Algorithms in Creative Technology

Main goal of this course is to achieve **fluency in programming**.
In the first module we treated language constructs as variables, assignments, loops, functions, scopes and classes. So far, these form the building blocks of your toolbox in programming (unless you have experience from a different source). If we take "language" as a metaphor you know by now the vocabulary and grammar of the programming language. Now, we want to go one step higher: the next bigger building blocks for programming are algorithms, that need integration skills on top of your current toolbox. Using the language metaphor, you should now learn to tell (short) stories.
We chose a number of "nice" algorithms to train this skill: some are from the book "Nature of Code" by Daniel Shiffman, http://natureofcode.com/book/. These algorithms are useful for CreaTe in any case: different versions of randomness, as e.g. needed for clouds or landscapes, moving flowers in the wind (giving the reason why you had to learn mass-spring-damper systems), particle systems making every application nicer, and flocks, very effective for video installations.

To achieve fluency you have to program in the first place, and program, and program, and program. Make each fault one time at least.
Therefore, we will not have so many lectures, but basically challenges and time to program. And there is an excellent team of student assistants, who are able to help you exactly at the moment when you run into a problem. Make use of them!!! Short introductory lectures will be given, especially at the beginning of each new topic. You can make your life easier when following a systematic way of programming, avoiding spaghetti code.

Getting help from student assistants

The concept of this course is that you do a lot of programming and get help when you run into a problem. Programming is a skill and needs a lot of practice and this is the way we believe that you learn programming fluency in the most effective way.
What we often see, however, is that students write messy programs and ask student assistants to debug them. This is not the right way to learn programming or make use of the time of the student assistants. When you ask a student assistant for help please make the structures you want to use clear beforehand, by, e.g., pseudo code you have written before, or a comment in the beginning of a method what this method should do, or a sketch by a flow diagram or a class diagram or what is appropriate here.
Learning how to debug (yourself) is an important skill in programming (and also all other making)!

Code quality

For the end assignment your program will be evaluated on several different aspects, and one very important is code quality. Often students think that it is sufficient when their program works, but when you cooperate with others, want to reuse code, or use your code later in a bigger context, code quality makes your life a lot easier.
Also, not all code presented on the internet has a high code quality. Even Shiffman's code is sometimes a bit "smelly". If you use code from the internet as a part of your assignments, you are also responsible for the quality of this code.
Typical issues we see in beginners' programs are the following, and you should try to avoid them:

- Communication is solved by global variables ("refrigerator communication") instead of using methods that pass parameters.
- Methods are too long, especially the draw.
- Variable names make only sense in the program owner's mind.
- var1,…., var7 used instead of creating an array
- Different classes do almost the same.
- Sets of variables/arrays are used where a class would be appropriate
- Classes that are related semantically are not related structurally: For example, for the rocket with the fire tail, an object of the class Rocket and an object of the class FireTail are only drawn at the same location, but belong to strictly separated classes. (The solution would be to have an object of the class FireTail as object variable in the class Rocket)
- Lousy documentation - not helpful, or stating the obvious.
- Interaction is not in the event handling part, i.e. mouseX, mouseY etc. are used within classes (or draw) and not passed as parameters of a method to the class, or similar with keyCode. All these interaction variables should be used in the dedicated methods like mousePressed(), etc.

Git Classroom and Grading

You are expected to upload your solutions to GitHub classroom. You can earn a third of your grade by having your assignments there before the due date.
On GitHub we will do automatic checks on plagiarism and code quality. You may work in teams of two people, if more people share the same code or submit code from the internet as their own, the usual procedures for plagiarism have to be started.
We will check some of the assignments also individually. In the oral exam in the end, we will come back to the assignments you made and evaluate them (see also under oral exam at the end of the document).
Due dates are only for black assignments. For blue and green assignments we assume that the intrinsic motivation is sufficient.
All assignments have to be uploaded to GitHub using a link given on Canvas. See also the GitHub intro in another file on Canvas.

There are 14 black assignments, 10 blue assignments, and 10 green assignment.
For a black assignment (before the due date) you receive 5/42 points (all adding up to 5/3 points), for each blue or green assignment you receive 1/15 point, (each colour adding up to 2/3 points), so altogether by having all assignments you get 3 points. It is obvious that assignments made not of sufficient quality will not result in points, which might be checked only retrospectively. Make sure with a student assistant that the solution you upload is sufficient, if unsure. For black assignments points will only be received if submitted before the due date (deadline).

The grade for the oral exam translates to maximal 6 points as follows:

points for the oral exam = (grade for the oral exam - 1) * 6 / 9

Altogether, this results in:

final grade = 1 + points for the assignments + points for the oral exam

Topic 1: Warming up

topics: arrays, loops and binary representation, PVectors
due date for black assignments: 3.5.2019 20:00

👉topic: recap arrays and how to lop through an array:
1. 1 Write for each of the following a program that defines an array of characters, and:
1.1.1. writes the content of the array on the Processing window, letter by letter
1.1.2. writes the content of the array on the Processing window, backwards order, i.e. starting with the last
1.1.3. counts how many occurrences of the letter 'e' are contained in the array
1.1.4. decides whether the array form a palindrome, i.e. reading from beginning gives the same "word" as reading from the end. An example for a palindrome is "ANNA".

👉topic: recap how to write a method/function and understand that a method/function can have a result type:
1.2. Make for each of the programs above (1.1.1-1.1.4) a function/method that is called from draw(), gets as parameter an array and does the same as the program above, where it has return type void for the first two examples, int for the third, and boolean for the fourth.

👉topic: recap random function and do another array exercise
1.3 Write a program with an array of integers of length 500, that are randomly initialized by values between 0 and 100 and write a function that counts how many numbers are above 50.

👉topic: more arrays and variations on loops
1.4 Write a program that gives as result the sum of all multiples of 5 below 1000.

👉topic: more arrays, variations on loops and a smart algorithmic idea
1.5 Write a program that gives as result the sum of all multiples of 3 and 5 below 1000. Note that numbers that are multiples of 3 and 5 at the same time should not be counted twice.

👉topic: string manipulation and Boolean number transformation
1.6 Write a program that transforms a binary number, e.g. 110101 represented as a string, to its decimal representation is defined as follows: $1 * 2^5 + 1 * 2^4 + 0 * 2^3 + 1 * 2^2 + 0 * 2^1 + 1$
(The more efficient way to calculate the transformation is the following: by processing the binary number from the front to the end using one loop:
((((1 * 2) +1) * 2 + 0) * 2 + 1) * 2 + 0) * 2 +1)
Using a Java function that transforms a binary representation to a decimal is not a solution to this assignment.

👉topic: nested loops
1.7 Display all binary numbers that can be written with 3 bits. Use 3 nested for-loops.

👉topic: smart nested loops
1.8 Assume that there are 9 GPS satellites visible (at a certain place, at a certain moment). To determine the position 4 satellites are necessary. Write a program that calculates all possible combinations of 4 satellites from the set of 9, and gives as output a list of these. (One possible solution has 4 nested loops.)

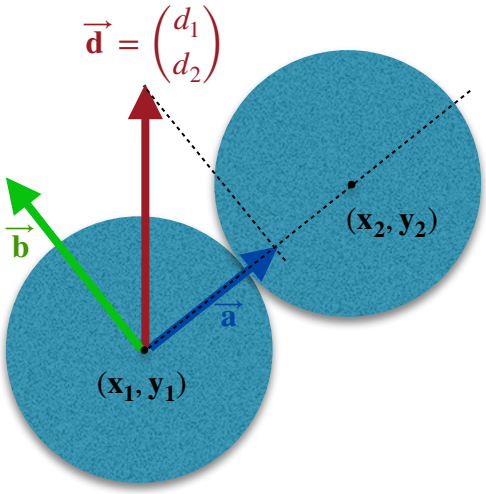
👉topic: challenge!
1.9 The prime factors of 13195 are 5, 7, 13 and 29.
What is the largest prime factor of the number 317584931803?

👉topic: PVectors
1.10 Take your bouncing creatures program from module 1 (the version where the creatures only bounce at the borders), and rewrite it with PVectors.
(Alternatively you can use the bouncing confettis pc4 that will be available on canvas).

Using a given class or API is a common task in programming. What you should learn is to read the description of the class, understand what the object variables are, and understand the methods and how to use them. This is one part of doing an exercise with PVectors. The other part of using PVectors is that this is really a useful class - it allows to pack components together that belong together, such as x-position and y-position, or speed in x-direction and y-direction. Once you know it, you will no longer want to fizzle around with the individual components of vectors as separate variables. It also makes your programs more readable.
As a step further, lots of physics is built around vectors: speed, acceleration, force. It is good to understand these concepts and use them easily.

You can find the PVectors in the references, and good (lengthy) introductions from Daniel Shiffman in Nature of Code http://natureofcode.com/book/chapter-1-vectors/
or from Khan academy at https://www.khanacademy.org/computing/computer-programming/programming-natural-simulations/programming-vectors/a/intro-to-vectors

👉topic: PVectors and their methods
1.11 The physics of bouncing between two balls (neglecting weight differences) is described by the following illustration and formula, where vector $\vec{d}$ describes the velocity of one ball (the bigger in the illustration) before bouncing, and vector $\vec{b}$ is the velocity of the ball after bouncing. Vector $\vec{a}$ describes the speed in the direction of the second ball, that contributes to the speed of this second ball after bouncing.



Define as $\overrightarrow{centers}$ the vector from (x_1, y_1) to (x_2, y_2), i.e., $\overrightarrow{centers} = \begin{pmatrix} x_2 - x_1 \\ y_2 - y_1 \end{pmatrix}$, $|\vec{a}|$ is the length of vector $\vec{a}$, and $|\overrightarrow{centers}|$ is the length of vector $\overrightarrow{centers}$,

then from the definition of the dot product for vectors we know:

I) $\vec{d} \cdot \overrightarrow{centers} = |\vec{a}| \cdot |\overrightarrow{centers}|$

$$\Longrightarrow |\vec{a}| = \vec{d} \cdot \frac{\overrightarrow{centers}}{|\overrightarrow{centers}|}$$

II) as $\vec{a}$ and $\overrightarrow{centers}$ have the same direction, the vector $\vec{a}$ is the same as the length of $|\vec{a}|$ multiplied by the normalised vector $\frac{\overrightarrow{centers}}{|\overrightarrow{centers}|}$:

$$\vec{a} = |\vec{a}| \cdot \frac{\overrightarrow{centers}}{|\overrightarrow{centers}|}$$

I) and II) combined give:

$$\vec{a} = \left(\vec{d} \cdot \frac{\overrightarrow{centers}}{|\overrightarrow{centers}|} \right) \cdot \frac{\overrightarrow{centers}}{|\overrightarrow{centers}|}$$

III) Finally, the vector $\vec{b}$ is the difference between vector $\vec{d}$ and vector $\vec{a}$:

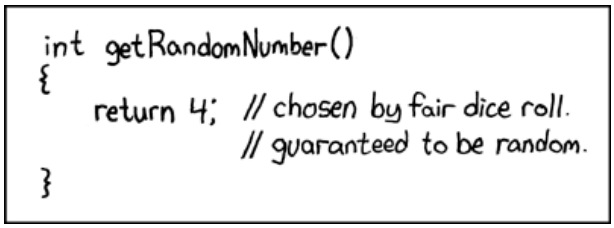
(Re)Write the method for bouncing balls using the formula above and PVectors.

Algorithms in Creative Technology

Topic 2: Randomness

topics: different sorts of randomness, repetition object orientation

due date for black assignments: 10.5.2019 20:00



by xkcd

In most programming languages a random function is built in. Real randomness is very difficult to program. In programming languages the random function is typically pseudo-random, that means, it is a function, giving always the same results for the same input parameters. The standard random functions available, i.e. uniform distribution, simulate evenly distributed randomness. It is used in the first 4 assignments here.
The second version of a random function that we will use here simulates a normal distribution.
The third kind of randomness we will look at is called Perlin noise. It is also only pseudo-randomly, but very useful to generate landscapes, clouds etc. Its main characteristics is some sort of continuity: in a little area around one point the random values generated differ only little.
See explanations and material here: http://natureofcode.com/book/introduction/
In Statistics you will learn more about distributions, later in the module. You can do the assignments here "naively", i.e. without deeper knowledge of distributions, but you should revisit the assignments later and be able to explain the intended effects of the distributions chosen in professional language in the exam.

👉topic: random function, array, distribution

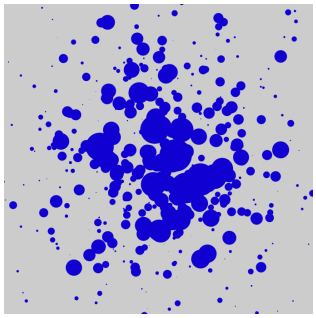
2.1 Use a random function to program a dice, e.g. each time you click the mouse a new random number between 1 and 6 occurs. Extend your program to count occurrences of each number. What is the distribution if you have 1000000 times the dice thrown (then without mouse click…)?

👉topic: random function, array, distribution

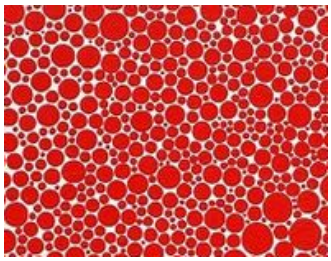
2.2 Change your dice in a way that the 6 gets a slight preference above the other numbers.

👉topic: normal distribution for location and size

2.3 To draw a colour splash use a normal distribution. More and bigger dots in the center, less and smaller the more far from the center. Both, position ***and*** size of the dots can be generated with random numbers with Gaussian distribution. The size would then be dependent on the distance to the center: bigger dots are in the center, the more far it gets away from the center, the smaller



the dots get - and this with a Gaussian distribution.
For the Gaussian number generator, however, take the example NOC_I_4_Gaussian from the examples from Shiffman, not the older version from the book!

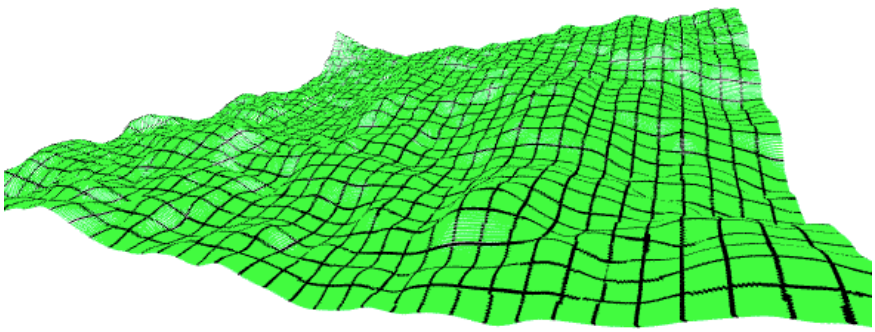


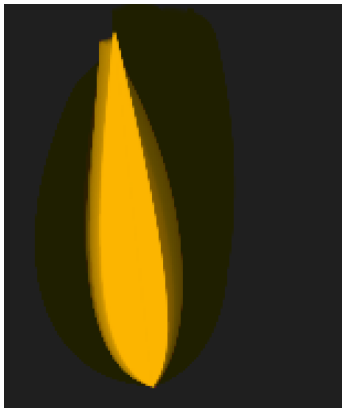
👉topic: random function, array of objects
2.4 The picture with the red dots (fragment from a picture of Yayoi Kusama) is a bit more sophisticated: the dots do not touch each other or overlap. Create such a picture randomly.

👉topic: 2D Perlin noise and seeds
2.5 Use the example Noise1D from the examples of the Nature of Code and extend it to two dimensions, i.e. the creature can not only walk right and left, but also up and down. Note that you need different increments in x- and y-direction, otherwise the creature will only move in diagonals. (why?).

👉topic: 3D Perlin noise
2.6 Use the example Noise2D, and extend it to three dimensions, i.e. the third dimension is time and the cloud changes in time.

👉topic: 3D Perlin noise
2.7 Make a landscape using two-dimensional Perlin noise explained in 1.6 of http://natureofcode.com/book/introduction/ The grid lines are necessary, otherwise the landscape looks exactly like the Noise2D example.

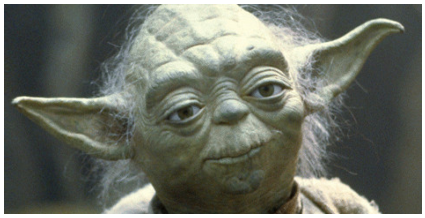




👉topic: Perlin noise in a different application
2.8 Create a moving flame using Perlin noise (and e.g. a bezier shape). Think about which parameters of a flame are most useful to modify.

Algorithms in Creative Technology

topic 3: Forces

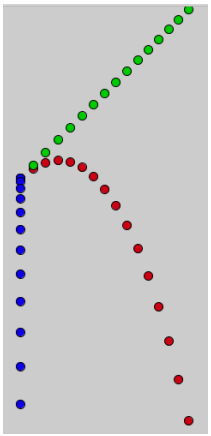


topics: forces, elementary vector operations

due date for black assignments: 17.5.2019 20:00

👉topic: speed, acceleration, gravity

3.1 Write an application of a catapult. For the simple case you can assume that you have a fixed force and a fixed direction in which an object is accelerated. The object has then a velocity that decreases by friction, and the force of gravity pulling it downwards. Use for the acceleration, force, gravitation, speed and position PVectors as data structure. The movement of the object is basically composed of two movements, that can simply be added up. One is the movement of the object in the direction of acceleration, together with friction resulting in a trajectory of the green ball in the illustration. The second is the force and acceleration by gravity as also in the bouncing ball example and illustrated by the trajectory of the blue ball. The combined (added) movement is the the trajectory of the red ball. (Solutions to slightly similar problems, copied from, e.g. Processing, fall under plagiarism.)



👉topic: speed, acceleration, gravity

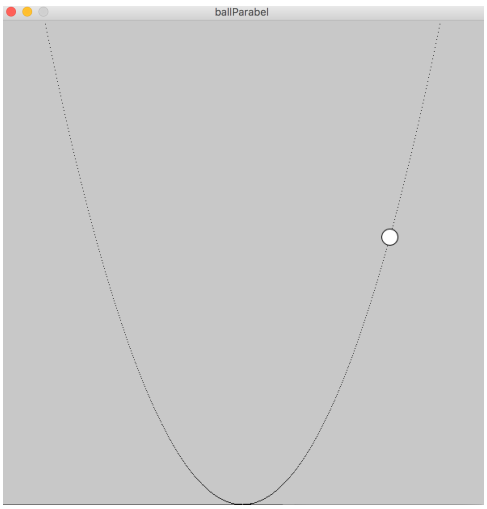
3.2 Make the catapult application of the previous assignment interactive where the user can, e.g., draw a catapult or bow, can determine the forces by the distance and direction he draws is, and the releases it. (Interpret "draw" in the bow and arrow sense, perhaps like in angry bird, not in the Processing sense!)

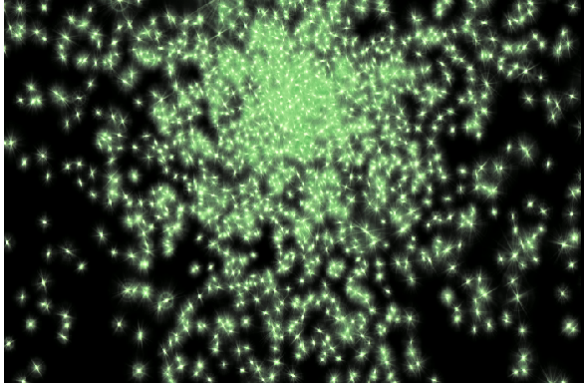
👉topic: speed, acceleration, gravity, frame rate independence

3.3 In all examples so far, we assumed time steps of 1 for our integration, leading to expressions like *position += speed*. With this time step assumption the speed of the visualisation depends on the frame rate of the execution. Doing the exercise physically responsible, we would need the definition of a *timestep* that depends on the frame rate, but makes the visualisation independent of the frame rate. The expression for the position would then change to *position += speed * timestep*.
Implement the catapult of 3.2 using such a time step.

👉topic: speed, acceleration, gravity

3.4 Assume a ball is rolling along a parabolic line as in the picture below, moved by gravity. Determine the force, velocity and position of the ball in each frame. Add friction, so that eventually the ball stops moving.
Using the methods of PVectors like normalize(), angleBetween(), mult(), add() makes life here much easier.





Algorithms in Creative Technology

topic 4: Particles

topics: lifespan, vector arrays and inheritance
due date for black assignments: 24.5.2019 20:00

Here, basically topics from chapter 4 of "Nature of Code" http://natureofcode.com/book/chapter-4-particle-systems/ are treated. For assignment 26 sections 4, 4.1 and 4.2 are relevant. For assignment 27 section 4.3. (array list) is also needed. For assignment 28, 4.4. is the basis.

👉topic: particle system, class dependencies

4.1 Goal of this assignment is to make a flying rocket (or other object) with particles as its tail. To achieve this effect make a simple particle system using a vector of particles. For the individual particles make a class. A particle is similar to a ball starting at some center, and heading randomly in some direction, (for the rocket example particles go preferably in the opposite direction of the rocket). A particle has a lifespan, which should be one object variable. After its life passed, a particle is not visible any more. You can create a nice visual effect when particles are fading during their life (or change colors or…).
Take care for the class structure: the rocket (or unicorn or…) should have a particle system as an object variable!.

👉topic: particle system, class dependencies

4.2 For a particle system using a vector array an example can be found in the example section of Processing, ArrayListParticles (and the following ones). Understand the array list concept and extend/change the example in the following ways:
- include different shapes of particles within one application/class (using some sort of flag, i.e. a variable for each particle that stores its form.).
- make a particle system which is one explosion, not a continuous stream
- let a new explosion take place on and at mouse click (or when two balls are bouncing or...)

👉topic: variation on the particles

4.3 Use the flame of assignment 2.8 to make a moving camp fire. Use each flame as a particle. (Try to make a nicer fire than I did - see picture.)

👉topic: efficiency of array implementations

4.4 Create two versions of a particle system that differ only in the implementation of the particles as array, resp. array list. Compare the performance.

👉topic: 3D Perlin noise as vector field for moving particles

4.5 Interpret 3D Perlin noise as a vector field and let particles flow through the vector field. There are numerous examples on the web: take them as inspiration and create your own beautiful image! (And: quote your sources!)

Algorithms in Creative Technology

topic 5: flocking

due date for black assignments: 4.6.2019 20:00

Swarm behaviour (flocking) is simple to program, but gives nice visual effects. A swarm is a collection of swarm members that are driven by a few principles:

coherence, i.e. a member does not move too far from the rest of the swarm

avoidance, i.e. a member does not bump into other swarm members or an obstacle

alignment, i.e. a member adapts its direction and velocity to its surrounding members

These three effects added up result in a (new) speed and direction of a swarm member. In each frame, each swarm member considers only its neighbours (and possibly a obstacle) to calculate a new speed and direction. It is not looking at the whole swarm, but only in its direct environment, i.e. all swarm members that have at most a certain distance.

Chapter 6 of "Nature of Code" describes in great detail the concepts and effects
In Processing an example is already given, (NOC_6_09_Flocking). You may use the example to build on your program. (And be aware of that when you use this example your solution will probably be less nice than when you write your own code from scratch - flocking algorithms were for me the first evidence that given code can restrict phantasy in comparison to self-written code.)

👉topic: playing with a given flocking algorithm

5.1 Choose flock members different from the triangles. Adapt the weights of the different effects that they reflect a kind of flocking that fits to your flock members, i.e. cows flock differently as fire flies or fishes. Adapting weights requires some playing to get the intuition for the effects. Do **not** let move flock members over the boarders (and appear on the other side of the screen, which is kind of non-intuitive). Add an obstacle that is avoided by the swarm. Change the flock movement such that the flock members (bolds) do not move over the side of the window, but "bounce".

👉topic: flocking from scratch

5.2 Write the code for flocking yourself, based on pseudocode on flocking.
Put the methods for the flocking behaviour (such as alignment, coherence etc. in the flock class, not the class of the flock members, as this behaviour only makes sense in the context of a flock).

👉topic: flocking and interaction

5.3 Add interaction, an enemy or attractor moved by the user. (Or any other concept of interaction.)

Algorithms in Creative Technology

topic 6 : Flower moving in the wind

due date for black assignments: 11.6.2019 20:00

topics: translating physics to Processing , rotational spring-mass-damper system

These are the most difficult assignments of the course. The intention here is not that you take the inverted pendulum from the "Nature of Code", but implement the mass-spring-damper system from dynamical systems. The physics you need for these assignments are known from the book "Dynamical Systems" (see, e.g., exercise 7.40 there).
The real goal of this assignment is to **model a flower or grass moving in the wind**, or, **alternatively**, **water drops falling on a water surface**. In the easiest version of this assignment one spring-mass system has to be made, a rotational one for the flower, a linear one for the water drops. The next step is to combine a number of such systems. For the flower case they are added on top of each other, approximating the natural look of a stem and a flower (where forces and velocity is passed between the systems as shown in the block diagram below). For the water drop case each surface point of the water is modelled by a mass-spring-damper system and when moving, a surface point receives also forces from its neighbours.

The equations for a rotational spring are:

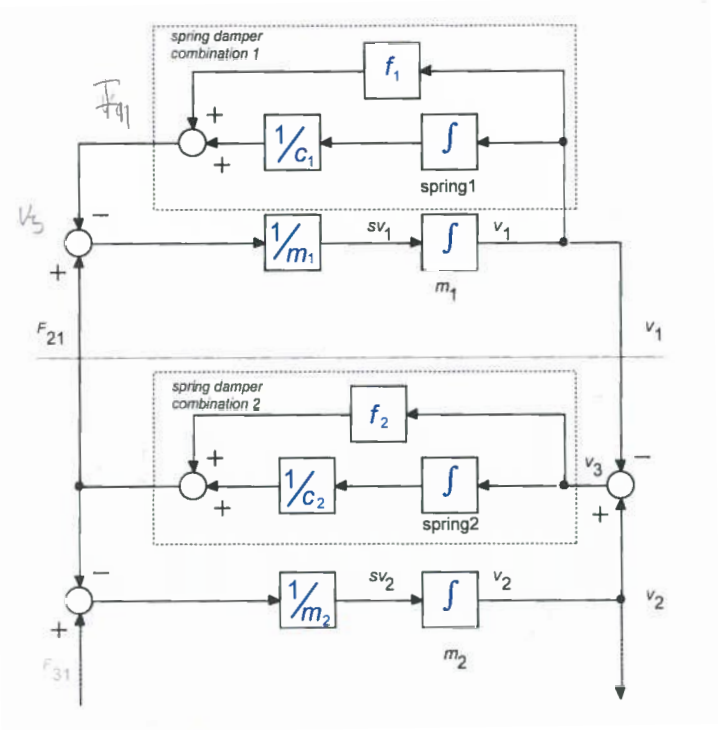
The torque is the force in the rotational case,
where c is a spring constant and ω the rotational speed:
$T = 1/c \int \omega \, dt$

The friction is described below, where f is a constant characterising the friction:
$F = f * \omega$

By the law of d' Alembert we have:
$T + F + m \, d\omega/dt = 0$, where m is the mass

Note that $d\omega/dt$ is the acceleration and $\int \omega \, dt$ the angular rotation (position).

The figure on the left shows a block diagram of two mass-spring-damper systems. In the first step, we concentrate on the lower one, below the dashed line (because this is the more generic one that can be stacked).
What we basically do, when programming the system, is the following: for each of the crossing points in the diagram, labelled with v2 and v3 on the right, and F21 on the left, we introduce a variable. Each "round" of calculation in Processing, these variables are then updated with new values as described by the diagram. In Processing we always take as step size for time simply 1. Accordingly, an integration is always represented as an addition (as we had earlier position += velocity,



the new position is the old position plus velocity in time interval 1 - try to do the integration in this assignment similarly.)
It is therefore useful to have an additional variable representing the point between spring2 and 1/c2, which physically is an angle (speed in a rotational system produces not a new position, but a new angle).
Altogether, the diagram above then represents already all the calculations you have to do (these are 4-5 lines of code in the end). What remains to do is to define all the variables and parameters needed and giving them suitable (initial) values. The mass-spring-damper system modelling a stake requires passing of forces and velocities from the system below and the system above, respectively.

See chapter 3 of "Nature of Code" http://natureofcode.com/book/chapter-3-oscillation/ as inspiration, and background on rotational movements etc., but be aware that the solution asked here is a different one.

Below are two sets of assignments that can be done as **alternatives**. Either you do the flower example or the water drop example. Or both :-)

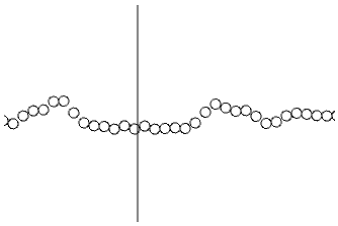
👉topic: translate a physical concept into code
6.1 Write a class for a rotational spring-mass-damper system and visualise it.
The choice of the proper parameters is crucial here, otherwise the systems easily behaves in a chaotic way. The parameters woking in my solution are: 100 for the spring constant, 0.1 for the damping constant.

👉topic: physics and smart method structure
6.2 Building upon assignment 6.1 make a class that adds a number of spring-mass-damper systems on top of each other, which will give the flower or grass in the end. Take care that the relevant forces and velocities are passed between the segments, i.e. just stapling does not reflect the physics!

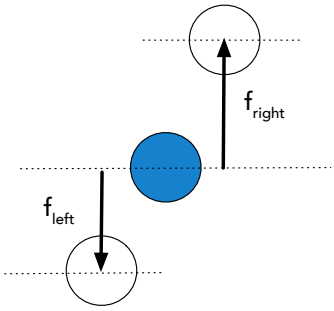
👉topic: multiple mass-spring-damper systems with external force by Perlin noise
6.3 Building on assignment 24 make a field of flowers or a grass field that is moved by the wind. Note, that the force of the wind is applied to the individual elements not at the same time, but with some delay.
Grass fields moved by Perlin-noise wind can look beautiful!!

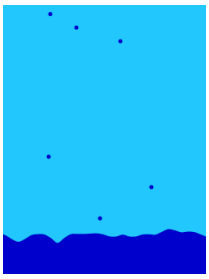
The next assignments are an **alternative** to the set above and the idea here is to visualise a water surface on which drops are falling creating small waves.
The water can be considered as an array (one-dimensional for 2D or two-dimensional for 3D) of mass-spring-damper systems. To model an individual mass-spring-damper system you can use a fragment of the block-diagram above and the related formulae.
However, passing forces between neighbouring mass-spring-damper systems works differently.

👉**topic: translate a physical concept into code**
6.4 Write a class for a linear mass-spring-damper system and visualise it. Introduce an interaction where the user can add force. Use the formulae from the lecture and be able to explain them!



👉**topic: physics and smart method structure**
6.5 Combine a number of single mass-spring-damper systems to a water surface. Each surface point has its own dynamics, as elaborated in 6.4. Additionally, it receives also forces from its direct neighbours. The force received by a neighbour is also a spring-force: the difference in location determines the displacement and, multiplied by the spring constant, gives the additional force from this neighbour's side.





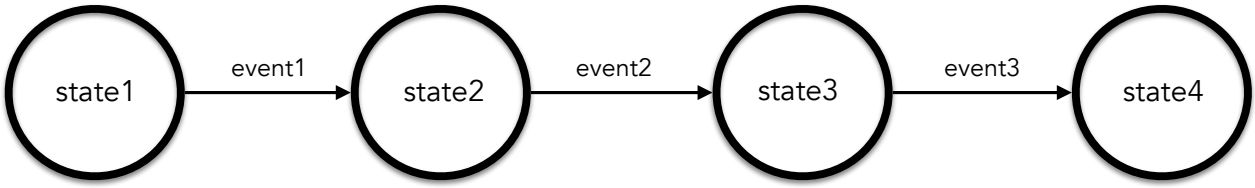
👉**topic: multiple mass-spring-damper systems with external force by Perlin noise**
6.6 Generate rain falling on a water surface and moving it.

Algorithms in Creative Technology

topic 7: State Machines

due date for black assignments: 14.6.2019 20:00

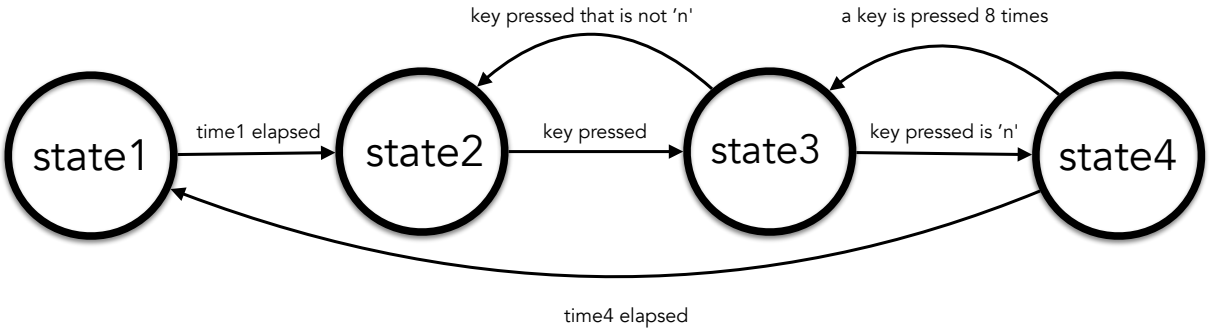
State machines are a concept of organising you designs and code. Imagine, you have a game with different levels. Each level could be considered as a state, where the game progresses from one state to another when a certain level of points is reached. Or you make a firework show with different phases, where after a certain time the next fireworks start. You can realise these effects without the concept of a state machine, but having the concept makes it conceptually easier.
Below is a diagram of a state machine.



An event can be a click of the mouse, a number of points reached, a time limit reached, or similar. Furthermore, a state may have different successors. Which one it enters, depends on what happens in the state before. Imagine a mouse click on a door, that lets the player enter the next room. Clicking on another door would lead to another room.

On blackboard we have an example of a state machine, that behaves as illustrated by the following drawing. It shows the different possibilities, you can modify it according to your needs.

👉**topic: see and understand a state machine example as it can be useful**
7.1 Use the state machine on Canvas to create an own example, of a firework, weather changing, etc.



The Endassignment

Below are 4 sets of requirements for your end assignment. Make sure you satisfy them all.

1. Scope of applications:
In the end assignment you are expected to combine at least 3 from the 4 topics in one application:

- **Randomness**, normal distribution **and** Perlin noise. Make sure that the application of the respective form of randomness is **meaningful**: if you, e.g., substitute plain randomness in some example simply by Perlin noise, which does not make a visual difference, then this would not be meaningful.
- **Forces**. This does not mean that the pure usage of PVectors is sufficient. Choose one of the examples with catapult or billiard or a mass-spring-damper system, or something at least equivalent in difficulty.
- **Particles**
- **Flocking**

Also the level of solutions you chose (black, blue, green) will influence or determine your grade (6,8,10).

2. Self written code & understanding

- At **least two (nontrivial) classes** have to be written by yourself, i.e., two classes at least not by Shiffman, and not from a tutorial (resulting in identical code for everyday who has written this code himself following the tutorial), or elsewhere from the web.
- **Flocking and particles should not both be from Shiffman** (or other sources on the web), at least one of them self written.
- It will count how much **code you have written yourself**. A major part of **significant complexity** has to be written by yourself.
- You should be **able to explain** the complete program (each of "*oh, that is already … weeks ago that I wrote this, I really cannot remember*", or "*my partner wrote that part*", or "*I tried a couple of things and this suddenly worked, I do not know precisely why*" are equivalent to a fail for the question raised.)

3. Complexity
The complexity of your program is determined by:

- How much interaction is there with the user.
- How much interaction is between the classes. Does an event in one class have an effect on objects of other class?
- How many classes are there. If you have written sufficiently self written code you can always improve your program by adding other assignments, code from somewhere else (but do not forget to give reference to your sources!!!)

4. Programming Style & Code Quality
The points below are the same as we had already in module 1 and 2.
When you have your final assignment working, a cleaning up will be necessary.

- Keep the main draw function as small as possible.
- Do not use unnecessary global variables, especially not for value passing.
- Have constants for your program as global variables (e.g. number of players, etc.), not hidden in classes.
- Do not hard code parameters that may change during the program development, e.g. make the size of an array a variable, and use this variable in all loops as bound. Also do not hardcode graphical elements that you might change.
- Do not hide user interaction in classes. Deal with user interaction in the following way: detect the interaction in the main part, in functions like void *mousePressed()* or similar. In the body of these functions you should call the event handling method of the respective class. Also avoid variables *mousePressed*, etc., use methods instead.
- Have things that belong together logically, in one class. For example, if you have a rocket with a fire tail, put rocket and fire tail in one class that organises them both, not in two different classes, where objects accidentally are at the same position).
- When classes become too large, split them in a logical way.
- Use arrays (and loops) when you have similar values.
- Use functions when you have similar code.
- Do not have unused variables.
- Do not have unused code.
- Have comments and a header (ask, if you do not know what this is.)

Often when programs grow during time they will not have an ideal structure following all the criteria above. After having you program working functionally, a round of cleaning up will be necessary in most cases.

Oral exam

In the oral exam you will show your end assignment, and answer questions on it.
We will also examine the course assignments, especially when not treated in the end assignment, i.e. we expect that you have done all assignments of the course of the level chosen. The versions of the code you have uploaded to GitHub will be the versions that we consider. If you have covered a topic in your end assignment, we will not look at your course assignments. A good strategy is therefore to cover as many topics in your end assignments as possible (for the storyline of your end assignment).