Extra exercises

1 Week 1

1.1 The dark side of the moon

Write a program that draws a moon orbiting the earth.

1. Draw an earth in the middle of the screen.
2. Draw a moon that revolves around the earth.
3. Make half of the moon black (the dark side). Tip: draw two half-circles with `arc`s.
4. Now make sure that the dark side of the moon is always facing away from the earth.

2 Week 2

2.1 Parametric curves (I)

It is possible to draw a circle by just using lines. The idea is that if you draw a line that is small enough, you won't see that it is not curved.

1. To practice, first draw a dotted line. Use a for-loop for this.
2. Now draw a line made of smaller lines (connect the dots).
3. Draw a line made of smaller lines that goes through the diagonal.
4. To draw a circle, draw many small lines from position (radius,angle) to (radius, angle + small difference). Draw a circle without using the circle function, using a for loop.

2.2 Parametric curves (II)

Download the following source code: http://liacs.leidenuniv.nl/ ruijlbjg/psy.pde.

1. Explain what the functions `step` and `sign` do.
2. The complicated functions `getx` and `gety` describe a curve, by giving the x and y coordinate for a given parameter t. You can imagine walking on the curve, starting at $t = 0$ all the way to the end, which is $t = 72 \cdot \pi$. Draw the curve.

2.3 Loading animation

1. Build a loading animation that adds 12 circles on a line, one by one. The colours should transition from red to green.
2. Do the same, only draw the circles in a circle now. Make sure the first circle is almost transparent, and the last opaque.

3 Week 3

3.1 Blurring

Write a program that blurs an image by averaging the colours of a pixel with its surrounding pixels. Make the level of blending, i.e., how much the colours of the surrounding pixels contribute, a variable.

3.2 Noise

Write a program that creates dynamic noise on an image, just like an analog tv with a bad connection. Every few frames, change the colours of the pixels by a small random amount. Make sure you apply the noise to the original image every time. If you don't do this, the image colours will drift so far from the original, that you will not be able to recognize it anymore.

3.3 Masses on a spring

Build a system of circles connected with lines to other circles. These lines are springs: if they become longer, they apply a force in the opposite direction. So, if you move one of the masses, the system should start to vibrate.

1. Build an array of 10 floats to keep track of the position of 10 circles. Initialize the positions to be in a horizontal line with equal distance between them. Draw the circles and draw lines connecting them.

2. Since we are going to apply forces of the spring, we need 2 more arrays: one for the velocity and one for the acceleration. Every frame, we are going to update the position, acceleration and velocity. Let us first consider the force. For a simple spring, we say the force is equal to the length of the string. Since each circle `i` is connected to two strings, we find the following:

$$a_i = (p_{i+1} - p_i) + (p_{i-1} - p_i) = p_{i+1} + p_{i-1} - 2p_i$$

where a_i is the acceleration of the ith circle in the current frame and p_i is the position of the ith circle. Every frame we have to update the three parameters:

$$p_i^{\text{new}} = p_i^{\text{old}} + v_i^{\text{old}} \Delta t \quad (1)$$
$$v_i^{\text{new}} = v_i^{\text{old}} + a_i^{\text{old}} \Delta t \quad (2)$$
$$a_i^{\text{new}} = p_{i+1} + p_{i-1} - 2p_i \quad (3)$$

where we assume that every frame lasts Δt seconds (say Δt=0.1) and where `old` means the values from the last frame.

3. Implement the above rules. Make sure the loop goes from 1 to 9, so the outer two circles are fixed. You will need 3 separate for loops.

4. When you press the mouse, move one of the masses, so that the system starts moving. It could be that the movement is so large that one of the masses crosses another one and the system goes crazy. You can combat this by adding friction: Make sure that v_i^{new} is multiplied by a factor less than one, for example 0.95.

4 Week 4

4.1 Binary search

1. Create a list of 20 random numbers and sort it using the `sort` function.

2. Create a function that returns true if a number is in the list and false if it is not.

3. Implement a binary search: test the element in the middle of the array. If it is smaller, check the top half, otherwise the bottom half. Keep on doing this until the number is found or it is concluded that the number is not in the list.

4. Write a recursive implementation.

4.2 Path in tree

1. Use a 2 dimensional array to represent the following triangle:

```
   3
  7 4
 2 4 6
8 5 9 4
```

2. Write a recursive function that finds the path from top to bottom with the highest value. This is defined as the sum of all visited nodes. In this case the past with highest value is $3 + 7 + 4 + 9$.

5 Week 5

5.1 Health and armour packs

1. Create a player class that keeps track of the health and armour score (both in range 0 to 100). Print the values of health and armour on the screen (not the console).

2. We are going to create a class for health and armour kits that can be picked up when you click on them. Since most of the functionality is the same, first create a base class called `Kit`. A `Kit` has variables for the position on the screen, a `draw` function, and an `onClicked` function. Implement a default draw function that draws a rectangle.

3. Randomly generate 20 Kits in a `ArrayList` and make sure to draw them all.

4. Now create a `HealthKit` and an `ArmourKit`. Override the `onClicked` so that the `HealthKit` restores health and the `ArmourKit` restores armour. Also give them a different draw function.

5. Instead of generating 20 random `Kit`s, generate random `ArmourKit`s and `HealthKit`s. Make sure they end up in the same `ArrayList` of `Kit`s.