AI 4 Games
Assessment Milestone 3
GUI framework with Qt

August 26, 2022

Objectives

1. Review `C++` programming.
2. Constructing a GUI with the "framework" Qt.
3. The software pattern MVC (Model-View-Controller).

Instructions

This practical laboratory has several parts. Each part concentrates on at least one learning objective. All these objectives are designed to set-up the starting code to develop a computer game where you can progressively add artificial intelligence to the characters in the game. The goal is that you would have more sophisticated monsters that chase the player, and that you can provide more capabilities than bomb to fight the monsters from the player. As you provide more abilities to the player you may need to provide more intelligence to the monsters (or classes of monsters, for example, monsters wiht a compass, with a sense of smell, with shooting capturing nets).

You will need to complete several steps. The instructions here are guidelines. There are many places you will have to experiment and fill in the gaps.

Subversion and teams

Your instructor may decide to group students in teams for this project. If that is the case, it is also advisable to have **Subversion** support the effort of the team. This means setting a subversion repository somewhere in a server machine where members of the team belong to a user group and users of that group shall have all privileges to act on t eh repository. This is a technical matter that is not the main subject matter of the course and will not be discussed further. We only insist that using version control in a team is truly useful and full documentation on `svn` can be found in

http://svnbook.red-bean.com/

Coding standards

Coding standard help understand the purpose of each element or component in a computer program. They also help structuring and reinforcing solid structuring practices of the code. The textbook "Artificial Intelligence For Games" spends several paragraphs in the first 2 chapters explaining the coding standards and the notation that will be used. Some of the standards are required by documentation generation tools like `doxygen`. The purpose of this exercise is to review the code provide it and make revision so it conforms to the standards we will detail next. Reviewing code this way will help you understand even more what the code does, and more importantly what elements are visible and what not externally to a class or module. In many cases, these standards are also needed because of the way the programming language allocates and deallocates memory. There is a tutorial associated with this lab on memory management. For example, each of the three following very popular object-oriented programming languages has a different approach to memory management. Recall `C++` leaves this responsibility completely to the programmer with maximum flexibility (`Java` has a garbage collector freeing the programmer completely of memory management responsibilities). `Objective-C` ha somewhat a policy in between with tools to keep counts on references.

1. Standard for identifiers and code.

    - Standard names (`AClass`,`_anAttributte`, `localVariable`, `aMethod`).
    - Names of methods shall be self-explanatory and without abbreviations.
    - No boolean getters `anAttribute()` and setters with `anAnttribute(newValue)`.
    - Access to Boolean values and testing the state of an object `isCreativeCommons()`, `is unLoaaded()`, `hasExpitred()`, while setters and imperatives to the object `followCreativeCommons()`, `unload()`, `expire()`, etc.
    - When doing `Unit-Testing` or `TDD`, the testing methods may be an exception to the rule and we will use a word of verb for what is being tested (and the conditions for the test). For example: `testAddTo List_whenNameOfArtistExists()`.

2. Accessibility to members

    - All attributes are `private` (or `protected` if it is justified).
    - Select the most restrictive visibility as (`private`, `protected` or `public`) for each method.

3. Implementing methods.

   - Short methods, using fast exists specially when we have nested `if/elses`.
   - Methods that are `getters`, and that do not modify the sate of the object are always declared constant with a `const` at the end of the declaration:

     `ReturnType myMethod(parameters)` **const**

     The compiler will then warns us if attributes are being modified or this method uses methods that are not const.
   - Parameters that are objects are passed by reference so we avoid the cost of copying (and allocation). The reference must be declared constant if the receiver methods no requires to modify the object.

     `void myMethod(` **const** `Type` **&** `object)`

     We can only invoke constant methods on an object received as constant. The compiler will ensure this and that way we have no side effects on the received object.
   - Parameters that are of basic types are not passed by reference, and also for basic references.

     `float myMethod(int value)`
   - We return constant references to return objects that are not basic:

     const `Type` **&** `myMethod()` **const** `return _member;`
   - We do clone (copy) objects we return when created locally inside a proprietary methods (name these methods informally as getters by construction).

     `Type myMethod() { Type result; ...; return result; }`

     The reason is we do not return references to objects that expire when out current context s a method terminates.
   - When the parameter passing or the return reference value transfer the responsibility of eliminating the object (and recycling its memory), we will use as a standard a counter of references.

```
// We transfer to the caller the object that was on the
receiver:
Type * extractObject();

// Create and transfer to caller (no longer in receiver)
Type * createType() const;

// Transfer to caller and no longer in receptor
void adopt(Type *);

// Transfer to the caller and elimination
void eliminated(Type *) const;
```

Important points

- Memory management is a central issue and particularly in C++ Therefore, review the following points.

  1. In C++ there is no *garbage collector* to eliminate un-referenced objects. Remember many objects are created and eliminated automaticly by the mechanisms of the runtime stack.
     - Local objects of a method are destroyed automaticly when we exit the context of the method.
     - All members that are not pointers (not references) of an object are created with the constructor and deallocated automaticly by the destructor.
     - All objects created using *new* are subject to memory management and must be deallocated with the call to `delete`. For this follow strict discipline and convention common in development tea,s to manage the responsibility of deleting objects.
  2. Conventions regarding memory allocation
     - We will tend to leave the responsibility of eliminating an object to the class that create it.
     - We may acquire the responsibility of an object by calling a method that inside has a `new`, only if we are clear of receiving is as a parameter or as the return value of the called method.
     - When we create an object and delegate the responsibility to `this` we must provide a method for cleaning the object and at least deal with this responsibility on the destructor of `this`
     - Never place two types of roles in the same pointer variable where in one case the pointer is a reference without responsibility to recycle memory and in another section of the code, there is the responsibility to recycle the object.
     - If an object pointer by a pointer is eliminated, it should be good practice to set e the pointer to null.
     - If we assign an object to a pointer and the pointer is not null, the object being pointed should be recycled there.
     - When an methods transfers responsibility of the memory management of and object, it should always do this to its caller.
     - When by parameter passing or by return value a transfer of responsibility is happening, adopt the convention of making a reference count.

```
// Transfer that does not affect this
Type * createType() const;

// Transfer into this
void adopt(Type *);
```

Task

The code provide it does not follow the conventions of codding indicated above on purpose. It is your job to review all the class names and method names and apply the corresponding editing changes to ensure the code complies with these conventions. Moreover, the code has no implementation of any memory management. Many classes that need it do not have a destructor, and therefore, no objects are recycled. Review the code provided carefully and ensure yourself that all objects not required are properly released and that there are no memory leaks.

Framework `Qt` for the GUIs

In order to get you started with `Qt` you are required here to complete the standard tutorial. We strongly encourage you to carry out the tutorial using a version control system. Each of the 14 stages of development should be completed. We strongly recommend you do not just download each file and execute it, but actually start with the very first file and edit each file yourself to achieve the next stage. At each stage perform a `commit` into the `svn` repository of the new achieved functionality. On the Web, the tutorial is in
`http://doc.trolltech.com/4.4/tutorials-tutorial.html` You must complete the 14 tutorial chapter. Again, we insist you do this tutorial by editing file into the next level. Do no just download the next stage of files. You will learn much more from reflecting on mistakes and actually putting in the lines of code.

1. Hello World!
2. Calling it Quits
3. Family Values
4. Let There Be Widgets
5. Building Blocks
6. Building Blocks Galore!
7. One Thing Leads to Another
8. Preparing for Battle
9. With Cannon You Can
10. Smooth as Silk
11. Giving It a Shot
12. Hanging in the Air the Way Bricks Don't
13. Game Over

14. Facing the Wall

As we suggested, do at least one `svn commit` at the end of each chapter. Each new version is committed, not each new file. Use tags to tags the chapter version in case you do more frequent commits. Look at for `svn` documentation
`http://svnbook.red-bean.com/en/1.1/ch04s06.html` A *tag* is a copy of the state of your files in a folder at a particular time. For grading purposes by your lecturer/tutor it is important to tag the version after the end of each chapter of the `Qt` tutorials. The tags should be `t1`, `t2`, ..., `t14`.

For programming with `Qt` there is a reference manual available on the WEB
`http://doc.trolltech.com/4.4/index.html`

> **Important:** You may need to install `Qt` for LINUX and/or MacOs. This is usually a simple process. Follow the instruction on the Qt web-site if you have problems installing but under LINUX is usually a standard package and also on MacOS, `port` can install it rather easy.
>
> Depending on what is installed the command to build projects with `Qt` may change to indicate the version installed. Sometimes it is `qmake-qt4` sometime is simply `qmake`.
>
> Typically, the setting up of a Makefile and the compiling is achieved with the three steps below:

```
qmake-qt4 -project
qmake-qt4
make
```

Task — the pattern MVC

Draw the class diagram of all the classes in the folder `PacManQt`. Study the code and make sure that you can follow what is the responsibility of each class. remember you have to make sure it compiles with the standards of coding practice.

You will see that the design and architecture is very similar with the version that didn't have a Graphical User Interface. Make sure you organize the class diagram to show th pattern MVC. The code supplied is a model to get you started on a game. Identify the classes that constitute the GUI nd the classes that constitute the *controller*.

Compile and run the code. While at the moment the graphics ar e not fancy and the prototype does very little it is the base architecture to start to make more intelligent characters.

Study also the dynamic behavior to identify the elements that constitute the *View*. Draw the corresponding UML interaction diagram that reflects the dynamic behavior of the program. You may not need to follow all that happens inside the classes of `Qt`.

You must submit both diagrams.

Documentation

Add targets to the corresponding `makefile` so that your project generates the documentation automatically.