VA: Graphical interfaces and event-driven programming

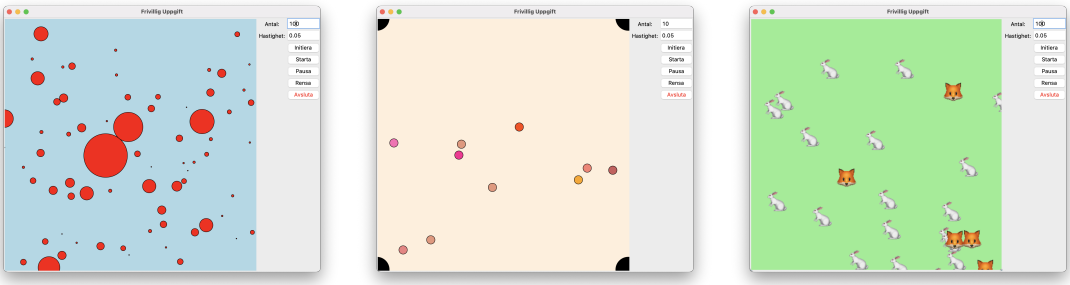
- This assignment is not mandatory. This assignment is for extra points during the exam. Extra points are given for this course instance only (exam or re-exam).
- This voluntary assignment concerns writing a program with a graphical user interface (GUI) and event-driven programming.
- To present VA, describe your code and demonstrate its functionality.
- When the assignment has been approved you should upload the code to STUDIUM.

Tip: If your program is a game, i.e. a TA can move an object, it is easier to pass the assignment. Especially, if it is a game for two e.g. table hockey.

Important: You may collaborate with other students, but you must write and be able to explain your own code. You may not copy code neither from other students nor from the Internet except from the places explicitly pointed out in this lesson. Changing variable names and similar modifications does not count as writing your own code.
Since the VA is included as part of the examination, we are obliged to report failures to follow these rules.

In this assignment you will write a program that simulates objects (e.g., balls) that move in a box. There should be a graphical user interface to specify certain properties (e.g., the number of balls and their radius), start, stop, etc.

What happens in the box is fully up to you. Some possible examples are shown below; You see the graphical interface and the sequence of events are illustrated graphically either in the same or a separate window.



The moving objects

We call the obects balls but they can be any type (atoms, turtles, foxes and rabbits, people, ...). The balls should have a position (a vector in the $x-y$-plane), a velocity (also a vector in the $x-y$-plane), and a size (e.g., a radius) and/or a type (e.g., fox/rabbit or different colors). You may add as many properties as you like.

The objects should move in the direction of the velocity vector. This velocity can be constant as long as the object does not collide with something. However, you can also change the velocity vector by introducing gravity or friction.

When there is a collision between objects and the walls of the box, you should typically change the velocity of the object by standard rules, that is, angle out is equal to angle in.

What happens when two objects collide with each other is up to you, but something should happen! Examples:

- They can bounce against each other (like billiard balls). Formula below.
- They can merge with each other to become a larger ball (like soap bubbles)
- One of the objects can remove the other object (like foxes and rabbits)
- The two object can, under suitable conditions, produce another object (like two rabbits become three)
- If one obejct is contaminated, it can transmit a disease to a non-contaminated object.

Requirement: The number of objects should change during the simulation. For example, if simulating a game of billiards one should have holes where the balls can disappear. Object can also have age attribute so they can die, for example with increased possibility with age.

Minimum requirements of the graphical interface

There must at least be buttons to start and stop the simulation. One can also have buttons to pause and resume (voluntary). There should at least be text fields to specify

- number of objects
- speed of the simulation (e.g., number of milliseconds between each move/frame)

The fields should have default values.

The fields should be read when clicking the start button. One should be able to start a new simulation, with other values, without restarting the program.

Graphical user interface (GUI)

We advice you to use `Tkinter` to build the GUI, since it is included in all Python distributions. However, you can use any framework you wish, but it might be harder to get help.

`Tkinter` documentation: https://docs.python.org/3/library/tkinter.html

`Tkinter` tutorial: https://realpython.com/python-gui-tkinter/

Other frameworks for graphical interfaces: https://wiki.python.org/moin/GuiProgramming

To get more insights about the possible code structure, here is a link to a student's project in the course instance HT24: https://scratch.mit.edu/projects/1085215068/. Your code should be written in Python.

Tips!

- Generally: Take small steps! Add and test one feature at a time! Use `git` to make backups!
- Classes like `Vector`, `Box`, `Ball`, `Fox` can be a good idea to use (and can often be tested

without a GUI).

- If object i should bounce with object j, then their new veolicity vectors $\mathbf{v}_i$ and $\mathbf{v}_j$ are given by the formulas,

$$\mathbf{v}_i = \mathbf{v}_i + \frac{(\mathbf{v}_j - \mathbf{v}_i) \cdot (\mathbf{s}_j - \mathbf{s}_i)}{|\mathbf{s}_j - \mathbf{s}_i|^2}(\mathbf{s}_j - \mathbf{s}_i)$$
$$\mathbf{v}_j = \mathbf{v}_j + \frac{(\mathbf{v}_i - \mathbf{v}_j) \cdot (\mathbf{s}_j - \mathbf{s}_i)}{|\mathbf{s}_j - \mathbf{s}_i|^2}(\mathbf{s}_j - \mathbf{s}_i)$$

  where $\mathbf{s}_i$ and $\mathbf{s}_j$ are the objects position vectors. The operation $\cdot$ is the scalar product. You can ignore the case when more than two objects collide at the same time.

To get more insights about the possible code structure, here is a link to a stu- dent's project in the course instance HT24: https://scratch.mit.edu/projects/1085215068/. Your code should be written in Python.