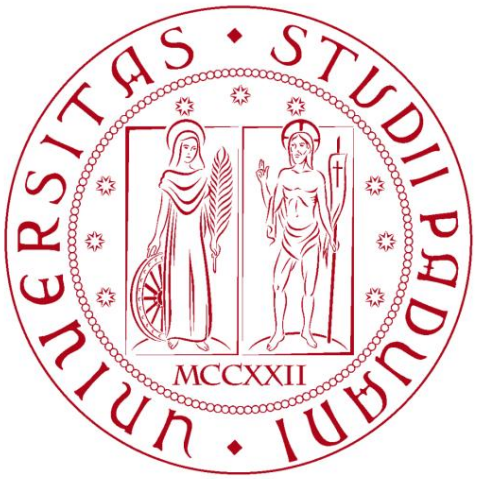


UNIVERSITÀ DEGLI STUDI DI PADOVA

UNIVERSITY OF PADUA
OBJECT-ORIENTED PROGRAMMING COURSE
AY 2023/2024

Picello Davide Serial number: 2034825

PaO project report - Zoo Simulator

Group members:

- **Picello Davide 2034825**
- Mahdi Mouad 2044222

Theme

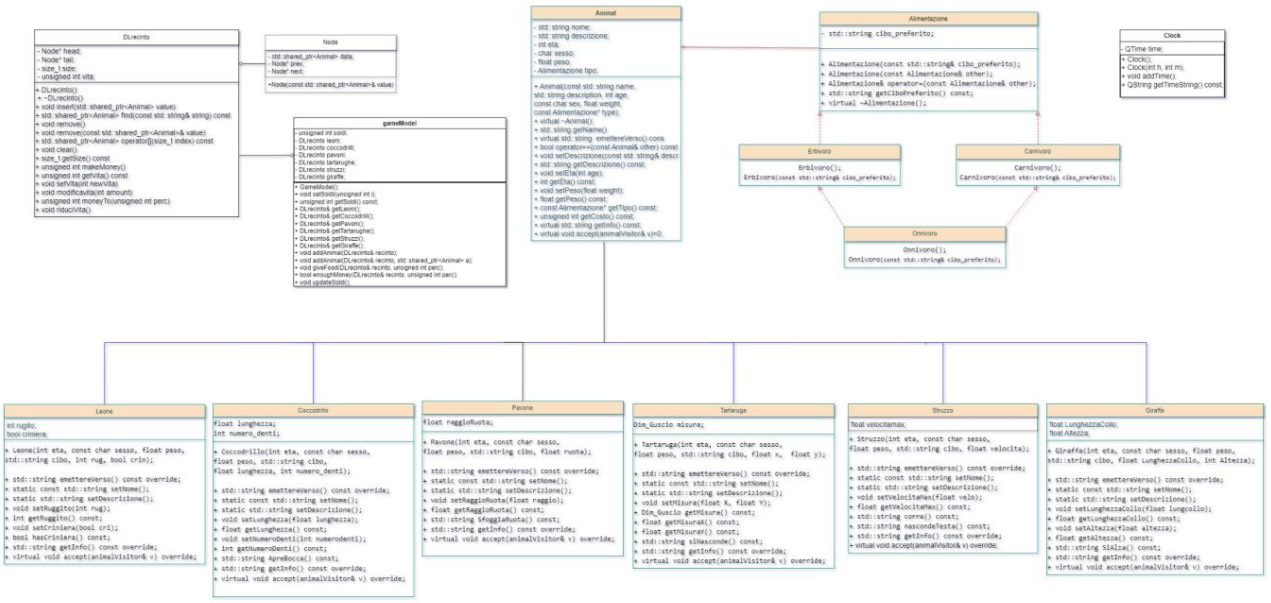
The theme of the project is, as you can imagine from the title, a small zoo-themed video game. This idea was conceived, and then subsequently chosen, precisely because it lends itself well to the use of polymorphism and inheritance.

The aim of the game is to manage your own zoo, being able to add animals, feed them and be able to see some specific information about each animal: some is common information such as a description of the type of animal, and some specific to each individual component, such as the name and the weight. Furthermore, each type of animal has its own attributes, for example, the lion has the power of its roar.
An animal has a specific cost to buy, but also generates a profit for each update of the game timer, as more animals are assumed to attract more visitors.

Logical model

The animal class hierarchy has a virtual base class “Animal” which includes the methods and attributes common to all animals. From this derive the derived classes crocodile, giraffe, lion, peacock, ostrich and turtle, which have their own methods and attributes.

The Animal class also contains a Nutrition class, inserted in the early stages of the project, even if then, in addition to telling us what type of diet the animal has and what its favorite food is, it has never found its current use.
Finally we also find the enclosure, or the container required in the project specifications, which is a dynamic doubly linked list that is used to keep track of the individual instantiated animals and their relative removals. The choice of a doubly linked list was not essential, but it was decided to leave it for pure educational purposes, to differentiate itself from a simple list.



The original quality image can be found in the folder.

Non-trivial use of polymorphism

To satisfy this constraint we have implemented the dynamic creation of the various windows for each animal via a **visitor.**
In particular, in the ***visitor folder,*** the ***animalDetails*** and ***animalVisitor*** files (which also work thanks to the pure virtual method in Animal ***accept(animalVisitor& v),*** then implemented in each subobject), contain the implementation of the visitor, which has the task of creating different windows based on the type of animal with each one having a different layout and different functions specific to that type.
For example: in the widget relating to the Lion, we have a particular layout with colors that symbolize it such as orange, and its specific function, that of roaring, simulated via popup with information on the power of the roar (an attribute specific to each instance of lion); however with the crocodile we have a different colour, green, and a different function, that is to open its mouth and show off its teeth.

We also tried to integrate sounds (for example the roar of the lion when its roar was activated) but, as far as we understood, we did not have the necessary QT module ***(QT Multimedia).***

Data persistence

The data persistence constraint was satisfied by saving, and loading, games via .xml files.

We have a "dataManager" class that takes care of saving, or loading, the data relating to a game: first it saves the general information such as the date, time and money, then it also stores all the data of each individual animal for each enclosure.

Features implemented

Gameplay

You have 6 enclosures, one for each type of animal in the game: crocodile, giraffe, lion, peacock, ostrich and turtle.
These enclosures spawn empty, and can be interacted with in a couple of ways:

Feeding animals

Using the appropriate button you can feed all the animals in the enclosure. Hunger/health is simulated via the life bar, visible from the map. If the health is at zero, an animal will be removed every time the game timer is updated (2 real seconds/2 minutes of the game) to simulate its death.
You can decide how much to feed the enclosure by choosing, via a pop-up, how much to raise the health value to (25%, 50%, 75% or 100%) and how many resources this requires.
Obviously, if you don't have enough resources for a certain option, it won't be available.

Add animals Using the

appropriate button you can add 1 animal of its type to the enclosure. This will require a certain number of resources based on the type of animal added, but this value also influences the profit an animal generates, for example:

> A lion requires €10 to purchase, but will then generate, with each update of the game timer, twice that same value. So, to have an enclosure with 4 lions we would have to spend €40, but this will generate €80 every update of the game timer.

As you can see, the revenue is enormous compared to the expense, this is done specifically to make the gaming experience easy and therefore have greater freedom on the choices you can make. To make the game more challenging it would be enough to have the user choose a difficulty level, and decrease the gain of the fence based on the chosen option.

Interaction with individual animals and search If you

click on the label of a single animal, or search by name using the search bar, the various attributes of the animal we have selected will be shown through a special window, automatically generated based on to the type of animal chosen.

Random data generation

Also thanks to the functions implemented in the ***"generate.h*** and ***generate.cpp"*** files we have created a mechanism that allows, when an animal is generated, attributes to be assigned that can be as truthful as possible: the weight of a crocodile, for example , is generated by choosing a random value between 2 extremes, which will obviously be different from the data used for peacocks.

Shotcut

During a game, if you press "esc" on your keyboard, a menu will be shown with 3 choices: - "Close without saving" which

closes the game without saving it - "Cancel" which only closes this newly created
menu, making us return to the game - “Save” to save the progress of the current game - By selecting this option
we will be asked to enter the name of the game.

If the name is already present, then we will be asked if we want to overwrite this save

Layout

The layout is flexible, with everything being placed in one large ***QGridLayout,*** which is flexible by nature. However we decided to keep a minimum size so as not to make everything too small.

Other features

Other interesting features that have already been discussed previously are: saving and loading games, window with specific data dynamically generated for each type of animal.

Hour reporting

Although the project specification file indicated around 50 hours of individual work, the actual total hours spent were well above that threshold, coming in at around 65/70 hours.

It must also be said, however, that approximately 80% of the time was spent on the graphics part done with QT, which took so much time for mainly two reasons:

- QT was a new framework for both members of the group. This therefore entailed an important "self-learning" phase of this new technology.

- Developing a video game, obviously, requires greater commitment regarding the graphics, and this project has absolutely not disproved this belief.

Subdivision of activities

The activities were, as much as possible, equally distributed between the two components, for example: the final visitors (six in total) were developed, 3 by me, and 3 by my partner.

Changes applied

There have been no changes as this is the first time this project has been delivered

Conclusions

The development of this project has been incredibly inspiring.
Although it took much longer than initially expected, I find this to be an extremely educational and, despite the considerable amount of total time involved, a very enjoyable experience.
In fact, if I had to do this project all over again, I would probably now be able to drastically reduce the hours spent.