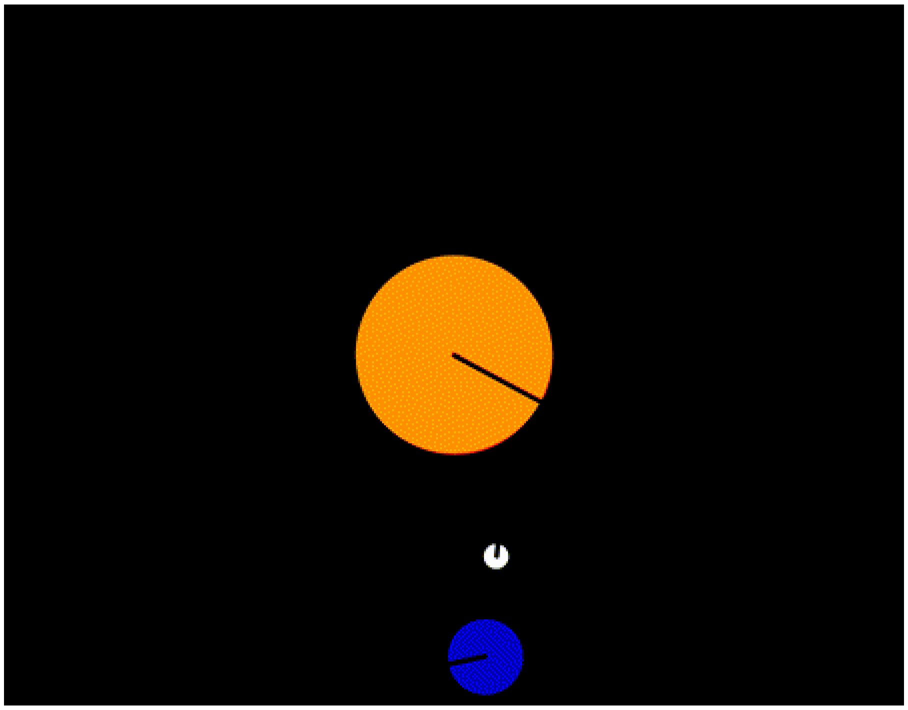
# Graphics Programming CM2030 Midterm Instructions

*24 Dec 2021*

*These instructions have been copied from Coursera by a fellow student. For official and updated information, visit https://www.coursera.org/learn/uol-graphics-programming/home/*



## Week 1 – Solar System

In this assignment the goal is to create a solar system like the one in the image above.

**Please note: This is a graded assignment. This is a draft of the assignment made available before course work submission is open, only minor changes may occur.**

### Where do I submit?

There are eight graded assignments like this on the course. This one and three more make up the midterm assignment and are submitted in week 10. Nevertheless, we strongly recommend that you complete this graded assignment in the week it has been assigned and not wait for the midterm submission. You need to master the material included in these before you continue with the next weeks. Once you complete it, save it somewhere safe, do not share online using the webspace and upload it with the others when prompted in week 10.

## Steps to complete

Start by running the ***solarSystemTEMPLATE*** sketch in Brackets.

What you'll see is a function called *celestialObj()* which draws any celestial object you want, whether it's the sun, a planet or a moon. It takes a color and a diameter. When you first run it, the sketch already shows the sun. There's also a variable called *speed* which you will use in order to rotate objects at various speeds. The variable speed is equivalent to the *frameCount* variable of p5js, the variable that measures how many frames have been displayed on the canvas.

Step 1: Create an earth of color blue and size 80. Move it to orbit 300 pixels. Hint: You should only use transformations to achieve this.

Step 2: Rotate the earth around the sun at velocity "speed". Make sure you use the radians() function inside the rotate() call.

Step 3: Make the earth spin around its axis at velocity "speed" too. You should now have an earth that rotates around its axis and around the sun.

Step 4: Add a moon of color white and size 30 to the earth at an orbit of 100 pixels. Make the moon rotate at velocity *-speed*2* so it spins the opposite way to the way the earth rotates. Make sure that your moon, while spinning around the earth, always shows the same side to earthlings (ie earthlings should not be able to see the so-called 'dark side of the moon').

Step 5: Make the sun spin around its axis, as it really does, at a slow *speed/3*.

Step 6: Extend the sketch by adding one more celestial body by implementing one of the ideas for further development.

Step 7: Submit the file to the 'Staff Graded Assignment: Mid-term assignment submissions' located in the end of week 10.

## Ideas for further development:

- What if the moon had an asteroid that was stuck in orbit around the moon? Can you add a smaller celestial body that is rotating around the moon.

- Pretend that earth, like Mars, has two moons. Can you add another different sized moon that rotates around the earth at a different speed with a different distance from the earth? This moon should also only show the one side to the earth.

## Marking rubric:

Step 1 - [1 points]: Earth is at the right orbit.

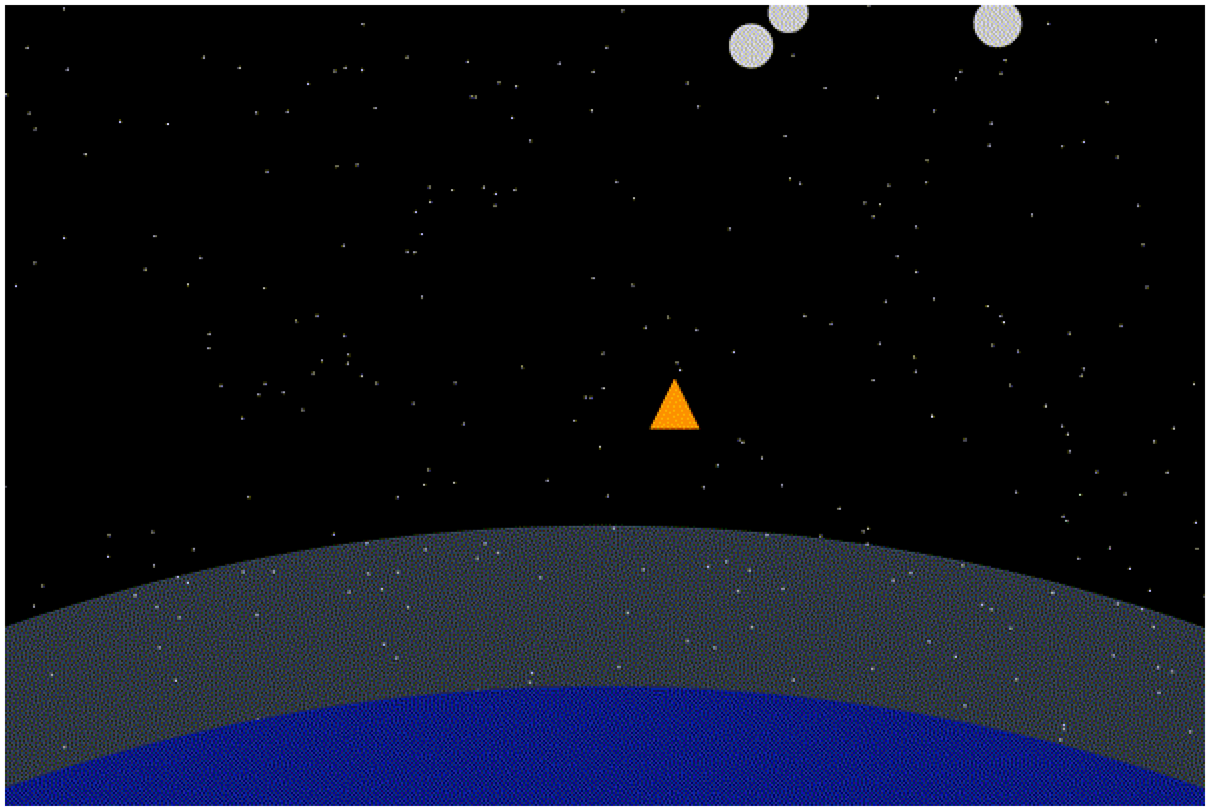
Step 2 - [2 points]: Earth rotates around the sun.

Step 3 - [2 points]: Earth rotates around its axis and all earthlings get to see the sun.

Step 4 - [2 points]: Moon is at the right orbit, rotates around the earth, always shows the same side (one point each).

Step 5 - [1 point]: Sun is spinning on its axis at the right speed.

Step 6 – [2 points]: Did the learner implement one of the ideas for further development correctly?



# Week 3 - Asteroid game clone

In this assignment the goal is to complete an asteroids game clone complete with realistic movement, gravity and collision checks as in the image above.

**Please note: This is a graded assignment. This is a draft of the assignment made available before course work submission is open, only minor changes may occur.**

## Where do I submit?

There are eight graded assignments like this on the course. This one and three more make up the midterm assignment and are submitted in week 10. Nevertheless, we strongly recommend that you complete this graded assignment in the week it has been assigned and not wait for the midterm submission. You need to master the material included in these before you continue with the next weeks. Once you complete it, save it somewhere safe, do not share online using the webspace and upload it with the others when prompted in week 10.

## Steps to complete

Step 1: In bulletSystem.js complete the function edges() by looping over the bullets array and for each bullet check if it has left the screen. If it has, remove the specific bullet from the array. (Hint: Use splice() to remove the bullet from the array and think about how you splice from an array correctly). Remember: If you want to call a function or use any variable that belongs to the class you'll need to prefix it with the word "this". So in this case it would be: this.bullets.myFunction();

Step 2: In spaceship.js update the interaction() function and fill out the missing information in order to simulate the correct behavior in the system. When the left arrow is pressed the spaceship should move left, when the right arrow is pressed it should move right and so on. Please note that the spaceship won't move until you have completed the next step too.

Step 3: In spaceship.js update the move() function similarly to how the move() function works in asteroidSystem. Make sure you limit the velocity vector to maxVelocity so that the spaceship does not accelerate too much. Notice how, unless we fire the rockets in the opposite direction, the spaceship will keep moving. There is no friction in empty space.

Step 4: In sketch.js complete the isInside() function that takes the location of two circles and their diameters and returns true if they overlap, false otherwise. You could check it works, by creating a dummy circle around the mouse and checking if isInside() returns true.

Step 5: In sketch.js complete the checkCollisions() function so that you check collisions between the spaceship and all asteroids. Hint: You'll need to loop over all the asteroids and use the inInside() function you just programmed. If it returns true then you'll call the gameOver() function. If you've done things right, if the spaceship is hit by an asteroid the game will end. Check before proceeding.

Step 6: In sketch.js add more functionality to the checkCollisions() function to check if an asteroid has crashed onto earth. If you do things right then the game should end when that happens. Check before proceeding.

Step 7: In sketch.js add more functionality to the checkCollisions() function to check if the spaceship has collided with the earth and if it has it ends the game. Check before proceeding.

Step 8: In sketch.js add more functionality to the checkCollisions() function to check if the spaceship is inside the atmosphere. If it is, the spaceship's setNearEarth() function is called.

Step 9: In spaceship.js complete the setNearEarth() function. When the spaceship enters the earth's atmosphere it's affected by the earth's gravity. Create a "downwards-pointing" vector of strength 0.05 which pulls the spaceship towards the earth. The atmosphere also introduces friction and the spaceship can't move forever like in empty space. It will decelerate unless it fires its engines. Create a force called friction that's 30 times smaller than the velocity of the spaceship, points the opposite direction to velocity and is then applied in the the opposite direction.

Step 10: In sketch.js add more functionality to the checkCollisions() function to check if any of the bullets of the spaceship have hit any asteroids. If they have, then call the destroy() function of the asteroid object, passing it the index of the asteroid to destroy.

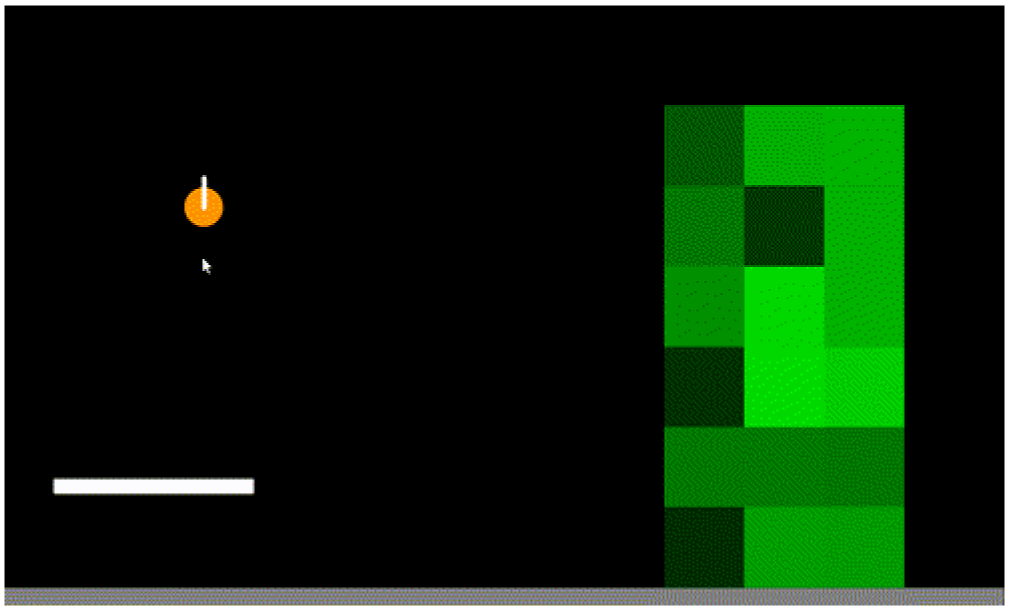
Step 11: Implement two of the ideas for further development below. Points given to ambitious learners.

## Ideas for further development:

- Make the spaceship pretty by adding jets thrusters which activate from the opposite side of movement just like on a real spaceship.
- Keep score of how many asteroids you have hit.
- Make the program get progressively harder by spawning asteroids more frequently as time goes by.
- Make it your own by customising the look of the game (make sure you maintain same functionality or you might lose points during grading).

## Marking rubric

- Step 1 - [1 point]: edges() function is correct.
- Step 2 - [1 point]: interaction() function is correct.
- Step 3 - [2 point]: Spaceship moves correctly. No friction in space and velocity limited by maxVelocity.
- Step 4 - [1 point]: isInside() function works as it should.
- Step 5 - [1 point]: If asteroid hits spaceship: game ends.
- Step 6 - [1 points]: If asteroid crashes on earth: game ends.
- Step 7 - [1 points]: If spaceship crashes on earth: game ends.
- Step 8 - [1 points]: If spaceship inside atmosphere: call the spaceship's setNearEarth() function.
- Step 9 - [2 point]: If spaceship inside atmosphere, earth pulls it towards it, air friction affects movement.
- Step 10 - [1 point]: If any bullet hits asteroid: destroy asteroid.
- Step 11 - [4 points]: Has learner implemented any of the ideas for further development?



# Week 5 – Angry Birds Clone

In this assignment the goal is to complete an angry birds clone game exhibiting the behaviors shown in the image above.

**Please note: This is a graded assignment. This is a draft of the assignment made available before course work submission is open, only minor changes may occur.**

## Where do I submit?

There are eight graded assignments like this on the course. This one and three more make up the midterm assignment and are submitted in week 10. Nevertheless, we strongly recommend that you complete this graded assignment in the week it has been assigned and not wait for the midterm submission. You need to master the material included in these before you continue with the next weeks. Once you complete it, save it somewhere safe, do not share online using the webspace and upload it with the others when prompted in week 10.

## Steps to complete

Start by downloading the template attached. The template code contains the sketch.js file which ties everything together and the physics.js file which contains physics-specific functions, moved to a separate file so that your code is cleaner. You'll write most of your code in the physics.js file.

## Creating a propeller system

<u>Step 1:</u> Amend the setupPropeller() function in physics.js to setup a static body of type rectangle at location (150, 480) of size (200, 15), similar to how the ground is created. Use the global variable *propeller* for this. The initial angle is equal to the global variable *angle* which we provided. Add the *propeller* to the world.

<u>Step 2:</u> In the drawPropeller() function set the angle of the propeller equal to the global variable *angle*. Set the angular velocity equal to the global variable *angleSpeed* already provided for you. Update the variable *angle* by *angleSpeed*. This will make sure the propeller actually moves at a rate of *angleSpeed* between frames. Draw the propeller using the drawVertices() helper function just like in the drawGround() function. In sketch.js, amend the keyPressed() function so that when the left arrow is pressed, the angle speed is incremented by 0.01. Do the opposite when the right arrow is pressed. If you've done things right when you start your sketch you should be able to control the propeller on screen using the left and right arrows.

<u>Step 3:</u> When 'b' is pressed a bird is created wherever the mouse is by calling the setupBird() function. Study that function. At the moment the program does not draw birds. Amend the drawBirds() function to loop over the birds array and draw them using the drawVertices() helper function. Use the isOffScreen() function to check if the bird has left the screen and, if it has, remove it from the physics world using the removeFromWorld() helper function provided <u>and</u> from the array. Remember to also decrement your for-loop counter in order not to break your code! Pressing 'b' should now create a new bird where the mouse is which flies away upon impact with the moving propeller.

## Creating a tower of boxes

<u>Step 4:</u> Amend the setupTower() function to create a tower of boxes as shown in the picture above. Tower should be six boxes high, three boxes wide. Each box should be of size 80x80 pixels. (Hint: Create a nested for loop and push bodies of type rectangle on the boxes array provided). Also push a random shades of green onto the colors array. We'll use those colors to draw the boxes later. Remember to add the boxes to the world.

<u>Step 5:</u> In the drawTower() function loop over the boxes array and draw each box using the drawVertices() helper function. Remember to use the random colors you created inside the colors array. You should now see a tower of boxes in different shades of green like in the image above.

## Creating a slingshot

<u>Step 6:</u> Amend the setupSlingshot() function to initialise the global variable *slingshotBird* as a body of type circle in a similar place as shown in the image above. Give the circle zero friction

and a restitution of 0.95. Set the mass of the *slingshotBird* as ten times its original mass, just like we have done for each of the birds in the setupBird() function.

Initialise also the global variable *slingshotConstraint* as a constraint that behaves and looks like the one shown above. Give it a stiffness of 0.01 and damping of 0.0001.

Remember to add both the bird and the constraint to the world and make sure they appear where they are in the image above.

<u>Step 7:</u> Amend the drawSlingshot() function and use the drawVertices() and drawConstraint() helper functions to draw the slingshot bird and slingshot constraint. If you've done things right you should now be able to control the slingshot with the mouse. Drag to extend, release mouse to release bird. Pressing 'r' resets the slingshot.

<u>Step 8:</u> Implement one of the ideas for further development.

## Ideas for further development

- Turn it into a game. Create a countdown. The player has a 60 seconds to remove all boxes from the screen. If they succeed, they win. If they fail, looping stops and a GAME OVER message is displayed.
- Make it your own by changing the colors and style.
- Add another physics object that adds to the gameplay in a fun way.

## Marking rubric

Step 1 - [1 point]: Propeller in right place approx.

Step 2 - [1 point]: Propeller moves with left/right arrow.

Step 3 - [2 points]: Birds are drawn when pressing 'b' and interact with propeller. Removed from world and array when off-screen.

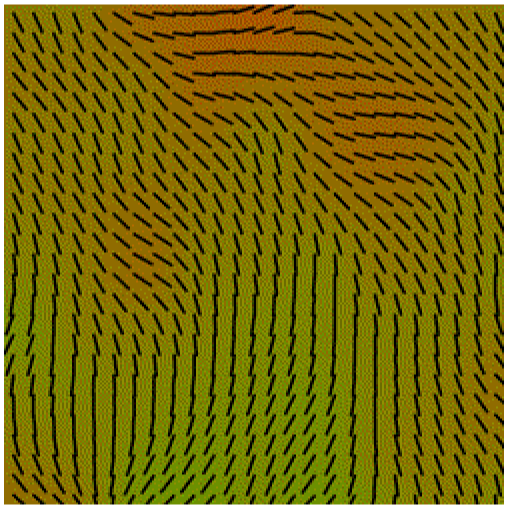
Step 4 - [0 point]: (points transferred to next step)

Step 5 - [3 points]: Box tower is in the right place approx. Has dimensions of example provided. Boxes use multiple colors.

Step 6 - [0 point]: (points transferred to next step)

Step 7 - [3 points]: Slingshot appears in right place. Has the right stiffness. Slingshot bird has enough mass to demolish tower.

Step 8 - [3 points]: One of the ideas for further development has been implemented in an impressive way. Full points for learners that challenged themselves.



# Week 7 - Noisy grid

In this assignment the goal is to complete a constantly evolving generative piece as shown in the image above.

**Please note: This is a graded assignment. This is a draft of the assignment made available before course work submission is open, only minor changes may occur.**

## Where do I submit?

There are eight graded assignments like this on the course. This one and three more make up the midterm assignment and are submitted in week 10. Nevertheless, we strongly recommend that you complete this graded assignment in the week it has been assigned and not wait for the midterm submission. You need to master the material included in these before you continue with the next weeks. Once you complete it, save it somewhere safe, do not share online using the webspace and upload it with the others when prompted in week 10.

## Steps to complete

Start by downloading the template project attached.

### Creating a colorful grid

Step 1: Amend the colorGrid() function and create a 25x25 grid of rectangles. The rectangles should be of width and height equal to *stepSize*, the global variable already provided. Set the fill to white and the stroke to black, to make sure the grid is correctly setup.

Step 2:

For each of the tiles generate a 3D noise value, using the tile's x and y coordinate as well as the frameCount so that the noise values change over time. Make sure you scale the input parameters of the noise function appropriately so that you get organic values out of it. Use that noise value to lerp (aka interpolate) between red and green.

Use the lerpColor() function (see p5.js examples) to do so. Remove the rectangle's stroke and use color returned from the lerpColor() function to fill the rectangle. If you've done things right you should see a grid of tiles with slowly changing colors.

If the color of your rectangles are changing rapidly or are not smooth like the gif above, double check how you have scaled the input parameters for noise().

Step 3: Make the grid color change speed be dependent on the x coordinate of the mouse (ie small mouseX, faster color changes).

### Creating a compass grid

Step 4: Amend the compassGrid() function to create a grid of 25x25 lines of length *stepSize*. Make sure each compass is at the center of each tile. By default they should all be pointing up. You should use translate() to move to the center of each grid.

Step 5: For each of the compasses generate a 3D noise value, using the compass' x and y coordinate as well as the frameCount so that the noise values change over time. Make sure you scale the input parameters of the noise function appropriately so that you get organic values out of it. Use that noise value with map() to generate an angle between 0 and 720 degrees. Use that value to rotate the compass. If you've done things right your compasses should be rotating while the sketch is running.

If your lines are not rotating in the pattern seen above check how you are scaling the input parameters of noise(). Also study how each line rotates in the gif above.

Step 6: Make the compass' movement speed be dependent on the x coordinate of the mouse (ie small mouseX, faster compass rotations).

Step 7: Implement two of the ideas for further development. Points given to ambitious learners.

## Ideas for further development

- Make it your own by changing the colors and style.
- Use the noise to change the color and length of the compass lines.

## Marking rubric

Step 1 - [1 point]: Tiles are of the right size and spread on canvas.

Step 2 - [2 points]: Colors change organically over time.

Step 3 - [1 point]: mouseX affects how fast the grid colors change.

Step 4 - [1 point]: Compasses setup correctly and at the center of each tile.

Step 5 - [2 points]: Compasses move organically over time.

Step 6 - [1 point]: mouseX affects how fast the compasses rotate.

Step 7 - [2 points]: Has learner implemented any of the ideas for further development?