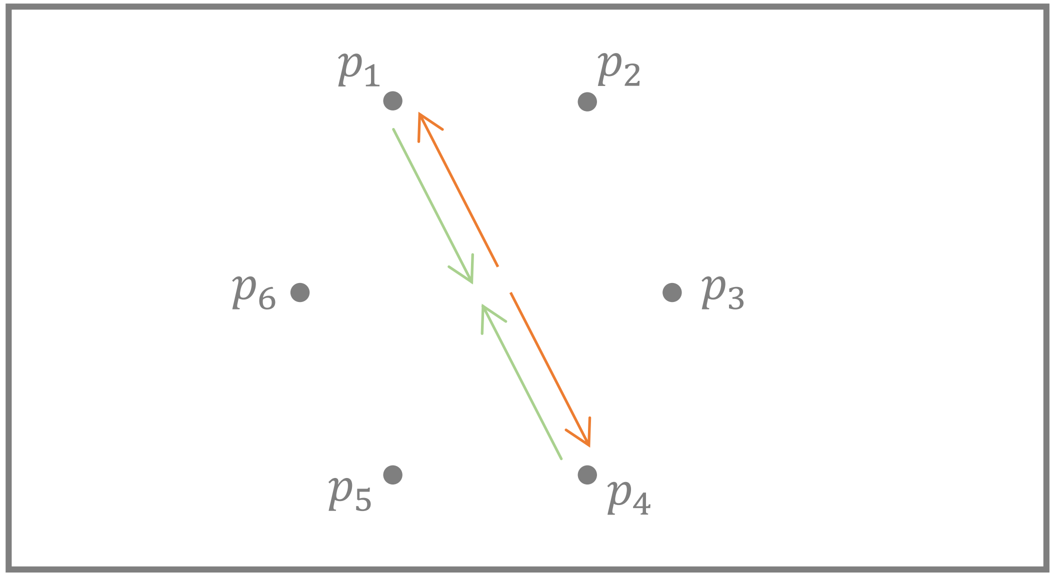
$$v_x^2 = v_{x0}^2 + 2a_x(x - x_0)$$

$$\Delta\vec{p} = \vec{F} \cdot \Delta t$$

$$\vec{p} = m \cdot \vec{v}$$

THE MEMBRANE



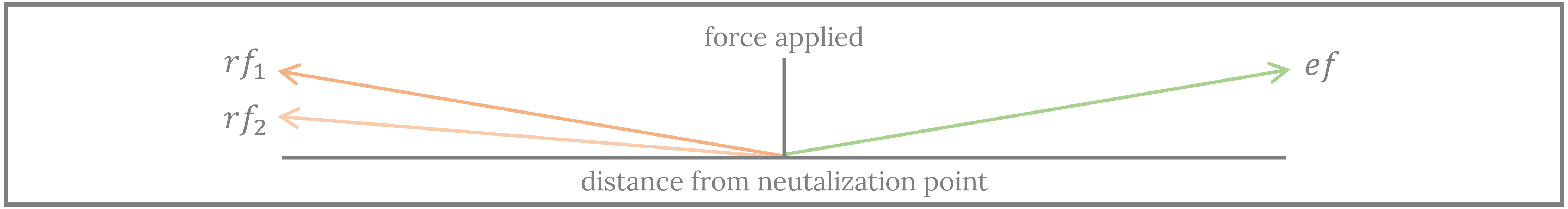


The membrane is currently comprised of six particles arranged as a regular hexagon. Every particle is connected to its neighbors by two forces: an elastic force and a repulsive force.

The magnitude of these two forces is dependent on the particles' distances from each other; as the distance between two particles increases, the elastic force between them increases and the repulsive force decreases.

The two forces cancel each other out at a specific distance between each pair of points. Thus, the circumference of the cell is the product of this distance and the total number of points in the membrane.

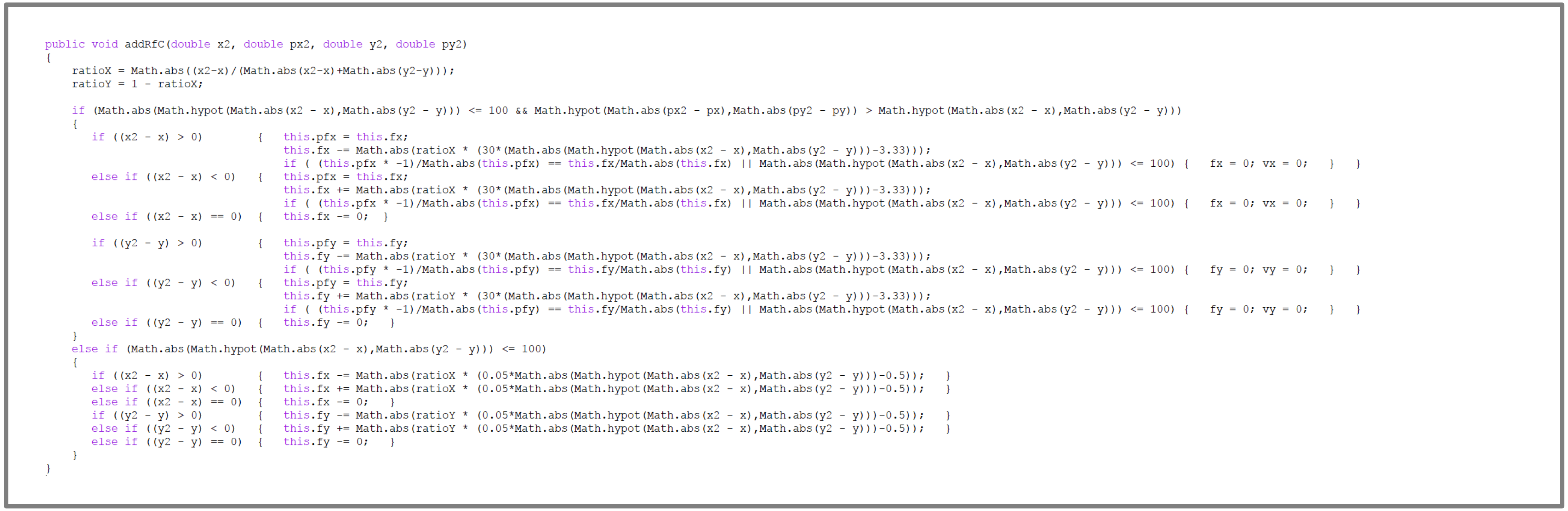
NEUTRALIZING FORCES



In the interest of accurately modeling the physical process by which an elastic barrier stabilizes, a system of functions was created to govern the elastic and repulsive forces between each pair of particles. The graph of the functions are shown above, and each operates as follows: as d, or the distance from the point of neutralization changes, two functions of equal magnitude act to propel the particles towards this point. This, however, creates a perpetually oscillating distance between the two particles. To counteract this, a second function is integrated into the repulsive force. It acts immediately after the primary repulsive function changes the direction of the particle; the particle is sent with perpetually decreasing force after each oscillation, appearing to stabilize at the point of neutralization.

REPULSIVE FORCE CODE

The following is the code used to implement the repulsive force along the cell's circumference.



```java
public void addRfC(double x2, double px2, double y2, double py2)
{
    ratioX = Math.abs((x2-x)/(Math.abs(x2-x)+Math.abs(y2-y)));
    ratioY = 1 - ratioX;

    if (Math.abs(Math.hypot(Math.abs(x2 - x),Math.abs(y2 - y))) <= 100 && Math.hypot(Math.abs(px2 - px),Math.abs(py2 - py)) > Math.hypot(Math.abs(x2 - x),Math.abs(y2 - y)))
    {
       if ((x2 - x) > 0)        {   this.pfx = this.fx;
                                    this.fx -= Math.abs(ratioX * (30*(Math.abs(Math.hypot(Math.abs(x2 - x),Math.abs(y2 - y)))-3.33)));
                                    if ( (this.pfx * -1)/Math.abs(this.pfx) == this.fx/Math.abs(this.fx) || Math.abs(Math.hypot(Math.abs(x2 - x),Math.abs(y2 - y))) <= 100) {   fx = 0; vx = 0;    }   }
       else if ((x2 - x) < 0)   {   this.pfx = this.fx;
                                    this.fx += Math.abs(ratioX * (30*(Math.abs(Math.hypot(Math.abs(x2 - x),Math.abs(y2 - y)))-3.33)));
                                    if ( (this.pfx * -1)/Math.abs(this.pfx) == this.fx/Math.abs(this.fx) || Math.abs(Math.hypot(Math.abs(x2 - x),Math.abs(y2 - y))) <= 100) {   fx = 0; vx = 0;    }   }
       else if ((x2 - x) == 0)  {   this.fx -= 0;  }

       if ((y2 - y) > 0)        {   this.pfy = this.fy;
                                    this.fy -= Math.abs(ratioY * (30*(Math.abs(Math.hypot(Math.abs(x2 - x),Math.abs(y2 - y)))-3.33)));
                                    if ( (this.pfy * -1)/Math.abs(this.pfy) == this.fy/Math.abs(this.fy) || Math.abs(Math.hypot(Math.abs(x2 - x),Math.abs(y2 - y))) <= 100) {   fy = 0; vy = 0;    }   }
       else if ((y2 - y) < 0)   {   this.pfy = this.fy;
                                    this.fy += Math.abs(ratioY * (30*(Math.abs(Math.hypot(Math.abs(x2 - x),Math.abs(y2 - y)))-3.33)));
                                    if ( (this.pfy * -1)/Math.abs(this.pfy) == this.fy/Math.abs(this.fy) || Math.abs(Math.hypot(Math.abs(x2 - x),Math.abs(y2 - y))) <= 100) {   fy = 0; vy = 0;    }   }
       else if ((y2 - y) == 0)  {   this.fy -= 0;   }
    }
    else if (Math.abs(Math.hypot(Math.abs(x2 - x),Math.abs(y2 - y))) <= 100)
    {
       if ((x2 - x) > 0)        {   this.fx -= Math.abs(ratioX * (0.05*Math.abs(Math.hypot(Math.abs(x2 - x),Math.abs(y2 - y)))-0.5));   }
       else if ((x2 - x) < 0)   {   this.fx += Math.abs(ratioX * (0.05*Math.abs(Math.hypot(Math.abs(x2 - x),Math.abs(y2 - y)))-0.5));   }
       else if ((x2 - x) == 0)  {   this.fx -= 0;   }
       if ((y2 - y) > 0)        {   this.fy -= Math.abs(ratioY * (0.05*Math.abs(Math.hypot(Math.abs(x2 - x),Math.abs(y2 - y)))-0.5));   }
       else if ((y2 - y) < 0)   {   this.fy += Math.abs(ratioY * (0.05*Math.abs(Math.hypot(Math.abs(x2 - x),Math.abs(y2 - y)))-0.5));   }
       else if ((y2 - y) == 0)  {   this.fy -= 0;   }
    }
}
```

ELASTIC FORCE CODE

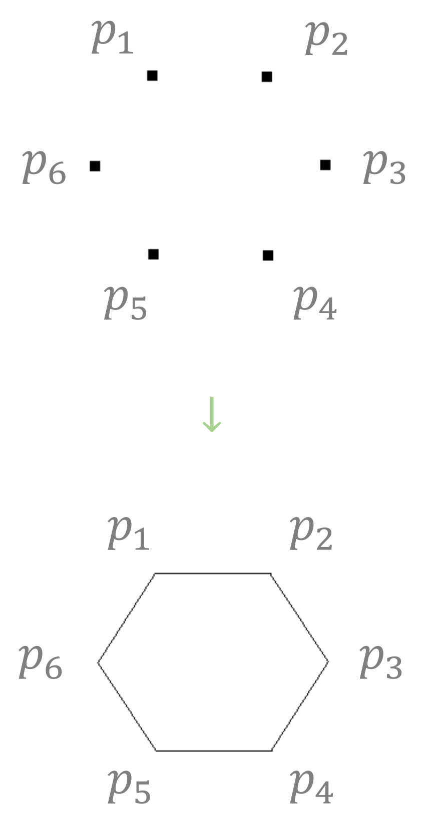
The following is the code used to implement the elastic force along the cell's circumference.

```java
public void addEfC(double x2, double y2)
{
    ratioX = Math.abs((x2-x)/(Math.abs(x2-x)+Math.abs(y2-y)));
    ratioY = 1 - ratioX;

    if (Math.hypot(Math.abs(x2 - x),Math.abs(y2 - y)) > 100)
    {
       if ((x2 - x) > 0)        {   this.fx += Math.abs(ratioX * (10*(Math.abs(Math.hypot(Math.abs(x2 - x),Math.abs(y2 - y)))-10)));   }
       else if ((x2 - x) < 0)   {   this.fx -= Math.abs(ratioX * (10*(Math.abs(Math.hypot(Math.abs(x2 - x),Math.abs(y2 - y)))-10)));   }
       else if ((x2 - x) == 0)  {   this.fx += 0;  }

       if ((y2 - y) > 0)        {   this.fy += Math.abs(ratioY * (10*(Math.abs(Math.hypot(Math.abs(x2 - x),Math.abs(y2 - y)))-10)));   }
       else if ((y2 - y) < 0)   {   this.fy -= Math.abs(ratioY * (10*(Math.abs(Math.hypot(Math.abs(x2 - x),Math.abs(y2 - y)))-10)));   }
       else if ((y2 - y) == 0)  {   this.fy += 0;   }
    }
}
```

VISUAL DISPLAY

As mentioned previously, the CELL program is a constantly painted and repainted applet. Each particle of the cell was initially drawn as a 9 x 9 pixel square, as seen to the right.

The particles were later changed to single pixel points, and were connected by lines to form a hexagon.



TROUBLESHOOTING

Multiple issues arose while attempting to create the basis of the CELL program, some of which remain unsolved. These problems and their solutions include:

Particles oscillating with increasing distance. →	Second reflexive force added.
Particle momentum not being conserved. →	Absolute values implemented.
Particles not retaining $\frac{x}{y}$ ratio. →	Distribution of forces altered.
Particles remaining fixed in one position. →	setVx/setVy methods removed.
Only diameter forces remaining stable. →	Unsolved.

THANKS FOR WATCHING!

Stay tuned for Quarter Two Evidence Packet