

Final Project

Advanced Programming Techniques

Introduction

This final project will take the rest of the semester. As evaluation, every team will present its work during the exam period in January. The team has 3 students, all having the same responsibility: *designing software in Qt that conforms to the requirements stated in this document*. The team defends the project as a team of 3 students, not alone. All software updates need to be committed to gitlab.kuleuven.be. The **deadline for the project is January 13th 23:59**. The list of commits of every team member will be used for evaluation. Every team should also have a uml_yyyymmdd.png of their UML on Gitlab, at least every two weeks this UML should be updated (and pushed to git) with the latest version. Give each major version its own date suffix. **Also keep track of your team's progress** in a readme.md file, where we can see what each team member is working on per week or use the issue-board in GitLab to manage your work progress.

The final goal is to create a **game-like application** where the protagonist needs to be able to navigate in a given 2-dimensional, grid-based world, attack various kinds of enemies and to gather the necessary health packs to survive. Navigation of the protagonist needs to be done manually (e.g., arrow keys). Additionally, there should be an **auto-play** mode where the protagonist tries to defeat all enemies automatically.

Moving in the world will take energy defined by the value of the tile you are moving to, defeating an enemy will need enough health (which can be increased by going to a tile with a health pack on it).

Your application needs to be able to switch between at least 2 different visualizations of the same world: a 2D graphical representation and a text-based representation. Make use of the **MV(C) design pattern** to help you in reaching this goal. Make your architecture suited to easily be extended with more types of enemies (each having different behavior) and even more ways of visualization.

You do not start from scratch for this project. On Toledo you will find a header file (world_v6) with the definition of the World – Tile – Enemy – PEnemy¹ – XEnemy² objects you will use, together with a **shareable library** implementing the defined methods. All these types are part of the model of your application. All of them will play a role in our game, each having their own rules. The library also contains functionality to generate the world you are playing in based on an image. Each pixel in this image will represent a tile in your grid-based world³. A pathfinding algorithm is also implemented in this library which will be useful for the autoplay functionality.

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 unknown type of enemy which will have specific behavior that you may choose yourself

³ The value of a tile is determined by the grey value of the corresponding pixel

Detailed specifications

Each team member is responsible for all the subtasks. Of course, you should split the work that needs to be done. We will look at the *readme.md* and GitLab commits to assess who has done what work.

Subtask A. “Graphical” representation



- Look at `QGraphicsView` / `QGraphicsScene` / `QGraphicsItem` for this task
- Each tile has a fixed dimension in pixels and has a color defined by the value of the tile. In the simplest setting the world is visualized as a scaled version of the image used to create the world.
- All other objects (protagonist, enemies, health pack) get a specific visualization with the same dimensions as a tile. Each type of enemy has a different visualization. Also 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 while poisoning a wider area of the world surrounding it. Your visualization should also animate this effect.
- For the protagonist, at least the following actions should be visualized: attacking, using a health pack, getting poisoned, moving, and dying. Minimally these actions should result in a brief color change in the protagonist, but you can go further and include full animations if you wish.
- You can control the protagonist with simple movements (up, right, down, left)
- The path followed by the protagonist in autoplay should be visualized in some way
- Instead of using a color, each tile can have an associated image (grass/brick/...)
- The architecture of your application should make it easy to add new types of “actors” in the world with different behavior.

Subtask B. “Text-based” representation

	H	H	H				
P		E				E	
x	E	E					
	\$						
		H	x		H		

- The world itself should have an ASCII art-like, string-based visualization. A simple printout of the events that are happening is insufficient. We still need to be able to see the world.
 - This text visualization should not simply be printed in the terminal but should be integrated in the applications overarching UI. For example, there could be a central window in your UI that shows your world, and two buttons ‘Graphical view’ and ‘Text view’ that switch the representation shown in that window.
 - The same protagonist actions should be visualized as in the graphical view (attacking, using a health pack, getting poisoned, moving, and dying). Again, the minimal approach is to briefly change the protagonist’s color, but you can use a different approach if you wish.
 - In this mode, all interaction with the protagonist is done by text commands instead of mouse clicks. The available commands should be easily extendable (we do not want to see a big if/else tree). Minimal required commands are *up*, *right*, *down*, *left*
 - *goto x y (this triggers the pathfinder)*
 - *attack nearest enemy*
 - *take nearest health pack*
 - *help (this prints a list of all available commands)*
 - Ideally a command should be understood as soon as it is unique (so typing a ‘r’ would be enough to go right), consider tab completion as a possible extension.
 - The architecture of your application should make it easy to add new commands.

Subtask C. “Overlay”

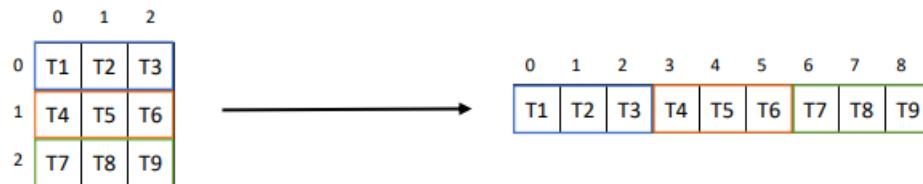
You can use a different image as a top layer with all tiles “behind” it as a data layer with information. For example, a map of a medieval town with some roads, houses and gardens with stone walls surrounding them. The data layer underneath this could have tiles with low values for the roads and infinite values for the stone walls, this way a visualization can be different from the data layer. Entering the door of a house loads a new map. There could be potholes or walking through several types of roads could be more tiresome (muddy roads). Think of a way to incorporate this into your design, you should have only one image for the complete map of one level. When zooming in into such a map the correlation between the top layer and the data layer should be maintained, your solution should be able to handle this.

Subtask D. “Levels”

The protagonist can move to special Tiles (Door/Portal) which will result in a different map being loaded. This loading of new worlds should be done in a short time. Think of a way to “cache” your map, the protagonist might go back and forward fast between two levels. How can you optimize your memory management?

There are of course many ways to link various levels, think of a convenient way for people to create the levels and how they can indicate to which level the door/portal goes.

Note that the world tiles returned by the library are stored in a specific order (see below).



Subtask E. Integration and extra features

User interface

You will need a basic UI which gives you the possibility to switch between the different views and control some settings. Your UI should support at least the following features:

- Show the protagonist’s health and energy bars
- Allow switching between graphical and text-based representation
- Allow the input of commands in text-based representation
- Zoom in/out on your world visualization

XEnemy

You should add a third type of enemy in your game, aside from regular Enemies and PEnemies (poison). Make sure your XEnemy has its own visualization. The behavior of your XEnemy is up to you. Some ideas:

- A transforming enemy, like a poltergeist, who changes the tile values near him to insane values or a morphing enemy that can go from harmless to boss-level difficulty.
- A moving enemy
- An enemy who does not die when killed, but resurrects as a zombie
- An enemy that needs to be hit three times, but teleports after each hit
- An enemy that grows stronger the longer it is on the board
- ...

Note that your XEnemy is part of your model layer, so you will have to extend the library for this. You cannot / are not allowed to change the source code of the library, so you will need to use inheritance here. Also, the library returns a vector of all enemies, but this will only contain regular enemies and PEnemies. You will need to find a way to convert a certain number of regular enemies to XEnemies.

Bonus features

If you have time and energy left, you can try adding some bonus features. Use your imagination here. Some suggestions:

- A Save / Load system which stores the level's current state
- Adding health packs to specific tiles with drag-and-drop
- New enemy types
- More than one protagonist
- Multiplayer support
- Animated Protagonist
- More animations
- An additional visual representation (different 2D visualization? 3D? Virtual reality?)
- ...

The library

On Toledo you will find a header file (`world_v6`) 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. 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.

Use the method `World::createWorld(QString filename, unsigned int nrOfEnemies, unsigned int nrOfHealthpacks, float pRatio)` to initialize your world and create a protagonist.

- `filename` is the name of an existing image. Use the **Qt Resource System** to add images to your project in a platform independent way.
- Every pixel of the image stored in `filename` will become a Tile in your world, the grey value of the pixel determines the value of the tile (and thus how difficult it is to traverse). This way, you can create new worlds by just drawing an image. We will also add a couple of images on Toledo that you can use to generate worlds.
- Your application should visualize all tiles using their position. The `value` attribute should be used to give a Tile a certain color.
- The 3 other parameters to `createWorld` define how many enemies and health packs you will have and what ratio of the enemies will be PEnemies.
- The method `getTiles()` returns a vector with `unique_ptr` to the tiles you just created.

Keep in mind that all data you get from the library are collections of `unique_ptr`. This is the 2nd version of a smart pointer available in 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. The moment that the library returns the collection, it transfers ownership to your application.

Planning

You can make your own planning for this project, but we expect you to loosely adhere to the following structure:

Week 6: Experimentation with Qt, integrating the library and making a first graphical version

Week 7: Design architecture and create application UML. **Milestone 1**

Week 8-10: Subtask A-B-C-D-E implementation

Week 11-13: Integration and extra features

January 13th 2024 23:59: Deadline final commit and push to gitlab.kuleuven.be **master branch**

During Exam period: Final presentation. **Milestone 2**

Be sure that every member can commit to Gitlab since the activity on GitLab will also be used as an evaluation criterium. Each task still has a certain degree of freedom where you may use your own creativity.

Do not leave the integration all the way to the end. Be sure that you can do early integrations using a SCRUM like approach.

Milestones

Milestone 1: Architecture (12/11/23)

Based on all the info you have until here create a class diagram of your application. Again, we will make use of Design Patterns to define our architecture. Here you should apply the Model-View (Controller) design pattern to make the visualizations as loosely coupled as possible with the underlying model.

Try again to be as detailed as possible, this class diagram will help you to divide the work among the different team members. Use smart pointers wherever possible. Use of ordinary pointers needs to be motivated!

This class diagram needs to be uploaded by Sunday, November 12th. **This UML is not graded**, but you will get feedback in the next lab session.

Milestone 2: Final code (13/01/24 23:59)

Every team will get 40 min. to present its final project. All code, resources (images, other data...) + final **UML** class diagram (as **png-image**) + status report need to be final and on Gitlab on gitlab.kuleuven.be. Please also send an e-mail to your lab coach to mention that your whole team participates in (or the *whole team* postpones) the presentation. The detailed schedule will be presented later.

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

Evaluation criteria

We will analyze your final uploaded code before the presentation and prepare questions beforehand. The final presentation consists of three phases:

Demo

You will start by giving a short demo of your project, demonstrating all the features. Make sure you can show at least the following aspects:

- Switch between 2D, overlay and text-based view during program execution
- Move protagonist with arrow keys or by clicking / goto command (using the given pathfinder)
- Autoplay feature
- XEnemy
- Visual representation of path
- Easily switch to new map by passing a DoorTile
- Of course, any bonus features you add will be factored into your score.

UML

Have a complete, accurate UML diagram of your code ready (you can just generate this from your code). Based on this, we will ask questions about your software architecture. We will focus on:

- Model-View(-Controller) architecture
- Correct use of Polymorphism
- OOP design principles (SOLID) think about extending with other actors, other commands, more complex visualizations...

Code

We will dive into the specifics of your code. Aside from the points mentioned above, we will also consider:

- Correctness
- Efficiency
- Avoiding unnecessary copies (pass by reference vs pass by value)
- Use of smart pointers