11.2 game @ VU

In june 2005 we started with the development of a game, nicknamed VU-Life 2, using the Half-Life 2 SDK. We acquired a Cybercafe license for Half-Life 2, with 15 seats, because we would like to gain experience with using a state-of-the-art game engine, and we were impressed by the graphic capabilities of the Half-Life 2 Source game engine. After some first explorations, we set ourselves the goal:

- to develop a game that could be used for promoting our institute, and
- to prepare a masterclass game development for high-school students.

Our first ideas concerning a game included a game in which the subject chases a target, a game where the subject has to escape, and an adventure game. In the end we decided for a less ambitious target, namely to develop a game which gives the subject information about our institute, by exploring a realistic game environment, representing part of our faculty. As an incentive, a simple puzzle was included which gives the subject information on how to obtain a 'hidden treasure', to be found in a specific location in the game environment.

1

With only about eight months time, we decided to do a feasibility study first, to gain experience with the Half-Life 2 SDK technology, and to determine whether our requirements for the game and the masterclass could be met. For the VU-Life 2 game, we can summarize our requirements as follows:

- the game must provide information about the faculty of sciences of the VU,
- the game environment must be realistic and sufficiently complex, and
- the interaction must be of a non-aggressive, non-violent, nature.

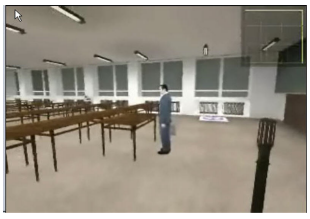
The last requirement has to do with the fact that the VU is by its origin a Christian university, so that anny agressive or violent interaction could hardly be considered to be an appropriate theme for a promotional game. For the masterclass, we stated the following requirements:

- it must be suitable for beginners, in particular high school students,
- it must explain basic texture manipulation, and
- offer templates for modifying a game level, and finally
- there must be a simple (easy to understand) manual.

The format for a masterclass for high-school students at our institute is three times two hours of instruction. The goal is to attract (more) students for the exact sciences. However, if the masterclass would be too complex, we would run the risk to chase potential students away, which would be highly counter-productive.

VU-Life 2 – the game

To give an impression of the game and how we used the Source game engine and the associated Half-Life 2 SDK, let's start with a typical game scenario, illustrated with a walkthrough.



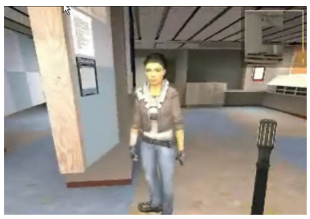
(a) lecture room

(b) lecture room

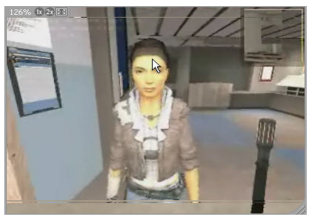
(c) student office

When starting VU-Life 2, the player is positioned somewhere in the game environment, such as a lecture room.. In the front left corner of the lecture room, middle right of left screenshot in the figure above, there is a place marked as an information spot. The information spot corresponds with one of the nine in the top right of the screen. The player is expected to detect this correspondence by exploring the game environment. The nine squares together form a puzzle, indicating, when all squares are filled, where the hidden treasure can be found. In other words, when the player visits all the nine information spots contained in the game environment, the player has solved the puzzle and may proceed to obtain the hidden treasure.

(a) student office



(b) student office



(c) student office

To visit all the information spots, the player has to explore the game environment, including another lecture room, the student administration office, and the student dining room. While exploring the game environment, the player may read information about the curriculum, meet other students, and encounter potentially dangerous individuals.

(a) restaurant (b) restaurant (c) restaurant

4

As illustrated in the figure above, the puzzle squares will gradually become filled, and when complete, the combined puzzle squares will indicate the location of the hidden treasure, which is the 7th row of chairs of the lecture room depicted previously.

Despite the fact that we intended to create a non-violent game, we must admit that the hidden treasure actually consists of obtaining the power to use weapons. Form our observations, and this was exactly what motivated us to include this feature, the use of weapons proved to be a most enjoyable aspect for the high school students playing the VU-Life 2 game, in particular when allowed to play in multi-user mode.

using the Half-Life 2 SDK – technical issues

The VU-Life 2 team had no prior experience with the Half-Life 2 Source SDK. Therefore we started by exploring three aspects of the Source SDK: level design with the Hammer editor, making game modifications, and importing (custom) models into Half-Life 2. During the exploration of these aspects we came across various technical issues, which we will discuss below.

level design First, we made various smaller levels. Each level was compiled and tested seperately so that it worked fine as a standalone level. The idea was to combine them, that is to create one large world containing the smaller levels. However, the initial coupling caused several compiling errors. After analyzing the errors, some important restrictons for building (large) levels became clear.

In the second part of the level compilation process called VVIS, a visibility tree of the level is made. This tree is used to tell the renderer what to draw from a given (player) viewpoint in the level. The amount of used brushes (the default shapes for creating a level) determine the size of the visibility tree. The bigger the tree, the longer VVIS will take to build the visibility tree at compile time and the more work the renderer has to determine what to draw at runtime. Therefore, the standard brushes should only be used for basic level structure. All other brushes that do not contribute to defining the basic level structure should be tied to so-called *func_detail* entities. This makes VVIS ignore them so that they do not contribute to the visibility tree, thus saving compiling and rendering time.

In addition, there is a (hardcoded) maximum to the number of vertices/faces you can use for a level. Each brush-based entity contributes to the number of vertices used. It is possible, however, to reduce the number of vertices used by converting brush-based objects to entities. This is done outside of the Hammer level editor with the use of 3D modelling software and the appropriate conversion tools.

With the above mentioned restrictions in mind we were able to create a relatively large level that more or less realistically represents the faculty of exact sciences of the VU campus. The key locations, as partially illustrated in the figures above, are the restaurant, lecture room S111, lecture room KC159, student office(s), and the multimedia room S353.

To give an impression of the overall size of the *VU.vmf* game level, as map information we obtained 6464 solids, 41725 faces, 849 point entities, 1363 solid entities, and 129 unique textures, requiring in total a texture memory of 67918851 bytes (66.33 MB).

game modifications Since a multi-user environment was required. we chose to modify the Half-Life 2 Deathmatch source code, The biggest challenge for modifying the code was finding out how to implement the features for VU-Life 2. To this end, relevant code fragments were carefully studied in order to find out how the code is structured and works. Furthermore, by experimenting, it was possible to get the features working. Below is a list of features for the VU-Life 2 Mod.

- *player properties* – players start out immortal, meaning that they cannot "die" while exploring the world. Furthermore, continuous sprinting is enabled, which allows the player to walk around faster.
- *puzzle HUD* – when the player starts out, the puzzle HUD is the only HUD element displayed.
- *puzzle setter* – allows puzzle parts to be displayed on the puzzle HUD.
- *weapon enabler* – allows weapons to be enabled/disabled for the player. Enabling the weapons also enables damage, and swithes from the puzzle HUD to the default Half-Life 2 HUD, which displays weapon and damage information along with a crosshair.

importing models Getting a model into the Half-Life 2 environment requires two steps:

- the model must be exported to the custom Valve format smd
- the model must be compiled from *smd* to *mdl* format

The first step required finding the correct plugin that allowed a conversion to the *smd* format. The second step required using Valve tool *studiomdl* and defining a *qc* file, which is used to specify properties for the compiled model. The default Valve tool *studiomdl.exe* proved to be difficult to work with, because it requires a lot of parameters have to be set. By using the StudioMDL 2.0 GUI, compiling the *smd* file was very easy. It sets the appropriate parameters, allowing the user to focus on the compiling of the model.

the masterclass – instruction and assignments

The masterclass consisted of three sessions, two hours each. In the first session, the (high school) students were given an overview and general instructions on how to accomplish the assignments, and were then set to play the VU-Life 2 game.

The assignments, as already indicated previously, were:

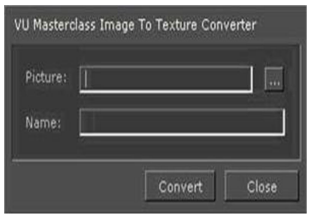
1. to modify an existing game level by applying different textures,
2. to create objects within an existing game level, and
3. (for advanced students only) to create a new level.

More complex assignments, such as creading a Mod, were considered to be outside of the scope of this masterclass.

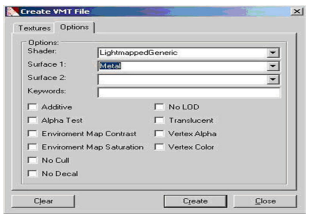
The overview and instructions given in the first session included:

- an overview of the history of games,
- a general introduction on modelling characters and objects,
- the use of the Hammer editor, and finally,
- an explanation of the assignments.

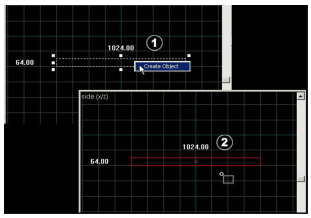
The history of games encompassed historic landmarks such as Pong, Tetris and The Sims, as well as a brief discussion of current games like Worlds of Warcraft, and Half-Life 2.



(a) converter



(b) VMT tool



(c) camera

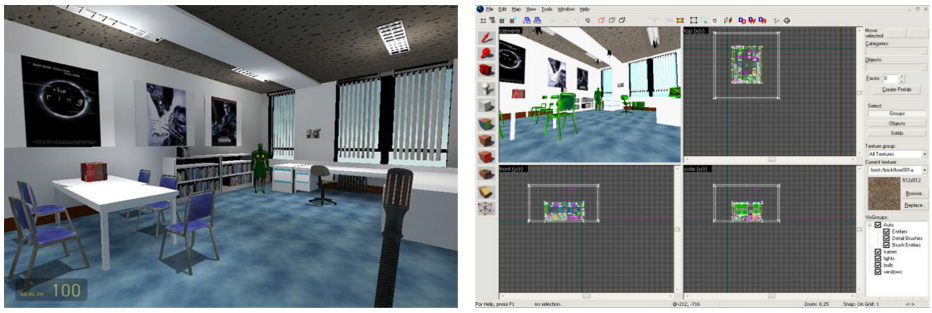
In the introduction on modelling an overview was given of the major tools, like Maya and 3DSMax, as well as a brief explanation of notions such as vectors, polygons, textures, lights, and skeleton-based animation.

Both the explanation of the use of the Hammer and the assigments were explicitly meant as a preparation for session two, in which the students started working on their assignments.

In addition to the oral overview and instructions, the students were given a manual, that was made available in paper as well as online, to prepare themselves for the assignments. The homework for the second session was to make pictures suitable for the application as textures in the *masterclass room*, which is depicted below.

To allow the students to easily apply their textures, a texture conversion tool, was offered, that converts and image file into a texture for a particular location

in the game level based on keywords, e.g. *mc_floor* for the texture on the floor of the *multimedia room*. Alternatively the students could use the VMT-Edit tool, and apply the texture using the Hammer editor.



(a) masterclass room (b) room in Hammer editor

6

The introduction on how to use the Hammer editor covered the basic tools, including the

- *block tool* – for creating simple object,
- *selection tool* – to select objects for texturing,
- *entity tool* – to select dynamic or interactive objects, and the
- *texture tool* – to apply textures to an object;

as well as how to compile a level into a map ready for play, including an explanation of the BSP (world), VIS (visibility), and RAD (radiosity) components.

The students were explicitly told that the assignments did not involve any programming, creating game AI, or modelling. (To learn these aspects of game development, they were simply adviced to sign up for our curriculum.) Instead, we told them, use your phantasy and be creative!

lessons learned

In the second session, the high school students started working with great fervour.

Somewhat surprisingly, all students worked directly from the (paper) manual, rather than consulting the online documentation, or the help function with the tool.

masterclass at work

7

In retrospect, what appeared to be the main difficulty in developing the masterclass was to create challenging assignments for every skill level. In our case, the basic skill level (modifying textures of a template level) allowed the high school students to start immediately. By having optional advanced assignments like creating your own objects, you can keep all students interested, since there are assignments to match the various skill levels.

competition To stimulate the participants in their creativity, we awarded the best result, according to our judgement, with a VU-Life 2 T-shirt and a CD with Half-Life 2. The results varied from a music chamber, a space environment, a *Matrix* inspired room, and a messy study room. We awarded the *Matrix* room with the first prize, since it looked, although not very original, the most coherent.

example(s) – *dead media*

In DeepTime, the *dead media* project is described, in a way that deserves to be presented without any transliteration. The following quotes characterize both the *dead media* project, as well as its context and implications for our (notions) of culture:

civilisation

> Media are special cases within the history of civilisation. They have contributed there share to the gigantic rubbish heaps that cover the face of our planet or to the mobile junk that zips through outer space.

dead media project

> Together with like-minded people, in 1995, Bruce Sterling started a mailinglist (at that time still an attractive option) to collect *obsolete software*. This list was soon expanded to collect *dead ideas*, or *dead artifacts*, and systems from the *history of technical media*: inventions that appeared suddenly and disappeared just as quickly, which dead-ended and were never developed further; models that never left the drawing board; or actual products that were bought and used and subsequently vanished into thin air.

machines can die

> Sterling's project confronted burgeoning phantasies about the immortality of machines with the simple facticity of a continuously growing list of things that have become defunct.

technology and death

> Once again, romantic notions of technology and of death were closely intertwined in the *Dead Media* Project.

How romantic we are, may be felt most intensely when our favorite machine breaks down, or threathens to be destroyed. This occurred to me recently, tripping over an electricity wire, connected to a thinkpad notebook. A small domestic drama occurred, and not even the promise of an improved (tablet) version of the same notebook could drive away the atmosphere of sadness and loss. Fortunately, after

some fiddling with a screw driver, the damage seemed to have been reduced to an occasional blue screen. Yet, the incident illustrates how deeply involved we are with our machines, even if by profession we should know better.

research direction(s) – *service oriented computing*

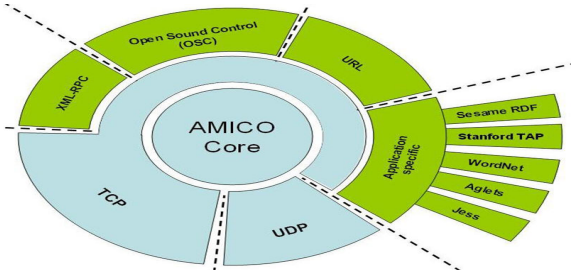
Currently, one of the leading open source game engines is Delta3D[1], which consists of a collection of (open source) software components, well-maintained by a team at the Naval Postgraduate School in Monterey (CA). This section, which was previously called *open source game development*, will look at the way existing game engines may be enhanced by using services, for example web services that may be invoked by a REST-protocol (using urls), SOAP (Sinple Object Access protocol) or XML-RPC.

In AMICOCALC, we present the following scenarion, or problem statement:

scenario(s)

> Michelle is a writer, and she writes a book about Dutch paintings. To collect necessary information, she analyses different sources about Dutch culture, many of them in Dutch language. However, being a beginner in Dutch, she often has to translate phrases from and to English. She also needs to find additional information about particular concepts and facts. Instead of using several tools, such as online dictionaries, definition books and search engines, she decides to compose a simple service that aggregates all the services she needs.

It should be not to difficult to think of similar scenarios in the context of games, for example language or culture training games. A generic solution to this type of scenario(s) is provided by AMICO[2] (Adaptable Multi-Interface COmmunicator), which is, according to its developer Zeljko Obrenovic, a generic platform, used to support rapid prototyping with heterogeneous software services, AMICO. AMICO is based on the publish-subscribe design pattern, which is well suited for integration of loosely-coupled components, and often used in context-aware and collaborative computing. A publisher updates a shared data repository without being concerned with whether any subscribers are listening for updates. In the loosely coupled model, components can run on different machines in a distributed environment. An architectural overview of AMICO is given in the figure belew.



[1] www.delta.org
[2] amico.sourceforge.net

8

Our example scenario illustrates the integrated usage of various software components and services. Service-oriented computing (SoC) has a relatively long history, and may be regarded to have its roots in object-oriented software development, to the extent that services can be considered as components which offer their functionality at a sufficiently high level of abstraction. [Eliens2000]. Services provide higher-level abstractions facilitating implementation and configuration of software applications in a manner that improves productivity and application quality. Most of the existing work in SoC is concentrated on Web services, which is often seen as a main way to bring business to the Web [Blevec07]. Service-oriented computing is, however, not limited to Web services, but embodies key principles such as loose coupling, implementation neutrality, flexible configurability, persistence, granularity, and teams. Services provide higher-level abstractions for organizing applications for large scale, open environments. Adding service abstraction on top of heterogeneous open-source components, for example, enables their easier integration with other systems [Obrenovic07b]. In the category of service-oriented computing solutions, we distinguish between 4+1 dimensions:

dimension(s)

- scope – individual vs. corporate
- platform – local vs. web-based
- functionality – data aggregation vs. interaction
- developers – end-user vs. professionals

Additionally we address the issue of

- licensing – OS vs. proprietary

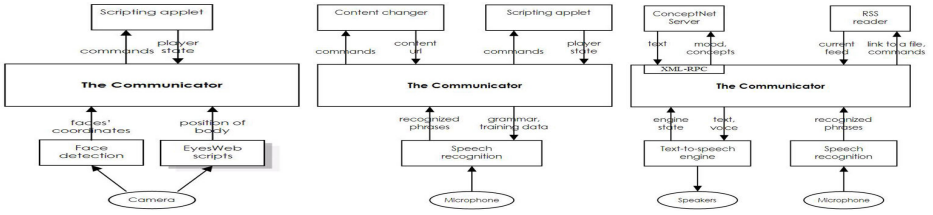
Dependent on how a particular application is positioned in the category of service-oriented computing solutions, different issues play a role. For example, for corporate applications, security is of vital importance. In our approach, which is more targeted to the individual end-users, security plays as consequently less important role. However, other issues, such as time-management, are crucial. We introduce the licensing dimension to emphasis that only a very limited number of the solutions available for the construction of (primarily web-based) services and mashups is available as an open-source projects [Mashups]. In a general fashion, service-oriented computing requires three mechanisms, or layers of functionality;

service-oriented computing

- service brokering – to connect services,
- service adapters – to transform data, and
- integration mechanisms – to deploy services.

Traditionally, due to their complexity, these layers of functionality necessitated the help of professional developers with expertise in CORBA, COM or more recently SOAP, to construct suitable end-user applications. However, with the recent surge of web-based services, exemplified in the increasingly growing number of mashups, service-oriented computing seems to have come closer to end-users. That is end-users with non-trivial programming or scripting skills. For example, some of the

existing end-user environments support extension of functionality by providing an access to Web services. Excel's Web Services [ExcelWS], enable end-users to create a code wrapper for Web services, and use functions from this wrapper within the spreadsheets. Limited forms of combinations of EUD and SoC can be found in Web based spreadsheets, such as Google spreadsheets [GoogleSS]. These environments enable receiving updates from remote sources for small number of predefined data types, such as stock prices and currency exchange rates. As a relatively new phenomenon, mashups introduce end-user service integration to create composed website or application that combines content from more than one source into an integrated experience, usually with a end-user interface. Content used in mashups is typically sourced from a third party via a public interface. Other methods of sourcing content for mashups include Web feeds (e.g. RSS or Atom), web services and Screen scraping. Many people are experimenting with mashups from Google, eBay, Amazon, Flickr, Yahoo [YahooPipes]. Although the dividing line between end-users and professional developers seems to become blurred, for example where end-users learn how to use scripting languages, we wish to emphasize that our notion of end-users is quite strict. An end-user should not be assumed to be able to do any scripting beyond simple formulas and an occasional conditional expression using the IF-THEN construct.



camera input voice input news reader

9

In comparison with the existing solutions, AMICO has a wider scope than most web-based mashups or EUD integration platforms, since it supports not only data-aggregation but, as illustrated in the figure above, provides support also for additional services, including interaction facilities involving speech recognition and TTS output. Yet, it is primarily aimed at individual users, supporting the integration of services for which both a service connection as well as data transformation functionality is available.

End-user development, as we will argue in section 11.4 becomes even more urgent,'where much of the content and functionality is actually provided by (a community of) users. How to provide a generic solution for user-added game content is however still an open problem, for which the service-oriented computing paradigm, however, seems to provide promising solutions.