CELL

Quarter 4 Evidence Packet

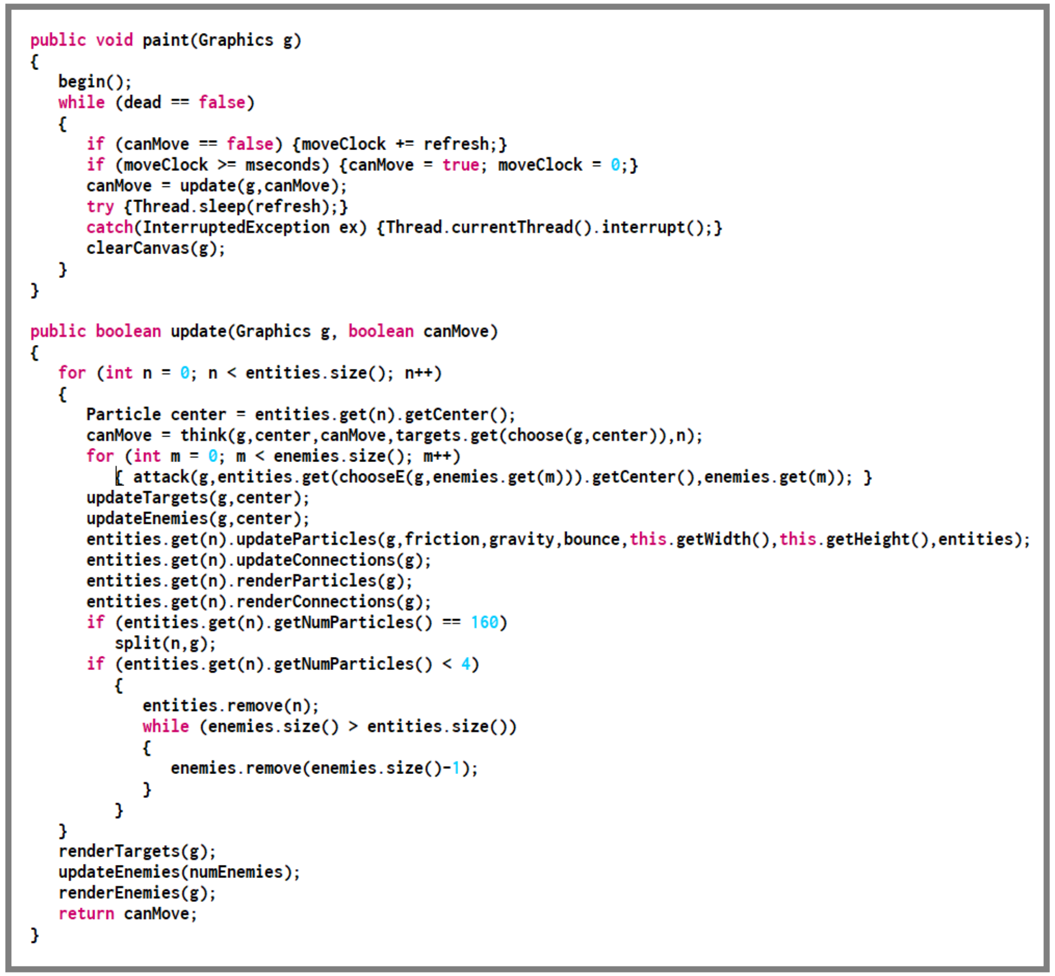
Owen Martin

PROGRAM STRUCTURE

For the first three quarters of AOIT III, the CELL program has been structured such that the behavior of the focus cell is determined considering other object's attribute values, namely the positions of targets and enemies.

In order to create more focus cells in the program, it was necessary to excise the code pertaining to cell behavior from the main class into a separate Entity class.

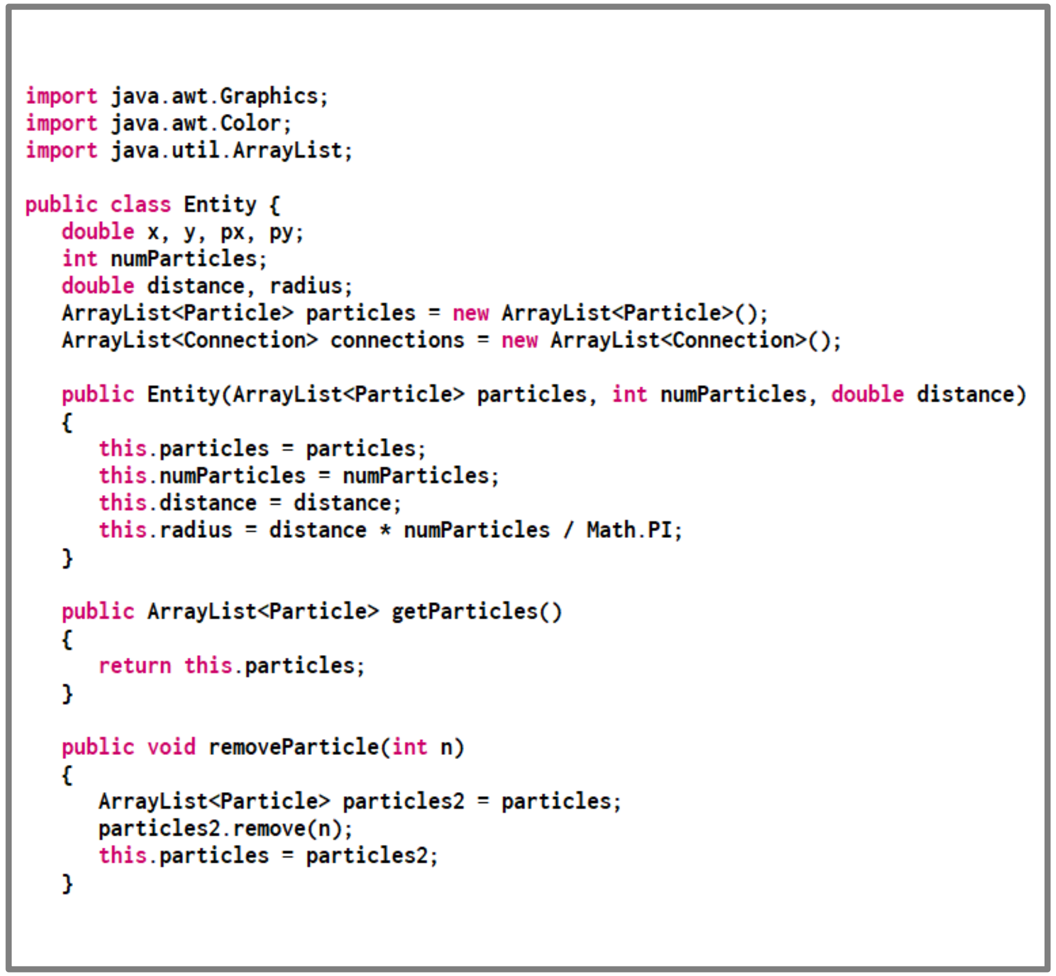
The main program was, therefore, reduced to contain mostly procedural methods called throughout the program's total runtime, as seen to the right.



```
public void paint(Graphics g)
{
   begin();
   while (dead == false)
   {
      if (canMove == false) {moveClock += refresh;}
      if (moveClock >= mseconds) {canMove = true; moveClock = 0;}
      canMove = update(g,canMove);
      try {Thread.sleep(refresh);}
      catch(InterruptedException ex) {Thread.currentThread().interrupt();}
      clearCanvas(g);
   }
}

public boolean update(Graphics g, boolean canMove)
{
   for (int n = 0; n < entities.size(); n++)
   {
      Particle center = entities.get(n).getCenter();
      canMove = think(g,center,canMove,targets.get(choose(g,center)),n);
      for (int m = 0; m < enemies.size(); m++)
         { attack(g,entities.get(chooseE(g,enemies.get(m))).getCenter(),enemies.get(m)); }
      updateTargets(g,center);
      updateEnemies(g,center);
      entities.get(n).updateParticles(g,friction,gravity,bounce,this.getWidth(),this.getHeight(),entities);
      entities.get(n).updateConnections(g);
      entities.get(n).renderParticles(g);
      entities.get(n).renderConnections(g);
      if (entities.get(n).getNumParticles() == 160)
         split(n,g);
      if (entities.get(n).getNumParticles() < 4)
         {
            entities.remove(n);
            while (enemies.size() > entities.size())
            {
               enemies.remove(enemies.size()-1);
            }
         }
   }
   renderTargets(g);
   updateEnemies(numEnemies);
   renderEnemies(g);
   return canMove;
}
```

THE ENTITY CLASS



```java
import java.awt.Graphics;
import java.awt.Color;
import java.util.ArrayList;

public class Entity {
   double x, y, px, py;
   int numParticles;
   double distance, radius;
   ArrayList<Particle> particles = new ArrayList<Particle>();
   ArrayList<Connection> connections = new ArrayList<Connection>();

   public Entity(ArrayList<Particle> particles, int numParticles, double distance)
   {
      this.particles = particles;
      this.numParticles = numParticles;
      this.distance = distance;
      this.radius = distance * numParticles / Math.PI;
   }

   public ArrayList<Particle> getParticles()
   {
      return this.particles;
   }

   public void removeParticle(int n)
   {
      ArrayList<Particle> particles2 = particles;
      particles2.remove(n);
      this.particles = particles2;
   }
```

The Entity class, containing primarily the code removed from the old main class, also shares many lines of code related to the Particle and Target classes.

Many methods pertaining to the entity's constituent particles were flushed out, including getParticles(), removeParticle(), and updateParticles().

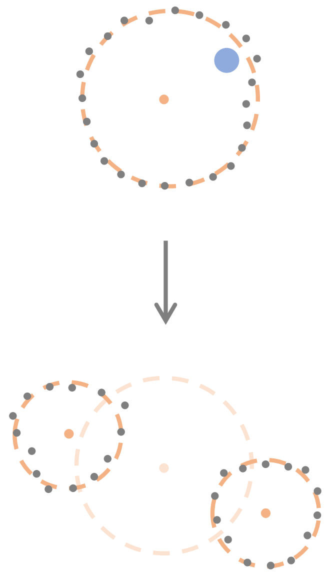
Attributes not included in the Particle and Target classes were also added, including (int) numParticles, (double) distance and radius, and (ArrayList<Particle>) particles and (ArrayList<Connection>) connections.

THE SPLIT() METHOD

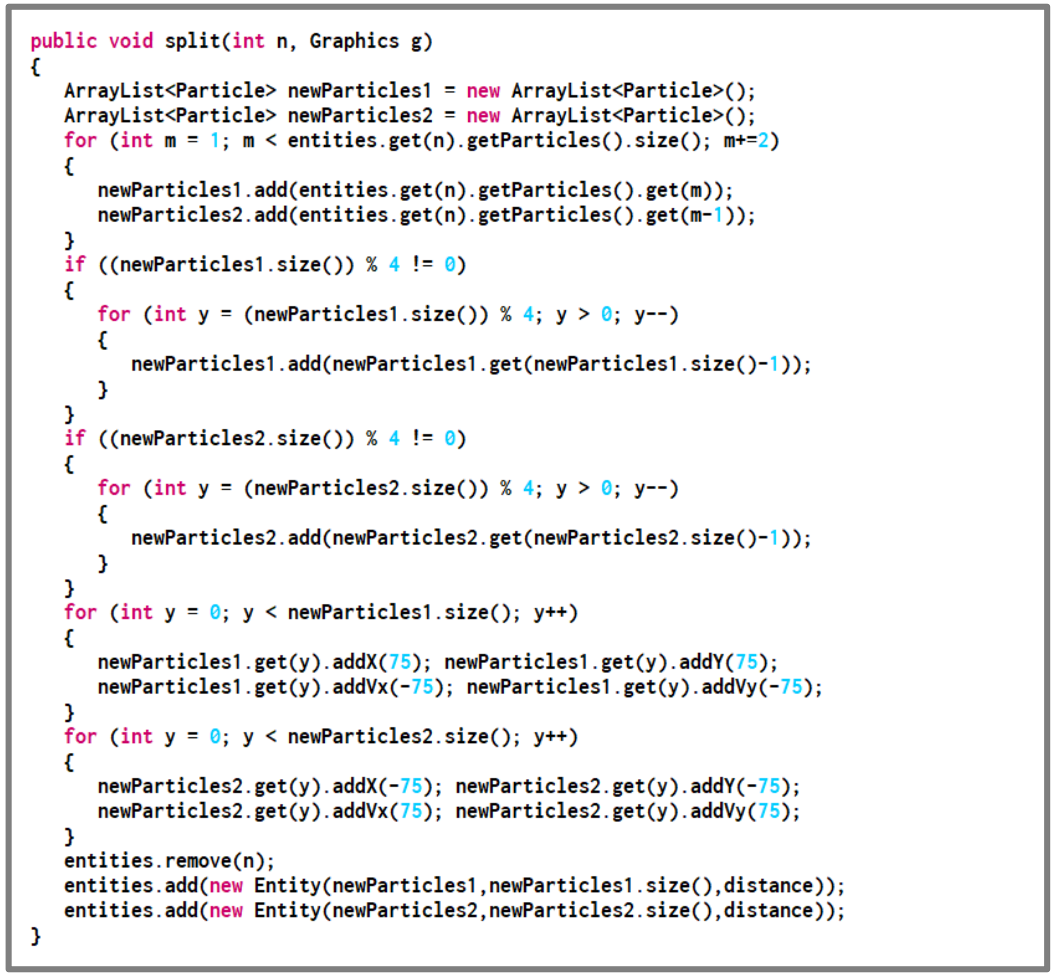
An appropriate representation of biological reproduction would require a suitable criterion for reproduction to occur. In the CELL program, this was decided to be number of constituent Particle objects.

The split() method was created to facilitate the process of entity reproduction. It is only called if the total number of particle objects in an entity's membrane equals an arbitrary number that is a factor of 16, for reasons articulated later.

When an entity fulfills the criterion for reproduction, its membrane particles are transferred completely into two separate entities, which are then propelled away from each other.

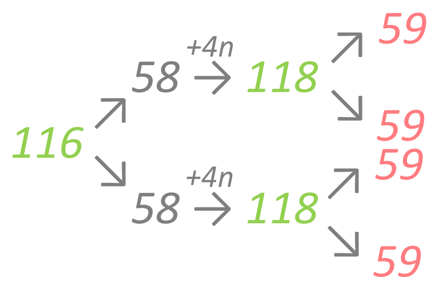

The before and after of an entity's split.

SPLIT() METHOD CODE

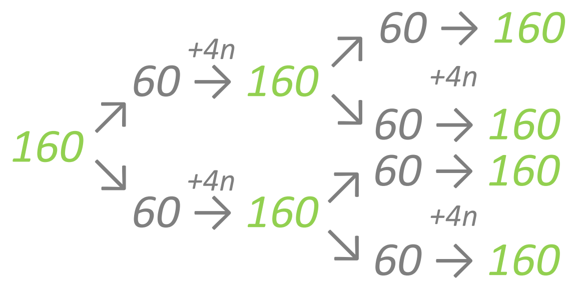


```java
public void split(int n, Graphics g)
{
   ArrayList<Particle> newParticles1 = new ArrayList<Particle>();
   ArrayList<Particle> newParticles2 = new ArrayList<Particle>();
   for (int m = 1; m < entities.get(n).getParticles().size(); m+=2)
   {
      newParticles1.add(entities.get(n).getParticles().get(m));
      newParticles2.add(entities.get(n).getParticles().get(m-1));
   }
   if ((newParticles1.size()) % 4 != 0)
   {
      for (int y = (newParticles1.size()) % 4; y > 0; y--)
      {
         newParticles1.add(newParticles1.get(newParticles1.size()-1));
      }
   }
   if ((newParticles2.size()) % 4 != 0)
   {
      for (int y = (newParticles2.size()) % 4; y > 0; y--)
      {
         newParticles2.add(newParticles2.get(newParticles2.size()-1));
      }
   }
   for (int y = 0; y < newParticles1.size(); y++)
   {
      newParticles1.get(y).addX(75); newParticles1.get(y).addY(75);
      newParticles1.get(y).addVx(-75); newParticles1.get(y).addVy(-75);
   }
   for (int y = 0; y < newParticles2.size(); y++)
   {
      newParticles2.get(y).addX(-75); newParticles2.get(y).addY(-75);
      newParticles2.get(y).addVx(75); newParticles2.get(y).addVy(75);
   }
   entities.remove(n);
   entities.add(new Entity(newParticles1,newParticles1.size(),distance));
   entities.add(new Entity(newParticles2,newParticles2.size(),distance));
}
```

SPLIT() CRITERIA



The breakdown of a flawed split criterion.



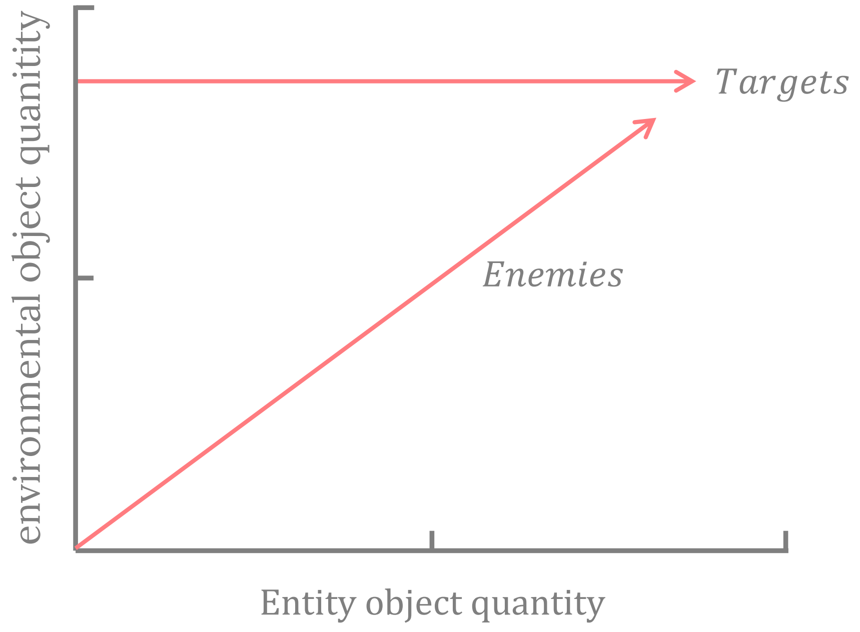
The breakdown of a correct split criterion.

The split() method operates by dividing the existing Particle objects in an entity's membrane into two separate ArrayLists belonging to two different entities.

Because each newly created entity must create connections between its particles to stay together, the number of particles in both new entities must be even. In light of this, the number of particles in the original entity's membrane must be a factor of four.

Additionally, because the newer entities themselves must reproduce, the number of particles in their new membranes must also be factors of four. In all, the original Entity must reproduce when it reaches a number that is a factor of 16.

ENVIRONMENTAL DYNAMICS



The graph of environmental object quanity versus Entity object quanity.

In order to ensure a long program runtime, certain environmental factors had to be made dynamic. For instance, a constant number of Enemy objects would be inappropriate if one were to cap the growth of an Entity population.

To achieve that end, it was noticed that one Enemy object per Entity object provided a stable ratio under certain conditions.

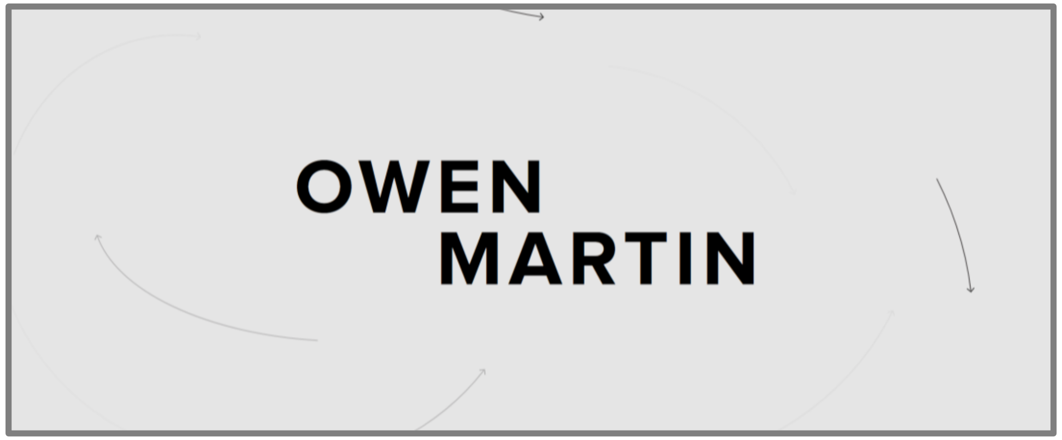
The number of Target objects, which serve as food to Entity objects, was kept constant to ensure that all Entity objects are not destroyed by the surplus Enemy objects.

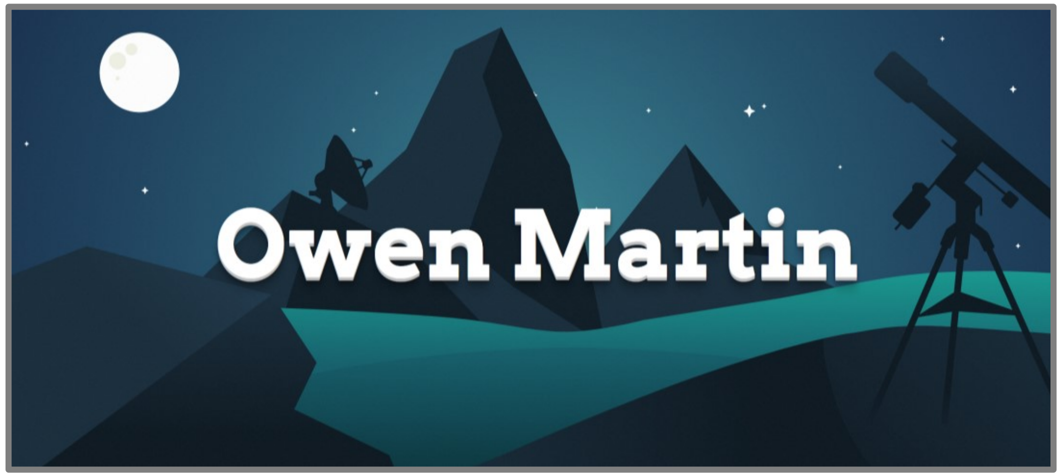
DIGITAL PORTFOLIO

The last portion of the AOIT III curriculum consists of updating one's digital portfolio to include all independent study materials. While I personally like my old portfolio, I did find some revisions to be in order.

One significant alteration took place in the homepage of the portfolio. The original name-and-arrows design (top right) was replaced with a perspective-based flat background (bottom right).

The new design features 3-dimensional text (code courtesy of Michael Garvey) and scalable SVG images converted from Microsoft PowerPoint.





THANKS FOR WATCHING!