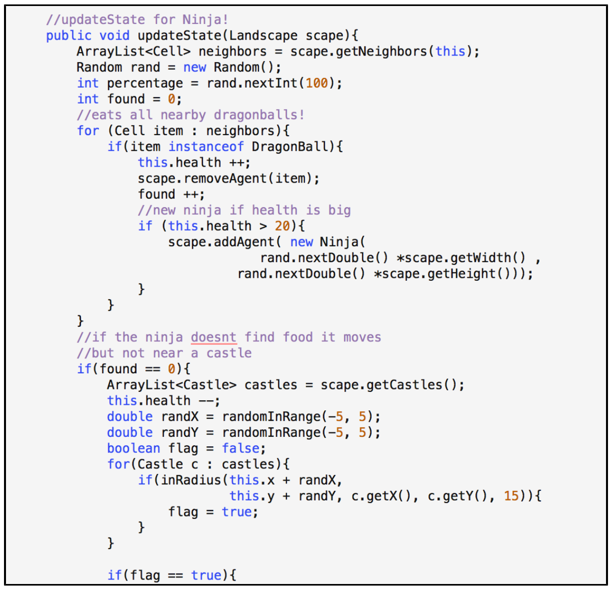
Project 5

In this project, we continued with our Cell and Landscape work but got a little more in depth. This week, instead of using a singly LinkedList, we are using a doubly LinkedList so we can traverse through the list in both directions. The main task of this project was to create our own simulation that modeled a landscape with a predator, a prey, a producer, and an obstacle and being able to make them all interact. To do this, we created a number of Cell subclasses and gave them all there individual updateState methods. We use the same framework from Project 4, but get more complex with the interaction of multiple cells within the landscape.

Task 1. The first part of this project was to plan out our idea for the simulation. I decided I wanted to create a simulation in which Ninjas tried to eat DragonBalls for health. The DragonBalls were created from a Gift box and the obstacles in the landscape were Castles. The Ninjas are predators, the DragonBalls are prey, and the Gifts are the producers.

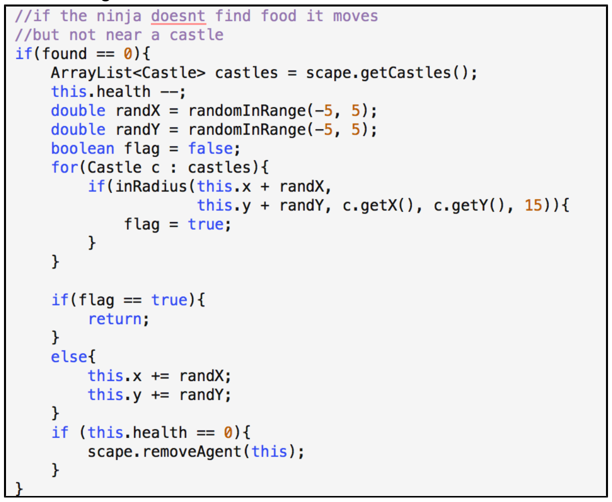
Task 2. For this task, we simply changed the parameters of all our updateState methods from a LinkedList list of neighbors to a new Landscape scape.

Task 3. This task took by far the longest amount of time. We had to write and code all the various subclasses we needed for our individual simulation. I had to write a class Ninja, DragonBall, Castle and Gift. Each one of the subclasses had an updateState method. The updateStates for each class was very different depending on the action of the Cell. For example, Ninjas eat DragonBalls for health, move randomly at a 10 percent chance, and die if there health gets too low. Here is a snippet of the code from the Ninja updateState:



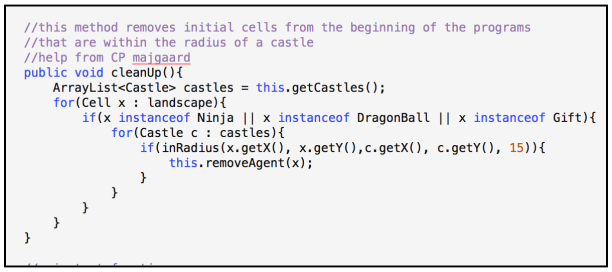
```java
    //updateState for Ninja!
    public void updateState(Landscape scape){
        ArrayList<Cell> neighbors = scape.getNeighbors(this);
        Random rand = new Random();
        int percentage = rand.nextInt(100);
        int found = 0;
        //eats all nearby dragonballs!
        for (Cell item : neighbors){
            if(item instanceof DragonBall){
                this.health ++;
                scape.removeAgent(item);
                found ++;
                //new ninja if health is big
                if (this.health > 20){
                    scape.addAgent( new Ninja(
                                rand.nextDouble() *scape.getWidth() ,
                             rand.nextDouble() *scape.getHeight()));
                }
            }
        }
        //if the ninja doesnt find food it moves
        //but not near a castle
        if(found == 0){
            ArrayList<Castle> castles = scape.getCastles();
            this.health --;
            double randX = randomInRange(-5, 5);
            double randY = randomInRange(-5, 5);
            boolean flag = false;
            for(Castle c : castles){
                if(inRadius(this.x + randX,
                            this.y + randY, c.getX(), c.getY(), 15)){
                    flag = true;
                }
            }

            if(flag == true){
```

Also in this task, one of the most important things I had to be careful of was making sure nothing entered the radius of the castle. To do this I did two things. First, in all the updateStates I wrote a similar for loop that had conditionals that basically asked if the new position a Cell is moving to is within the given radius of a Castle Cell. Here is what that code looked like:



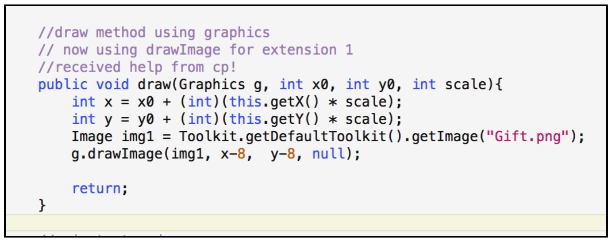
```java
//if the ninja doesnt find food it moves
//but not near a castle
if(found == 0){
    ArrayList<Castle> castles = scape.getCastles();
    this.health --;
    double randX = randomInRange(-5, 5);
    double randY = randomInRange(-5, 5);
    boolean flag = false;
    for(Castle c : castles){
        if(inRadius(this.x + randX,
                    this.y + randY, c.getX(), c.getY(), 15)){
            flag = true;
        }
    }

    if(flag == true){
        return;
    }
    else{
        this.x += randX;
        this.y += randY;
    }
    if (this.health == 0){
        scape.removeAgent(this);
    }
}
```

Then, secondly I created a cleanUp method that removes any initial cells that are created within the radius of a castle. Here is that code:



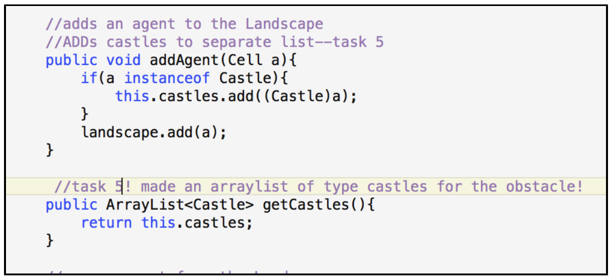
```java
//this method removes initial cells from the beginning of the programs
//that are within the radius of a castle
//help from CP majgaard
public void cleanUp(){
    ArrayList<Castle> castles = this.getCastles();
    for(Cell x : landscape){
        if(x instanceof Ninja || x instanceof DragonBall || x instanceof Gift){
            for(Castle c : castles){
                if(inRadius(x.getX(), x.getY(),c.getX(), c.getY(), 15)){
                    this.removeAgent(x);
                }
            }
        }
    }
}
```

Task 4. This task asked me to make sure each of my individual classes could draw itself uniquely. Here I completed Extension 1 by learning to use PNG Images and updating my draw method. I had to research how to use Images in java and how to resize my icon .pngs. Here is a snippet of my new draw method:



```java
//draw method using graphics
// now using drawImage for extension 1
//received help from cp!
public void draw(Graphics g, int x0, int y0, int scale){
    int x = x0 + (int)(this.getX() * scale);
    int y = y0 + (int)(this.getY() * scale);
    Image img1 = Toolkit.getDefaultToolkit().getImage("Gift.png");
    g.drawImage(img1, x-8,  y-8, null);

    return;
}
```

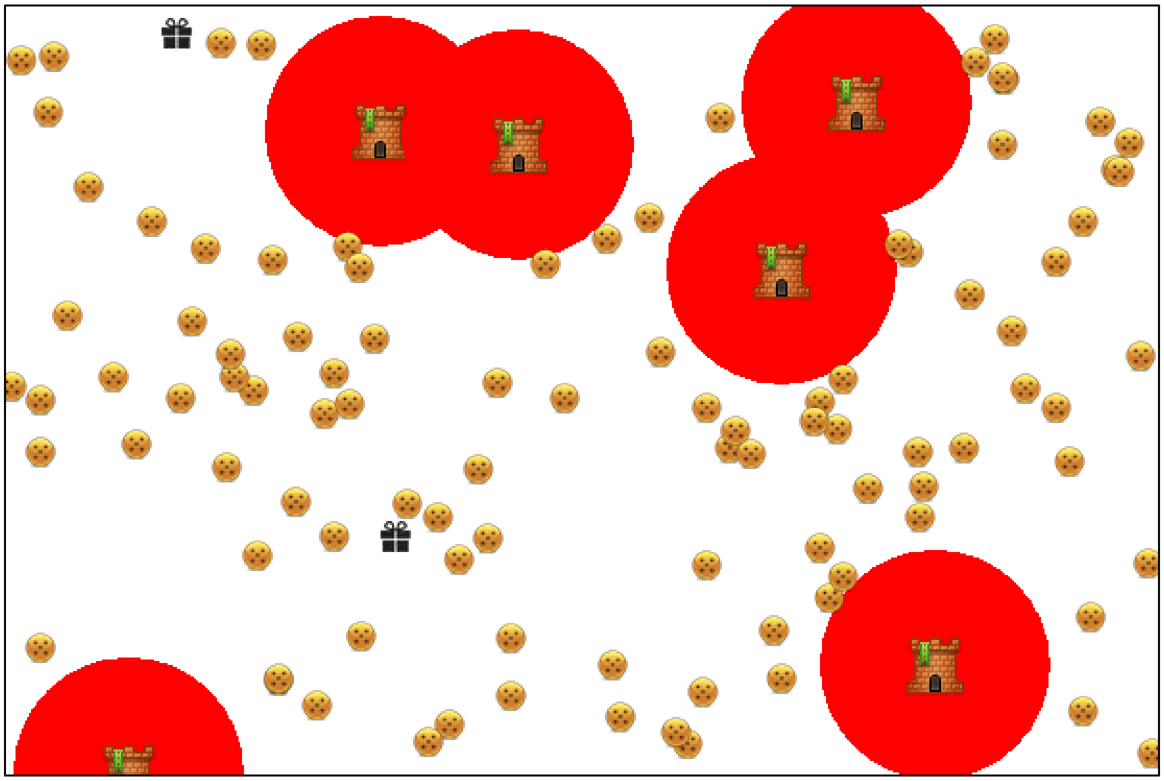
Task 5. For this task, we were asked to make sure we updated our Landscape class regularly. We got a suggestion to create a separate list in Landscape for obstacles. I did this. Here is the code:



```java
//adds an agent to the Landscape
//ADDs castles to separate list--task 5
public void addAgent(Cell a){
    if(a instanceof Castle){
        this.castles.add((Castle)a);
    }
    landscape.add(a);
}

 //task 5! made an arraylist of type castles for the obstacle!
public ArrayList<Castle> getCastles(){
    return this.castles;
}
```

Task 6 & 7. I sort of combined these two tasks. Instead of writing a initialization method within my Landscape class, I simply made the initialization of the beginning landscape within the main method of Landscape display.

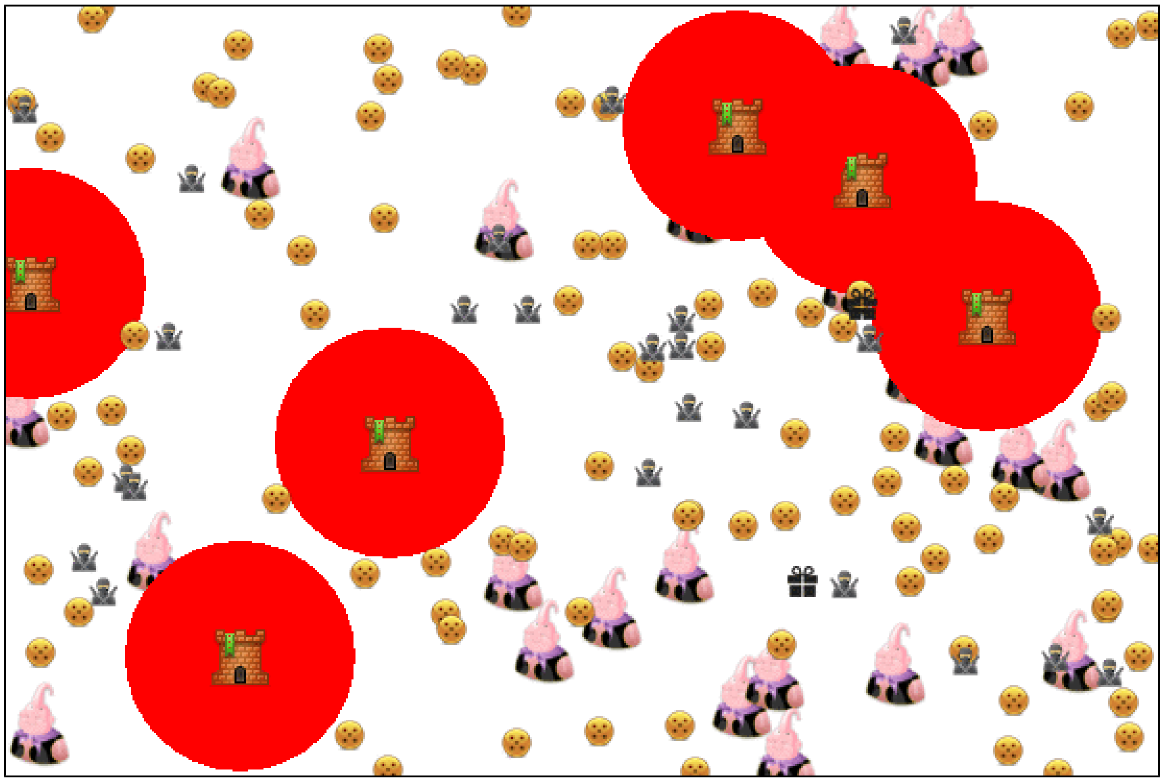
Task 8. For this task, we are asked to run the simulation 100 times and include a representation of our simulation. Here is a GIF of my simulation:



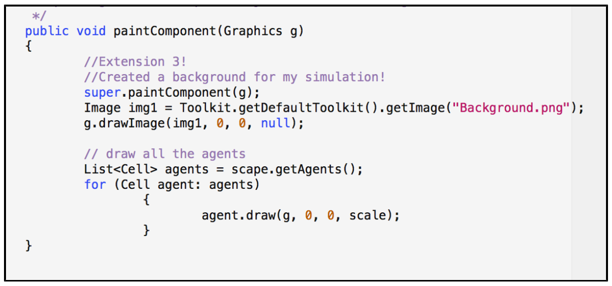
Task 9. I had to experiment a lot with initial values. At first, i was creating way too many Ninjas to start and all the DragonBalls were disappearing. Then, I had the opposite problem where I was starting with too many DragonBalls and the Landscape didn't look right. Then, I had to play with the radius of each individual Cells isNeighbor function. At first, the Ninjas were looking for food in an extremely small radius and dying off completely! Not good. But after playing with radius values, initial population values, and updateState percentages, I found a suitable simulation.

Extension 1. I described this extension earlier in the Project. I used creative icons for an interesting visualization in my simulation.

Extension 2. For this extension, I created another type of Predator named BOO! He eats everything but castles and cannot die. He has a very special updateState and quickly takes over and dominates and entire population. Here is a GIF of his behavior:



Extension 3. For this extension I added a background to my simulation. Here is the code in LandscapeDisplay:



```
 */
public void paintComponent(Graphics g)
{
        //Extension 3!
        //Created a background for my simulation!
        super.paintComponent(g);
        Image img1 = Toolkit.getDefaultToolkit().getImage("Background.png");
        g.drawImage(img1, 0, 0, null);

        // draw all the agents
        List<Cell> agents = scape.getAgents();
        for (Cell agent: agents)
                {
                        agent.draw(g, 0, 0, scale);
                }
}
```

What I learned: I became more proficient with using the same type of Cell and Landscape interaction from the past few projects. Also, I learned a lot from this project becasue I was doing a lot of it without much strict guidance or rules. It was a "create your own" type project and it really forced me to learn it well. I became better at visualizing and planning updateState methods. Also, I learned more about graphics and how to draw images. Next, I learned how to properly implement a doubly LinkedList. In the end, this project was a great finale to a number of weeks of cell and landscape type projects and the predator/prey theme of life was great.

Who helped me: I received help and guidance from CP Majgaard. I worked alongside CS231 classmate Steve Parrott.