Final Project Media Processing

This final project will take the rest of the semester. As evaluation every team will present its work in week 14.

The final goal is to create a game-like application where the protagonist needs to be able to navigate in a given world, to be able to attack all enemies and to gather the necessary health-packs. A flexible strategy needs to be developed, to increase the chances of the protagonist in an unknown world. The application will be build in several steps, each having a minor milestone to reach.

Use the Model-View design pattern to make the visualization as loosely coupled as possible with the underlying model.

Use smart pointer wherever possible. Use of ordinary pointers need to be motivated!

1 How to use an external library and visualize the world (W6+7)

On Toledo you will find a header file with the definition of the World – Tile – Enemy objects you will use. The implementation of all available objects can be found in a shareable library (or a dll if you prefer that). So the first step is to adapt your project settings to be able to use these things. Check the qmake manual to find out what need to be done. The given libworld.so is a 64 bit Linux library, if you need another version, you need to get the source code and build the project yourself. Keep in mind however, that this should be considered as a "given" file: *you are not allowed to change any functionality of the given code.*

Once your project can be build and you are able to get from the World class the basic information of your game: the tiles, enemies, health-packs and protagonist. Then you need to visualize everything: e.g. display every tile as a square of a greyvalue relative to its value and use something more "special" for the enemies, health-pack and protagonist (e.g. an image). Use a different visualisation for a non-defeated and a defeated enemy. The library will not guarantee that health-packs and enemies are at different locations, so your application have to check for this and when it happens you should replace the conflicting health-pack to another random position until the conflict disappears. Avoid hard-coded sizes in your visualization, your view should be resizable in some sense. Your application should look nice with different worlds of different sizes.

Make use of the functionality of the QGraphicsScene/QGraphicsView classes. The best matching examples available from Qt is "diagramscene", although too complex at some points. Make a good separation between the model of your application (the info you get from the library) and the visualisation (everything deriving from QgraphicsItem that you put in your QGraphicsScene)

At the end of this subtask you are able to visualize the world with its tiles, enemies, health-packs and a protagonist and you are able to resize the world.

2 Path-finding (W7+9)

In the final application the protagonist will not be controlled by the player but will have sufficient intelligence to navigate in an unknown world, populated with health-packs and enemies. So you need to develop a path-finding algorithm to calculate the most appropriate way to get from position (xstart, ystart) to position (xend, yend). There are of course many ways to define what the most appropriate way is, but we will use a combination of distance and the difficulty to move to a certain position (e.g. you can see every weight of a tile as a height value. Your protagonist will use more energy to climb a steep mountain than to follow a more or less flat path.) Tiles with value == 1.0 must be seen as unpassable.

A* is the most used path-finding algorithm. We will not use an existing version, but implement our own using the most appropriate data structures and generic algorithms available from STL and/or

Qt. Both team members need to be able to answer all detailed questions about this algorithm: choice of data structure, efficient implementation... Keep in mind that this is a quite demanding application, so use all possible ways to optimize its performance.

In your implementation, you may assume that the world tiles are ordered in the collection you get from createWorld(). So your left neighbour is at index-1, your top neighbour at index-nrofCols... Don't forget to take into account the border conditions.

At the end of this subtask your protagonist should be able to move from its start position (0,0) to the bottom-right tile using a minimal cost path. This should be visualized as an animation with a fixed update frequency and overlaying/changing the color of the tiles you passed by. It should be possible to adapt the relative weight factors of the current cost and the heuristic cost to clearly visualize the effect of the map. It should also work with different worlds of different sizes.

3 Strategy (W10+11)

Finally your protagonist will face a world populated with enemies and health packs. You need to develop a strategy to try to defeat all enemies with a minimal number of moves (= fastest).

Implement minimal following strategy: select nearest enemy, find out if you have enough health to defeat him/her, otherwise find first a suitable health-pack. Of course navigating in the world has also a cost: your energy level will decrease with the cost to move to the next tile (= related to their difference in value). As a consequence you only have a limited field-of-view in which you can look for enemies and/or health-packs. Once you have defeated an enemy, your energy level is restored to maximum. The visualization of the enemy needs to be changed after it is defeated and the tile becomes unpassable.

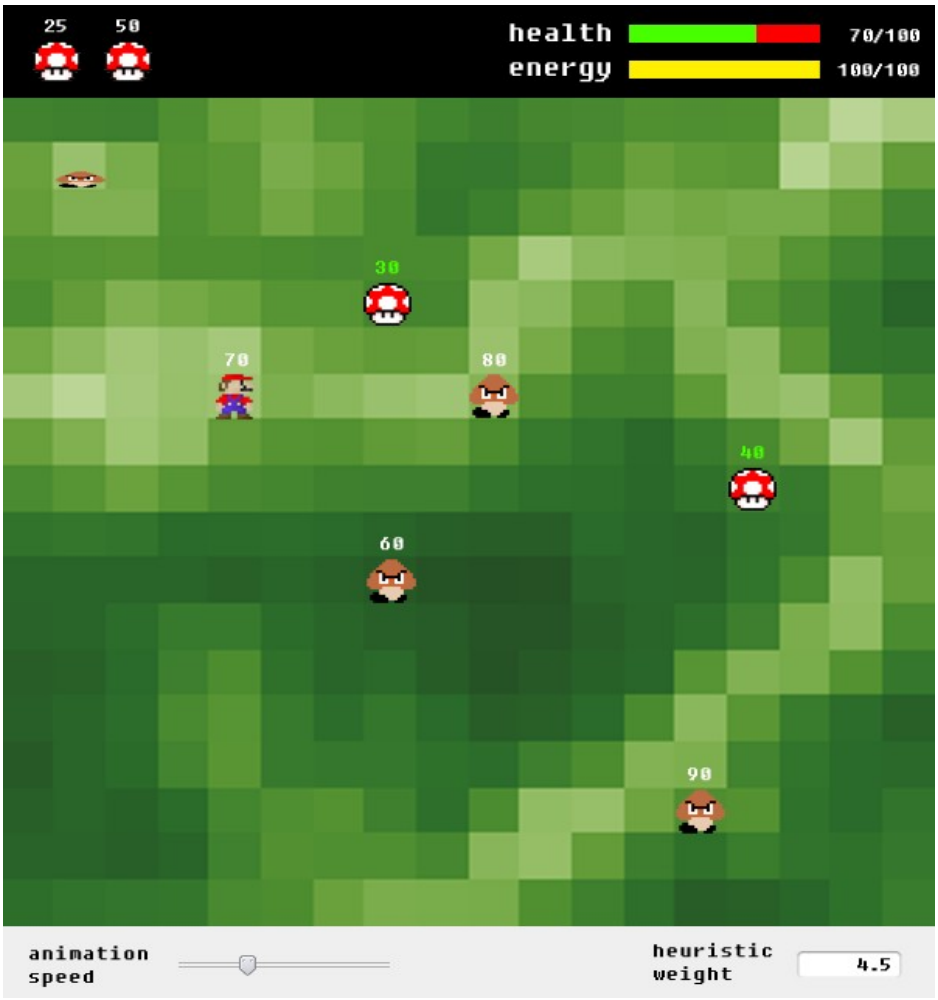
You repeat this scenario until all enemies are defeated (you win) or you are not able to defeat one any more (you lose).

At the end of this subtask you should be able to play the game: create a world with a given number of tiles, enemies and health-packs. Let the strategy decide what the protagonist should do. Finally he/she will win or loose.

4 User Interface and Interaction with your world (W8+9)

Give your protagonist object the possibility to move: use the arrows keys to move to 1 of the 4 neighbouring tiles, if possible and use a mouse click on a tile to find a path to the selected destination and execute the animation. Be sure the health level is updated correctly.

In the UI of your game both the health level and the energy level need to be visualized. You need also 2 UI elements with which you can control the “play speed” (how fast move the protagonist over the world) and the relative weight of the heuristic part in the cost calculation of the A* algorithm.



At the end of this subtask you are able to play the game with a decent UI. You are able to control your protagonist with the arrow keys. You are able to control your protagonist with the mouse. You are able to start your strategy (from menu, button and shortcut).

5 Extras (nice to have features)

- pause the game
- save / restore the state of a given game
- replay the game
- In stead of using the random positions of the health-packs you get from the library, put them first in a repository. This repository needs of course to be visualized (a number of icons). Expand your application to be able to drag-and-drop your health-packs on a specific tile in your world. If your protagonist enters this tile, his/her health level will be increased by the amount of the specific health-pack. A good starting point here is the “drag and drop puzzle” example
- more complex strategies, where you optimize multiple moves
- putting 2 protagonists in the world which try to get the highest score
- use QtQuick for UI
- ...

6 Final Presentation (W14)

Every team will get 30 min. to present its final project. All code, resources (images, other data...) + final UML class diagram (as image) needs to be final by Monday, January 5th 2015 9am on svn.studev.groept.be. The detailed schedule will be presented later. The code will be evaluated on a 64 bit Linux environment.

If no working (compilable and runnable) project is available by that date, students are not allowed to the presentation and need to rework their project for the 3rd examination period in September.