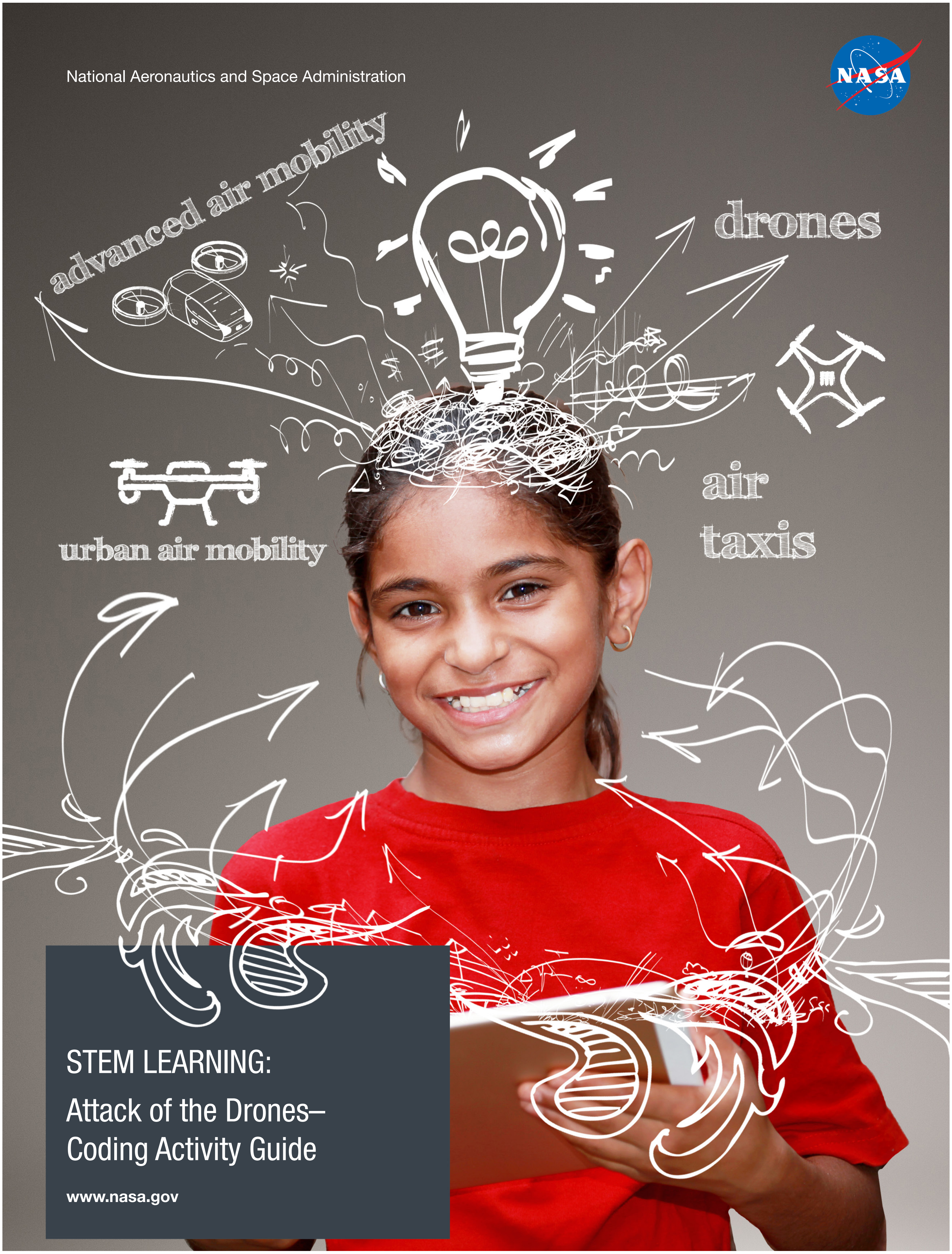
National Aeronautics and Space Administration



STEM LEARNING:

Attack of the Drones–Coding Activity Guide

www.nasa.gov

OVERVIEW

In this activity, students use Scratch to create a side scrolling game in which they pilot a drone through traffic while maintaining an acceptable altitude. The activity guide is set up as a tutorial that takes students through the process of making the game. There are additional features that can be added to further challenge students who are ready for it. The activity engages students in computational thinking and problem solving.

Standards

CCS.MATH.CONTENT.7.EE.B.4: Use variables to represent quantities in a real-world or mathematical problem, and construct simple equations and inequalities to solve problems by reasoning about the quantities.

Next Generation Science Standards

Science and Engineering Practices

- Developing and using models
- Using mathematical and computational thinking

Crosscutting Concepts

- Scale, proportion, and quantity

Materials

- Computer or tablet with internet access (or download all content to run locally)
- Downloadable graphic files (see links at the end of this guide).

Free Scratch account, available at *http://scratch.mit.edu*

Grade Level

5th–12th

Suggested Time

1–3 hours

Management

1. This activity can be used to introduce students to coding or as an enrichment activity after students have learned about the various coding blocks. The "Additional Resources" section contains other NASA activities for practicing block-based programming.
2. Due to the length of this activity guide, it is recommended that students access it electronically to minimize printing requirements.
3. This activity guide takes students step-by-step through the process of creating a side scrolling game using Scratch. This tutorial style activity is appropriate for students with any level of programming experience. Once students get a part of their program running, encourage them to experiment with modifying the code to see what it does. This helps students "see" what various commands and variables do by seeing how they affect the code's performance.
4. After completing parts 1 through 7 of this activity guide, the program will be a fully functional (though simplistic) game. The activity can be considered complete at this point, especially for beginning programmers. The guide continues with four more advanced parts that require more complex code and encourage higher-level thinking. These can be found in parts 8 through 11 of this guide. They are designed so that students can add as few or as many of these features as desired. They do not need to be added in order, so any of these parts can be assigned/used independently. The directions in these more advanced parts are less detailed to challenge the students.
5. As students write their program, they should have it beta tested by another student. Beta testers should look for errors in the program's execution and provide feedback to the programmer.
6. If students complete the task with time remaining, they can be assigned additional higher-level requirements to keep them challenged. Several suggestions can be found in the "Extension Ideas" section of this guide. Optionally, students can come up with their own requirements and change their code to meet these new requirements.

Background Information

NASA is leading the nation to quickly open a new era in air travel called Advanced Air Mobility, or AAM. The vision of AAM is that of a safe, automated, and affordable air transportation system for passengers and cargo in both urban and rural settings.

Vertiports are locations from which AAM aircraft can land or take off. Air taxis transport small groups of passengers from a vertiport near where they are to one located near their destination.

Package delivery drones are capable of delivering a range of products such as food, medical supplies, retail purchases, and more to their final destination. They fly autonomously, meaning they are not piloted while transiting. According to recent NASA-commissioned market studies, by 2030 there will be as many as 500 million flights per year for package delivery services.

In order to safely control aircraft in this new airspace, NASA has helped develop technology to create virtual barriers, or **geofences**, around areas where aircraft aren't allowed to fly. This is important because of the dangers posed to people and equipment by drones or other aircraft flying in these restricted areas. For example, drones flying too close to airports interfere with the safe landing and taking off of the planes.

PART 1 – SET IT UP

1. Download the image file, unzip it, and save it to a computer

2. On the Scratch website (*https://scratch.mit.edu/*), sign in and click "Create" to create a new project. Scratch can also be downloaded and run locally, removing the need for an active internet connection while working with Scratch.

3. In the "Stage" box in the lower right of the screen, place the mouse cursor over the blue button and select "Upload Backdrop." Select the "Backdrop Color" file included in the files that were downloaded.

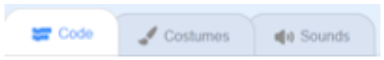
4. Next to the "Stage" box on the screen, there is a box for sprites. There should be a sprite of a cat there by default. **Right click** on the picture of the cat and select "Delete" to remove it.

5. Place the mouse cursor over the blue button in the "Sprite" box and select "Upload Sprite." Select the "Drone" picture that's included in the files that were downloaded.

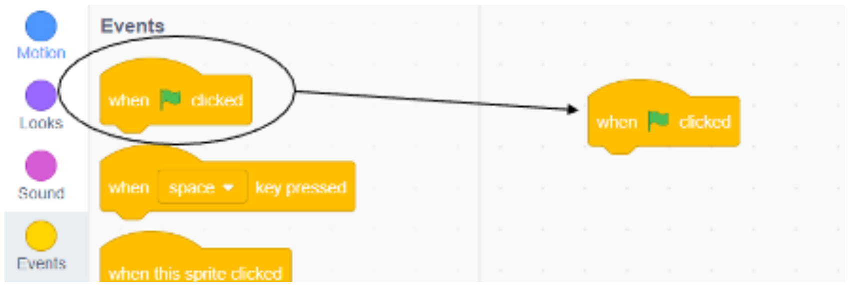
6. Using the same method as described in the previous step, add two more sprites: "Title" and "Begin Button."

7. Once the project is coded, users will start it by clicking on the green flag icon. But, code needs to be added first to tell Scratch what to do.

8. Click on the "Title" sprite in the "Sprite" box so it is selected. In the main box on the left of the screen, there are three tabs shown at the top. Make sure the "Code" tab is selected.



9. Located on the left are the different commands that can be used for coding. Start by clicking on the light orange "Events" button. Then, drag the "when flag clicked" command into the programming space.



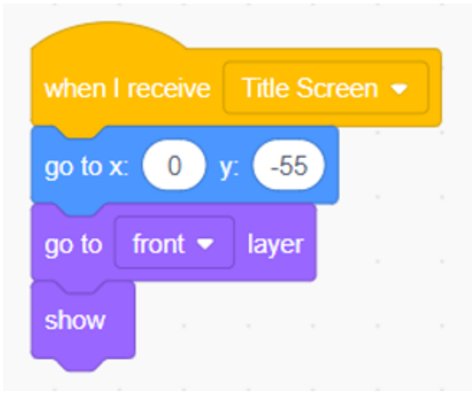
10. Scratch uses a command called "broadcast message 1" to trigger events throughout the code. Drag the "broadcast message 1" command over so it connects to the bottom of the "when flag clicked" command. By default, the message that will be broadcast is called "message 1." To change the name, click on the dropdown menu located in the "broadcast message 1" block. Select "New message" and name the message "Title Screen." Once the game is started, it will broadcast this message to set up the title page.

11. Drag the "when I receive message 1" block into the programming space. Select the dropdown menu button on this command, then select "Title Screen." Anything attached to this block will happen whenever the program broadcasts "Title Screen."

12. Under the “Motion” commands on the left, drag the “go to x: y:” command over and attach it to the bottom of the “when I receive Title Screen” block. Change the x-value to 0 and the y-value to 65. This moves the title screen to its location on the screen.
    Note that the origin, at position (0,0), is located in the center of the backdrop.

13. Under the “Looks” command, drag the “go to front layer” command (you may have to scroll down to find this command) and attach it under the “go to x: y:” command. Under that, attach the “show” command.

14. Select the “Begin Button” sprite in the “Sprite” box to enter the code to run when “Title Screen” is broadcast. The code should move it to position (0, -55), move it to the front layer, and show the sprite.



```
when I receive [Title Screen ▾]
go to x: (0) y: (-55)
go to [front ▾] layer
show
```

15. Select the “Drone” sprite and add similar code to move it to position (0, -10), move it to the front layer, and show the sprite when “Title Screen” is broadcast.

16. This is a good time to test the code. Run the code by clicking on the green flag 🚩 icon. If everything has been coded correctly, the title, drone, and begin button should line up nicely on the screen.

PART 2 – ADD AND ANIMATE THE GROUND

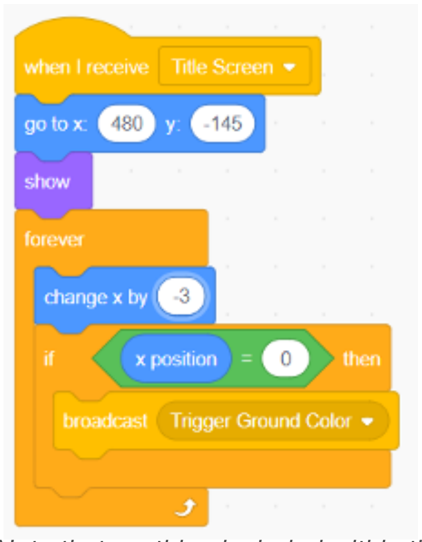
1. The backdrop in Scratch is always 480 pixels (or dots) wide. In order to have the ground scroll across continuously, two copies of the “Ground Color” sprite need to be uploaded in the “Sprite” box. Scratch will automatically change the name of the second ground sprite to “Ground Color2.”

2. Add code for the “Ground Color” sprite to make it move to position (0, -145) and show once “Title Screen” is received.

3. For the “Ground Color2” sprite, code it to move to (480, -145) and show once “Title Screen” is received. This places it off the screen to the right, but that changes once the ground sprites get animated.

4. Click on the green flag icon to test the coding. The ground should be positioned at the bottom of the backdrop.

5. To animate the ground, the ground sprites will move to the left in three pixel increments. Select the “Ground Color” sprite. Under the code that runs when “Title Screen” is received, attach the “forever” command that is found in the “Control” commands. Inside that command, add the “change x by 10” command found in the “Motion” commands. Change the value from 10 to -3. The negative sign moves it to the left on the screen.

   ```
   when I receive [Title Screen ▾]
   go to x: (480) y: (-145)
   show
   forever
       change x by (-3)
       if <(x position) = (0)> then
           broadcast [Trigger Ground Color ▾]
   ```

   Note that anything included within the “forever” command will continue to run until the code stops it.

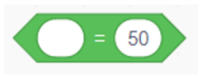
6. Repeat the previous step for the “Ground Color2” sprite.

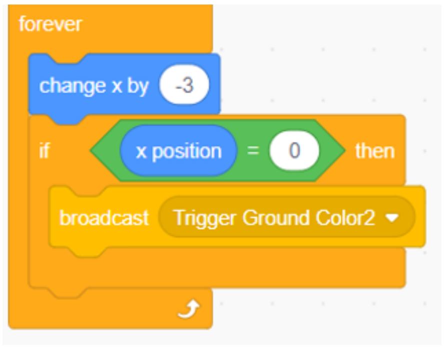
7. Test the code by clicking the green flag icon. The ground should scroll to the left until both ground sprites have gone by. It stops after both sprites have gone by.

8. To keep the ground scrolling, code needs to be added that takes a sprite that has gone off the screen to the left and places it back to the right of the screen. This will be accomplished using an “if...” command.

   Note that an “if...then” command checks to see if something is true. If it is, whatever code is right after the “then” portion of the command runs. If not, the code doesn't run.

9. In the code for the sprite "Ground Color," drag an "if...then" command from under the "Control" commands and attach it under the "change x by -3" command. Next to the word "if" is a blank space. Under the "Operators" commands, drag the "=" command into this space.





Fill the empty oval in the "=" command with "x position" (located in the "Motion" commands) and change the number from 50 to 0. Under the "if..." command, drag over a "broadcast message1" command. Click on the dropdown box on the broadcast command and add a new message named "Trigger Ground Color2."

10. Select the sprite "Ground Color2" and add the same code, except change the name of the message to be broadcast to "Trigger Ground Color."

11. Now the code detects when one of the ground sprites fills the screen. When it does, it broadcasts a message telling the other ground sprite to move to the right of the screen. To actually move the sprite, code needs to be added.

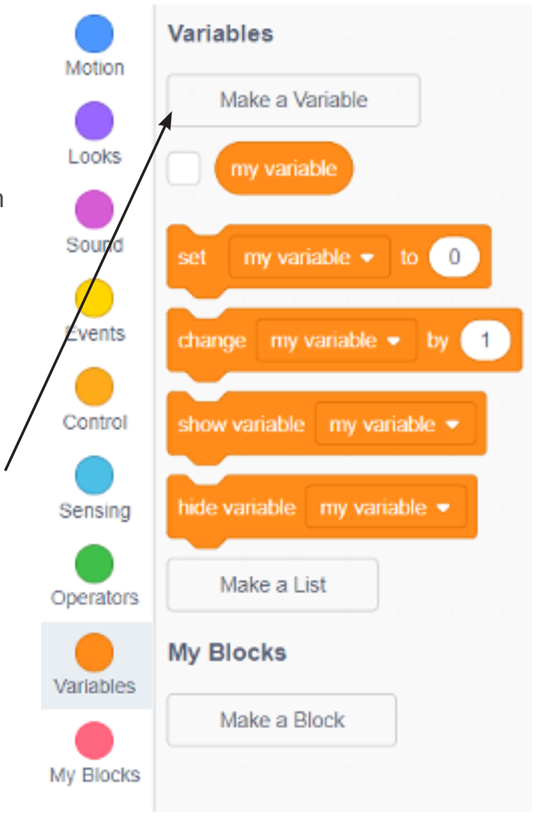
12. Select the "Ground Color" sprite. From the "Events" commands section, drag the "when I receive message1" command into the programming area. Select the dropdown button in this command and change the message to "Trigger Ground Color." Under the "when I receive..." command, add the command to move it to (960, -145).

13. Select the "Ground Color 2" sprite and do the same, except choose "Trigger Ground Color2" in the "when I receive..." command.

14. Click on the green flag icon to test the code. The ground should continually scroll.

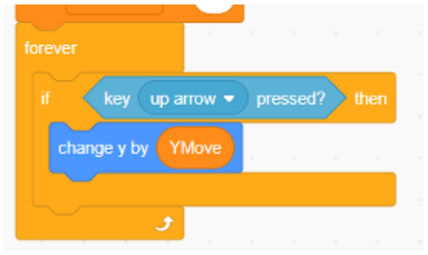
PART 3 – START THE GAME/MOVE THE DRONE

1. When a user clicks the “Click to Begin” button on the title screen, the game should start.
2. Select the “Begin Button” sprite in the “Sprite” box. Drag the “when this sprite clicked...” command from the “Events” commands into the programming area.
3. Once the button is clicked, the button should be removed from the screen and a message should be broadcast to start the game. From the “Looks” commands section, add the “hide” command. Then, add the “broadcast...” command from the “Events” commands. Change it to broadcast a new message and name this message “Start Game.”
4. Select the “Title” sprite and add the code to hide this sprite when the “Start Game” message is received.
5. Click the green flag icon to test the code. When the “Click to Begin” button is clicked, the drone and ground should be the only sprites left on the backdrop.
6. At this point, code can be added to move the drone around while playing the game. To do this, a variable will be used to tell the program how many pixels to move the sprite whenever an arrow key is pressed. A variable is given a name and holds a numerical value to be used in the program.
7. Click on the “Variables” commands and click “Make a Variable.” Name the variable “XMove.” Select the “For all sprites” option and click the “OK” button. Repeat this step and create another variable named “YMove.”
8. Select the “Drone” sprite in the “Sprite” box. Drag the “when I receive message1” command into the programming area and change it to “when I receive Start Game.”
9. In the “Variables” commands there is a command “set my variable to 0.” Drag this under the “when I receive Start Game” command. Change it so that its sets the variable “XMove” to 2. Under that, add the “hide variable my variable” command from the “Variables” commands section and change it so it hides the variable “XMove.” Repeat this step to set the variable “YMove” to 2 and hide it.

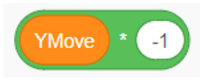


10. Under those, add a “forever” command. Inside this “forever” command is where the code goes to move the drone. There are four arrow keys that can be pushed, so four “if...then” statements need to be added.
11. For vertical (up and down) movement, the code needs to check if the up arrow or the down arrow were pressed. Inside the “forever” command, add an “if...then” command. In the blank of this command, add the “if... key space is pressed” from under the “Sensing” commands. Use the dropdown box to change the “space” in this command to

"up arrow." Inside the "if... then" command, add a "change y by 10" command from the "Motion" commands. Click on the "Variables" commands on the left. Then, drag the "YMove" variable into the "change y by..." command (to replace the 10).



12. Repeat the previous step to add another "if... then" after the previous "if...then." This time, the code checks to see if the "down arrow" is pressed. And, instead of moving the sprite by YMove, it needs to move the same distance in the opposite direction. So, use the multiplication command from the "Operators" commands. Set it to multiply "YMove" by -1, which changes the direction of movement. Place this expression into the oval in the "change y by..." command.

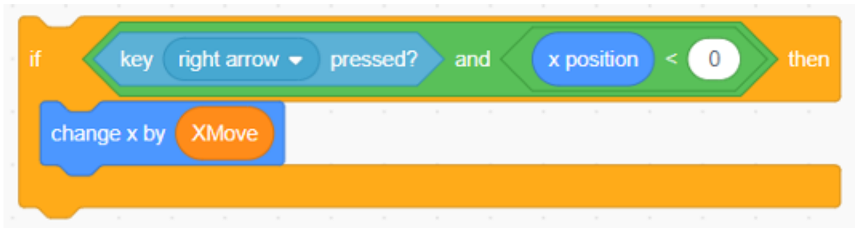


13. This is a good point to test the game. The drone can fly all the way up and all the way down, including through the ground. This will be addressed in part 4.

14. The drone also needs to be able to move horizontally (left and right). The code will be written to keep the drone within the left-hand part of the backdrop. This means that two things must be checked before the drone moves: was a right or left arrow key pushed and is the drone still in the allowable area on the backdrop.

15. To move the drone to the right, add a third "if...then" command after the previous two. Inside the "if...then" command, add a "change x by 10" command and change the 10 to the variable "XMove."

    For what is being checked by the "if… then" command, add an "and" box from the "Operators" command section. In the first empty space, add an "if key right arrow pressed?" command. In the second space, the code to see if the drone is to the left of the center of the screen needs to be entered. This is done with a "<" box from the "Operators" commands section. When the "and" is complete and placed in the "if… then" command, it should look like this:



16. Enter the code to move the drone left. This time, check to see if the left arrow was pressed and if the x-position is greater than -140. If both conditions are true, the drone should move by ("XMove" multiplied by -1).

17. Test the program to see if the drone moves in all four directions. The "XMove" and "YMove" values can be changed from 2 and the code restarted to see how it affects the flight.

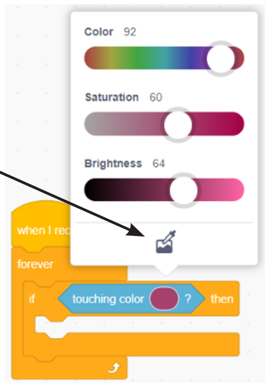
PART 4 – CHECK TO SEE IF THE DRONE IS TOO HIGH OR TOO LOW

1. Per federal regulations, drones can only be operated below 400 feet without a special exemption. In the game, the code needs to continually check the altitude and end the game if any part of the drone goes above 400 feet. The altitude is marked on the backdrop, with the green line representing 400 feet.

2. Select the "Drone" sprite in the "Sprite" box. Another piece of code should be added here that runs when the "Start Game" message is broadcast. Drag the "when I receive..." command into the programming area and set it to "when I receive Start Game." Under it, add a "forever" command so that the code continually checks to see if the drone hits the green line.

3. Add an "if...then" command inside the "forever" command. Drag the "touching color...?" command from the "Sensing" commands section into the empty block of the "if...then" command. Click on the color in the "touching color...?" command and select the eyedropper at the bottom of the pop-up box.

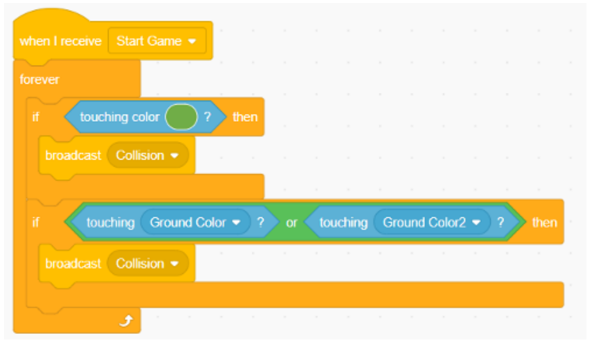
   Move the eyedropper over the backdrop until it is on the 400-foot line. Click it and the "touching color...?" commands now will check to see if the drone touches this line.



4. Inside the "if...then" command, code it to broadcast a new message that should be named "Collision." This message will inform all parts of the code that the drone hit something. In this case, it hit the 400-foot line.

5. Code must also be added to continually check whether the drone has hit the ground. Another "if...then" command needs to be added after the "if touching color...?" command that was just added. Inside the "if...then" command, add the "broadcast Collision" command.

6. To determine whether or not the drone hit the ground, the code will check whether it has touched either the sprite "Ground Color" **or** the sprite "Ground Color2." To check this, use the "or" block located in the "Operators" commands. In each of the spaces, insert the "touching mouse pointer?" command found in the "Sensing" command section. Use the dropdown boxes to change it from "mouse pointer" to "Ground Color" in one and "Ground Color2" in the other.



```
when I receive Start Game
forever
  if touching color ? then
    broadcast Collision
  if touching Ground Color ? or touching Ground Color2 ? then
    broadcast Collision
```

7. For the code to respond to the collision, code must be added. Drag the "when I receive message1" command into the programming area. Change it to "when I receive Collision." Under it, add the "stop all" command found in the "Control" command section.

8. Test the program several times to ensure it stops if the drone flies too high or too low.

PART 5 – ADD OBSTACLES

1. Air taxis are small, electrically powered aircraft that can carry passengers. Flying below 400 feet, they travel between vertiports (locations in a city designed for them to land at and take off from). These aircraft take priority over small drones, so drones must maneuver so that they do not impede the air taxis' flights.

2. In this game, air taxis will fly in at random heights as the game progresses. In the "Sprite" box, add the "Air Taxi" sprite.

3. Code needs to be added to control how this new sprite responds to the different broadcast messages. Start by dragging the "when I receive message1" command into the programming area. Change it so that it responds to "Title Screen" and add the command to hide the sprite. This way, the air taxis don't show up on the title screen.

4. Next, add another "when I receive message 1" command and change it to "when I receive Collision." Under this, add the "stop all" command. Click on the dropdown box to change it to "stop other scripts in sprite." This is necessary so that it stops checking for collisions once one has been detected.

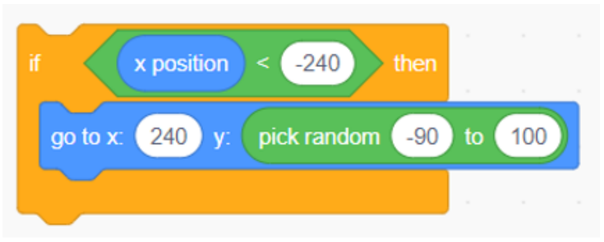
5. Add a third "when I receive message1" command and change it to "when I receive Start Game." From the "Controls" command section, add the "wait 1 second" command. This gives the user a brief pause to get situated before air taxis start appearing.

6. Next, add a "go to x:... y:..." command. Change the x-value to 240, which is the right edge of the backdrop. For the y-value, drag the "pick random 1 to 10" command from the "Operators" command section into the oval. Change the 1 to -90 and the 10 to 100. This places the "Air Taxi" sprite at a random height within the backdrop.

7. Under the "go to x:... y:..." command add the command to show the sprite.

8. Next, code needs to be added to move the air taxi across the screen, detect when it goes off the left-side of the screen, and then send a new air taxi from the right-side of the screen. Since this happens continually, add a "forever" command. In it add a "change x by -5" command to make it move.

9. Also in the "forever" command, add an "if...then" command. Code it to check if the x-position is less than -240 meaning it had gone off the screen. If so, it should move it to an x-value of 240 and a random y-value between -90 and 100.


```
if x position < -240 then
    go to x: 240 y: pick random -90 to 100
```

10. The code needs to also continually check to see if the drone has hit the air taxi. Under the previous "if...then" command, add another "if...then" command. Code it to check if it is touching the "Drone" sprite and, if so, it broadcasts the "Collision" message.

11. Run the program to ensure it is working properly.

PART 6 – KEEP SCORE

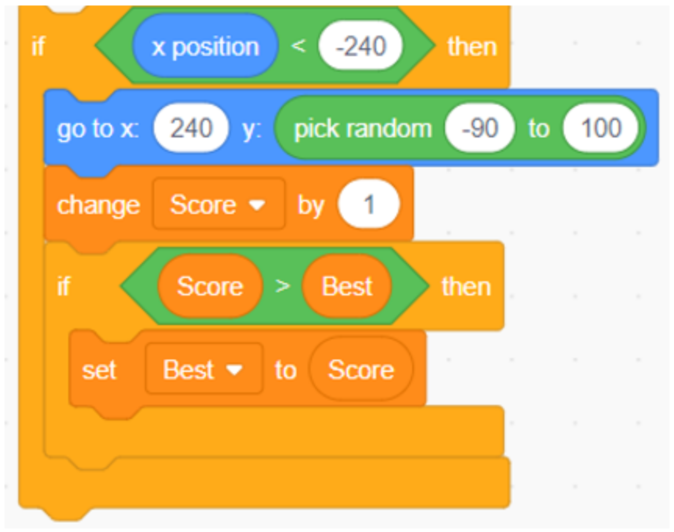
1. An important part of most games is keeping score. For this game, a point will be added every time an obstacle moves all the way across the screen. To keep score, two variables will be added: “Score” will keep track of the score and “Best” will keep track of the high score. In the “Variables” section of the commands, make these two variables. You will see them displayed on the backdrop and can drag them to wherever you like on the backdrop.

2. Select the “Title” sprite in the “Sprite” box. In the code that runs when the green flag icon is clicked, add the command “set Best to 0” (from the “Variables” commands section) before the “broadcast Title Screen” command. This will reset the best score every time the game starts. When the “Try Again” button is added later, it will restart the game without resetting the best score.

3. Under the “when I receive Title Screen” command, add the commands “show variable Score,” “show variable Best,” and “set Score to 0” so that they will be displayed throughout the game and the score will reset when the title screen is displayed.

4. Select the “Air Taxi” sprite. In the code section, find the code that moves the sprite to the right-side of the screen when it exits the left side. Every time this happens, the score should increase by 1. At the same time, the best score needs to be compared to the current score. “If the current score is higher, then the best score should be replaced with the current score.”

5. After the “go to...” command that moves the sprite, add the command “change Score by 1” to increase the score. Then, add an “if...then” command that checks to see if the score is higher than the best score. If so, the best score is updated to reflect the score.



```
if x position < -240 then
    go to x: 240 y: pick random -90 to 100
    change Score by 1
    if Score > Best then
        set Best to Score
```

PART 7 – RESTART GAME

1. Once the game ends, the user needs to receive a message saying that the game is over and giving them the opportunity to try again. For this, two new sprites should be added. In the “Sprite” box unload the sprites “Game Over” and “Try Again.”
2. Select the “Game Over” sprite. This sprite must respond to two messages, “Title Screen” and “Collision.” Add the command “when I receive Title Screen” and add the “hide” command under it.
3. Next, add the “when I receive Collision” command. Under it, add the commands “go to x: 12 y: 75” and “show.”
4. Select the “Try Again” sprite. Add the “when I receive Title Screen” command. Under it, add the “hide” command.
5. Add the “when I receive Collision” command. Under it, add the commands “go to front layer,” “go to x:0 y: 25,” and “show.”
6. From the “Events” command section, drag the “when this sprite clicked” command into the programming area. This tells the program what to do when the button to restart the game is clicked. Under it, add the command “broadcast Title Screen.” This will return the user to the title screen without resetting the high score. This enables the user to play again.
7. Test the game to ensure everything is working.

SUMMARY

At this point, the game (though simplistic) is fully functional. Try adjusting the values coded into the game to see how they affect the game’s performance.

Parts 8 through 11 contain directions for adding additional features to the game. They make the game more challenging for the user. Each of the parts is independent, so they do not need to be added in order and they do not all need to be added to function properly. The directions for these parts is less detailed than the direction for parts 1 through 7.

PART 8 – SPEED UP THE AIR TAXI

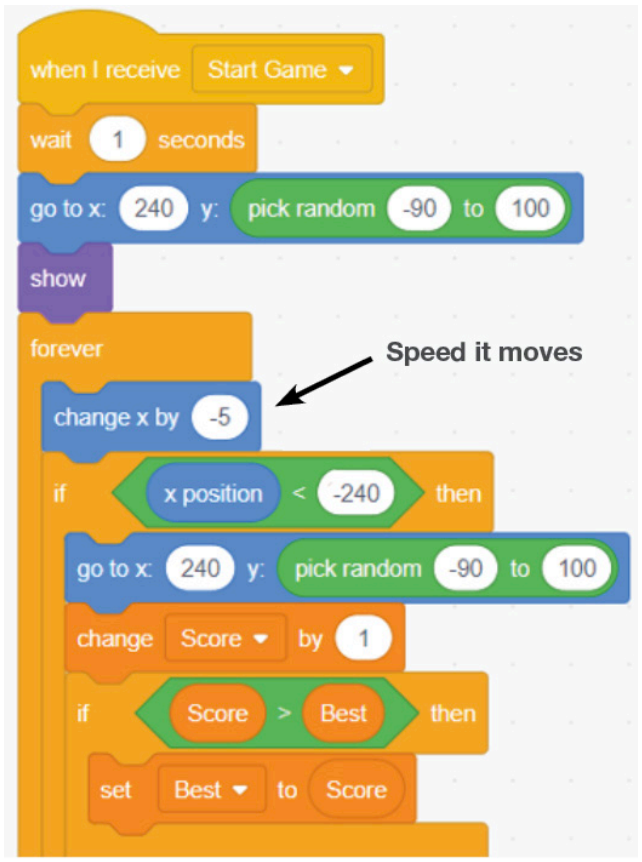
1. The code for the air taxi moves it across the screen 5 pixels at a time. The code to do this can be found in the “Air Taxi” sprite’s code, specifically the code that responds to the message “Start Game.” Changing the number in the “change x by -5” command changes the air taxi’s speed. Try different values to see the difference with different numbers.

2. Since this is a game, it makes sense to have the air taxis slowly increase in speed as the game progresses. To accomplish this, the code must be modified.

3. Instead of using a number in the “change x by -5” command, using a variable makes it easier to change the speed. This variable must be added and an initial value given to it. In the “Variables” command section, add a variable and call it “Air Taxi Speed.” Then, in the code that responds to “when I receive Start Game” in the “Air Taxi” sprite, add the command to set the variable “Air Taxi Speed” to -5 and to hide the variable.

4. Now that the variable has been set up, modify the code so that it changes the x-position of the sprite by the variable “Air Taxi Speed” instead of by -5.

5. Test the game to make sure it still works the same way.

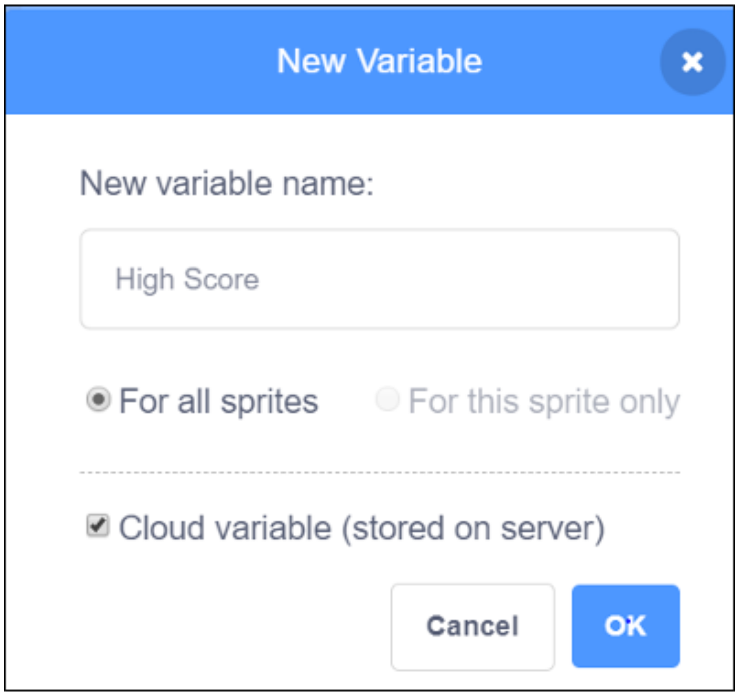


```
when I receive Start Game
wait 1 seconds
go to x: 240 y: pick random -90 to 100
show
forever
    change x by -5
    if x position < -240 then
        go to x: 240 y: pick random -90 to 100
        change Score by 1
        if Score > Best then
            set Best to Score
```

6. In the same block of code, there is a command “change Score by 1” which increases the score every time the air taxis fly off the left-side of the screen. Just below this command is where the variable “Air Taxi Speed” needs to be changed. Add the command to change the variable “Air Taxi Speed” by -0.5. This command will speed up the air taxi every time an air taxi goes off the screen.

7. Test the game. It should be a more challenging game now!

PART 9 – CREATE A GLOBAL HIGH SCORE


New Variable

New variable name:

High Score

● For all sprites ○ For this sprite only

☑ Cloud variable (stored on server)

1. Throughout the code, a number of variables are used. The variables in this program hold numerical values and are given names with which they can be used and changed. These variables are created when the program runs and they cease to exist when the program ends.

2. To create a global high score, a new type of variable is required. Create a variable called “High Score.” When creating it, click the box to make it a “Cloud variable.” This type of variable does not get created when the program runs and destroyed when it ends. Instead, the value for this type of variable is stored on the server so that it can be accessed by anyone running the program.

3. When this variable is created, a box will appear on the backdrop showing the variable name and its value. Move the box to where it looks good.

4. In the “Sprites” box, click on the “Title” sprite. In the code that responds to the “Start Game” message, there are two commands that display the score and the best score. Just below these, add the command to show the variable “High Score.”

5. In the “Sprites” box, click on the “Air Taxi” sprite. In the code that responds to the “Start Game” message, there is an “if...then” command that checks to see if the current score is greater than the best score. If the score is higher, it replaces the best score with the current score. Just below this, add another “if...then” command that checks to see if the current score is greater than the high score. If it is, the high score needs to be replaced with the current score.

6. Test your program.

PART 10 – ADD GEOFENCED AREAS

1. When NASA helped design the Advanced Air Mobility system to control aircraft, another feature called geofencing had to be created. Geofencing is a means of creating virtual barriers around specific areas. This keeps the aircraft from going into areas that could be hazardous.

2. To code geofences into the game, the “Geofence” sprite included in the downloaded files must first be uploaded to the “Sprite” box.

3. In the “Code” tab of this sprite, code needs to be added so that it will respond to the following three messages: “Title Screen”, “Start Game,” and “Collision.”

4. In the code that responds to the “Title Screen” message, add the command to hide the sprite.

5. In the code that responds to the “Collision” message, add the command to stop the other scripts in this sprite.

6. In the code that responds to the “Start Game” message, add the commands to:

   - Wait 8 seconds.
   - Go to a spot with an x-value of 240 and a y-value that is randomly between -90 and -150.
   - Show the sprite.

   Then add a “forever” command that contains the code to:

   - Change the x-position of the “Geofence” sprite by -3. This number is the same as the ground sprites so it stays in the same place on the ground.
   - Check to see if the x-position is less than -240. If so, move back to a spot with an x-value of 240 and a y-value that is randomly between -90 and -150 and change the variable “Score” by 1.
   - Check to see if the “Geofence” sprite touches the “Drone” sprite. If so, broadcast the “Collision” message.

7. Test the program to make sure it works properly.

PART 11 – ADD PACKAGE DELIVERY DRONES

1. Another important aspect of Advanced Air Mobility is package delivery drones. These small, unmanned aircraft carry packages containing food, medical supplies, retail purchases or other items for delivery to the end user.

2. To code package delivery drones into the game, the “Package Drone” sprite included in the downloaded files must first be uploaded to the “Sprite” box.

3. In the “Code” tab of this sprite, code needs to be added so that it will respond to the following three messages: “Title Screen,” “Start Game,” and “Collision.”

4. In the code that responds to the “Title Screen” message, add the command to hide the sprite.

5. In the code that responds to the “Collision” message, add the command to stop the other scripts in this sprite.

6. In the code that responds to the “Start Game” message, add the commands to:

   - Wait 2 seconds.

   - Go to a spot with an x-value of 240 and a y-value that is randomly between -120 and 100.

   - Show the sprite.

   Then add a “forever” command that contains the code to:

   - Change the x-position of the “Package Drone” sprite by -4.

   - Check to see if the x-position is less than -240. If so, move back to a spot with an x-value of 240 and a y-value that is randomly between -120 and 100 and change the variable “Score” by 1.

   - Check to see if the “Package Drone” sprite touches the “Drone” sprite. If so, broadcast the “Collision” message.

7. Test the program to make sure it works properly.

EXTENSION IDEAS

There are many features that can be added to this program. The following list contains a few ideas that can be assigned:

- Add a different background. Or, have the background change to night (a possible backdrop is included in the downloaded files) when a certain score is reached.
- Add a visual effect, available in the “Looks” commands section, to the title screen.
- Have the program display a message telling the user why they lost (hit an air taxi, flew too high, etc.).
- Add a variable that uses a slider to adjust the speed of the drone the user controls.
- Included in the downloaded files is a file called “planes.” To show that these planes operate above 400 feet, have them scroll to the right above the 400-foot altitude line.
- Change the flightpath of the air taxi so that it doesn’t fly in a straight line.

ADDITIONAL NASA RESOURCES FOR PRACTICING BLOCK-BASED CODING

- Package Delivery Drone Simulation
  - *https://www.nasa.gov/sites/default/files/atoms/files/aam-package-delivery-drone-simulation-activity-guide.pdf*
  - NASA’s Aeronautics Research Mission Directorate developed this activity in which students create a geofenced area on a map and program a package delivery drone to fly around this area in order to deliver a package.
- Explore Mars with Scratch:
  - *https://www.jpl.nasa.gov/edu/teach/activity/explore-mars-with-scratch*
  - NASA’s Jet Propulsion Lab designed this activity that takes a student through the steps of creating a simple game where the user controls a rover on Mars.
- Crew Orbital Docking Simulation:
  - *https://www.nasa.gov/stem-ed-resources/crew-orbital-docking-simulation-coding-sim.html*
  - Designed as part of the Commercial Crew Program, this NASA activity challenges students to create a simulation of a commercial spacecraft docking with the International Space Station.

GRADING RUBRIC

Rubric Category	Score
Program Execution and Output • Program simulates a drone flying. • Program meets the coding requirements (mark the requirements met): ☐ Backdrop and sprites are positioned appropriately. ☐ The ground is added and animated. ☐ The drone moves around the left side of the screen. ☐ The game ends if the drone goes too high, goes too low, or hits an obstacle. ☐ The game keeps and displays the current and best score. ☐ After losing the game, the user can restart it.	________/4
Program Execution and Output • All errors and bugs are eliminated. • Program is well organized and code runs efficiently. • Program goes beyond the minimum requirements.	________/4
Project Management • Project completed on time. • Time given is used productively to work on project. • Programmer had others beta test the program to help find errors.	________/4
TOTAL	________/12

4 (Advanced)	=	All criteria are met and followed with very few mistakes
3 (Proficient)	=	Most criteria are met with few mistakes
2 (Developing)	=	Many criteria are not met and/or there are many mistakes
1 (Beginning)	=	Most criteria are not met
0 (Very Little to No Effort)	=	Very little to no effort was put forth to meet criteria

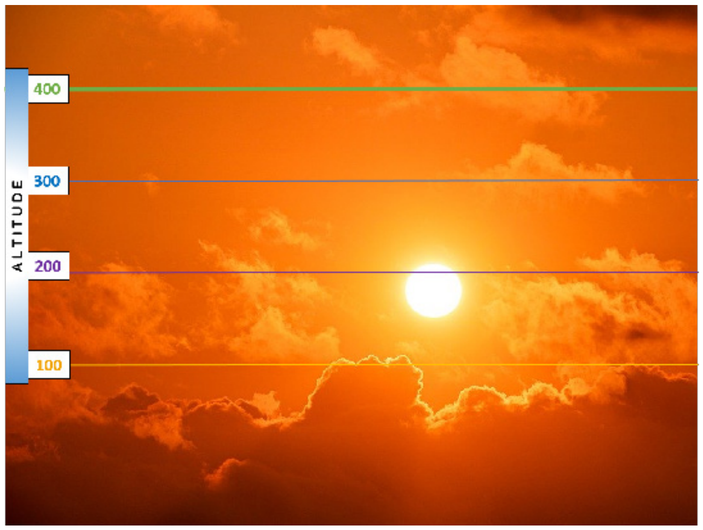
Air Taxi Download

Begin Button Download

Drone Download

Try Again Button Download

ALTITUDE
400
300
200
100
Backdrop BW Download

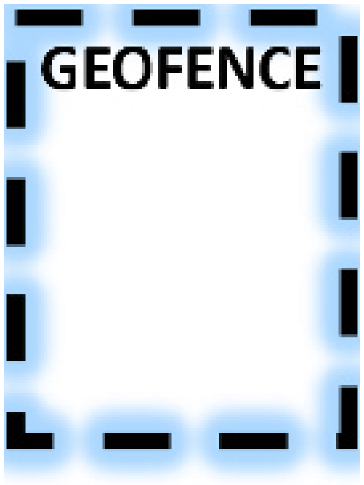


Backdrop Color Download

City Download

Ground B&W Download

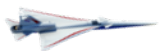
Ground Color Download



Geofence Download

NASA Logo Download

Package Drone Download



Planes Download

ATTACK OF THE DRONES

Title Download

GAME OVER

Game Over Download

National Aeronautics and Space Administration

Headquarters
300 E Street SW
Washington, DC 20546

www.nasa.gov

EP-2020-04-509-HQ