Practical Assignment 4

Game development

March 26, 2012

The objective of this assignment is twofold. First you will once again implement additional functionalities to your game engine. Secondly you will use your game engine to program a simple game that shows all these functionalities. No VS solution is provided, you start from the current state of your engine.

Objective 1: Upgrading the game engine

In this final assignment you will again implement additional functionalities to your game engine. You have to implement at least three new functionalities among the selection given below. You have to select (at least) one functionality in each of the following tasks. It is advised to already have an idea of the game you want to program (see Objective 2) before starting the following implementations as they might partially depend on the game genre.

Task 1.1: Advanced Graphics, HID or Audio

You have to implement (at least) one of the following functionalities:

- **Advanced Graphics** By now the renderer should be able to deal with primitive shapes or simple objects. Of course a real game engine has to provide much more graphical possibilities. In this task you will do so. For example you can design a manager of 3D meshes (loaded from different file formats), textures and sprites. You can also implement a particle system, explosions and object breaking. Any kind of special effect that you find relevant for a game engine is a potential candidate.

- **Advanced HID** For most PC games, a keyboard and a mouse is good enough. But when you want something a bit more fun or more immersive, additional input devices can be used. In this task you will upgrade your `InputManager` to take additional devices into account. You can for example implement an interface for a webcam or:

Joystick	:	http://msdn.microsoft.com/en-us/library/ee416842(v=vs.85).aspx
XBox 360 controller	:	http://msdn.microsoft.com/en-us/library/ee417003(v=vs.85).aspx
Kinect camera	:	http://www.microsoft.com/en-us/kinectforwindows/develop/

- **Audio** To greatly improve the ambiance of your game, an audio manager can be added to the game engine. The game will then benefit from additional clues about the gameplay thus improving the gaming experience. Looking at the hardware or hardware abstraction layer is not a goal of the course, then you will use pre-existing drivers or libraries. Your task is to integrate this driver/library in your game engine. You should be able to load audio files and play, pause and stop them according to game related events. You can also mix sounds and have 3D positioning. Here are few links to audio managers that can easily be integrated to your engine but you are free to use any driver/library you find suitable.

IrrKlang	:	http://www.ambiera.com/irrklang/
OpenAL	:	http://connect.creativelabs.com/openal/
DirectMusic/Sound	:	http://msdn.microsoft.com/en-us/library/hh309469(v=vs.85).aspx

IrrKlang is for instance very easy to use, and can be summarized by the following code:

```
#include <irrklang.h>
//...
irrklang::ISoundEngine * soundEngine = irrklang::createIrrKlangDevice();
//...
soundEngine->play2D(audiofilename);
//...
soundEngine->drop();
//...
```

Task 1.2: Network, Scripting or GUI

You have to implement (at least) one of the following functionalities:

- **Network** The lecture 13 extensively presents the typical architecture used in network games. Using these information you will implement a server/client architecture for single-peer (2 players) or multi-peer (average sized LAN) games. You will use Win32 sockets in UDP or TCP mode. You can find more on their usage on the msdn website:
  `http://msdn.microsoft.com/en-us/library/ms740673(v=vs.85).aspx`
  You can also use a third-party library to help you design this component. But be aware that it is not going necessarily to save time as you will have to get used to the library. Programming a simple server/client architecture will most probably take you less time. Here are few links to network libraries that can easily be integrated to your engine but you are free to use any library you find suitable.

RakNet	:	`http://www.jenkinssoftware.com`
GNet	:	`http://developer.gnome.org/gnet/2.0/`
GNE	:	`http://www.gillius.org/gne/`

- **Scripting** The lecture 14 introduces the scripting mechanics. You will use a scripting language, an already existing one, to program a part of the game logic (*e.g.* scenario or dialog). It is advised to use either Python (see `http://docs.python.org/py3k/c-api` for the embedding API) or Lua (see main page `http://www.lua.org`). In any case, we can summarize the usage of an embedded scripting language by the following pseudo-code:

  ```
  #include <ScriptLanguageHeader> // e.g. Python.h
  //...
  Initialize(); // e.g. Py_Initialize();
  // Open script, write input, execute script, read output
  Finalize(); // e.g. Py_Finalize();
  //...
  ```

- **GUI** A proper game usually has a graphical user interface. It might consist of only few buttons to click, or of very complex nested menus. From the current design of your game engine, you could imagine adding functions to the `Renderer` class, and trying to implement a generic GUI system using functions of the actual renderer. This will be too much work and not the purpose of this assignment. Therefore you can use a third-party library to do that job for you. Your task will be to integrate it in your game engine and be able to create buttons and menus, a HUD, special effects *etc.* Here are few links to GUI libraries that can easily be integrated to an OpenGL application but you are free to use any library you find suitable.

CEGUI	:	`http://www.cegui.org.uk`
GLGooey	:	`http://glgooey.sourceforge.net`
Glam	:	`http://glam.sourceforge.net`

Task 1.3: Physics, Animation or AI

You have to implement (at least) one of the following functionalities:

- **Physics engine** Since about a decade, games make use of physics. From simple ballistic rules and bouncing objects to complex active physics based character animation, virtual worlds are nowadays highly dynamic. The usual way of solving collisions and active physics is to model the virtual objects with primitive shapes (cylinders, capsules, boxes or spheres) augmented with joints, mass, inertia and center of mass. The laws of motion are then applied and solved in real-time calculating their responses to the environment (external forces and collisions). The goal of this task is not to develop a physics engine, as it will be again too much work and not the purpose of the assignment. You will therefore use a third-party library. The goal of the task is to integrate that physics component to your game engine and to use it in your game. You can for example use it to handle collisions and forces (at least gravity) or to create ragdoll characters. Here are few links to physics libraries that can easily be integrated to your engine but you are free to use any library you find suitable.

ODE	:	`http://www.ode.org`
Havok Physics	:	`http://www.havok.com/try-havok`
PhysX	:	`http://developer.nvidia.com/physx`

- **Animation engine** Imagine a game where nothing is moving, it will probably be a bit boring. The animation component is usually responsible for moving the entities in the virtual world. You could of course do it as you do it now, *i.e.* everything in the `update` function of each entity. Using an animation engine, each entity (and its associated scene node) is still responsible for its own state, but if the entity must follow a specific path or set of transformations, then the animation engine is responsible for the manipulations of these pre-defined data. For example the animation engine will provide functionalities to load motions and skeleton files, to play / pause / stop these motions, to generate smooth sequences or interpolation of motion clips *etc.* The goal of this task is to program a simple animation component. The entity will for example ask the animation component to load a motion, and the entity will then access the motion state (transformations to apply to the scene node) using a local time parameter. To help you in that task you can search on the Internet for code samples to load commonly used motion file formats (*e.g.* `http://3dgep.com/?p=1053` to load MD5 files, and `http://www.oshita-lab.org/software/bvh` for BVH files).

- **AI engine** An other important component in a game engine is the artificial intelligence system. You do not want your enemies to be too stupid (making the game not challenging enough) as well as not too smart (so that you do not lose every time). That balance is usually achieved by designing an AI component to centralize the different available behaviors. This component is typically in charge of calculating the paths the entities will take, providing the different strategies and a framework to model an autonomous entity. Your task will be to program such functionalities. You can for example implement a simple path planning algorithm such as the A* algorithm (`http://en.wikipedia.org/wiki/A*_search_algorithm`) or the steepest descent algorithm. You can propose different update strategies and change them in real-time according to game related events. You can also implement an autonomous entity with a simple finite state machine (`http://en.wikipedia.org/wiki/Finite-state_machine`). Different models can then be used such as the *Perception - Decision - Action* loop or the *Belief - Desire - Intention* model. The combination of a proper AI component and a scripting functionality can provide a very good separation between the game engine and the game logic.

Task 1.4: Basic engine tuning

When the feedback on your assignment 3 is available, please take into account as many comments as possible. Definitively solve the major errors and then tune your code to correct the smaller mistakes you made or improve on its flaws. During your demonstration you will probably have questions and comments, you can also tune your engine according to this feedback.

Objective 2: Game development

The primary goal of the course is the programming of game engines, not games. Of course to demonstrate that your game engine works properly you need an applicative scenario. The second objective of this assignment is to program a very simple game that is making use of the functionalities of your game engine. Finally what you want to show is not a game but how good your game engine is working.
The final game should nevertheless be a real original game, with a starting point, a goal to achieve and an end (win or lose). The game should be fun to play for a couple of minutes. Finally you will have to prepare a demonstration of your game engine, so you need to think about a scenario that shows as many functionalities as possible. More information about the demonstration will be given during the last lecture.

More

The mandatory requirement is to implement three tasks out of nine. If you have extra time, you can implement more than three. Do not forget to list the completed tasks during your demonstration.

Submitting your work

The submission deadline is Thursday April 19 at 11:59pm. To send us your work, you will use the CS-UU submission system available at `http://www.cs.uu.nl/docs/submit`. Please DO NOT send your assignment by email.

Design your code in an object-oriented fashion. So, use classes and inheritance where appropriate. Please include sufficient comments to the code so that at least the role of each function and class in your program is clear. Include as much error checking as appropriate. Double check that there is no memory leak. Make sure that you do not copy objects when not necessary.

If your project does not compile and run in both debug and release mode, there will be no grading. The fulfillment of the tasks and the design of your classes will be the major grading factors. Beside that, the final game (look & fun *etc.*), the demonstration of your engine and the resulting quantity and quality of the game engine functionalities will be taken into account. The quantity and quality of the game assets (meshes, textures, audio files) will not be a major criteria.

Please make sure to *clean* your solution (no temporary files and no .sdf). The assignment consists of the solution file(s) (.sln), the project files (.vcxproj, .ico, *etc.*), the source files (.cpp and .h), the asset files (audio, textures, *etc.*), the external libraries (Libs and plugin folders) and the files related to the demonstration (.ppt, .pdf, .avi, *etc.*). Create a .zip or .rar archive of your assignment files and upload it using the submission system. After a successful submission you should receive a confirmation email in your student email account. If not, please contact Jeroen Fokker.