For office use only
T1 _____
T2 _____
T3 _____
T4 _____

For office use only
F1 _____
F2 _____
F3 _____
F4 _____

2017

20th Annual High School Mathematical Contest in Modeling (HiMCM) Summary Sheet
(Please make this the first page of your electronic Solution Paper.)

Team Control Number: 8478
Problem Chosen: A

Please paste or type a summary of your results on this page. Please remember not to include the name of your school, advisor, or team members on this page.

Within the 36 hours, we constructed a model for an outdoor aerial light show that will organize 272 drones into the formations of a Ferris wheel, dragon, and a firework display. By mathematically analyzing the flight path of the drones, and taking external factors such as regulation, collisions, launch area, and viewer perception into account, we formulated an efficient model for the light show.

In both the Ferris wheel and the dragon design, we chose to make most of our models by spacing the drones along lines and curves. The Ferris wheel was a combination of line segments and concentric circles and the dragon used reference points spaced along a sine curve. Both used parametric equations to calculate the movement of the drones when animating the designs. The third design started with drones clustered in the center and used both defined and randomly-generated angles to create a visually appealing scatter firework effect with a less complex model.

We also considered launch area, air space, display angle, safety considerations and regulations alongside the duration of the show to provide an optimum viewing experience and practicality for the city. We determined that we would require a 333 ft x 333 ft x 30 ft air space and 920 ft^2 of launch area. Finally, we considered possible improvements we could make on our model to reduce the number of assumptions required and more comprehensively model the show. With these animations as the models of our light show, we will be able to effectively carry out this event.

PROBLEM A: DRONE CLUSTERS AS LIGHT DISPLAYS

Table of Contents:

Restatement of the Problem:

The integration of drones for light shows, alongside modes of delivery, have seen soaring popularity with modernity: both literally and physically. With the employment of drones in events like the Superbowl, and Intel light shows, drones' uses have widespread implications that are quickly gaining momentum. In an order to implement these drones in light shows, adherence to regulations alongside quantifying and monitoring the path for a set number of drones is crucial.

For an outdoor aerial light show that will display a Ferris wheel, a dragon, and another unique image, various factors need to be taken into account including the required airspace, launch site, safety considerations, and flight path.

Our objective is to mathematically delineate the flight path for the drones and form the desired figures in the sky. We plan to achieve this goal by quantitatively analyzing the shape of the displays while calculating their path. By investigating these formations, we hope to have an efficient model for drone light shows that adhere to the standards and regulations.

Assumptions and Justifications:

1. Drones act as a point source or single pixels of light.

- Each drone has one light, and all are far enough away from the audience so that only the light on each drone is visible. In the Intel video, they used this technique to model the placement of the drones as shown in Figure 2.

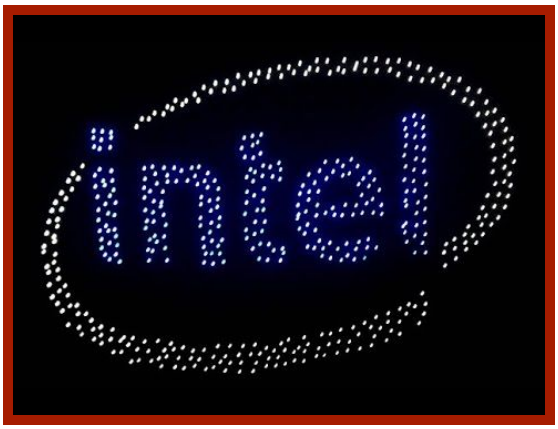


Figure 1. The allotted distance from the viewer demonstrates the miniscule nature of the drones in the sky from a set distance: the drones resemble points on a plane.

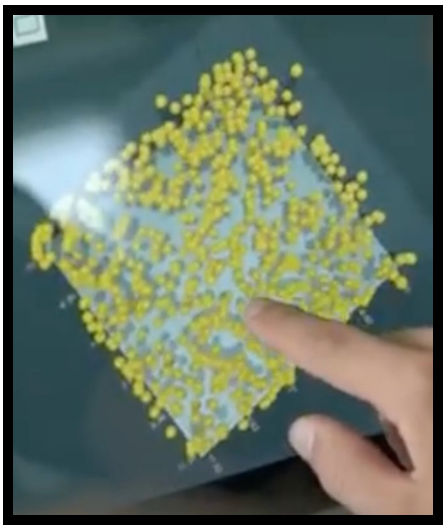


Figure 2. Intel in their "Intel's 500 Drone Light Show Video," showcased the technique of marking the drones as points, or pixels.

2. No drones lose connection with their operator or lose battery during the display.

- For the purpose of our model, no drones fail during the display because our model does not account for filling in gaps. Intel drones are equipped with a GPS to return to their starting point if they fail so this assumption would not harm the safety of the audience if it was incorrect. We also assume that the drones will remain in range while in the intended performance area. The drones also return to their starting point if they go out of range or lose connection.

3. Drones do not veer off course or collide with each other if their paths do not cross.
    - For the purpose of our model, drones operate in their intended manner. While we do take into account their proximity by making sure drones are at minimum 2 feet apart to account for their size, their proximity does not otherwise affect performance or course.
    - For the 3rd design only, randomly determined directions sometimes cause the drones to cross. This iteration of our model is not robust enough to account for this eventuality, so we assume drones do not collide at all in this design only.

4. The audience will be at a location from where they can watch the display as it was meant to be seen (i.e. viewers will be able to see a 2D display as if watching it on a screen).
    - This allows us to simplify our models by calculating 2D positions and paths while allowing the entire audience to experience the full show.

5. Drones can display any color of light.
    - Intel's drones have an LED that can display 4 billion color combinations. We assume our drones will have these same features.

6. No obstructions will be made to the drone's path, including but not limited to airplanes, trees, wind, or weather.
    - Drones are not be permitted in a 5-mile radius from airports and public areas such as national parks. Our light show will not account for any external factors in the air including avian wildlife.
    - The drones will be placed and moved as if there is no wind or weather interference, which could disorient the drone. We assume wind and weather (such as climate, rain, storms, etc.) are not factors. This allows us to focus more clearly on the problem, because both wind and weather are relatively random natural phenomena that we cannot accurately predict and implement. Considering its effect would distract us from creating our models.

7. There is no disturbance to the visibility of the drones during the light show. The show is performed on a clear night for the optimal viewing experience and the distance of the drones do not put them out of visible range.

- We assume that the launch site will be far enough away from other lights to prevent them from obstructing the display. This is so that the formation of drones that we account for will not be hindered in any way by other external lights, which would detract from our show.

8. There is already available land for viewing of the show
    - In order to have an accurate estimation for the amount of land that will be used for the launch site, we will assume that there will be ample land to view the drone light show. While we will factor the viewing site into the viewer's perspective of the light show, the amount of land for the attendees will be presumed to be sufficient and apart from the drone launch area.
9. FAA clearance is obtained
    - As Intel obtained special permission from the Federal Aviation Administration to launch the drone swarm, we have assumed that we can attain this same clearance because we use fewer drones in our display. Our embodiment should not be hindered by the inability to attain this, and would distract from the implementation of our drones for the light show.
10. We get permission to use the land
    - An additional assumption is that we would have attained permission to use the land for the drone show. As we would need a large open space, for both the viewers and the launch, we would want to ensure that we have procured access to this space. In order for everything to run according to plan, a non-disrupted area to initiate the show is an utmost requirement.

Part I: Models and Solutions

Using our assumption that drones act as point sources or single pixels, we created our models by defining and transforming the coordinate location of each drone. We assumed that drones operate exactly as intended with no external interference, so we only accounted for the size of the drones when determining the scale and distance between the drones for the first 2 models. We assumed that all viewers would be at a location where they could view the show as if projected on a screen (i.e. would be properly located to view a 2D display). We assumed that drones could display any color, so the colors in our third design could be arbitrarily chosen.

We simulated each of our models using the graphics.py module in Python. For the purposes of our model, each pixel is equal to 0.5 feet for all designs. All angles are in radians. The coordinate system of graphics.py places the point (0,0) in the top left corner with x increasing towards the right and y increasing downwards.

Ferris Wheel Model:

Variables:

x_o (x_o in code): initial x-value in pixels (of each unit)

y_o (y_o in code): initial y-value in pixels (of each unit)

For each spoke:

θ_o (O_o in code): initial angle in radians (of each spoke)

t: time in radians (used for parametric equations)

Solution:

In the code, we used the center of the concentric circles as the reference point. However, when designing the model, we used the end of the left leg of the Ferris wheel as the origin. Initially we planned our model in Desmos, and this draft can be seen in the Figure 3. The Ferris wheel composed of 2 circles, 8 spokes, 2 legs, and 8 carriages. The circles were represented on Desmos with an equation of $(x-25)^2+(y-44)^2=484$ and

$(x-25)^2+(y-44)^2=9$. The "legs" of the Ferris wheel were represented by two line segments. The line segments of our model have a linear equation of $y=2x$ with a domain of [0,22] and a linear equation of $y=-2x+100$ with a domain of [28,50]. The 8 spokes consist of 4 line segments total. These segments all intercept the center of the circles and the endpoints of the segments all intersect the larger circle. The equations for these line segments are: $y=44$ with a domain of [3,47], $x=25$ with a range of [22,66], $y=x+19$ with a domain of $[25-\frac{22}{\sqrt{2}}, \sqrt{242}+25]$, and $y=-x+69$ with a domain of $[25-\frac{22}{\sqrt{2}}, \sqrt{242}+25]$. We then used this draft to create our Ferris wheel model which can be seen in Figure 4. The z coordinate for each drone is as follows: 1.5 for the drones that make up the inner and outer circles, 0 for the drones that make up the spokes, and -1.5 for the drones that make up the legs and feet. This is to prevent possible drone collisions at points of intersection between the spokes, circles, and legs.

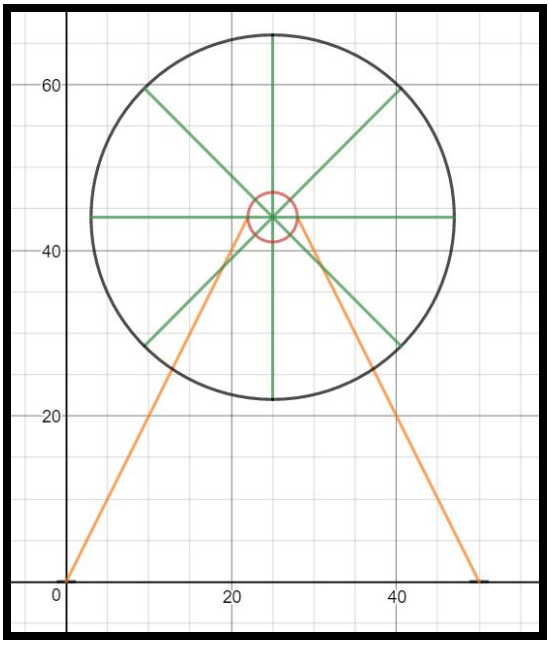


Figure 3. Our rough model of the ferris wheel model.

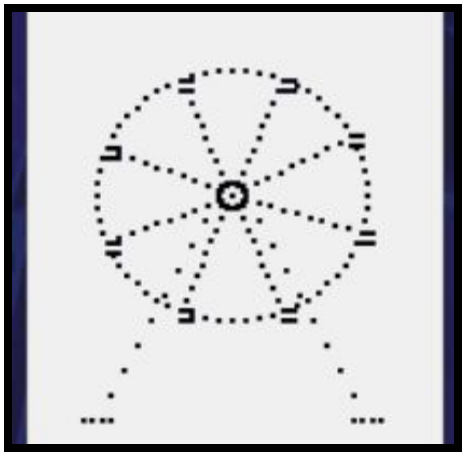


Figure 4. The model we created using Figure 3 as a plan.

The two circles are concentric and have a radii of approximately 253.96 feet and 34.63 feet. There are a total of 64 drones in the larger circle and 18 drones in the smaller circle. Since the circumference of the circles would be 44π and 6π, the drones for the circles were

spaced accordingly in our model since there must be an integer amount of drones. There are 8 spokes consisting of 8 drones each connecting the smaller and larger circles. Then, there are two legs on the Ferris wheel with "feet" at the end. The "feet" are each 4 drones long. The legs are each 8 drones long. Finally, there are 8 carriages on the Ferris wheel which move around in a circular fashion. These carriages consist of 6 drones. In total, we used 219 drones including a drone added in the center to create this model.

Algorithm:

variables: same as above

parameters: t_int = $\pi/32$ - interval between successive 'frames' of the animation, arbitrarily chosen for smoothness

The code we used to test our model and equations is included in Appendix item C3, and the algorithm is explained here. We used python and the graphics module to display and animate our design.

We used the same base class, or python object, called Group for each group of drones, initialized with the parameters x_o, y_o, xshift, and yshift. Respectively, these are the initial x and y coordinate of the reference point, and the x and y coordinates of the drones relative to the reference point. Each spoke on the Ferris wheel uses a subclass called Spoke that inherits from Group, initialized with the parameters x_o, y_o, O_o, and shift.
A variable t is created to represent time in radians and the current angle of the spoke. Since the spokes follow a circular motion, we used polar coordinates to represent the locations of the drones: shift contains the r coordinates and t is the θ coordinate of the locations of the drones relative to the reference point. Each spoke has a corresponding carriage object, which used the Group class. The carriage uses the spoke's last point as its reference point.

When initialized, the class would display each drone as a black point in the specified location. The drones that make up the circles and legs are stationary (they use the Group

class), while the drones that make up the spokes and carriages follow a circular motion (they use the Spoke class). The Spoke class includes a method called tick to move itself and its corresponding carriage. This method increments t by $\frac{\pi}{16}$ radians, then calculates the polar coordinates of each drone using r coordinates from the initial shift list. The reference point (the center of the wheel) doesn't move, so calculating the new coordinates uses a very similar process as calculating the initial coordinates. The method then converts each coordinate to rectangular, calculates the difference between the new and current coordinates, and moves each point to their new location.

In our code, we defined a list spokeShift with incremental values (see appendix C) to be used as relative r coordinates for each drone that make up each spoke. We initialized 8 Spoke objects using a reference point (75, 75), angles in $\frac{\pi}{4}$ radian increments, and the list spokeShift. Secondly, we defined 4 lists outerShiftX, outerShiftY, innerShiftX, innerShiftY for the relative locations (see appendix C) of the drones that make up the outer and inner circles, respectively. We then initialized an outer wheel and inner wheel Group object using the reference point (75, 75) and these lists. Finally, we defined 2 lists legShiftX and legShiftY for the relative locations of the drones that make up the legs and feet, and initialized a legs object using the reference point (75, 75) and these lists.

Finally, an infinite loop rotates the wheel by calling each spokes' tick() method every 0.05 seconds.

Dragon Model:

Variables:

x_o (x_o in code): initial x-coordinate in pixels (of each reference point)

θ_o (O_o in code): initial phase of sine wave in radians (of each reference point)

y_o (y_o in code): initial y-coordinate in pixels (of each reference point)

i: integer (used to generate starting points)

t: time (in radians, used for parametric motion)

x : x-coordinate in pixels

y : y-coordinate in pixels

Solution:

We based our dragon design and movement around a parametric sine wave of the general form:

$x = ki$	$x = x_o + kt$
$\theta = ci$	$\theta = \theta_o + ct$
$y = a + bsin(\theta)$	$y = a + bsin(\theta)$
For initial positions and	for movement.

We used i over the range [0, 28) and the constant parameters k, a, b, and c as follows:

k=10px a=200px b=20px c= $-\pi/4$ radians

The constants are in px because the program we used to model our design uses pixels, and we decided that each pixel is equal to 0.5 feet in the real world for the purpose of our model.

This created a sine wave with amplitude 20, rest position of 200, and period 80, a representation of which is shown below:

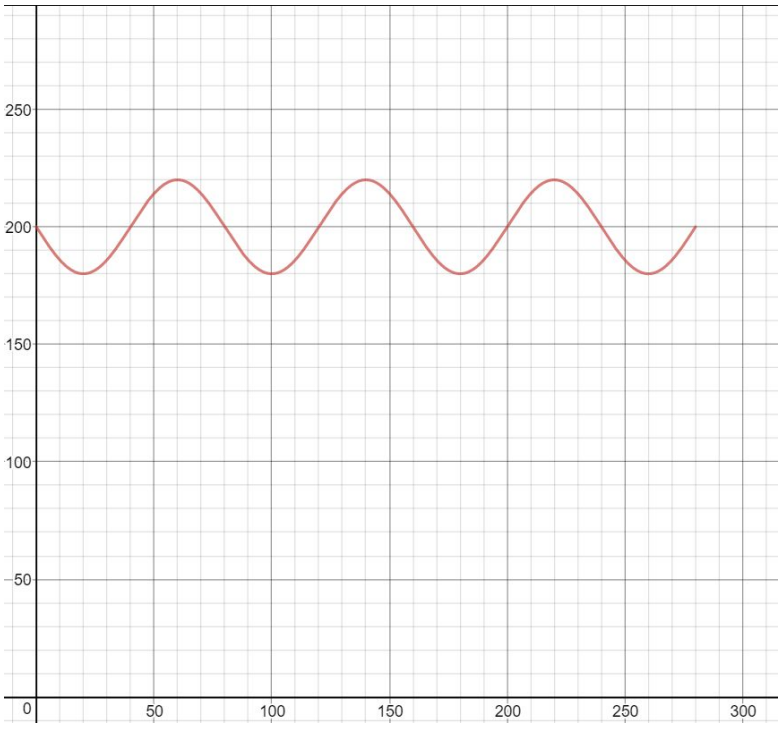


Figure 5. Sine wave with above parameters.

The design uses a total of 136 drones. Because we used many more drones in previous designs, we decided that we could make 2 dragons using a total of 272 drones. The second dragon would fly below the first and use the same equations with a shift in the y-value. We calculated 28 initial reference points using the integers from 0 to 27, the coordinates and phase of which are included in Appendix item D.2.

The body of the dragon consists of 97 drones in 27 vertical columns as follows; the columns with the least drones will be further back to create a 'tapered' effect in the tail.

Number of drones in column	# of Columns
4	20
3	4
2	2
1	1

The top point of each body column will serve as a reference point located on the sine curve. Their initial phases and positions are determined by the first 27 reference points calculated using integer i from 0 to 26. The rest of each column is located directly under the reference point, each drone 7 pixels under the one above it. The z-coordinate of each column is determined as follows to prevent collisions between columns:

$$z = \{0 \, for \, even \, i, \; 1.5 \, for \, odd \, i\}$$

The head consists of 39 drones. Its reference point is in the neck and its initial location is determined by the 28th integer value of i, or 27, to be $(270, \; 200 - 10\sqrt{2})$. The coordinates of the other points were determined by sketching the design in MS Paint, then scaling the relative coordinates down by a scale of 5. The exact and relative coordinates of each point of the head are in Appendix item D.1, and the sketch is included here:

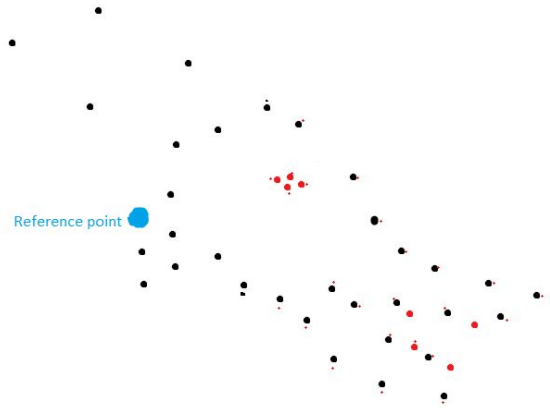


Figure 6. Sketch of dragon's head and reference point. Points in red will light up a different color than points in black to contrast features.

The movement of each reference point is determined by the following parametric equations as stated above ($x = x_o + kt$ $\theta = \theta_o + ct$ $y = a + bsin(\theta)$ with k=10, c=$-\frac{\pi}{4}$, a=200, and b=20). x_o and θ_o are the initial x-coordinate and phase as determined by the first set of equations. The new locations of each of the 28 reference points are determined by incrementing t by $\frac{\pi}{64}$ every 0.05 seconds, and the other points in each group (either a column of drones or the head group) are moved so that they stay in the same positions relative to their respective reference points.

If we had more time, we would determine equations that could make the dragon design move in a circle so the animation could continue for an extended period of time without having to reverse direction, move backwards, or move excessively slowly. We could also include an animation to move the mouth of the dragon or an animation for shooting fire.

If we were to make the dragon design move in a circle, we could write equations for the polar coordinates $(r,\ \theta)$ of each reference point, similar to the following:

$\theta = kt$

$r = a + bsin(ct)$

K is a constant, representing a constant rotational speed in θ. A is a constant representing the circular equilibrium state of the wave, b is a constant representing amplitude, and c is a constant affecting the period of the wave. We could convert these equations to cartesian form as $(rcos(\theta),\ rsin(\theta))$ if we wanted to continue using graphics.py. However, this would require us to calculate the new coordinates of each point of the dragon's head individually which would require much more complex code or careful consideration to simplify. The body segments could be calculated in a similar method if the non-reference points decreased in r instead of y. Circular motion of our design would likely be easier using a graphics system that allows rotational transformations.

Algorithm:

We used the code in Appendix C to test this design. x_o, y_o, and θ_o are represented as x_o, y_o, and O_o respectively in the program. The algorithm for the code is as follows:

1. Define functions x and y to return x and y coordinates at time t using above equations for motion (given the parameters x_o and y_o).

2. Define class (named 'group') with the parameters x_o, y_o, θ_o, xshift, and yshift. Each instance of this class represents a group of drones that stay in the same position relative to each other. Xshift and yshift represent the x and y coordinates of the drones relative to the reference point.

- When initialized: Start counter t at 0, convert initial coordinates and lists of relative coordinates into a list of points, display points, store first point as reference point rPoint (with shift value 0 for both x and y, representing no difference from the reference).
- Move method (moves group by incrementing t counter): Increment t by $\frac{\pi}{32}$, calculate new x and y coordinate of the reference point using the x and y functions, calculate difference between new coordinates and current coordinates, move all points in group using difference as movement vector.

3. Define lists head_xshift and head_yshift (relative locations of points in head group) using data from Appendix item D.1.

4. Define lists gen_xshift and gen_yshift as [0,0,0,0] and [0,7,14,28] respectively, representing relative locations for up to 4 drones in a column.
5. Populate lists x_start, O_start, and y_start with values generated from the equations for initial position and phase ($x = ki\ \theta = ci\ y = a + bsin(\theta)$) for integers i from 0 to 27 inclusive.
6. Initialize group objects for each body segment and head and add objects to list. The 1-drone body segment takes i-value 0, 2-drone body segments take i-values 1-2, 3-drone body segments take i-values 3-6, 4-drone body segments take i-values 7-27, and the head takes i-value 27.
7. Use move method on each object in list of group objects, wait 0.05 seconds, repeat.

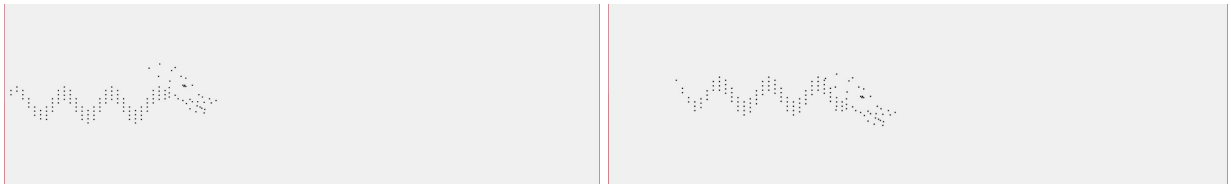


Figure 7. Initial location of the drones and location after some movement.

Firework Model:

Variables:

x_o : initial x-coordinate (center)
y_o : initial y-coordinate (center)
t: time in pixels
θ : angle in radians
i: integer (used to generate starting direction)

Solution:

For our final design, we considered writing the HiMCM logo or our school logo in the sky with drones, but, realizing that writing our school logo would be against the rules, we ultimately settled on creating a firework design. This model is also in the 2D plane. Because we used some random paths in this design, our current model is not robust enough to account for avoiding collisions, so we assumed that drones do not collide and collision avoidance is an improvement we could make in the future.

We based the number of drones in this design around the number used in our previous designs. We had a maximum of 272 drones in our dragon design, so we used 260 drones for this design. All the drones were initially clustered in the center, defined as the point (500, 500) in a 1000 by 1000 px graphics window. Each drone's light is off at the beginning of the show. The drones will be deployed in 13 waves of 20, each wave consisting of 5 groups of 4 drones. There are 2 phases of motion, the first of which can be modeled by the following equations:

$$\theta = i * \frac{2\pi}{5}$$

$$x = t * cos(\theta)$$

$$y = t * sin(\theta)$$

We used integer i over the range [0, 5) to simulate 5 groups evenly spaced in different directions moving away from the center linearly at 1px/t. t is incremented in 0.05 second units, so the speed is 20px/s. Each wave turns on a yellow light when it is deployed. Waves are deployed every 5 cycles of t, so 4 waves are deployed every second.

The second phase of motion starts once each group reaches a distance of 60 px from the center (t=60). The 4 drones in each cluster break off in random directions (chosen by picking a random number between 0 and 2π) and begin flashing randomly between pink, blue, and green. The equations of motion are the same as those for the first part, but the θ value for each individual drone is a randomly-chosen number.

The drones continue flying until they reach the edge of the performance area, then return to their launchpad. To program this, we can set the range of the drones as the area of the performance area, and once they reach the edge, drones that go out of range are programmed to return to their home point, or launchpad.

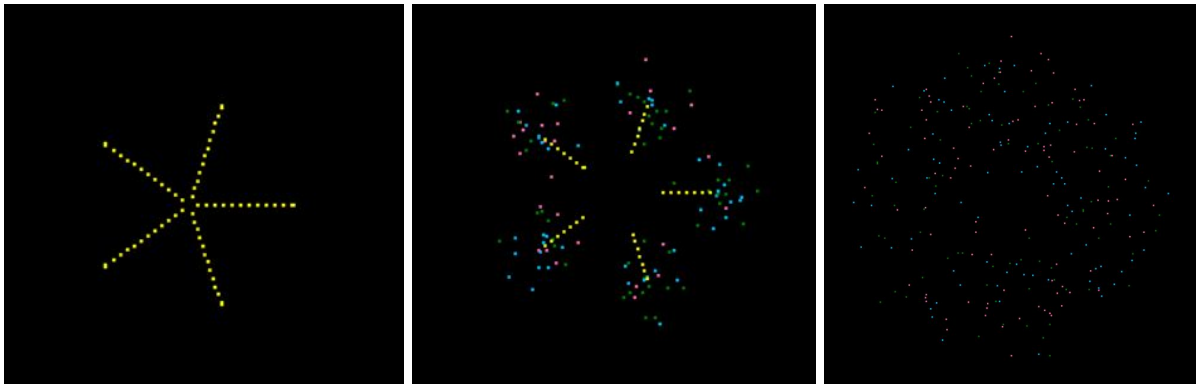


Figure 8. The groups of drones start out moving away from the center like spokes of a wheel, split off in random directions, then disperse. Background is black so different colors are visible.

Algorithm:

x: x-component of movement vector

y: y-component of movement vector

The code we used to test this model is in Appendix E.2. We altered the equations to calculate movement recursively to simplify the program. The algorithm is as follows:

1. Define a list of directions dirs for the first movement (before the groups split) using the equation $\frac{2\pi}{5} * i$ for integer i from 0 to 4.
2. Define class (named 'wave') to represent each wave of drones
    - When initialized: draw 20 points representing drones clustered at the center point (500, 500), randomly populate list dirs2 with 20 values between 0 and 2π (directions for each drone after splitting from group), set t = 0 (counter variable for object), set color of points to yellow.
    - Move1 method (first phase of movement): For each of 5 groups: fetch direction from list dirs, calculate x and y using cosine and sine of direction, move all drones in group using movement vector, increment t counter
    - Move2 method (second phase of movement): Define list colors including hot pink, deep sky blue, and green. For each drone: set color of point to random color from colors, fetch direction from list dirs2, calculate movement vector using cosine and

sine of direction, move drone using movement vector (repeat for all 20 in wave using different directions). T is no longer incremented because it is only used to determine when to switch from phase 1 to phase 2.

3. Create empty list initList of initialized wave objects.
4. Initialize wave object, add to list
5. For all objects in initList, use either move1 method (t of object <= 60) or move2 method (t of object > 60) to move all drones 5 times with 0.05 second delay between moves.
6. Repeat 4 and 5 until 13 waves are initialized. No more waves are initialized after this.
7. Use move1 or move2 methods (same logic as above) on all objects in initList, wait 0.05 seconds, repeat.

Part II: Requirements and Considerations:

In order for the 3-display light show to perform optimally, we had to address a variety of conditions including the amount of drones we would implement, the launch area, the amount of air space, the duration of the light show, and important safety considerations.

For the best viewer experience, we needed to tilt our displays so that they can best be seen from the ground. Because all of our designs are modeled as mainly 2D displays with only some variation in the third dimension, they can be treated for this purpose as 2D planes. The FAA regulations prohibit drones from flying over 400 up and our displays are 333 ft high, so the middle of the display has to be somewhere from around 170 ft to 230 ft high. We chose to use 200 ft for our purposes. We calculated display angles with vertical as 90 degrees for various viewer distances so that the viewer's' line of sight to the middle of the display would be perpendicular to the display angle as follows:
(90 - arctan(200/dist))

Viewer Distance (ft)	**Display Angle (°)**
50	14°
100	27°
150	37°

200	45°
250	51°
300	56°

For the amount of drones used for the show, we took two main factors into account, including current regulations on quantities of drones, along with the required drones to have an accurate depiction of a dragon, a Ferris wheel, and a firework display. Seeing as Intel required specialized clearance in an order to implement their designs for their 500 drones light show, it would not be feasible for us to utilize an amount of drones that has never been tested before. We simply would not have the resources/time to be able to negotiate this value.

Furthermore, our designs of a dragon, a Ferris wheel, and a fireworks display utilized a maximum of 272 drones for the body of the figures. With this amount of drones having been utilized in previous light shows, it is not hard to imagine that this number can be used in our circumstance: this magnitude of drones is far from Intel's, and pales in comparison to drone light shows that have been displayed internationally (almost 1,000 drones at once). Additionally, this number of drones is more economically feasible as this equipment can cost up to thousands of dollars, and can be rented to accommodate for this fact.

To launch all of our drones we would need an open, level space that is at least approximately 0.021077891355 acres, or 918.1529 square feet large, according to the average size of a drone (560mm by 560mm). As the drones would need to be monitored on-site with a pair of individuals - per FAA regulations - an additional launching space to observe the drones and initiate them would be required. These are the bare necessities alongside the drones that we would need to have for the light show.

Our required air space will be scaled from the animation window (1000, 1000). Since each drone is approximately 1.8 ft in length, we decided that drones on the same plane (same coordinate in the z-direction) must be at least 2 feet apart to prevent collision.

There are several safety considerations that must be put in place to ensure the light show runs smoothly. Firstly, our embodiment ensures that the drones will not have any technical faults while in-air, however, it is still vital that we pay attention to the proximity between the drones. To address this, we scaled the pixels in our simulated models so that the drones are approximately 3 feet apart and some of them are closer or further back in the z-direction. In the dragon design, we scaled so that each pixel is equal to 0.33 feet.

When considering the air space requirement, we used the NRG Stadium as a maximum size because it has hosted a similar show in the past (Super Bowl 2017). In that show, the drones were flying above the stadium, so we used the stadium floor area to estimate the air space used in that show. According to the NRG Park website, the stadium floor is about 90,000 square feet. After multiplying our 1000 px window by our conversion factor, we get an air space of about 333 ft by 333 ft. Because most of our display is on a 2D plane or intended to be viewed as 2D with a third dimension only to keep drones from getting too close, we only need 30 ft in the third dimension to encompass our launchpad and extra buffer space.

The duration of the drone light show was 3 minutes and 33 seconds, not accounting for animation transitions. As the light show will be relatively short, we may be able to manipulate the delay between frames of our animation which can lengthen or shorten out the duration of the show. The time frames of the light shows are utilized as variables in the Ferris wheel, dragon, and firework code to manipulate the smoothness and duration of the animation. We could also repeat each model's animation multiple times to lengthen the show.

Strengths and Weaknesses of Our Models:

Strengths:

- Our model accurately perceives where each drone is as one pixel in a coordinate grid represents one drone, and the location of the drones as oriented in the sky, is scaled accordingly.
- Our model only requires a number of drones (272) that is within a range that has already been permitted by the FAA. This would alleviate the legal clearance process for our light show.
- We optimized the amount of drones needed to create our model to reduce the cost of renting the drones. Given that drones are highly expensive, with Intel drones costing around 35,000 dollars, this number allows us to still have an enjoyable yet cost-effective show.

Weaknesses:

- Our model did not factor the viewer's angle of the drone show, however, this is an embodiment that could readily be addressed in the future with the rotation of the plane according to the height of the drones above the ground, and their distance from viewers.
- The model fails to account for weather and other extraneous factors that could affect drone flight, which is not entirely accurate as there will inevitably be meterological conditions that could impact the flight and course of the drones.
- The model additionally is limited by its 2-D infrastructure, which can be problematic in terms of potential collisions between the drones, however, we mediated this by assuming the drones would be pre-programmed to avoid this course of nature.
- We didn’t account for the drones returning to their launchpad, which will be required for the closing of the show; however this does not have a direct impact on the outcome of our animation.
- Our model does not address the transition between successive animations, however this can be readily implemented by having them mapped to the points of the next display.

A Letter to the Mayor of our Findings:

Dear Mayor,

Thank you for giving us the opportunity to design an outdoor aerial light show for our great city. We have come up with three possible sky displays, a dragon, a Ferris wheel, and a firework display which utilize a total of 272 drones: this display will last approximately 3 minutes and 33 seconds not accounting for transitions. By displaying traditional designs using modern technology, and finishing with a design that emulates traditional fireworks, we wanted to highlight the transition from past to future that is rapidly gaining momentum.

In order to make this display a success, we will require a sizeable launch area . To ensure the safety of the civilian onlookers, we require at least 920 square feet of land as a launch area. It would be practical to have some more than this minimum amount of space, as per FAA regulations there will need to be multiple people operating the course of the drones. This land is a requirement additional to the viewing area, which will be separated from this part of the launch area. This calculation was derived with the an approximate size of a drone, which is around 560 mm by 560 mm. Just like fireworks, our display must consider the safety of the viewers.

We also require a certain amount of air space to safely display our show. By looking at previous shows, we decided that our 2D designs would use a 333 x 333 foot area. We also need 30 feet in the third dimension as a safety buffer and to encompass our launch area. Our designs are scaled so that each pixel is 0.33 feet.

Our display is careful to take into account the safety of civilians. Though current FAA regulations do not allow us to fly drones over an audience despite the exceptions that allow us to fly multiple drones with one pilot at night, we addressed the proximity between the drones to further prevent collisions that could potentially be harmful to the viewers.

Although our model assumes the drones will not have any technical faults in the air, we took into account the proximity of each drone. In our models, we designed our models so each drone would be 3 feet apart and some drones were placed closer or further back in the z-dimension.

We believe this display is feasible both economically and logistically. Previously, Intel has performed a light show utilizing 500 drones, the most done in the US. Since we only use a maximum of 272 drones, we will be able to obtain clearance from the FAA because we are using an amount of drones that is under the amount that has been used before and we have taken safety into account. In addition, the lowered amount of drones allows us to reduce cost because currently, the cheapest drones on the market sell for around $50 dollars. Though we will likely rent the drones instead of buying them for this performance, this number of drones will allow us to put on a stunning performance without taxing the budget of this city.

As of now the light show, again without transitions, amounts to 3 minutes and 33 seconds. However, if there is a need to lengthen or decrease the time of the show this can be readily done and modeled. We would do this by manipulating the time frames for the Ferris wheel, dragon, and firework models.

While traditional fireworks shows are a classic on special occasions and greatly entertaining to an audience, high-tech alternatives are becoming more and more numerous and feasible to perform every day. We think that performing a drone light show will be an excellent opportunity to showcase this city taking a step into the future and symbolize technological growth in this modern age.

Sincerely,
Team #8478

Appendix A:

Python:

Our models were created through Python.

IMPORTANT:

- All code must be kept in the same folder with the Graphics.py library in a folder called “graphics”.
- All programs were written on Windows machines and some seem to work incorrectly on Apple Macs.

Item A.1: Graphics.py

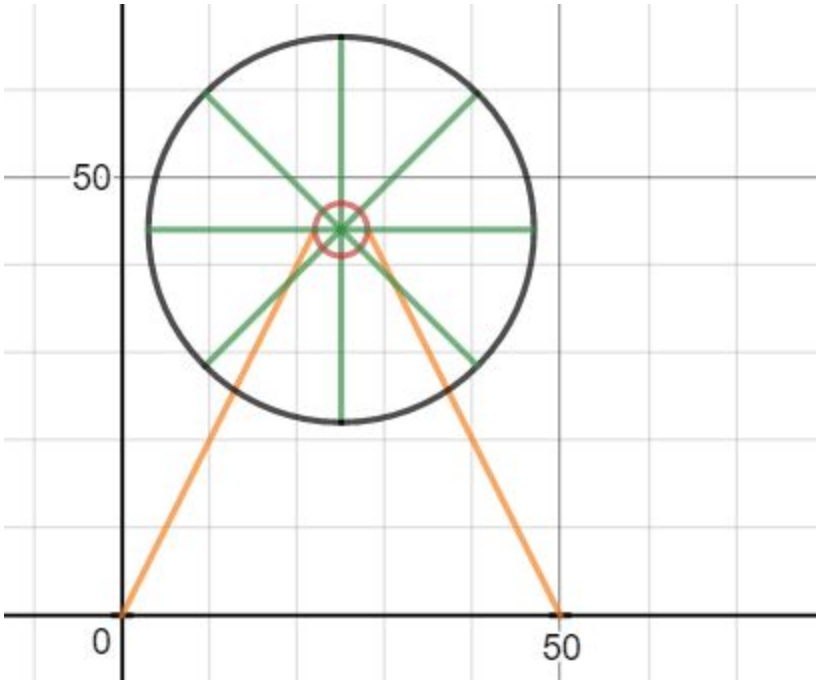
We used a python library known as Graphics.py to visualize our models. Documentation and installation for Graphics.py can be found here:
http://mcsp.wartburg.edu/zelle/python/

Appendix B:

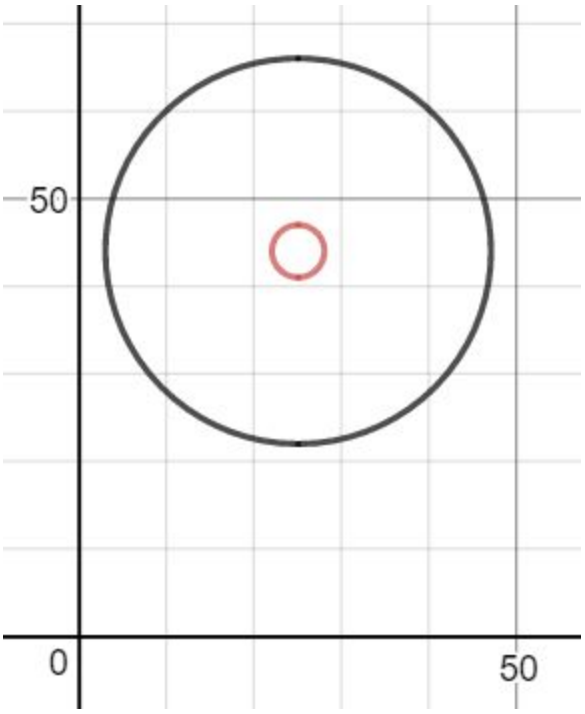
Desmos:

We used the Desmos online graphing calculator to draft the Ferris wheel design.

Item B.1: Ferris Wheel Model



Item B.2: Equations of Circles on Ferris Wheel



1
$$(y-44)^2+(x-25)^2=22^2$$

2
$$(y-44)^2+(x-25)^2=9$$

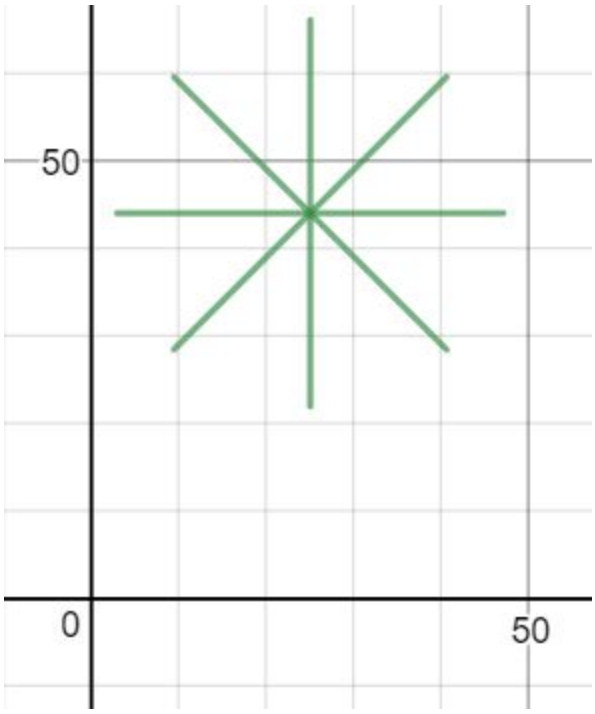
Item B.3: Coordinates of Endpoints of Spokes on Ferris Wheel

x_6	y_6
25	22
25	66

x_5	y_5
3	44
47	44

x_8	y_8
$25 - \frac{22}{\sqrt{2}}$	$44 - \frac{22}{\sqrt{2}}$
$\sqrt{22 \cdot 11} + 25$	$\sqrt{22 \cdot 11} + 44$

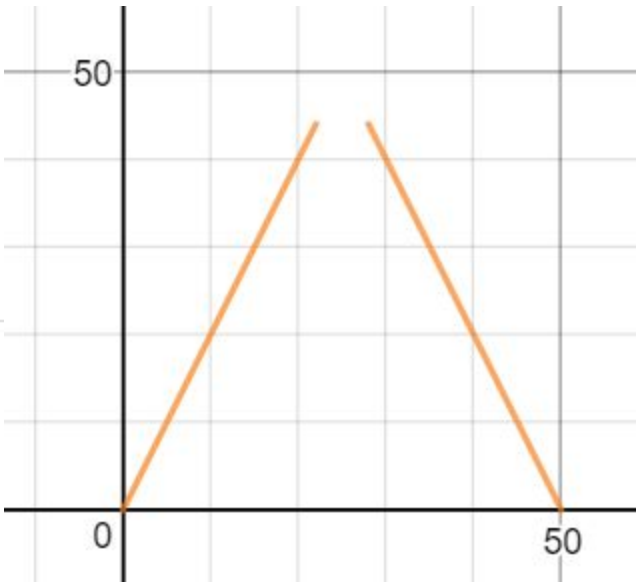
x_9	y_9
$25 - \frac{22}{\sqrt{2}}$	$44 + \frac{22}{\sqrt{2}}$
$\sqrt{22 \cdot 11} + 25$	$44 - \sqrt{22 \cdot 11}$



Item B.4: Coordinates of Endpoints of Legs on Ferris Wheel

x_3	y_3
28	44
50	0

x_1	y_1
0	0
22	44



Appendix C:

Ferris Wheel Model:

Item C.1: Relative and Exact Coordinates of Drones in Spokes and Wheels

Reference Point: (75, 75)			
In these coordinates, the positive y axis extends downwards.			
r coordinates for spokes:	[10, 15, 20, 25, 30, 35, 40, 45]		
theta coordinates for spokes:	[i*pi/4 | i is an integer, 0 <= i <= 7]		
A drone is placed at (75 + r*cos(theta), 75 + r*sin(theta)) for every r and theta from these sets.			
theta coordinates for outer wheel:	[i*pi/32 | i is an integer, 0 <= i <= 63]		

A drone is placed at (75 + 50*cos(theta), 75 + 50*sin(theta)) for every theta from this set.			
theta coordinates for inner wheel:	[i*pi/9 | i is an integer, 0 <= i <= 17]		
A drone is placed at (75 + 5*cos(theta), 75 + 5*sin(theta)) for every theta from this set.			

Item C.2: Relative and Exact Coordinates of Drones in Legs and Feet

Legs & Feet			
X (relative)	Y (relative)	X (absolute)	Y (absolute)
5	0	80	75
10	10	85	85
15	20	90	95
20	30	95	105
25	40	100	115
30	50	105	125
35	60	110	135
40	70	115	145
45	80	120	155
54	90	129	165
52	90	127	165
48	90	123	165
46	90	121	165
-5	0	70	75
-10	10	65	85
-15	20	60	95
-20	30	55	105
-25	40	50	115
-30	50	45	125

-35	60	40	135
-40	70	35	145
-45	80	30	155
-54	90	21	165
-52	90	23	165
-48	90	27	165
-46	90	29	165

Item C.3: Code for Ferris Wheel Model:

```
from graphics import *
from math import sin, cos, pi
import time
#python does math in radians

def main():
   class Group:
      def __init__(self, x_o, y_o, xshift, yshift, win):
         self.x_o = x_o
         self.y_o = y_o

         self.points = [Point(x_o + xshift[i], y_o + yshift[i]) for i in range(len(xshift))]

         for p in self.points:
            p.draw(win)

      def move(self, dx, dy):
         for p in self.points:
            p.move(dx, dy)

   class Spoke(Group):
      def __init__(self, x_o, y_o, O_o, shift, win):
         Group.__init__(self, x_o, y_o, [shift[i]*cos(O_o) for i in range(len(shift))],
                  [shift[i]*sin(O_o) for i in range(len(shift))], win) #converting polar to x-y

         carriageX = [0, 0, 2, 2, 4, 4]
         carriageY = [0, 4, 0, 4, 0, 4]
```

```
        self.carriage = Group(self.points[len(self.points)-1].getX(),
self.points[len(self.points)-1].getY(),
                              carriageX, carriageY, win)
        self.t = 0_o
        self.shift = shift

    def tick(self):
        self.t += pi/32
        self.draw(self.t)

    def draw(self,t):
        points = [Point(self.x_o + self.shift[i]*cos(t), self.y_o + self.shift[i]*sin(t)) for i in
range(len(self.shift))] #first point no shift from x_o, y_o

        for i in range(len(points)):
            dx = points[i].getX() - self.points[i].getX()
            dy = points[i].getY() - self.points[i].getY()

            self.points[i].move(dx,dy)
            if (i == len(points) - 1):
                self.carriage.move(dx,dy)

win = GraphWin('woo wheel', 150, 200)

spokeShift = [5*i for i in range(11)]
spokes = [Spoke(75, 75, i*(pi/4), spokeShift, win) for i in range(8)]

outerShiftX = [50*cos(i*pi/32) for i in range(64)]
outerShiftY = [50*sin(i*pi/32) for i in range(64)]
outer = Group(75, 75, outerShiftX, outerShiftY, win)

innerShiftX = [5*cos(i*pi/32) for i in range(64)]
innerShiftY = [5*sin(i*pi/32) for i in range(64)]
inner = Group(75, 75, innerShiftX, innerShiftY, win)

legShiftX = [5, -5, 10, -10, 15, -15, 20, -20, 25, -25, 30, -30, 35, -35, 40, -40, 45, -45,
        55, 52, 48, 45, -55, -52, -48, -45]
legShiftY = [0, 0, 10, 10, 20, 20, 30, 30, 40, 40, 50, 50, 60, 60, 70, 70, 80, 80, 90, 90,
```

```
            90, 90, 90, 90, 90, 90, 90, 90]
    legs = Group(75, 75, legShiftX, legShiftY, win)

    while True:
        for s in spokes:
            s.tick()
        time.sleep(0.05)
main()
```

Appendix D:

Dragon Model:

Item D.1: Relative and Exact Coordinates of Drones in the Head:

X	Y	Xdif	Ydif	Xscale	Yscale
197	**304**	0	0	0	0
197	340	0	36	0	7.2
197	375	0	71	0	14.2
230	280	33	-24	6.6	-4.8
230	325	33	21	6.6	4.2
230	360	33	56	6.6	11.2
60	116	-137	-188	-27.4	-37.6
144	185	-53	-119	-10.6	-23.8
154	80	-43	-224	-8.6	-44.8
251	135	54	-169	10.8	-33.8
238	225	41	-79	8.2	-15.8
281	109	84	-195	16.8	-39
281	343	84	39	**16.8**	**7.8**
333	186	136	-118	27.2	-23.6
308	377	111	73	22.2	14.6
370	202	173	-102	34.6	-20.4
348	392	151	88	30.2	17.6
474	398	277	94	55.4	18.8
529	407	332	103	66.4	20.6
587	411	390	107	78	21.4
626	390	429	86	85.8	17.2
573	374	376	70	75.2	14
514	359	317	55	63.4	11
480	340	283	36	56.6	7.2
427	261	230	-43	46	-8.6
378	414	181	110	36.2	22
405	382	208	78	41.6	15.6
429	399	232	95	46.4	19
407	458	210	154	42	30.8
465	436	268	132	53.6	26.4
459	483	262	179	52.4	35.8
507	456	310	152	62	30.4
526	497	329	193	65.8	38.6
494	444	297	140	59.4	28
533	465	336	161	67.2	32.2
360	258	163	-46	32.6	-9.2
357	272	160	-32	32	-6.4
374	268	177	-36	35.4	-7.2
347	262	150	-42	30	-8.4

Item D.2: Coordinates of Initial Locations of Reference Points:

i	x	theta (phase)	y	
0	0	0	200	1 drone
1	10	0.785398	214.1421	2 drones
2	20	1.570796	220	
3	30	2.356194	214.1421	3 drones
4	40	3.141593	200	
5	50	3.926991	185.8579	
6	60	4.712389	180	
7	70	5.497787	185.8579	4 drones
8	80	6.283185	200	
9	90	7.068583	214.1421	
10	100	7.853982	220	
11	110	8.63938	214.1421	
12	120	9.424778	200	
13	130	10.21018	185.8579	
14	140	10.99557	180	
15	150	11.78097	185.8579	
16	160	12.56637	200	
17	170	13.35177	214.1421	
18	180	14.13717	220	
19	190	14.92257	214.1421	
20	200	15.70796	200	
21	210	16.49336	185.8579	
22	220	17.27876	180	

23	230	18.06416	185.8579	
24	240	18.84956	200	
25	250	19.63495	214.1421	
26	260	20.42035	220	
27	270	21.20575	214.1421	head

Item D.3: Code for the Dragon Model

```
from graphics import *
from math import sin, pi
import time

#parametric x and y functions
def x(t, x_o):
   return x_o + 10*t

def y(t, O_o):
   return 200 + 20*sin(O_o - t*pi/4)

def main():
   win = GraphWin('Kawaii Dragon', 1000, 1000)

   class group: #class used to draw and move group of points
      def __init__(self, x_o, y_o, O_o, xshift, yshift): #parameters initial x, y, phase of reference point, list of point coordinates relative to reference point including reference point
         self.t = 0 #initialize time as 0
         self.x_o = x_o
         self.y_o = y_o
         self.O_o = O_o

         self.points = [Point(x_o + xshift[i], y_o + yshift[i]) for i in range(len(xshift))] #generate list of points by adding shift to reference value

         #first point in lists of shift coordinates is the reference point, with 0 shift in x and 0 in y
         self.rPoint = self.points[0] #define reference point for movements
```

```
        for p in self.points:
            p.draw(win) #display all points

    def move(self):
        self.t += pi/32 #increment time by pi/64
        x_new = x(self.t, self.x_o)
        y_new = y(self.t, self.O_o)

        xincr = x_new - self.rPoint.getX()
        yincr = y_new - self.rPoint.getY() #determine movement vectors to reach coordinates
at new time

        for p in self.points:
            p.move(xincr, yincr)

  head_xshift = [0, 0, 0, 6.6, 6.6, 6.6, -27.4, -10.6, -8.6, 10.8, 8.2, 16.8, 16.8, 27.2, 22.2,
        34.6, 30.2, 55.4, 66.4, 78, 85.8, 75.2, 63.4, 56.6, 46, 36.2, 41.6, 46.4, 42,
        53.6, 52.4, 62, 65.8, 59.4, 67.2, 32.6, 32, 35.4, 30]
  head_yshift = [0, 7.2, 14.2, -4.8, 4.2, 11.2, -37.6, -23.8, -44.8, -33.8, -15.8, -39, 7.8,
        -23.6, 14.6, -20.4, 17.6, 18.8, 20.6, 21.4, 17.2, 14, 11, 7.2, -8.6, 22, 15.6,
        19, 30.8, 26.4, 35.8, 30.4, 38.6, 28, 32.2, -9.2, -6.4, -7.2, -8.4]#coordinates for dragon
head
  gen_xshift = [0,0,0,0]
  gen_yshift = [0,7,14,21] #general coordinates for body segments: use first n values for
segment of n drones

  x_start = [10*i for i in range(28)]
  O_start = [-i*pi/4 for i in range(28)]
  y_start = [200 + 20*sin(O_start[i]) for i in range(28)] #define list of 28 starting
coordinates and phases on sine curve

  dragon = [] #initialize list of group objects

  dragon.append(group(x_start[0], y_start[0], O_start[0], gen_xshift[:1], gen_yshift[:1]))
#add 1-drone tail segment
  for i in range(1,3): #add 2 2-drone tail segments
    dragon.append(group(x_start[i], y_start[i], O_start[i], gen_xshift[:2], gen_yshift[:2]))
  for i in range(3,7): #add 4 3-drone tail segments
    dragon.append(group(x_start[i], y_start[i], O_start[i], gen_xshift[:3], gen_yshift[:3]))
```

```
    for i in range(7,27): #add 20 4-drone tail segments
        dragon.append(group(x_start[i], y_start[i], O_start[i], gen_xshift, gen_yshift))
    dragon.append(group(x_start[27], y_start[27], O_start[27], head_xshift, head_yshift))
#add head

    while True:
        for d in dragon:
            d.move()
        time.sleep(0.05)

main()
```

Appendix E:

Fireworks Model:

Item E.1: Relative and Exact Coordinates of Drones in Fireworks Model:

Item E.2: Code for Fireworks Model:

```
from graphics import *
from math import cos, sin, pi
import time
from random import randrange, choice


def main():
    win = GraphWin('Firework (/◕ワ◕)/*:･ﾟ✧', 1000, 1000)
    win.setBackground('black')


    dirs = [2*i*pi/5 for i in range(5)]


    class wave:
        def __init__(self):
            self.points = [Point(500,500) for i in range(20)]
            self.dirs2 = [randrange(0,360)*(pi/180) for i in range(20)]
            for p in self.points:
```

```
            p.draw(win)
        self.t = 0
        for i in self.points:
            i.setOutline('yellow')

    def move1(self): #move in clusters of 5 after initial burst
        for i in range(5):
            pdir = dirs[i]
            x = cos(pdir)
            y = sin(pdir)
            for p in range(4*i, 4*i+4):
                self.points[p].move(x,y)
        self.t += 1

    def move2(self): #clusters split off after initial burst
        colors = ['hot pink', 'deep sky blue', 'green']
        for i in range(20):
            self.points[i].setOutline(choice(colors))
            pdir = self.dirs2[i]
            x = cos(pdir)
            y = sin(pdir)
            self.points[i].move(x,y)

initList = [] #list of initialized wave objects
for d in range(13):
    x = wave()
    initList.append(x)
    for i in range(5): #5 iterations before new wave
        for n in initList:
```

```
            if n.t <= 60:
                n.move1()
            else:
                n.move2()
        time.sleep(0.05)
    while True:
        for n in initList:
            if n.t <= 60:
                n.move1()
            else:
                n.move2()
        time.sleep(0.05)

main()
```

Works Cited

Fly for Fun under the Special Rule for Model Aircraft. (2017, July 31). Retrieved November 18, 2017, from https://www.faa.gov/uas/getting_started/fly_for_fun/

M. (2017, February 05). Intel's 500 Drone Light Show! Retrieved November 18, 2017, from https://www.youtube.com/watch?v=jNIAzeU8POQ

"NRG Stadium." *NRG Park*, www.nrgpark.com/nrg-park-facilities/nrg-stadium/.

Stephens, Ric. "Typical Drone Sizes." *Flickr*, Yahoo!, 22 Mar. 2016, www.flickr.com/photos/ricstephens/25365498343/in/photostream/.

Zelle, John M. *Python programming: An Introduction to Computer Science*. Brantford, Ont.: W. Ross MacDonald School Resources Services Library, 2010.