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. A team has 3 students, each having a specific subtask as final responsibility.

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 different kinds of 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 built 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 pointers wherever possible. Use of ordinary pointers needs to be motivated!

1 Intro Qt Widget application + How to use an external library (W6)

On Toledo you will find a header file (world_v1) with the definition of the World class you will use. The implementation of all available functionality 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 needs 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 a "given" file: *you are not allowed to change any functionality of the given code.*

This first subtask is also an introduction to create a Qt Widget application using features from the Qt Core and GUI environment: Qt Designer, signals and slots, Qwidget, QgraphicsScene/QgraphicsView.

Use the method world::createWorld(QString fileName) to get a vector of smart pointers to Tiles. This method needs the name of an existing image. Put the image on the map where the executable is located or use the Qt Resource System to be platform independent. Your application should visualize all tiles using their position. The value attribute should be used to give a Tile a certain color. Use the QgraphicsScene/QgraphicsView functionality to reach the wanted visualization. Give your tile a certain display size (avoid hard-coded values) so that the final result will be a scaled version of the image you started with.

The best matching example available from Qt is "Diagram Scene Example", although too complex at some points.

At the end of this subtask you have a running application using the world library which is capable of visualizing all tiles resulting in a scaled view of the original image.

2 Complete world library and definition of architecture (W7)

On Toledo you will find a header file (world) with the definition of the World – Tile – Enemy – Penemy[1] - Xenemy[2] objects you will use, together with a shareable library implementing the defined methods. They all will play a role in our game having some rules. At the start of the game you should be able to select the image which will represent the world you are playing in. Moving in the world will take energy defined by the difference in value between

1 Penemy is an enemy which is poisoned. It will lose its poison when attacked and finally it will die

2 Xenemy is a yet unknwon type of enemy which will have specific behaviour

the 2 tiles involved, defeating an enemy will need enough health (which can be increased by going to a tile with a health-pack on it). Navigation of the protagonist can be done both manually (e.g. arrow keys) and automatically (through pathfinding)

You need to visualize everything. In other words, display every tile as a square with a color depending on the value of the Tile object and use something more "special" for the enemies, healthpacks and protagonist (e.g. an image). Use a different visualization for non-defeated and defeated enemies. Penemies need a more complex visualization. Once they get attacked, their poison level will gradually (but in a random time) go to zero. The specifiic rules to visualize an Xenemy will be given later. Your visualization should animate also this effect. Choose also a representation of your protagonist and put it on tile (0,0).Your application needs to be able to switch between 2 different visualizations of the same world, so be sure that your chosen architecture is suited for this requirement (use of MV(C) design pattern will help you in reaching this goal. Make your architecture suited to easily be extended with more types of enemies (each having different behavior) and more ways of visualization.

Once your project can be built, make sure you are able to get from the World class the basic information of your game (the tiles, (poisoned)enemies, health-packs and protagonist), and store this information in the appropriate objects.

Keep in mind that all data you get from the library are collections of unique_ptr. This is a 2nd version of a smart pointer available in the STL. Like the name suggests, with unique_ptr you can have only 1 pointer referring to the pointee. This means that you cannot assign unique_ptr's to each other, otherwise you would have 2 pointers pointing to the same pointee. This will later be covered in more detail in the lectures. The moment that the library returns the collection, it transfers ownership to your application. How can you handle a collection of unique_ptr?

```
for (auto & enemy : enemies) //you only get a reference to the smart pointer, no change of
                             //ownership
```

or you assign them to whatever type you want to use by std::move, e.g

```
std::shared_ptr<Enemy> test = std::move(en.at(0));
//std::move transfers ownership, the pointer in the collection is no longer pointing to a valid
//Enemy object
```

At the end of this subtask you have a clear understanding of your architecture using a Model-View approach, you are able to use the world library to get all necessary starting information. You have a UML class diagram showing this architecture which needs to be handed in to get feedback.

3 3 different subtasks (W8, 9+10)

When you have defined your architecture each of you can work on a separate task.

A) Visualization using QGraphicsView/QGraphicsScene

You should have the possibility to move your protagonist to a neighboring tile using the arrow keys. Avoid hard-coded sizes in your visualization, your view should be resizable. Your application should look nice with different worlds of different sizes.

You should have appropriate visualizations for all involved parties. At least enemies and health-packs should have a different visualization once they are defeated/used and the

Penemy objects should have some animated visualization on their poisoning effect. While a PEnemy is losing its poison, it will contaminate the environment (a new state for a Tile with a specific visualization) and if your Protagonist is on a contaminated tile, it will loose health. You are free to define the visualization, the rate, the effect on health…

Make use of the functionality of the QGraphicsScene/QGraphicsView classes.

B) Simulating a terminal environment

Your final application should be able to switch between 2 different visualizations. For the second one, we want to simulate a terminal environment, meaning that you can only use textual symbols to show everything on the screen. You are completely free to decide how you will reach this: will it only be text-based with specific commands as

```
> show prota
      protagonist at position (25, 36), health value = 96, energy = 58

> select enemy
      nearest enemy found at position (36, 59), strength = 63
> move (36, 29)
```

or will you show (part of) the world using textual symbols like

```
+--+--+--+--+--+--+--+--+
|P|  |  |  |E|  |  |  |
+--+--+--+--+--+--+--+--+
```

or something completely different… But anyhow, you need again a way to move your protagonist to a neighboring tile and “see” in some way the new state of your game.

But of course both visualizations should use the same underlying model, when switching I’m still playing the same game. Be sure that you can switch visualizations while playing the game, not only a selection at the start of the game.

C) Path finding

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 value 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 a value of infinity must be seen as impassable.

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. You 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 neighbor is at index-1, your top neighbor at index-nrofCols… Don't forget to take into account the border conditions.

As a benchmark you will need to find your way into a big maze (a picture of a maze with dimensions 2400 x 2380) in a “reasonable” time. Experience from the past shows that this is possible in less than 1 second.

4 User interface and integration of subtasks (W11+12)

Generate the main UI of you application. You start by default using your graphics visualization. Add necessary information about the state of your protagonist (energy, health), buttons to switch visualization, change the speed of the animation and control the parameters of the A* algorithm.

Add the possibility to navigate to a specific position (needed to check the efficiency of your path finding algorithm in a big maze) and meanwhile visualizing the path you are following. Be sure the health level is updated correctly.

Add the possibility to start the strategy (see next subtask) and to load another world (with a given number of enemies, health packs).

5 Strategy (W13)

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

Minimally implement the 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 (=difference in values) of the tile you walk to. 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 impassable.

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. Switch from 1 visualization to another...Finally he/she will win or lose.

6 Extras (nice to have features)

- pause the game
- save / restore the state of a given game
- replay the game
- Instead of using the random positions of the health-packs you get from the library, put them first in a repository. This repository of course needs 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
- ...

7 Final Presentation (W14)

Every team will get 40 min. to present its final project. All code, resources (images, other data...) + final UML class diagram (as image) needs to be final by Friday, December 28 2018 23.59 on gitlab.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.