Grundlagen der Programmierung - Winter 2021/2022

Assignment 10 - Programming in the Tileset Universe

This assignment is the final one of this course and we put all the pieces together. Use all your great new programming skills to create a game based on the provided tileset framework! You are now capable of using classes for different types of game characters or objects as well as graphic commands, media files, conditions, loops, formulas, arrays, and functions to realize your game design in detail.

This assignment is a group assignment (2-3 persons). Please distribute all work evenly among all team members. Everybody has to contribute significant parts of the code.

Deadline (EXTENDED TO THE NEXT THURSDAY)

Original deadline: Sunday, February 13 2022, 20:00.

New deadline: Thursday, February 17 2022, 20:00.

Submission Details (Important)

- Please combine all files into one *ZIP* file named *gdp_10_surnames.zip* and upload it in Stud.IP to the folder *Upload/Assignment 10*. Example: Your group consists of people having the name *Schmidt*, *Meyer*, and *Mueller*. Then you should submit a file named *gdp_10_schmidt_meyer_mueller.zip*
- The ZIP file should contain a folder that has the name *gdp_10_surnames* (see example at previous bullet point). Inside the folder, there should be one *PDF* file with the description of your game. We will correct this PDF file.
- In addition, we want to have your Processing program in the ZIP file so we can easily start your game. For this assignment, *your code does not need to be included in the PDF*!
- Please do not forget to mention your names in the PDF file. Please also list the time (rough estimate in hours) you needed for the assignment.

Exercise

Entering the Tileset Universe (important information, no exercise)

The type of game we want to realize is a tileset game, i.e. a game where the game world (i.e. the level) is composed of 2D square tiles from a set of predefined tiles. Many older games use this technique, for example Pacman and Super Mario Land.

Therefore, we have programmed a class `Map` which you should use and which maintains such a tileset level. The class provides methods that allow the game logic, i.e. your program, to access the level to make the game characters interact with the level in a way you define. The game characters themselves are completely in your program and outside the scope of this class.

The program `blockeditor` allows to create, view, and edit such a tileset level. You are supposed to use it, not to understand it. The program `gamedemo` is an example for a simple tileset game, simpler than the one you are supposed to program.

Look at `gamedemo` 's level using `blockeditor` , the graphics in `images` and the processing program `gamedemo` itself and understand how it works. Keep in mind:

- each level is stored in a `.map` -file and can be viewed and edited with `blockeditor`
- every tile (e.g. floor, wall, trapdoor, etc.) is identified with a unique letter. (in `gamedemo` : `F` for the floor, `W` for the wall, `E` for the goal ...)
- Everything outside the actual level is the special tile `_` .
- To draw a tileset level, the `Map` class needs graphics for each tile type. It therefore loads the graphics from the `images` folder in the same directory as the level file. These images must be `.png` files with the name of the tile set type, i.e. a letter. So in case of `blockeditor` , the `images` folder contains `E.png` , `H.png` , `S.png` ...
- *All tiles must be of the same size and square!*
- The graphics `_.png` (when it exists, not in `blockeditor`) is shown in empty tiles, in particular outside the defined level.

The `Map` class is used by `gamedemo` or later your program to

- draw the level (`draw`)
- query, which block is at a specific point (`at, atPixel`)
- change the block at a specific point(`set, setPixel`)
- query, whether a specified rectangle contains a certain tile typ (`testTileInRect`). This is used to let the game character or objects interact with the level.
- query, whether a specified rectangle consists only of certain tile types (`testTileFullyInsideRect`).

Game Concept and Presentation

Develop a concept for your game, which you will program in the final assignment. The game must be tileset-based using the `Map` class and one or several levels designed with `blockeditor`. Apart from the level itself, i.e. the `Map` object, there must be at least two types of game characters or objects types, for instance a player and opponent(s). Of one of these types there must be several objects or characters with a number that varies over time, e.g. opponents that appear, move around for a while, and finally disappear.

Hold a short presentation (a few minutes) about your idea in your Tutorium on February 3/4. This presentation is not obligatory and is not awarded with points. However, to get valuable feedback from your tutor as well as from your fellow students, it is highly encouraged!

Your presentation should address the following points:

- What is the goal of the game?
- Which player characters exist and how are they supposed to be controlled?
- What other objects are part of the game?
- Design (at least preliminary) graphics for the tiles and make a (preliminary) level using `blockeditor`.
- Explain specifically of what tiles the level shall consist and what each tile should do.

Program Your Game! (15 Points)

Program your game in Processing. Remember to add all graphics, sounds, etc. needed by the program into the appropriate folder inside the .zip file so we can directly start the game. Please pay attention to the following aspects:

- It has to be an Actual Game that can be Played

Please pay attention to required aspects of the game that have been described in Assignment 9. Your game must be startable and should not crash. Overall, it should make some sense as a game.

- It has to be a Tileset Game

Your game has to be a tilest game and has to make use of the provided `Map` class. However, you are free to choose the size and the content of the tiles. You are also free to extend the `Map` class. Please document any changes that you made.

- Classes for Game Character(s) and Other Objects

Program self-defined classes for the different types of game characters and other objects (at least two). Use an `ArrayList` to handle a multitude of game characters / objects of the same type. Of one of these types there must be several objects or characters with a number that varies over time, e.g. opponents that appear, move around for a while, and finally disappear.

Try to adhere to the "information hiding" principle, i.e. that the classes "know" as little as possible about the environment they are used in and that the main program "knows" as little as possible about the implementation of the classes.

- Window and File Size Constraints

The window size of your game should not exceed `1024 x 768`.

Please make sure that the included images and sound files are as small as possible (but as large as necessary) in terms of their file sizes as there is a maximum file size for a Stud.IP upload. Do not upload your game to any external hosing service, if it is too large. Instead, decrease the resolution of the included images or use a better compression for your sound files (MP3 instead of WAV, for instance).

Please note that it is not a requirement to have a game that makes use of images and sounds.

- Coding Style

In lecture 14, we will probably see and discuss common programming mistakes and examples for a bad coding style. Solutions for these common issues will be explained. Respect these guidelines and do not make these typical mistakes.

Examples (from previous lectures):

- Use global variables only for information that should be available globally in your program. If it is possible and reasonable, declare variables as local variables.
- A class should not assume the existance of variables and functions outside its scope (unless these are global Processing elements).

- Comments and Formatting

Please take care of a proper formatting of your program. Use the style of the example programs as well as Processing's *Auto Format* option.

The program must contain comments that explain what your code does. All functions and classes defined by you should have comments that explain what the function / class does. If anything has parameters or returns a value, there have to be additional comments that explain the parameters and/or return values. Furthermore, each global variable or class member also requires a comment.

- Game Structure

The game itself should display information about its controls. This could be right at the start or after the user has pressed something like a "help" key.

It should be possible to restart the game, i.e. to play again without ending and restarting the whole program. Please be careful to reset all relevant variables of your game, such as the score or any energy levels. Otherwise, things might go wrong.

It is not required (but also not forbidden) to have multiple levels.

- Documentation

Please submit - together with your game - a short documentation that describes your game. Please also add screenshots of all relevant screens of your game.

The documentation must list, who of the team has implemented which parts of your game.

It would be great, if we could show your game other students! Thus, your documentation has to contain answers to the following questions:

- Can we make your game (including the source code) available to the other students in *this* course?
- Can we show the output of your game to the students of *this* course?
- Can we show the output of your game in future courses?

There are no points for "correct" answers, but you should answer these questions in your documentation.

- Respect Copyright

Do not use any images or sounds that have a license that does not allow their usage for your game or of which you do not know the license. For every external image or sound, please list the source and the license in the documentation.

Grading

The points for the assignment distributed as follows:

- 8 - A game that can be played, uses the Map class, and respects all aforementioned contraints
- 5 - A proper use of different Classes and an ArrayList
- 2 - Documentation

However, points will be deducted for issues such as

- Bad coding style
- Bad formatting
- Buggy controls
- Too few comments
- Significantly uneven distribution of coding amoung group members
- Program crashes
- Copyright issues
- ...

Changelog of this Assignment

Version 1.1 - January 24, 2022

Set an extended deadline.

Version 1.0 - January 17, 2022

Initial document for winter 2021/2022.