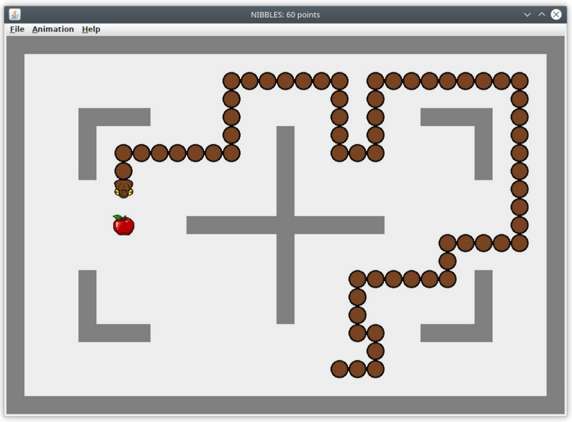
Project NMCGJ 2025-2026: Nibbles Game

The aim of the project is to design and implement a video game, called Nibbles. Nibbles is one of the legendary games like Pacman and Tetris. Despite it simple rules, there are many variants on the market.

Gameplay



Nibbles is a game in a two-dimensional world, where one or more virtual snakes live. The player has to navigate its snake through this world using the keyboard. A snake can move in four directions (left, right, up, and down) but not diagonal. There are not only snakes in this world, but also walls and food packages.

The player must navigate its snake such that it consumes as much food as possible. Each time the snake eats food, it grows a little bit in length and some points are collected. When a food package is eaten, it disappears and a new food package appears at a different random position.

A snake dies when it hits a wall or a snake (including its own body). The game is over when all snakes are dead. When playing with more than one snake, the longest snake is the winner; this is not necessary the snake that survived till the end.

The figure shows a possible snapshot of the game. The game board is completely surrounded with a wall (dark gray) and there are also some walls in the interior. There is a food package (a red apple) in the left part of the board, and we play with just a single snake (with a brown body and the head can be distinguished by its different shape and yellow eyes). At the moment of the snapshot, the snake has eaten already 12 food packages.

Problem

The game Nibbles has to be implemented for this project. The implementation needs to be flexible, so that the number of snakes and walls can be easily modified. For walls, also the position and length must be adaptable. Note that there are several “board objects” (namely snakes, walls, and food packages) which share several common properties like a position, color, etc. This suggests that these properties could be collected in a class, and that concrete classes for these objects could be constructed by means of inheritance.

The board must be completely surrounded by a wall (like in the example), and a certain number of walls can be placed in the interior of the board. There must be a single food package on the board. A snake has an initial length of 5 blocks. Moreover, its initial position on the board has to be chosen randomly, but such that the snake cannot die after its first step.

When dealing with different snakes, they should have a different color.

In each step of the game, each (living) snake should check whether the corresponding player has pressed a key to change its direction. If this is the case, then the snake moves one position in this new direction, otherwise it continues to move one position in the current direction. Moving the snake actually means that only the head and tail are changing. If the snake dies (because there was a collision with some object on the board that is not a food package), then the snake stops to move...

Every time a snake eats a food package, its length grows 5 blocks. This growth happens step-wise: the snake grows only one block in each step. This means that only the head moves but not the tail of the snake during the next 5 steps. When a food package is eaten, it should disappear and a new food package should appear at a different random position (which is not yet occupied by another object).

In order to simplify the task, the following classes have been prepared.

BoardPanel	A panel for working with a graphical gameboard.
BoardFrame AnimationBoardFrame KeyAnimationBoardFrame	A frame containing a BoardPanel ... with animation ... with keyboard interaction.

The class BoardPanel

The class `BoardPanel` provides a panel for simple drawings arranged into a board consisting of a certain number of rows and columns. This class is very similar to the class `GraphicsPanel`, but is more efficient in drawing and removing specific objects at certain board positions.

BoardPanel(int rows, int cols)	Constructs a board panel.
getRows()	Returns the number of rows of the board.
getColumns()	Returns the number of columns of the board.
isInside(int i, int j): boolean	Checks whether the position (i, j) is inside or not.
clear()	Clears the complete board.
clear(int i, int j)	Clears the component at board position (i, j).
drawRectangle(int i, int j, Color color, int density)	Draws a filled rectangle at board position (i, j) with a given color and density/size.
drawOval(int i, int j, Color color, int density)	Draws a filled oval at board position (i, j) with a given color and density/size.
drawLine(int i, int j, Color color, int dir, int density)	Draws an axis-aligned line at board position (i, j) with a given color, direction, and density/size.

drawCross(int i, int j, Color color, int[] dirs, int density)	Draws an axis-aligned cross at board position (i, j) with a given color, array of directions, and density/size.
drawImage(int i, int j, Image image, int density)	Draws a scaled version of a given image at board position (i, j) with a given density/size.
repaint()	Repaints the board panel

The class BoardFrame

The class `BoardFrame` provides a frame with a board panel. It inherits from the class `JFrame` which is part of the package `javax.swing`. The following methods are available in the class `BoardFrame`, which is very similar to the class `GraphicsFrame`:

BoardFrame(String title, int rows, int cols, int size)	Constructs a board frame with a given title, and the dimensions of the board.
start()	Visualizes the frame.
close()	Closes the frame.
setTitle(String title)	Sets the title of the frame.
setMenuVisible(boolean visible)	Indicates whether the menu bar is visible or not.
setGraphicsDimension(int width, int height)	Sets the preferred dimension of the board panel.
getBoardPanel(): BoardPanel	Gives the board panel of the frame.

Some simple dialog frames are also available:

showMessageDialog(String msg, String title)	Shows a dialog with a given message.
showInputDialog(String msg, String init): String	Shows a dialog requesting a string.
showInputDialogInt(String msg, int init): int	Shows a dialog requesting an integer.
showInputDialogDouble(String msg, double init): double	Shows a dialog requesting a floating point.

Some useful methods for reading and writing images

readImage(File file): BufferedImage	Reads an image from a file.
writeImage(File file, BufferedImage image)	Writes an image to a file.

To change the actions of the menu bar, the following methods should be overridden:

clearBoard()	Called by the menu File > New.
loadGraphicsFile(File file)	Called by the menu File > Open.
saveGraphicsFile(File file)	Called by the menu File > Save.

The class AnimationBoardFrame

The class `AnimationBoardFrame` is prepared for making animated 2D graphics. It is a subclass of the class `BoardFrame`, and starts a new thread for each animation. The following specific methods are available to run an animation:

playAnimation()	Plays the animation when not currently active or resumes the animation when paused. It has no effect if the animation is already running.
pauseAnimation()	Pauses the animation.
stopAnimation()	Terminates the animation.
isAnimationEnabled(): boolean	Indicates whether the animation is enabled or not.
isAnimationPaused(): boolean	Indicates whether the animation is paused or not.
setAnimationDelay(long millis)	Sets the delay time (in milliseconds) between each animation step.

The class `AnimationBoardFrame` is an abstract class, and provides an animation environment. It requires the implementation of the following abstract methods (similar to the class `AnimationGraphicsFrame`):

animateInit()	Initializes a new animation (is called before the start of the animation).
animateNext()	Executes the next step in the animation.
animateFinal()	Finalizes the animation (is called after the end of the animation).

These methods have to be implemented by a subclass and define the specific animation.

The class KeyAnimationBoardFrame

The class `KeyAnimationBoardFrame` is prepared for interacting with the keyboard. It is a subclass of the class `AnimationBoardFrame`. The class is listening to key events, and whenever a key is pressed the following abstract method is called:

processKey(KeyEvent e)	Processes the given key event.

This method has to be implemented by a subclass and specifies the action that has to be undertaken when a key is pressed. Information about the specific pressed key is passed by means of an instance of the class `KeyEvent`.

The class `KeyEvent` is part of the package `java.awt.event`, and is designed for passing key information. Each key has a specific key code, and can be retrieved by the method

getKeyCode(): int	Returns the code associated with the key in this event.

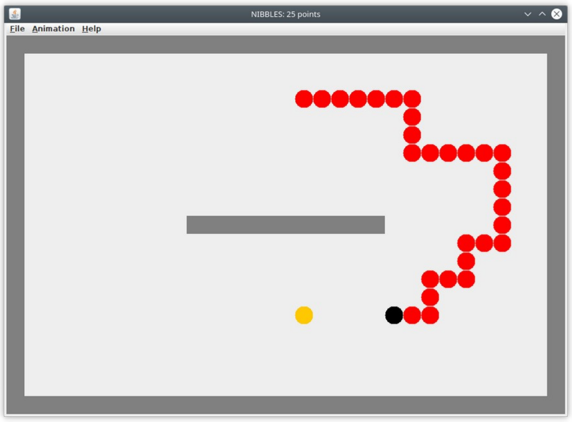
The integer key codes are stored in static constants in the class `KeyEvent`. Some useful examples are given by

VK_<C>	Code of the key with the letter or number <C>. This can be A-Z or 0-9
VK_UP, VK_DOWN, VK_LEFT, VK_RIGHT	The codes of the arrow keys.

More info can be found on the webpage: https://docs.oracle.com/en/java/javase/25/docs/api/java.desktop/java/awt/event/KeyEvent.html

Minimal requirements

- The game works with a specific wall-configuration and with a single snake (player).
- The implementation must allow to place the walls, the snakes, the food on the board at arbitrary places. For the walls and snakes, the position and length must be adaptable.
- Simple graphical shapes (rectangles, ovals, lines, etc.) are fine for the visualization of the walls, the snakes, the food. However, use different shapes and/or colors to distinguish between the different board objects.
- A good Java program design.



Suggestions for further options

- The option to load a user-defined wall-configuration from a file. For this purpose, you can use the classes `CoordinateIO` and `Coordinate` (see the pre-project for their description).

- The option to play with different players.
- The option to play with different levels of difficulty (for example, more walls, higher speed of the snake, etc.). When a certain number of food packages are eaten, the player(s) move to the next game level.
- The option to play with different “lives”. Whenever the snake dies, it can resurrect consuming one of its “lives” and the game ends when none of those “lives” remains. You can add the possibility to increase the number of “lives” by eating special food packages.
- Instead of composing the board objects (the walls, the snakes, the food, ...) of simple graphical shapes (rectangles or ovals) of different colors, they could also be composed of more fancy images. This could be done with the aid of the pre-defined methods `readImage` and `drawImage` from the classes `BoardFrame` and `BoardPanel`, respectively. Their use is illustrated with the following example code:

```
BoardFrame frame = new BoardFrame("My Board Frame", 10, 10, 40);
BufferedImage image = frame.readImage(new File("MyImage.png"));
BoardPanel board = frame.getBoardPanel();
board.drawImage(1, 1, image);
frame.start();
```

- Surprise me...

Notes

- A good Java program is not just a program that produces “the right result”; it should also be designed properly. In a good program design, every class (and method) should be responsible for a single well-defined job.
- Do NOT modify any of the pre-defined classes `BoardPanel`, `BoardFrame`, `AnimationBoardFrame`, `KeyAnimationBoardFrame`, `CoordinateIO`, and `Coordinate`.

Practical Information

The deadline for the project is **January 11, 2026**. Your report of the project can be turned in electronically by sending an email to speleers@mat.uniroma2.it. Such a report should include:

- the source code of your program;
- a class diagram of your program;
- an overview of your program design decisions.

Good luck and have fun!

Hendrik Speleers