Assignments

1. Events

☞ Before starting this assignment, read the theory about **Events**.

Write a program using a SnakeUserInterface of 40×30 which has the following features:

- Clicking on a square results in a piece of wall to be placed on that square.
- Pressing the space bar erases all the walls.
- The program prints the name and data of all events that occur.

2. Animation

☞ Before starting this assignment, read the theory about **Animations**

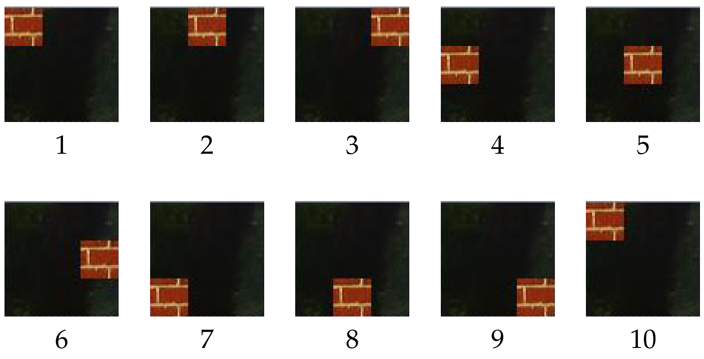
The goal of this assignment is to make an animated program in which a piece of wall moves across the screen. The piece of wall starts out on (0,0) and moves right a square at a time. Upon reaching the end of a row, the piece of wall will move to the first square of the next row. When the piece off wall reaches the end of the last row, it is transferred back to the initial (0,0) position.

On top of this make sure the program implements the following features:

- The animation should slow down 0.5 frames per second when $\leftarrow$ (left arrow) is pressed.
- The animation should speed up 0.5 frames per second when $\rightarrow$ (right arrow) is pressed.
- The piece of wall should change into a green sphere (a part of a snake) when g is pressed. Pressing g again will revert the change.

Use the SnakeUserInterface for this assignment.

Example



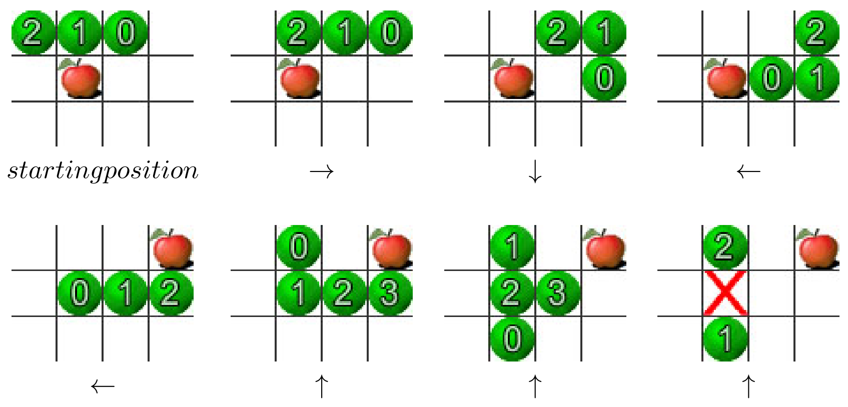
Graded assignment

3. Snake

☞ Before starting this assignment, read the theory about **Stepwise refinement**. In addition, before starting with programming, a draft of the program **has** to be approved.

A logical step forward from interactive animated programs is games. The goal of this assignment is to program the classic computer game, *Snake*.
The goal of Snake is to create a snake as long as possible. This is achieved by guiding the snake to apples, lying about on the field. The snake cannot stop moving, and dies whenever it hits something. Because the snake is growing longer and longer as the game progresses, it is increasingly difficult to avoid collisions with the snake itself.
At the start of the game, the snake consists of two pieces at the coordinates (0,0) and (1,0). As said before, the snake is always moving. At the start of the game, it moves to the right. When the user presses one of the arrow keys, the snake changes direction.
At every moment in the game, there is always an apple somewhere in the field. If the snake hits an apple, the snake becomes one piece longer in the next screen refresh. A new apple is placed on a random location, excluding all places covered by the snake.
When the snake reaches the end of the screen, it will re-emerge at the other end.

Example The example below shows a short game of snake, played on a 4x3 field. The game to be designed in this assignment will have a field measuring 32x24. The arrow indicates in which direction the snake is traveling. The numbers on the snake indicate its position in the row.



This assignment uses the SnakeUserInterface.

Bonus Edit the program in such a way that it accept a level as input. A level defines a number of walls, which the player has to avoid. Levels can be found on Canvas. The structure of these files is as follows: first the coordinates at which the snake is initialized are given followed by an =. Next, the initial direction of the snake is given, again followed by an =. Finally, all the coordinates at which to place walls are given.

Coordinates are formatted in the following way: one coordinate per line, in the format: x<space>y. The initial direction is one of four strings: `"L"` (Left), `"R"` (Right), `"U"` (Up) of `"D"` (Down).
An example of a piece of such a file:

```
1 0
0 0=R=3 3
4 3
5 3
6 3
7 3
8 3
etc...
```