Abstract

Title: Digital University – Serious Gaming

After following a lecture in one of the lecture rooms you can relax in one of the meeting places. You can look to your right to see a student picking his nose, you silently laugh. Luckily he can't hear you. Suddenly you hear your mother shouting: "Diner!". Within a second you're back in reality, back from the Digital University!

The assignment:
Design and build a virtual Delft university, starting with a lecture room and a meeting place. Eventually there needs to be some interactivity between agents, objects and other players. To accomplish this we need 3D software for modeling/building the 3D environment. After building the environment we can start adding some interaction between the user and the agents or objects in the classroom. The interaction, physics and game play are defined in the game engine. So besides finding a good and functional 3D software package, we also need to look for a game engine which can interact with the 3D modeling tool.

Student information:
Sven Anker
Rotterdam University / CMI
0773594@student.hro.nl

Rob van der Kamp
Rotterdam University / CMI
0772800@student.hro.nl

Supervisors:
Dr. Ir. M.M.M. Abd el Ghany
Rotterdam University / CMI
G.J. de Jonghweg 4-6
3015 GG Rotterdam
abdmm@hro.nl

Dr. drs. L.J.M. Rothkrantz
Delft University of Technology
Mekelweg 4
2628 CD Delft
L.J.M.Rothkrantz@ewi.tudelft.nl

Acknowledgements

First of all we would like to thank our supervisors Ir. M.M.M. Abd el Ghany and Dr. drs. L.J.M. Rothkrantz for the time, the feedback and all the effort they put in us. We thank them for their support, guidance and attention during our internship period.
We also thank them for making it possible for us to work at the Delft University of Technology.

Also many thanks to the technical support staff, Bart Vastenhouw, and Ruud de Jong for their support while facing hardware and software problems.

Sven Anker
Rob van der Kamp
Delft
December 2007

Contents

Abstract………………………………………………………………… 1

Acknowledgements…………………………………………………….. 2

Table of contents

1 Research……………………………………………………………….. 6
1.1 Game engine research………………………………………………. 6
1.1.1 Game engine features scheme…………………………….... 6
1.1.2 Features list………………………………………………. 6
1.1.2.1 Modeling requirements………………………… 6
1.1.2.2 Engine requirements…………………………… 7
1.1.2.3 Game play requirements……………………….. 10
1.1.2.4 Extra……………………………………………. 12
1.2 Research explanation…………………………………………….. 13
1.2.1 Why did we use this research approach?........................... 13
1.2.2 Why these engines?.. 13
1.2.3 Why these features?... 14
1.2.4 Why did we use this information display method?............ 14
1.3 Chosen engine……………………………………………………. 14

2 Blender…………………………………………………………………. 16
2.1 Blender information……………………………………………… 16
2.2 Game Blender information……………………………………… 16
2.3 Blender features and benefits…………………………………….. 17
2.3.1 Features………………………………………………….... 17
2.3.2 Benefits…………………………………………………... 18
2.4 User interface (UI)……………………………………………….. 19
2.5 Interaction by using Blender……………………………………... 22
2.5.1 Game Blender…………………………………………….. 22
2.5.2 Sensors, controllers and actuators logic bricks………….... 23
2.5.2.1 Sensors……………………………………….. 23
2.5.2.2 Controllers……………………………………... 23
2.5.2.3 Actuators……………………………………….. 23
2.6 Physics within Blender………………………………………….... 23
2.6.1 Particles…………………………………………………... 23
2.6.2 Fluid simulation…………………………………………... 24
2.6.3 Soft body…………………………………………………. 26
2.6.4 Rigid body……………………………………………….... 27
2.7 System requirements…………………………………………….... 27

3 Digital University………………………………………………………... 30
3.1 What do we want?.. 30
3.2 The design……………………………………………………... 30
3.2.1 The idea………………………………………………... 31
3.2.2 Sketches (pen & paper, 2D)……………………………… 32
3.2.3 Digital design (2D)……………………………………… 32
3.3 3D environment………………………………………………... 33
3.3.1 Lecture room……………………………………………... 33
3.3.1.1 Objects in the lecture room……………………... 33
3.3.1.2 Where does the interactivity take place?............. 33
3.3.1.3 Why is the lecture room important?.................... 34
3.3.1.4 Object screenshots and explanation……………. 34
3.3.2 Corridor………………………………………………... 34
3.3.2.1 Objects in the corridor………………………… 34
3.3.2.2 Where does the interactivity take place?............ 34
3.3.2.3 Object screenshots and explanation……………. 34
3.3.3 Meeting place…………………………………………….. 35
3.3.3.1 Objects in the meeting place…………………… 35
3.3.3.2 Where does the interactivity take place?............ 35
3.3.3.3 Why is the meeting place important?.................. 36
3.3.3.4 Object screenshots and explanation……………. 36
3.4 Building and animating the characters…………………………… 36
3.4.1 3D Modeling……………………………………………... 36
3.4.2 Armatures……………………………………………….... 36
3.4.3 Animations………………………………………………... 37
3.4.4 Game logic……………………………………………….. 37
3.4.5 Mouselook………………………………………………... 38
3.4.6 Bots………………………………………………………. 38
3.4.7 Path Node………………………………………………… 38
3.5 Lesson materials………………………………………………...... 39
3.6 User Interface Digital University………………………………… 41

4 Conclusion and recommendations…………………………………… 42
4.1 Conclusions………………………………………………………. 42
4.2 Recommendations……………………………………………….. 42

References………………………………………………………………. 43

List of abbreviations……………………………………………………. 44

Appendix……………………………………………………………….. 45

A. Features scheme…………………………………………………........ 46

B1. Sensors………………………………………………………………. 48

B2. Controllers………………………………………………………………… 50
B3. Actuators………………………………………………………………… 51

C. Sketches………………………………………………………………… 54

D1. Screenshot lecture room……………………………………………..... 56
D2. Screenshot corridor…………………………………………………... 57
D3. Screenshot meeting place…………………………………………….. 58

E. Building and animating the characters……………………………… 60

F1. MouseLook.py…………………………………………………………… 64
F2. os.py………………………………………………………………… 66
F3. ntpath.py…………………………………………………………… 71
F4. Stat.py……………………………………………………………… 78
F5. UserDict.py………………………………………………………… 80
F6. callFMV…………………………………………………………….. 80
F7. hasFMVfinished……………………………………………………. 81
F8. playFMV……………………………………………………………. 81

G. Internship report - Digital University

Chapter 1 Research

1.1 Game engine research

Game engines are very important for the game industry. All the interaction, movements, explosion, animation and cameras are controlled by the game engine. So it's very important for a game developer to choose the right engine. For our project we need a game engine with as much as possible options. We need to control the movements of the player, add mass and gravity, control animations and control the cameras.
So before we can start modeling and coding we need to choose an engine which is capable of doing all these things.

The following pages contain all the research documentation we've done considering the game engine. Differences between the engines will let use decide which engine we'll going to use for our project. First of all we need to find multiple game engines. We need to make a list of features; features we consider to be useful for the game engine.
By making a list of these features, we can see what the strengths and weaknesses are of the selected game engines.

After doing the game engine research we can choose a game engine to work with.

1.1.1 Game engine features scheme

Appendix A. Features scheme, page 43 – 44.

1.1.2 Features list

The following pages will explain all the features written in the "Game engine features scheme" (chapter 2.1). All the features are "numbered" from A – V, just like in the scheme. We'll describe every feature and explain why they are useful for our project.

1.1.2.1 Modeling requirements

A. Modeling environment

Modeling environment is also known as a "Model Program Tool" for creating 3D models. This modeling program/tool normally includes: building tools, rendering and analyzing options. Sometimes the modeling environment is integrated in the game engine. These modeling environments are also known as: model-intergraded programs. The Characters, objects and terrain are build with these modeling tools.

Our project needs: (integrated) modeling tool for building characters, surroundings and objects.

B. Rendering system

Rendering systems are used to make a digital image from a model. Textures, lights and shadows become visible. In Serious Gaming this is the last major step, giving the final appearance to the models and animation. There are lots of rendering systems available. Some are integrated in modeling and animation packages, some are stand-alone, some are free open-source projects. There are two types of rendering: pre-rendering and real time rendering. Pre-rendering is a computationally intensive process that is typically used for movie creation, while real-rime rendering is often done 3D video games which rely on the use of graphics cards with 3D hardware accelerators.

Our project needs: a rendering system (integrated in the modeling tool) for rendering the game scenes. This is very useful for creating a realistic environment.

1.1.2.2 Engine requirements

C. Animation

An animation is a simulation of movement created by displaying a series of pictures or frames. Animation is one of the main ingredients for Serious Gaming. Many software applications make it possible to create animations which you can use in your game. Walking cycles, explosives and tree & scenic animations are examples of computer animations.

Our project needs: a game engine with an animation option; to make it possible for our characters to walk through the virtual environment and to make the game look more realistic.

D. Collision

In serious gaming collision detection involves algorithms for checking for collision or intersection of two given solids. Simulating what happens once a collision is detected is sometimes referred to as "collision response", for which see “physics (physics engine)”. Collision detection algorithms are a basic component of 3D video games. Without them, characters could go through walls and other obstacles.

Our project needs: a game engine which has collision detection for preventing the character to walk through walls and other object.

Collision detection is also needed to detect collision between the other objects in the virtual environment.

E. Physics (physics engine)

Computer animation physics or game physics involves the introduction of the law of physics into a simulation or game engine, for purpose of making the effect appear more real to the observer. A physics engine is a computer program that simulates Newtonian physics models, using variables such as mass, velocity, friction and wind resistance. It can simulate and predict effects under different conditions that would approximate what happens in real life or in a fantasy world.

Our project needs: a game engine which has physics for simulating gravity, mass, friction etc. Just to make the game look and feel more realistic.

F. Body dynamic

There are two types of body dynamic: soft body dynamic and ridged body dynamic. Soft body dynamics is an area of physics simulation software that focuses on accurate simulation of a flexible object. The object is deformable, meaning that the relative positions of points of the objects can change. Friction, gravity, collisions, springs, wind are some of the forces who can influence the behavior of an object. Clothes, hair, sand and water are examples of soft bodies. Ridged body dynamic stands for a solid object of finite size which deformation is neglected. The distance between any two given points of the ridged body remains constant in time regardless of external forces.

Our project needs: mostly ridged body dynamic for the characters and objects.

G. Coding / Scripting

After modeling the objects in the virtual environment you can start coding/scripting. Most of the game engines support one or more coding languages. Other engines are working with scripting languages, where you don't have to code anything in the engine. Objects can start moving, transform or disappear by scripting them.

Our project needs: a coding/scripting language to control the game engine.

H. Multiplatform

Multiplatform applications are applications which can run on multiple computer platforms. A multiplatform application can run on all common platforms, or simply more than one. These days everyone has got different needs and wishes and that's why there are many different operating systems that vary in options. Engines also work on certain operating systems. The most engines work on the popular operating systems like Windows, Mac OS and Linux. It is important for an engine to support as many operating systems as possible so the software developed on the engine can reach a wide public.

Our project needs: a game engine which runs on multiple operating systems (multiplatform). So the game can be widely spread. Plus, IF this project is taken over by other students it needs to be possible to run/edit/model on all the other available platforms.

I. Import 3D files

Some engines don't have their own 3D modeling program attached to it, so then its necessary to be able to import files from an extern modeling program. Modeling programs like 3Ds Max, Maya and Milkshake are popular programs used by 3D modelers. If these programs files are supported by the engines, they can be imported to be used in the software.

Our project needs: a game engine who can import 3D files. We prefer to use an engine with an integrated 3D modeling tool. So there will be no problems using the 3D models in combination with the game engine.

J. Costs

There is a great variety of engines. Engines made by companies in order to earn money by selling the engines to game developers and engines made by hobbyist so developers with a low budget or hobbyist can use them for free to develop games. Depending on the purpose of your software you will have to consider which type to choose. The commercial engines may be expensive but they got all the latest technologies included. The free engines may not have all the latest technologies included, but most of the time they do have great support and a big community who helps to improve the engine.

Our project needs: an inexpensive game engine.

1.1.2.3 Game play requirements

K. Interactivity

Interaction is the kind of action that occurs as two or more objects have an effect upon one another. In the gaming world this is the same. For example, if a car crashes into a building, the building gets damaged. So the interaction between the 2 object is that the building and the car got damaged as the result of a crash.
To make a game or simulation realistic you should be able to pick up objects, talk to other players or Non playing characters. This all is called interaction.

Our project needs: a game engine which makes it possible for objects in the virtual environment to "communicate" with each other. If an object does something with a second object, both of the objects need to react on each other.

L. Movement

The movement in a game is depended on the type of game, but is always one of the most important features in a game. Movement in a game is for example, walking with a character. The movement in a game is mostly controlled with the keyboard or mouse.

Our project needs: a game engine which supports a keyboard (and probably a mouse) to control the movement of the character / player.

M. View (Camera)

The view in a game is controlled by cameras. Some games got cameras that can be controlled by the user and some don't. The view is also defined by the type of game. There are first person, third person and "free camera" games. In first person camera view you see what is in front of the object you are controlling. That means looking trough the eyes of a character or see what is in front of a car. In third person camera view, the camera is behind the character or object you control. For example: looking at the back of a character. The "free camera" view means that the user can control the camera by itself. This camera view is mostly used in games that have a big area to overview. The view in a game is mostly controlled with the computer mouse.

Our project needs: different cameras, for different camera views. Some camera's need to follow our character, others need to stay focused on one particular spot.

N. Artificial Intelligence

Artificial Intelligence is an object or character that is acting out of his own. This object or character is programmed to react on his environment. It is programmed to have a mind of his own, based on certain factors it moves and/or undertakes actions. An AI controlled character is always trying to maximize his chances of survival and/or success. Many games that involve enemies with weapons use advanced AI controlled characters that attack the player when spotted or when the player attacks them. There are various combinations of AI and gaming. AI also is a part of interaction as you can interact with computer controlled characters.

Our project needs: objects which are artificial intelligence. They need to react on the actions made by the player.

O. Audio & Video

Audio and video make a game more realistic. Sounds of doors opening, guns firing and the running engine of a car. All these sounds add realism to the game. What would a game be without sound? Sound can express happiness and drama.
Video's in games make it possible to explain certain things or give an intro to the game. All games these days got storylines which they show trough videos.
Videos in games also show commercials or promotion material of well known companies.

Our project needs: a game engine which supports audio and video. We prefer a game engine with integrated audio and video options. We want to be able to stream video in-game. We also want the objects/characters in-game to interact with the player, using sounds.

P. Menu Building

When you start a game you always begin in the main menu. The menu displays the options of the game. For example: Single player, Multiplayer, Options, and Exit. A menu makes it easier for the user to navigate to the program and also gives the first impression of the game.

Our project needs: an option for building menus. We need a menu to start the game, to edit game options and stop/exit the game.

Q. Multiplayer

The most games have a storyline to follow or single player missions, but some players want to compete against other players just like them. Multiplayer makes it possible for people from all over the world to play together. The AI in these days games are very advanced but the human brain still is different from the computers "brain".

Our project needs: a game engine which supports multiplayer options. Like an integrated networking engine. So multiple players can join the server and interact with each other in-game.

R. Text Display
By displaying text in a game the player can get all kind of information like; what to do, where to go or who to talk to. Some engines can display text by using bitmap images. Others engines got the option "text display" integrated.

Our project needs: an option to display text in-game.

1.1.2.4 Extra

S. Documentation
In order to program and configure the engine properly, knowledge of the engine is needed. When the programmers of the engine are not that well known with the engine they can look up certain subjects in the engines documentation. Having documentation delivered with the engine thereby is a big plus when purchasing an engine.

Our project needs: lots of documentation made by the creators of the game engine, plus documentation written by users.

T. Tutorials
When the documentation of an engine (if supplied with the engine) isn't very clear on some points tutorials can help out. Tutorials are examples of a certain subject explained step by step. Good tutorials are easy to follow and can be a big help when work with the engine.

Our project needs: tutorials written by the creators of the game engine, plus tutorials written by users. This will probably increase our level of success. This is the fastest way to learn the basic of a modeling tool or game engine.

U. Community/Forum
If u encounter problems while developing software with an engine and the documentation doesn't contain an answer to the problem you can always ask the community of that engine for help. The

community of an engine mostly consists of users that use the particular engine. They got experience with the engine and may have encountered the same problem before and can help you to solve the problems. These communities' share there experiences on forums.

Our project needs: a game engine with a community behind it, willing to help others.

V. Online support
Online support is the kind of support where u can contact the developers of the engine for questions considering the engine. This might be useful if no one can help you out with a problem or if you have any other question considering the engine. Online support is a service towards the buyer of the engine and not every engine developer has this service available.

Our project needs: a game engine with some kind of online support. This isn't necessary, but it can be very helpful when you're having problems.

1.2 Research explanation

We'll now explain why we used this kind of research, why we selected the engines and why we made the "Game engine features scheme".

1.2.1 Why did we use this research approach?

We are researching different engines because of the great offer in engines. These days more and more engines are being developed. There are free engines and engines you will have to pay for. Our goal was to find the best engine for our project, but we couldn't just take any engine. We preferred a free engine, but there are lots of free engines and they're not all the same in options and quality.
This research has to point out which engine is best fitted for our project.

1.2.2 Why these engines?

Why did we pick these particular engines to research and not other engines? It's pretty easy to answer; we chose these engines based on what engines we found on the internet and our own experience with game engines. Most of the engines are free of use. We also chose some engines that cost money to purchase in order to see if there is a big difference between freeware and commercial engines. 3D Blender and Second Life were known to us. Irrlicht, Virtools and Torque are engines that we found on the internet and Half-life is a commercial engine.

1.2.3 Why these features?

Not every engine has all the latest techniques, but engines that do have all the techniques are too expensive. Looking at all techniques we needed for our project we came up with a list of all features that the engine needed to contain. So we know which engine is the best for our project.

1.2.4 Why did we use this information display method?

We wanted the information of the research to be displayed as good as possible. That is why we chose to display the information in one big table. The engines are displayed on the top of the table and all features are displayed on the left. When an engine contains the feature noted on the side a cross is placed in the cell of the table where the feature and engine come together. This way of displaying information makes it easy to quickly see which engine has what features.

1.3 Chosen engine

Choosing the right game engine for our project wasn't that easy. There are many game engines available, but there are only several good engines.
We needed to research which game engine is the best for our project. We started our research with making a list of features which our game engine needs to support. When the list was completed, we started looking for game engines that met the features of our list. We found several engines on the internet and we used game engines that we already knew. We listed the engines in a table, together with the features. By placing crosses in the table if a game engine supported a feature, we could easily see which game engine supported what features.

Eventually we compared the engines depending on which features they have. We didn't only take a look at the features, but also if the engine wasn't too difficult to work with. According to the time schedule there isn't much time to learn the basics of programming the engine. We also looked if the engine had a modeling environment included. This makes sure that the models will work with the engine. Extern modeling programs may make things more difficult then with an integrated modeling tool. Combining all these requirements we chose the engine which, to us, seemed to be the best engine for our project.

Game Blender is the game engine we chose. Game Blender is a game engine which is very complete; it contains a modeling environment and an environment where games can be created.
Blender contains all the features we need for our project, Blender is free to use and has a great community that supports it.

Chapter 2
Blender

2.1 Blender information

Blender is open-sourced, community developed software based in the Netherlands. The software allows for the design and development of 3D models, animations, photo realistic graphics, architectural walk-throughs, and 3D games. The growing success of the blender program comes from contributors world-wide. Each contributor works freely to enhance the program. Blender is comparable to programs like Maya, Lightwave, and 3D Studio. The biggest difference is that Blender is free.

Blender was developed as an in-house application by the Dutch animation studio Neo Geo and Not a Number Technologies (NaN). It was primarily authored by Ton Roosendaal, who had previously written a ray tracer called Traces for Amiga in 1989. The name "Blender" was inspired by a song by Yello, from the album Baby.
Roosendaal founded NaN in June 1998 to further develop and distribute the program. The program was initially distributed as shareware until NaN went bankrupt in 2002.
The creditors agreed to release Blender under the terms of the GNU General Public License, for a one-time payment of €100,000. On July 18th 2002, a Blender funding campaign was started by Roosendaal in order to collect donations and on September 7th 2002, it was announced that enough funds had been collected and that the Blender source code would be released. Blender is now an open source program being actively developed under the supervision of the Blender Foundation.
The Blender Foundation initially reserved the right to use dual licensing so that, in addition to GNU GPL, Blender would have been available also under the "Blender License", which did not require disclosing source code but required payments to the Blender Foundation. However, this option was never exercised and was suspended indefinitely in 2005. Currently, Blender is solely available under GNU GPL.

Blender has a great verity of options. Starting with the modeling tool integrated in the game engine (Game Blender). This has some advantages comparable to the other modeling tools and game engines. These advantages of Blender will be explained later on.

2.2 Game Blender information

Game Blender is a sub-application of Blender, the popular open source 3D application, used to make games using Blender. It is an outgrowth of the application that Blender once was, which was a 3D application to make games for the Sony Playstation. The new Game Engine was written from scratch in C++, including support for standards like Python scripting and OpenAL 3D sound. Blender, being programmed in C and Game Blender in C++ kept development strictly separated.

Erwin Coumans and Gino Van Den Bergen developed Game Blender in 2000. The goal was to make a saleable commercial product that users of the freeware Blender could use to create games and real-time presentations. These games could either run as stand-alone applications or embedded in a web page; using a special plugin created from the Game Blender sources. An alpha version of the Internet Explorer browser plugin is on preview, and Firefox and COLLADA support is under consideration. Game Blender is used by inserting "logic bricks," "controllers" and "actuators" to control the movement and display of objects in the engine. Game Blender is also able to be extended via the Python programming language.

After version 2.37a was released, the game engine was almost completely stable, but it wasn't until version 2.41 that a complete and stable version of the Blender game engine was released. Currently, a team is working on developing Blender, releasing many new additions and changes periodically. Version 2.42 shows even more additional features being implemented into the game engine, including integration of the Bullet Rigid Body Dynamics and Vehicle Physics.

2.3 Blender features and benefits

Based on the research done in chapter 3 “Game engine research”, this chapter extricates the abilities and features of Blender. Blender has a great variety of options. Starting with the modeling tool integrated in the game engine. This has some advantages comparable to the other modeling tools and game engines. These advantages of Blender will be explained later on.

2.3.1 Features

Blender has a lot of build-in features, these features are listed below:

- Modeling features
    - Modeling environment
    - Rendering system
- Game Engine features
    - Animation
    - Collision
    - Physics
    - Body Dynamic
    - Coding/scripting
    - Multiplatform
    - Import 3D files
    - Costs
- Game play features
    - Interactivity
    - Movement
    - View (camera)
    - AI
    - Audio & video support

- Menu building
- Multiplayer
- Text display

- Extra
  - Documentation
  - Tutorials
  - Community/forum
  - Online support.

These features are explained in the document: "Game engine research". By having all these features, Game Blender has advantages comparing with other engines. These advantages are listed in chapter 2.2 "Benefits".

2.3.2 Benefits

The game engine of Blender, Game Blender; has some benefits comparing with the other game engines we found. They are as following:

1. Integrated environment; with modeling, animation and game play.

2. Framework with a collection of modules for interactive purposes like physics (rigid body dynamics), graphics, logic, collision simulation, sound and networking.

3. GameObjects can behave autonomously by having a set of tools called LogicBricks and properties. Properties act as the memory, sensors are the senses, controllers are the brain and actuators allow for actions in the outside world (muscles). So there can be interactivity by using the predefined sensors and Logic Bricks.

   Logic Bricks:
   - Sensors are like the senses of a life form; they react on key presses, collisions, contact with materials, timer events or values of properties.
   - Controllers are collecting events from the sensors and are able to calculate them to a result (brain).
   - Actuator performs actions on objects (motion etc.).

   The logic is connected (wired) with the mouse, Sensors to Controllers and Controllers to Actuators. After wiring you are immediately able to play the game! If you discover something in the game you don't like, just stop the game engine, edit your 3-D world and restart. This way the development time is drastically cut down!

4. Virtual reality, consisting of content and behaviors (physics, animation and logic).

5. Blender acts as a complete development tool for interactive worlds including a game engine to play the worlds.

6. Powerful scripting language Python for more advanced game play control.

7. True Multiplatform, All flavors of Windows, Linux, FreeBSD, BeOS, Irix and more.

8. Blender is still being used by a large group of people. So Blender has a large community, which is willing to help you with any problem.

9. Zero costs.

2.4 User interface (UI)

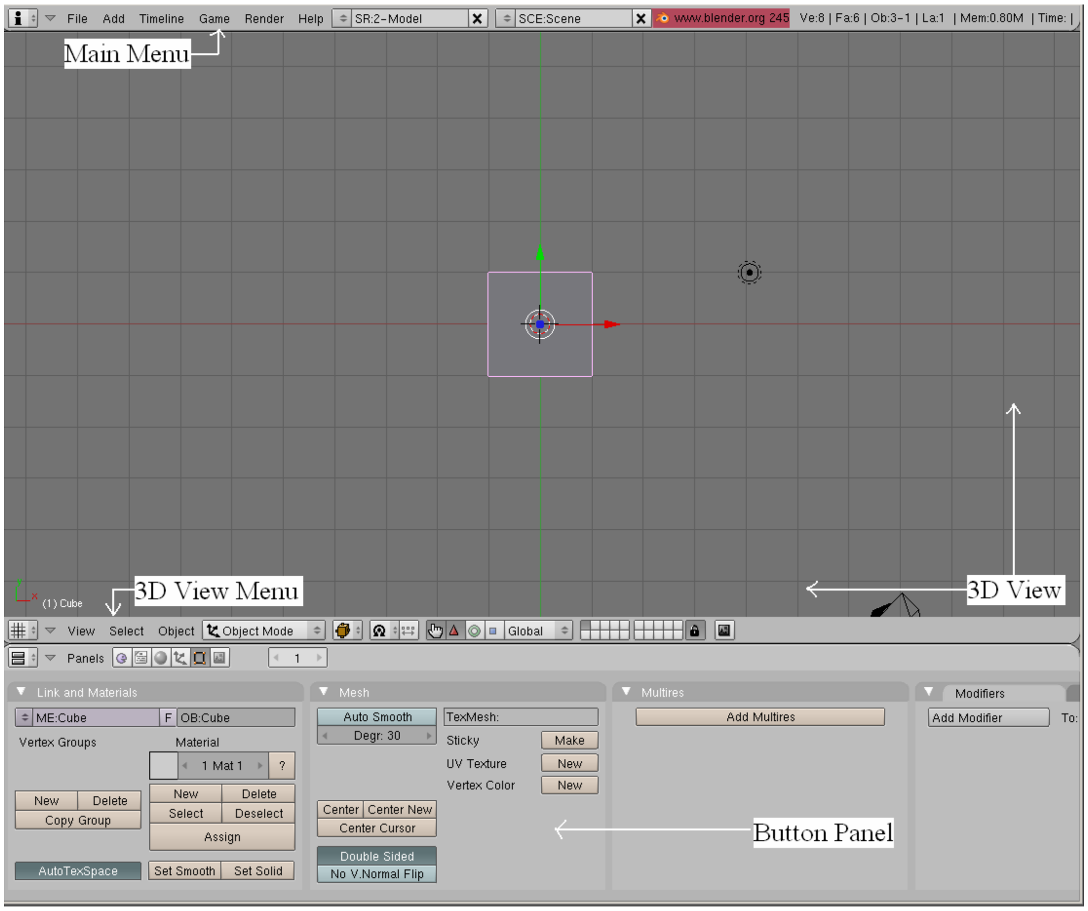


Figure 2.4.1 User Interface Blender

- The main menu panel is located at the top of the screen.
- The main 3D view is shown as a grid area. It shows the current scene from the top.
- The menu for the 3D view is located below it.
- At the bottom of the screen, you will see the Buttons Panel. The GE has its own panel.

Blender's user interface is based on splittable and joinable windows. The main system is basically a grid with edges splitting the parts. The edges can be freely manipulated. Each window has a window type.
Actually even the main menu at the top of the screen is a window. It is a window that contains settings of Blender. The window can be dragged downwards to reveal these settings. The menu can be put anywhere you like on the screen or even disabled. This is a big difference compared to conventional software.
Each window contains a header. Header contains basic menus and commands. You can move the header by using middle mouse button for instance to find more commands. Menus contain the shortcuts to the commands and can hence be used as reference when needed.
One of the most important windows in Blender is the Buttons Window. Buttons Window consists of panels that can have subpanels. Panels have been categorized and their contents may vary depending on mode you are working in. You can find essential commands and tools such as rendering settings there.

Blender has several window types:
- *Scripts Window*: provides access to all registered Blender Python scripts and a place for GUI scripts to draw in.
- *File Browser*: Blender will use this window whenever it asks you to load and save.
- *Image Browser*: like the File Browser, but shows thumbnails of image files it finds on disk.
- *Node Editor*: a newly introduced and very exciting way to handle materials borrowed from Softimage|XSI’s powerful render tree editor.
- *Buttons Window*: this is easily the most complex and the most used window type. It is worth your time to become very familiar with it. Many of Blender’s powerful features are found inside.
- *User preferences*: provides the Blender main menu. Also allows you to manipulate things like; mouse and widget display and user interaction, themes, autosave, OpenGL lighting, memory, system sound, video, file paths etc.
- *Text Editor*: a very simple plain-text editor with syntax highlighting and editing features for Python scripting.
- *Audio Window*: used for audio sequencing.
- *Timeline*: a new window to help with animation editing and playback.
- *Video Sequence Editor*: postproduction editing.

- *NLA Editor*: The Non-Linear Animation editor is one of the most powerful features in Blender. With it you can blend actions and objects IPO's together.
- *Action Editor*: this is another useful, and well-used, animation editor.
- *IPO Curve Editor*: IPO is short for interpolated. All animation is interpolation between keys, or values at a specific time. Values include: position, rotation, color, action etc. Learning about IPO's is very important for animation.
- *3D View*: this is the window where you can create, edit, position and animate your objects.

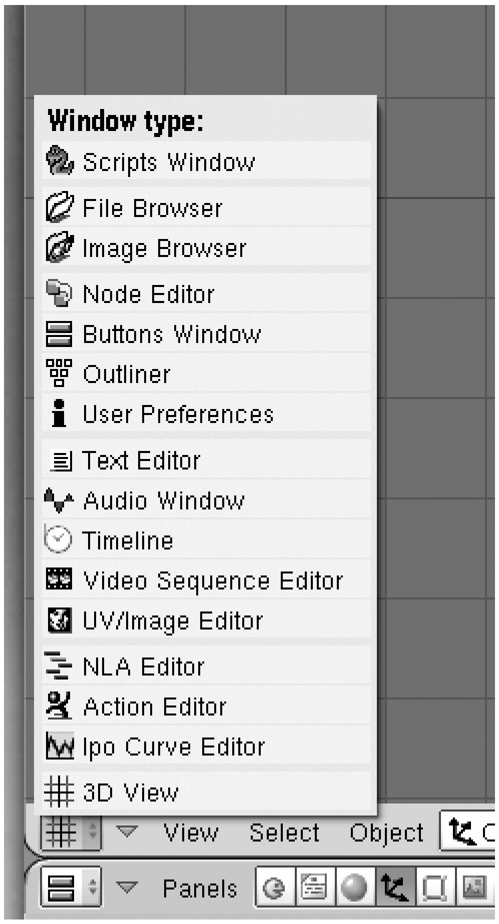


Figure 2.4.2 Window types Blender

Blender uses a widget to move, rotate or scale objects. A widget is sometimes called a manipulator. A widget has three colored handles to drag or manipulate objects. Each handle is color coded to identify one of the three coordinate axes:

- Red: x-axis
- Green: y-axis
- Blue: z-axis.

By dragging one of these handles, the object will move, rotate, scales along the axis of the handle (dragging a red handle moves, rotates or scales the object along the x-axis).

Blender has three widgets:

Transform Rotate Scale

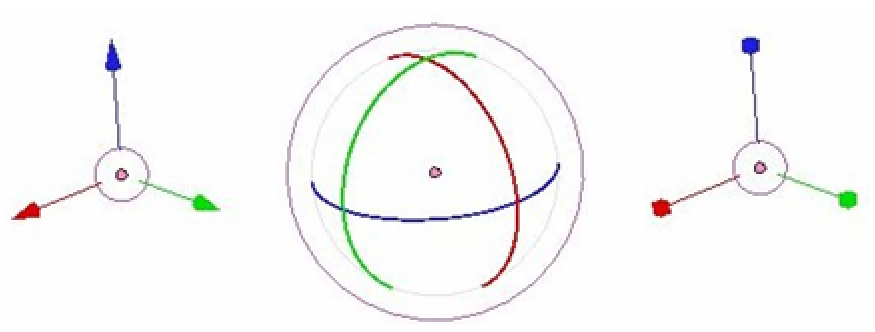


Figure 2.4.3 Widgets

2.5 Interaction by using Blender

Interaction is the main ingredient for gaming. Blender uses the game engine, Game Blender, for creating this interaction.

2.5.1 Game Blender

The game engine of Blender 3D is called: “Game Blender”. The engine controls all the movements made by objects, creates gravity; using the physics engine and controls the interaction between the objects. To make the objects react or interact with each other, they need to be scripted. Game Blender is used by inserting “logic bricks”. These logic bricks can be a sensor, controller or actuator to control the movement, and display of objects in the engine. Game Blender is also able to be extended via the Python programming language.

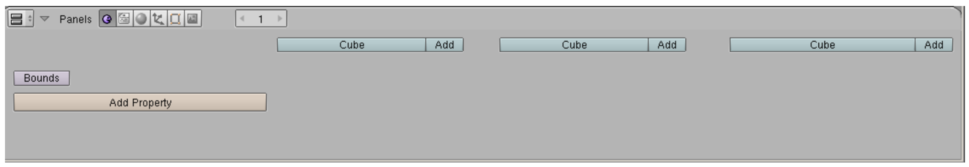


Figure 2.5.1.1 Game Logic Control Panel

Blender uses a visual click-and-drag system to create basic game interactions.

- **Sensors;** A sensor will detect some form of input. This input could be anything from a key press, a joystick button or a timer that triggers every single screen update (or frame) of the game.
- **Controllers;** Controllers are used to link Sensors to Actuators. They allow for some more complex control over how sensor and actuators interact with each other.
- **Actuators;** An actuator will actually carry out an action within the game. This can include moving an object within a scene, playing an animation, or playing a sound effect.

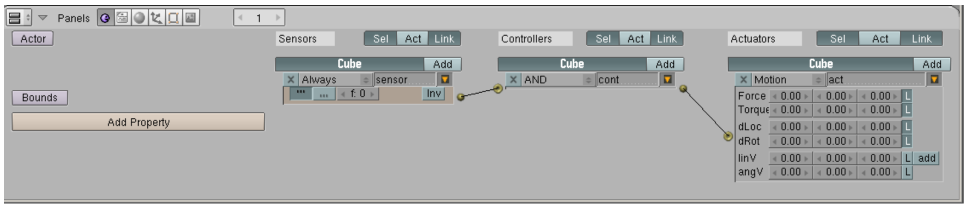


Figure 2.5.1.2 Game Logic Control Panel with a Sensor, Controller and Actuator selected for the object: Cube.

2.5.2 Sensors, controllers and actuators logic bricks

Game Blender has 28 different sensors, controllers and actuators. All these logic bricks make it possible for objects to start "thinking" for their own.

2.5.2.1 Sensors

Appendix B1. Sensors, page 45 – 46.

2.5.2.2 Controllers

Appendix B2. Controllers, page 47.

2.5.2.3 Actuators

Appendix B3. Actuators, page 48 – 49.

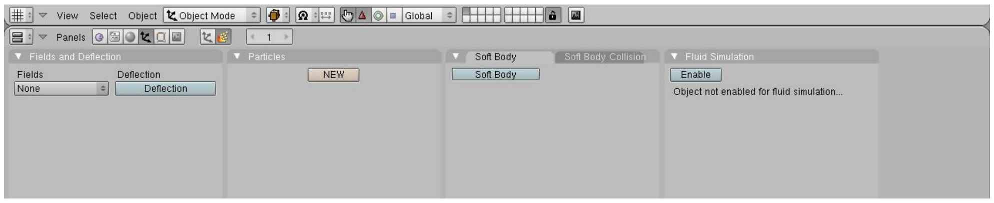
2.6 Physics within Blender

A physics engine is a computer program that simulates Newtonian physics models, using variables such as mass, velocity, friction and wind resistance. It can simulate and predict effects under different conditions that would approximate what happens in real life or in a virtual world.

2.6.1 Particles

To create particles, you will first have to enter the Object menu (F7) or press the icon with the three-way arrows (the one in a row of 6 icons), which is

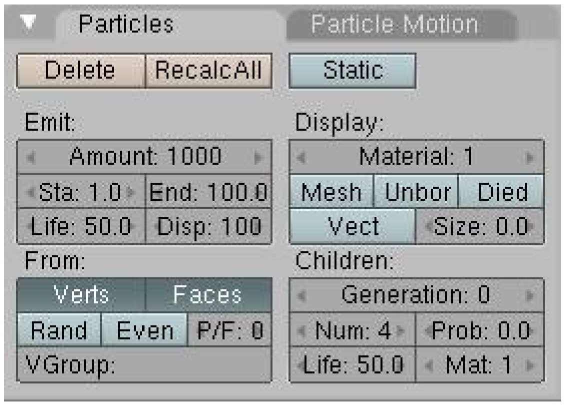
the icon for the object menu. Then u will have to select the Physics button which is the icon with the orange/yellow dots. When u clicked these icons, a menu will show up with 4 fields.
These fields are Fields and Detection, Particles, Soft Body and Fluid simulation. To enable Particles press the NEW button in the Particles field.



Object not enabled for fluid simulation...

Figure 2.6.1.1 Physics buttons

When the NEW button is pressed, a new menu appears. The menu is divided in 4 submenu's, Emit, Display, From and Children. The Emit submenu is the most important menu for the creation of the particles. In this menu you set the amount of particles and how long these particles live and when they end. Particles are mostly textured, because otherwise you won't be able to see the particles. You can also use objects like spheres or cube's to visualize the particles.



Amount: 1000
Sta: 1.0
End: 100.0
Life: 50.0
Disp: 100
Material: 1
Size: 0.0
P/F: 0
Generation: 0
Num: 4
Prob: 0.0
Life: 50.0
Mat: 1

Figure 2.6.1.2 Particles

- *Amount*: The total number of particles that will be emitted
- *Sta*: The starting frame of emission
- *End*: The last frame of emission
- *Life*: How long the particles will exist after emission
- *Disp*: Percentage of particles displayed and calculated in 3DView.

2.6.2 Fluid simulation

To create a Fluid Simulation, you will first have to enter the Object menu (F7) or press the Object menu icon (The three-way arrows icon (the one in a row of 6 icons)). Then u will have to select the Physics button which is the

icon with the orange/yellow dots. When u clicked these icons, a menu will show up with 4 fields.
These fields are Fields and Detection, Particles, Soft Body and Fluid simulation. To enable Fluid Simulation press the Enable button in the Fluid Simulation field.

If the Enable button is pressed a menu will appear. This menu contains 6 buttons: Domain, Fluid, Obstacle, Inflow, Outflow and Particle.

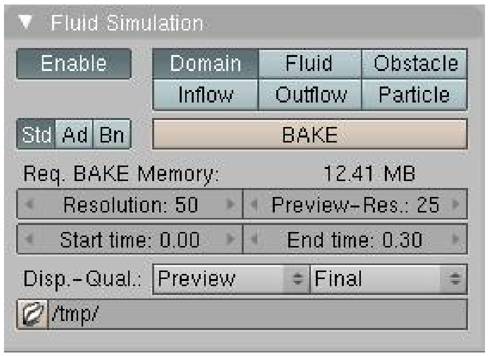

Fluid Simulation
Enable | Domain | Fluid | Obstacle | Inflow | Outflow | Particle
Std | Ad | Bn | BAKE
Req. BAKE Memory: 12.41 MB
Resolution: 50
Preview-Res.: 25
Start time: 0.00
End time: 0.30
Disp.-Qual.: Preview | Final
/tmp/

Figure 2.6.1.3 Fluid simulation

To create a Fluid Simulation there have to be at least two objects. One object is the Domain which normally is a cube. The Domain is the space where the fluid simulation is performed. All fluid objects outside of it are ignored, and the fluid can not flow out of the domain.
The second object is the fluid. The Fluid object is always placed within the domain. When this scene is simulated the Fluid will release within the domain and act like a real fluid would.

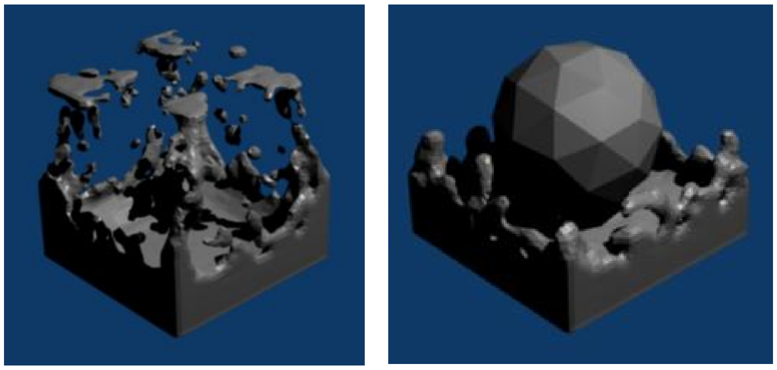
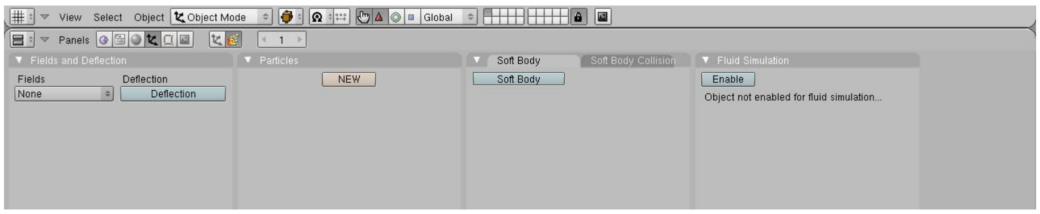


Figure 2.6.1.4 Fluid simulation - animated

2.6.3 Soft body

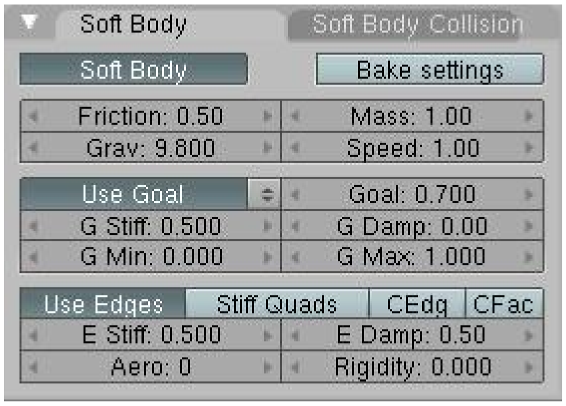
To make an object Soft Body, you will first have to enter the Object menu (F7) or press the icon with the three-way arrows (the one in a row of 6 icons), which is the icon for the object menu. Then u will have to select the Physics button which is the icon with the orange/yellow dots. When u clicked these icons, a menu will show up with 4 fields.
These fields are Fields and Detection, Particles, Soft Body and Fluid simulation. To enable Soft Body press the Soft Body button in the Soft Body field.



Fields: None
Deflection: Deflection
Object not enabled for fluid simulation...

Figure 2.6.3.1 Physics buttons

When the Soft Body button is clicked a menu appears with some parameters for the Soft Body. The most important parameters are Friction, Mass, Grav and Speed.
Adjusting the parameters gives different results regarding to how the object behaves.



Friction: 0.50
Mass: 1.00
Grav: 9.800
Speed: 1.00
Goal: 0.700
G Stiff: 0.500
G Damp: 0.00
G Min: 0.000
G Max: 1.000
E Stiff: 0.500
E Damp: 0.50
Aero: 0
Rigidity: 0.000

Figure 2.6.3.2 Soft body settings

- *Friction*: A generic force against movement that acts on all vertices. A value of zero means no Friction.
- *Mass*: The mass of the body in kilograms. Will be shared equally among all vertices. A higher mass will make the object harder to stop, and the action of force fields will be smaller.
- *Grav*: The local gravity, it's always pointing the negative z-axis.
- *Speed*: A tweak used while solving the movement. Don't modify, unless you have a good reason to do so.

2.6.4 Rigid body

To make an object Rigid Body, the following actions need to be executed. First you will have to select the object that has to become Rigid Body.
While having the object selected, enter the Logic menu (F4). You can also press the icon for the Logic menu which is the icon with the purple Pacman. By clicking this icon the Logic menu appears. This menu contains 4 fields, Actor/Bounds, Sensors, Controllers and Actuators. For Rigid Body only the Actor/Bounds menu is needed.

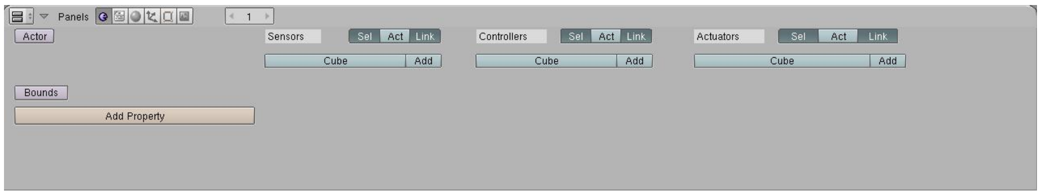


Figure 2.6.4.1 Logic panel

In the Actor/Bounds menu press on the Actor button in order to display new options. Two extra buttons will appear; Ghost and Dynamic. For the Rigid Body function we will have to choose Dynamic. When the Dynamic button is pressed two extra buttons will appear, Rigid Body and No sleeping. Press the Rigid Body button and your object will become Rigid Body.

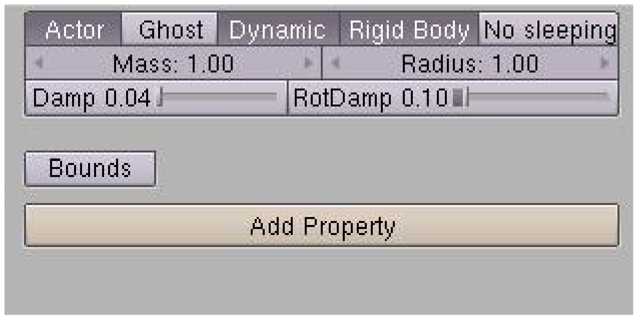

Actor | Ghost | Dynamic | Rigid Body | No sleeping
Mass: 1.00 | Radius: 1.00
Damp 0.04 | RotDamp 0.10
Bounds
Add Property

Figure 2.6.4.2 Actor

2.7 System requirements

Operating Systems:

- Windows 98, ME, 2000, XP or Vista
- Mac OS X 10.2 and later
- Linux 2.2.5 i386
- Linux 2.3.2 PPC
- FreeBSD 6.2 i386
- Irix 6.5 mips3
- Solaris 2.8 sparc.

Optimal specs for hardware:

- 2 Ghz dual CPU
- 2 GB Ram
- 1920 x 1200 px Display with 24 bit color
- 3 button mouse
- Open GL Graphics Card with 128 or 256 MB Ram.

Chapter 3
Digital university

3.1 What do we want?

When creating a Digital University there are several crucial segments. Of course you will need a system for interactivity, so people can interact with each other. You will also need some kind of Internet system so students will be able to log on to the University. You also need sound effects, but after all one of the most important segments is a 3D environment. Without a 3D Environment there is no University. The users of the University won't be able to see anything and won't be able to follow lessons as there is no visual teaching material. The users would also not be able to see any teachers, students, books etc.
Without a 3D Environment there visually wouldn't be a Digital University.

This report will contain information regarding the 3D environment of the Digital University TU Delft. The report will explain the choices and decisions that were made during the development of the 3D environment. All the information in the report is supported by matching illustrations. The report will attend to the design of the Digital University and the different areas the Digital University exists of. These points will be highlighted in their own chapters and subchapters. We will also share our view and experiences on building and designing the Digital University 3D environment.

3.2 The design

Before we can start modeling objects in Blender 3D, we need to start thinking about a plan. Therefore we need to make a design of the 3D environment, which includes all the objects we're going to build, an interactivity plan (painted on the map) and we need to find out where the cameras should be positioned within the environment.

There are three design phases:
1. The idea
2. Sketches (pen & paper – 2D)
3. Digital design (2D).

The first phase (*phase 1. The idea*); during the first phase the assignment will be defined. Defining the assignment makes there will be no discussion possible about the assignment later on. After the assignment has been defined, we can have a look at how the 3D Environment should look like in game. By writing down these ideas, they can later on be used to create sketches.

The second phase (*phase 2. Sketches*); during the second phase several sketches will be made to give us an impression of the (to develop) 3D environment.

The third and last phase (*phase 3. Digital design*); this phase will give us the final 2D impression of the 3D environment. All the models and objects will be created according to the 2D digital drawings.

3.2.1 The idea

The assignment is documented in our report: "The Assignment". It tells us what to do and provides us some guidelines. The assignment summarized:
Design and build a virtual TU Delft, starting with a lecture room and a meeting place. Eventually there needs to be some interactivity between agents, objects and other players.
To accomplish this we need to write down some ideas, about how the environment should look like.

3 different areas:

- **Lecture room:**
  - § A room to meet other students and teachers
  - § A room to follow lectures:
    - Presentations on demand
    - Should be able to ask the teacher questions
  - § A room with lots of interaction:
    - Interaction between player and teacher
    - Interaction between students (bots/agents - virtual students):
      - o Bots/agents notice when you enter the room
      - o Bots/agents notice when you look at them
      - o Bots/agents and the player should be able to communicate with each other (chat or voice)
    - Interaction between objects in the room (able to touch etc.).

- **Corridor:**
  - § An area to walk from the lecture room to the meeting place
  - § An area to meet other students (lots of bots/agents walking around)
  - § An area with interaction between bots/agents:
    - Collision detection
  - § An area which provides information (using posters etc.).

- **Meeting place:**
  - § A place to meet other students and teachers
  - § A place to relax and eat
  - § A place with interaction:
    - Interaction between the player and other students
    - Interaction between objects in the room (pick up a glass etc.)
    - Bots/agents notice when you enter the room
    - Bots/agents notice when you look at them

- Bots/agents and the player should be able to communicate with each other (chat or voice).

3.2.2 Sketches (pen & paper, 2D)

The ideas are defined and approved by Mr. Prof. Rothkrantz and Mr. Ir. Abd el Ghany, so now we can start sketching the environment on paper. These sketches represent the 3D environment (in 2D). They give us an impression how it would look like in Blender 3D after modeling all the objects. Heights, lengths, widths and scale aren't important for the sketches.
Appendix C. Sketches, page 50.

3.2.3 Digital design (2D)

The sketches can now be repainted on the computer. This is the last chance to modify the designs. All the digital designs/illustrations are drawn in Jasc Paint Shop Pro 9.

The digital designs:

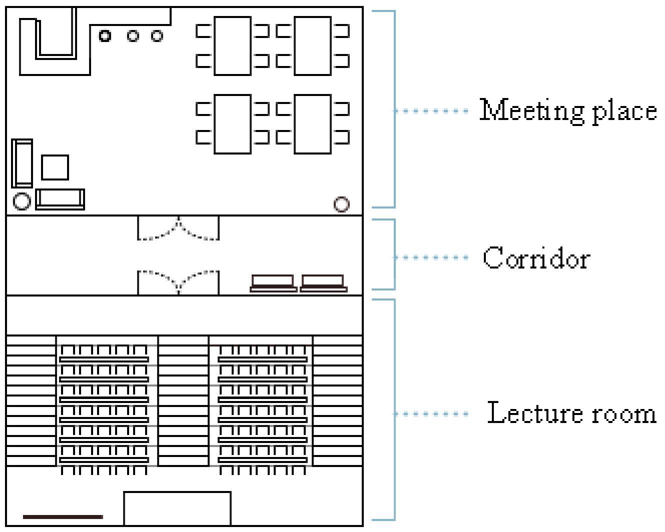


Figure 3.2.3.1 Digital design - top view

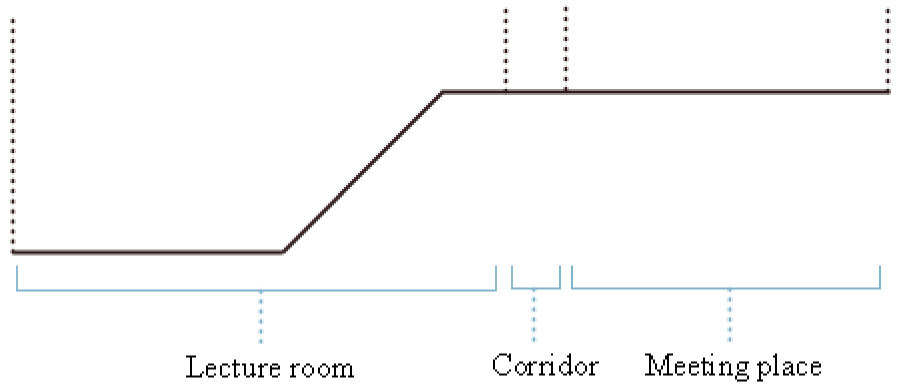


Figure 3.2.3.2 Digital design - side view

3.3 3D environment

Now we have some idea of how the 3D environment should look like, so we can start modeling in Blender 3D. Most of the objects are made from scratch, but some are modeled in 3D Studio and later on imported into Blender. The Blender libraries aren’t really useful for our project so we need to search for FREE user-created objects.
The following chapters will explain why we created the lecture room, the corridor and the meeting place.

3.3.1 Lecture room

The lecture room is the room where users can follow lectures. This is the main room in the current version of the digital university, without it, the digital university would not have any educational meaning.

3.3.1.1 Objects in the lecture room

- Projector screen
- Lecture room seating and desk
- Teacher’s desk
- Door.

The lecture room’s most important objects are the lecture room seating’s and desks, and the projector screen. These objects are most important because of their vital function is this room. The lecture room seating’s make sure the users are able to follow lectures comfortable and the desks create a place for the user’s (exercise) books. The projector screen creates an opportunity to show presentations or other media to the users.

3.3.1.2 Where does the interactivity take place?

The interaction in the lecture room takes place at the lecture room seating’s and desks, the projector screen and the door. There is also possible interaction between the users or between a user and the teacher. There could also be possible interaction between users and certain objects like books or pencils.

3.3.1.3 Why is the lecture room important?

The lecture room is the main reason of creating the digital university. With the digital university students will be able to follow lectures while being at home or anywhere else on a computer. The lecture room represents the real lecture room as on the university, and creates a feeling that users are still following a lecture on school.

3.3.1.4 Object screenshots and explanation

Appendix D1. Screenshot lecture room, page 51.

3.3.2 Corridor

The corridor separates the lecture room from the meeting place. Its function is the connection between the lecture room and the meeting place. Users can use the corridor in order to get from the one to the other room. Later on the corridor will be extended and will function as a connection between multiple rooms.

3.3.2.1 Objects in the corridor

- Bench
- “Poster”(Optional).

The only objects that the corridor contains are 2 benches. Users can use these benches to rest before college starts, just to relax on or have a chat before college starts.
An optional object for the corridor would be a “digital” poster. This poster could contain information about upcoming events or information regarding the university.

3.3.2.2 Where does the interactivity take place?

The interactivity in the corridor takes place in whole corridor. There can be interaction when a user wants to sit down on one of the benches. A number of agents continually walk certain paths in the corridor, a user can walk into one of these agents, on which the agent will respond to the user.

3.3.2.3 Object screenshots and explanation

Appendix D2. Screenshot corridor, page 52.

3.3.3 Meeting place

The meeting place is the place where people get together before/after/during college to meet each other, having a drink, relax, to study or to have lunch.

3.3.3.1 Objects in the meeting place

- Couch
- Bar & Bar Stool
- Sink
- Dining Table & Dining Table Chair
- Bin
- Side Table
- Coffee Table
- Wine Glass
- Knife
- Fork & Plate
- Book
- Clock
- Wall Decoration
- Plant.

The most important objects of the meeting place have to be the dining table & dining table chair and the bar & bar stool. These objects are both socializing areas within the room. Users enter this room and will mostly use it to enjoy a drink or have something to eat. The users use the dining table & dining table chair to enjoy their food/drink comfortable; thereby the users also have an opportunity to meet new users. The bar is also a social point, people are waiting for their drink or are enjoying their drink while being at the bar and meet new users. The meeting place also contains a corner with 2 couches, this is area where users can relax after college or have a chat with the other users.

3.3.3.2 Where does the interactivity take place?

The meeting place contains several interactivity points. All the interactivity takes place between:

- Interaction with an object: user picking up a glass
- Interaction with a bot/agent: having a dialog with and agent. An agent reacting on actions made by the user.
- Interaction with another user: having a real-time chat with another user.

There isn't a certain spot where the interactivity takes place, only the interaction with objects may have a prefixed spot. When the user interacts with an agent and/or user the place of interaction isn't defined. An agent could move (if the agent has AI) and so could a user.

3.3.3.3 Why is the meeting place important?

The meeting place is of great importance when it comes to socializing with other users.
The users will be able to communicate with the other users (students, teachers etc.) while enjoying their virtual cup of coffee. The meeting place represents all the social activities that would normally take place throughout the whole university. All these activities are now focused into one area, as where normally people would meet each other at different locations. After a digital lesson, students can gather in the meeting place and evaluate the lesson or discuss whatever they would like to. The meeting place would also be a good area for students to interact with the teachers after college.

3.3.3.4 Object screenshots and explanation

Appendix D3. Screenshot meeting place, page 53.

3.4 Building and animating the characters

In this chapter we will explain how we created the characters that we use in our 3D environment, how we modeled the characters and what we used to animate the characters. We will also explain how we created interactivity between objects and how we created paths.

3.4.1 3D Modeling

The first thing we had to take care of in order to create a character, was modeling the character. Our choice to model the character by our own, and not using an existing model, is because of the high polygons used in existing models. We created the body majorly by using the functions "Extrude"," Scale" and by editing the positions of the vertices.
The whole character actually exists out of one mesh, which is the torso. All other parts of the body were extruded out of the torso or out of the extruded parts. After the whole body was roughly created by extruding and scaling, all parts of the body were "smoothed".
All the parts were "smoothed" by hand (editing of vertices positions) and on the end smoothed by Blender's smooth function.
Appendix E. Figures E1 – E3, page 54.

3.4.2 Armatures

After creating the body of the character, we need to add a bone structure to be able to continue. An armature is another name for bone, or bone structure. By creating a

bone structure for the character, we will be able to animate it which leads to a more realistic view when the character is moving.
The creation of an armature is relatively easy. We simply add an armature which represents the spine or "the torso bone". From there we simply extrude other armatures for the other parts of the body. After creating an armature for every part of the body, we named all these parts. This naming of parts will be of great use later on.
Now there is a model of the character and armatures, and by connecting these armatures the character will be able to move the body parts realistic.
By "parenting" the body to the armature, and thereby creating name groups, we link the armatures to the body. Now that the armatures and the body are linked, we assign which bones to move what part of the body. After this all is done, moving an armature will move the corresponding body part.
Appendix E. Figure E4, page 54.

3.4.3 Animations

Animating a character is the next step to a more realistic appearance. By animating the character we made, we can create a walk animation that will be triggered after (for example) pressing a button.
Animating a body which contains armatures is relatively easy. All that needs to be done is placing the armatures in the required position and then "lock" them in place. The position of the armature can be changed in the "pose mode" of the armatures. The locking of the armatures can be done by using a timeline.
This timeline uses the number of frames as "time". We lock the position of all the armatures on the wanted frame position by creating a key frame in the timeline. By repeating this action we created a full animation of the character moving his arms and legs and thereby making a walk movement.
Appendix E. Figures E5 –E6, page 54 – 55.

3.4.4 Game logic

In order to implement the character we created into our "game" we used the game engine of blender to do so. Adding certain functions or options to objects in the game engine, is done by using Logic Bricks. These logic bricks are pre-programmed blocks, where the user can adjust certain variables or select options with the help of a drop-down-menu.
By using these logic bricks we created the possibility to control our character with the w,s,a,d keys and thereby make him walk forwards, backwards, left and right.
The logic bricks system exists out of 3 categories: sensors, controllers and actuators. These categories all have their own different logic bricks and by connecting these 3 categories of bricks to each other, we can "program" certain actions.
For example: Sensor (keyboard) AND Motion. By connecting these 3 bricks, we can bind a key to let the object move in a direction we want to.
Appendix B1 – B3 (sensors, controllers and actuators), page 45 – 49.

3.4.5 Mouselook

By using the game logic, the character can move forward, backward, left and right. But what if we want to turn the character around or look up and down. One way to have a character make these movements is by using the mouse for this action. With the help of the programming-language Python, we can make a script to be able to make these movements. By choosing the Python controller logic brick, a python script can be used to control the sensor and actuator logic bricks. By using a certain combination of actuators and sensors together with the script, we were able to control the character's body rotation and view.
Appendix F1. MouseLook.py, page 58 – 59.

3.4.6 Bots

In our environment we used computer controlled characters (Bots), in order to create a more realistic environment. These bots represent teachers or other students and respond on their surroundings. With the help of logic bricks we can make the bots respond on anything we want to. For example, they stop moving if the "player" crosses their path. This detection is done by adding sensors to the object that look out for objects with the property "player". They could also be able to detect each other, to make sure they won't collide or react on a collision.
The paths that these bots follow are predefined paths. More information of the paths that these bots follow will be explained in the Path Node section.
Appendix E. Figure E7, page 55.

3.4.7 Path Node

In order to move the bots in a certain pattern, we had to create a Path. A path is a predefined route of waypoints. In Blender we created cubes as waypoints and gave the cubes two properties: 'pathnode' and 'switch'. The 'pathnode' property is to make sure that the bot will only "see" waypoints with the property 'pathnode'. The property 'switch' was added to switch the waypoints on or off in the game.
The bot has also got two properties, but those are different then the waypoint's properties. These properties are: 'node' and 'speed'. The property 'speed' is used to define if the bot should move or not. When 'speed' is 0 the bot stands still, and when 'speed' is 1 the bot moves with a constant speed. The property 'node' makes sure the bot knows which waypoint is next.
The idea of the path is that the bot moves to the first waypoint and when the bot hits the waypoint the property 'node' will increase with +1. Each waypoint has a certain value of 'node' linked to it, so if 'node' reaches the corresponding value the bot will move towards that waypoint.
This method may sound complicated or like a lot of work, but this is by far the only way to create a path in blender, without using python or any plug-in.
Appendix E. Figures E8 – E10, page 55 – 57.

3.5 Lesson material

The lecture room is the most important room for creating a digital university. With the digital university students will be able to follow lectures while being at home or anywhere else on a computer. The lecture room has several objects, including a projector screen. The projector screen shows us an image with key bindings which represent a video, image or sound.

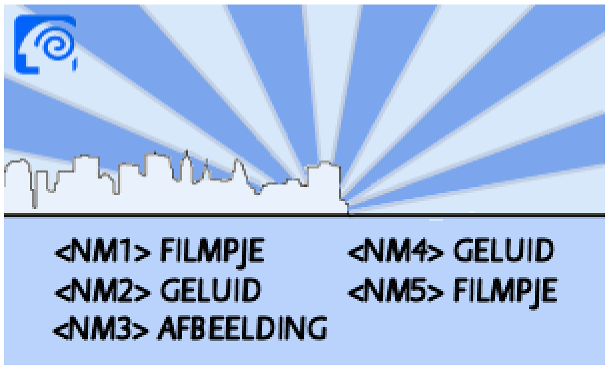
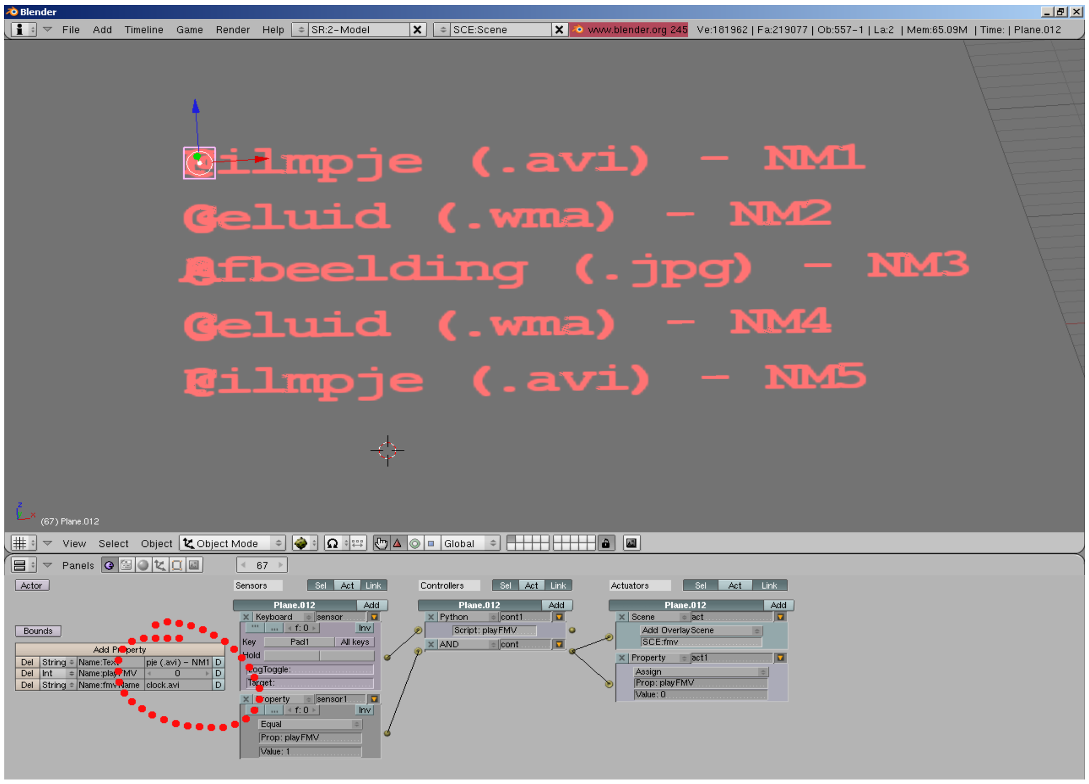


Figure 3.5.1 Image shown on the projector screen

Adding video cut scenes (movie clips that play in between segments of the game) in the Blender Game Engine (BGE) is easy by using the tool developed by Keith Gearty. Keith Gearty has developed a simple to use Python tool called "FMV-ed" to do just that. Keith pointed out that FMV-ed doesn't actually add the video to Blender, it merely calls the Windows Media ActiveX controller needed to display the cut scene. This means a pop-up opens on top of Blender, automatically plays the video, sound or show the image and than closes itself. By pressing the spacebar, the video pauses. By pressing the spacebar for the second time, the video continues. By pressing Escape, the player closes. The tool supports/plays .avi, .mpeg, .wma, .gif and .jpg files. The current license is closed-source freeware, provided as is, without warranty.

FMV-ed information summarized:
Author: Keith Gearty
Company: Gorgan Studios
Email: gorgan_almighty@yahoo.co.uk
Source: http://www.blendenzo.com/faqMovies.html.

By using this tool, the teacher is able to add his or her own lesson material to the digital university.


Blender
File Add Timeline Game Render Help SR:2-Model SCE:Scene www.blender.org 245 Ve:181962 | Fa:219077 | Ob:557-1 | La:2 | Mem:65.09M | Time: | Plane.012
View Select Object Object Mode Global
Panels 67
Actor
Bounds
Sensors Sel Act Link
Controllers Sel Act Link
Actuators Sel Act Link

Filmpje (.avi) - NM1
Geluid (.wma) - NM2
Afbeelding (.jpg) - NM3
Geluid (.wma) - NM4
Filmpje (.avi) - NM5

(67) Plane.012

Add Property				
Del	String	Name:Text	pje (.avi) - NM1	D
Del	Int	Name:playFMV	0	D
Del	String	Name:fmvName	clock.avi	D

Sensors – Plane.012: Keyboard sensor; f: 0; Inv; Key Pad1; All keys; Hold; LogToggle:; Target:
Property sensor1; f: 0; Inv; Equal; Prop: playFMV; Value: 1

Controllers – Plane.012: Python cont1; Script: playFMV
AND cont

Actuators – Plane.012: Scene act; Add OverlayScene; SCE:fmv
Property act1; Assign; Prop: playFMV; Value: 0

Figure 3.5.2 Objects “behind” the projector screen

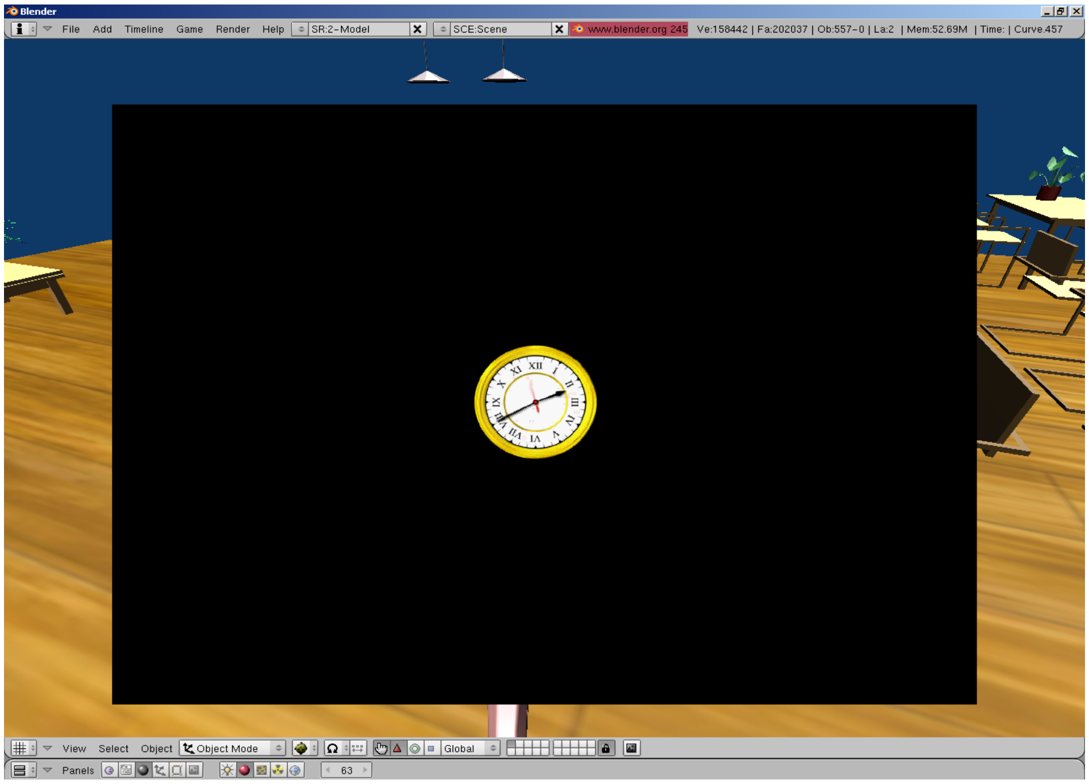
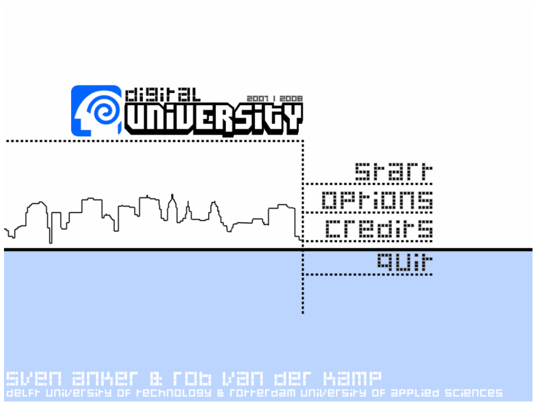

Blender
File Add Timeline Game Render Help SR:2-Model SCE:Scene www.blender.org 245 Ve:158442 | Fa:202037 | Ob:557-0 | La:2 | Mem:52.69M | Time: | Curve.457
View Select Object Object Mode Global
Panels 63

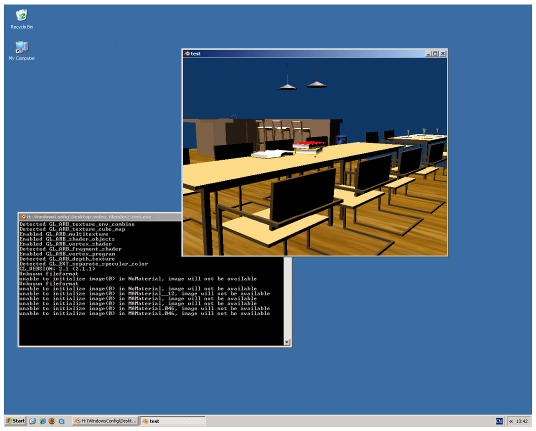
Figure 3.5.3 Pop-up; showing a video in Blender

The Python scripts can be found in *Appendix F1 – F8, page 58 – 75*. Comments included.

3.6 User Interface Digital University

After converting the Blender file to an executable file, the interface for the player gets something smaller, but it takes less time to render and start the game. For the most user friendly version of our game, the main menu doesn't load before starting the game. You can play the game directly after you started it.
The following page shows us two screenshots of the main menu and "game interface" for the player.


digital
UNIVERSITY
2007 | 2008
start
options
credits
quit
sven anker & rob van der kamp
delft university of technology & rotterdam university of applied sciences
Figure 3.6.1 Main menu


Recycle Bin
My Computer
test
H:\WindowsConfig\Desktop\video_Blender2\test.exe
Start
H:\WindowsConfig\Deskt...
test

```
Detected GL_ARB_texture_env_combine
Detected GL_ARB_texture_cube_map
Enabled GL_ARB_multitexture
Enabled GL_ARB_shader_objects
Enabled GL_ARB_vertex_shader
Detected GL_ARB_fragment_shader
Enabled GL_ARB_vertex_program
Detected GL_ARB_depth_texture
Detected GL_EXT_separate_specular_color
GL_VERSION: 2.1 (2.1.1)
Unknown fileformat
unable to initialize image(0) in NoMaterial, image will not be available
Unknown fileformat
unable to initialize image(0) in NoMaterial, image will not be available
unable to initialize image(0) in MAMaterial__12, image will not be available
unable to initialize image(0) in MAMaterial, image will not be available
unable to initialize image(0) in MAMaterial, image will not be available
unable to initialize image(0) in MAMaterial.046, image will not be available
unable to initialize image(0) in MAMaterial.046, image will not be available
```

Figure 3.6.2 User Interface player

Chapter 4
Conclusions and recommendations

4.1 Conclusions

After working with Blender for about 4 months you can say we're well known with most of the features and options offered by Blender. We discovered new options every day and I'm sure we still didn't use all the features Blender is offering.
We thought Blender had all the features as mentioned in A*ppendix A. Features Scheme, page 43 - 44*. But Blender did not turn out to be like we expected. We had the most problems with the interactivity part of Blender. The main goal of this project was to create an interactive Digital University, where students and teachers could interact with each other and any object in the university. Blender has a small amount of possibilities regarding to interactivity though. With the logic bricks we could give objects sensors and actuators so they could respond on certain objects or actions but these options were too limited. The only option to add some more interactivity is by using python scripts but then we would have to write scripts our own, which wasn't an option due to lack of time. We also reached a certain point in Blender where the contents of the environment got to big for the game, as the game took very long to load and the performance of the game was very low. So if we had managed to create an interactive environment, it still would have been a failure as the game wouldn't be to run on an average computer.
Blender does have a great usability. It is very easy to create 3D objects and to manipulate them. The game engine also isn't hard to use because of the logic brick system. Blender also has a great collision and physics engine which are easy to use in combination with the logic bricks.

Not all of our research goals are realized. Unfortunately "*making the virtual environment interactive*" did not worked out as planned, due to an unexpected time consuming problem.

4.2 Recommendations

So creating a simple game is relatively easy to do in Blender. But if you wish to create a complicated game with all kinds of features then we advise to use a commercial engine as they are far more extended.

References

http://www.blender.org	- Blender main page
http://www.gameblender.org	- Game Blender forum
http://wiki.blender.org	- Blender Wiki
http://blenderartists.org	- Blender Artist forum
http://www.blendenzo.com	- Blender tutorials
http://www.devmaster.net/engines	- Game engines
http://www.blendermasters.com	- Blender tutorials
http://www.secondlife.com	- Second Life
http://irrlicht.sourceforge.net	- Irrlicht engine
http://www.virtools.com	- Virtools
http://www.garagegames.com	- Torque game engine
http://developer.valvesoftware.com	- Valve Developer Community

List of abbreviations

2D	Two Dimensional
3D	Three Dimensional
AI	Artificial Intelligence
API	Application Programming Interface
BGE	Blender Game Engine
GNU GPL	GNU General Public License
GUI	Graphical User Interface
HRO	Rotterdam University of Applied Sciences
PC	Personal Computer
SDK	Software Development Kit
TU Delft	Delft University of Technology
UI	User Interface
URL	Uniform Resource Locator

Appendix

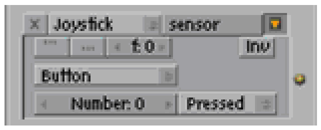
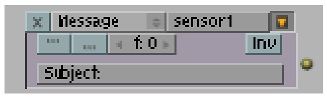
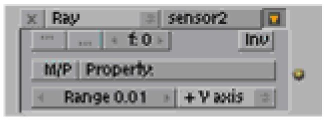
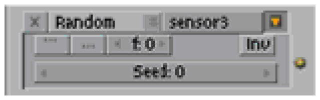
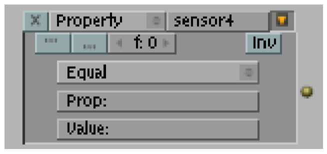
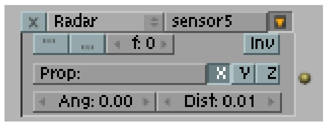
A. Features scheme

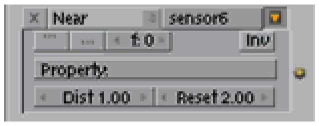
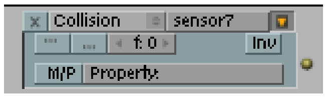
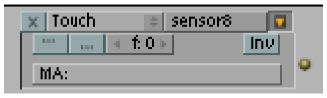
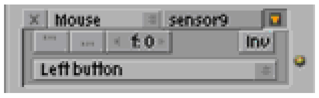
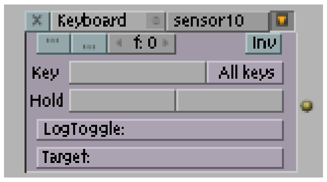
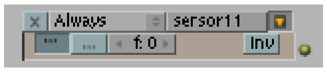
	Features	3D Blender	Half-Life	Second Life	Irrlicht	Virtools	Torque
	Modeling Requirements:						
A	Modeling Environment	x	/	x	/	x	/
B	Rendering system	x	x	/	x	x	x
	Engine Requirements:						
C	Animation	x	x	x	x	x	/
D	Collision	x	x	x	x	x	/
E	Physics (physics engine)	x	x	x	x	x	x
F	Body dynamic	x	x	x	x	x	x
G	Coding / scripting	C/C++, Python	C/C++	Linden Scripting Language (LSL)	C++, C#, VB.Net	C/C++, Virtools Scripting Language	C/C++, Torque Script
H	Multiplatform	x	x	x	x	x	x
I	Import 3D files	3D Studio, AC3D, COLLADA, DEC Object File Format, DirectX, Lightwave, MD2, Motion Capture, Nendo, OpenFlight, PLY, Pro Engineer, Radiosity, Raw Triangle, Softimage, STL, TrueSpace, VideoScape, VRML, VRML97, Wavefront, X3D Extensible 3D, xfig export	XSI, Max and Maya .smd	3D Studio Max, Maya, Autocad and Blender	3DS, Milkshape, COLLADA, Maya, DeleD, DirectX .X, FSRad .oct, Cartography shop 4 .csm, Pulsar LMTools .lmts, My3DTools 3 .my3D, Quake 2 models, 3DS Max, Gile[s], Blender	3ds Max, Maya, XSI, Lightwave, Collada	Milkshape, 3DStudio Max, and Blender
J	Costs	Free	On request	Free	Free	$9000-$10000	$150/ $290
	Game play requirements:						
K	Interactivity	x	x	x	x	x	/
L	Movement	x	x	x	x	x	x
M	View(Camera)	x	x	x	x	x	x
N	AI	x	x	x	/	x	x
O	Audio & video support	x	x	x	x	x	x

P	Menu building	**x**	/	/	**x**	/	**x**
Q	Multiplayer	**x**	**x**	**x**	/	**x**	**x**
R	Text display	**x**	**x**	**x**	**x**	**x**	**x**
	Extra:						
S	Documentation	**x**	**x**	**x**	**x**	/	**x**
T	Tutorials	**x**	**x**	**x**	**x**	/	**x**
U	Community / forum	**x**	**x**	**x**	**x**	/	**x**
V	Online support	**x**	**x**	**x**	**x**	**x**	**x**

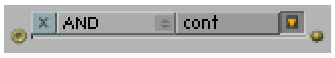
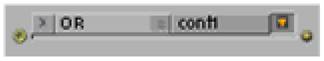
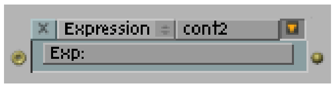
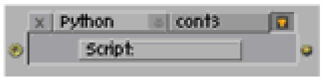
x = this option is supported by the game engine.
/ = this option is not supported by the game engine.
A-V = each feature has his own character. The character corresponds with the character in the “Features list” (chapter 2.2).

B1. Sensors

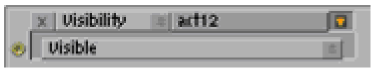
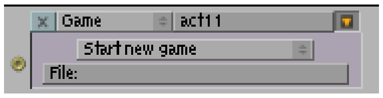
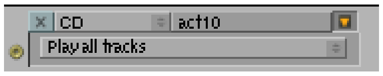
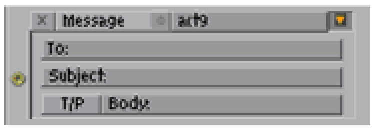
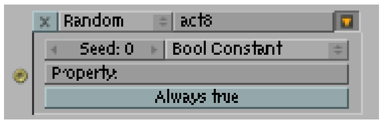
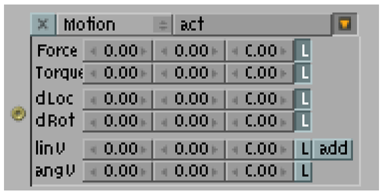
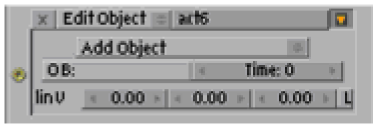
Name	Figure	Description
Joystick		Triggers when either a joystick button is pressed, or when joystick is moved along a certain direction (left/right, up/down).
Message		Triggers when a message is received.
Ray		This will trigger when an object is detected along a certain axis. You can additionally check for the detected object having a certain material or property value.
Random		Triggers randomly, change seed for different sequences numbers.
Property		Triggers when a property changes, is between certain min and max values, or is equal or not equal to a certain value.
Radar		Triggers when an object is detected within a certain range (distance and angle). You can specify a property that the object must have.

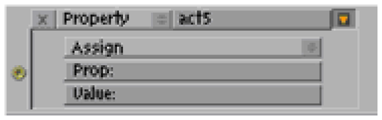
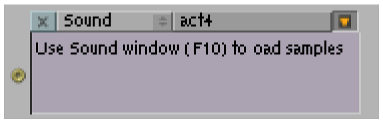
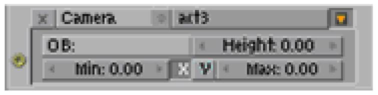
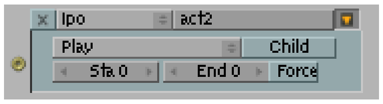
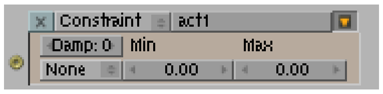
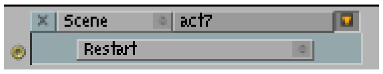
Near	 Near sensor6 f: 0 Inv Property: Dist 1.00 Reset 2.00	Triggers when a object is detected within a certain distance. You can specify a property that the detected object must have.
Collision	 Collision sensor7 f: 0 Inv M/P Property:	Triggers when the object is in collision with another object. You can specify a material or a property that the collided object must have.
Touch	 Touch sensor8 f: 0 Inv MA:	Triggers when an object is touching another object. You can specify a property that the touched object must have.
Mouse	 Mouse sensor9 f: 0 Inv Left button	Triggers when certain mouse event occur, such as mouse button clicks, mouse movement etc.
Keyboard	 Keyboard sensor10 f: 0 Inv Key All keys Hold LogToggle: Target:	Triggers when a certain key is pressed.
Always	 Always sensor11 f: 0 Inv	Triggers every single frame.

B2. Controllers

Name	Figure	Description
AND		Runs the connected actuator if all of the connecting sensors are triggered.
OR		Runs the connected actuator if any of the connecting sensors is triggered.
Expression		Evaluates an expression.
Python		Runs a python script.

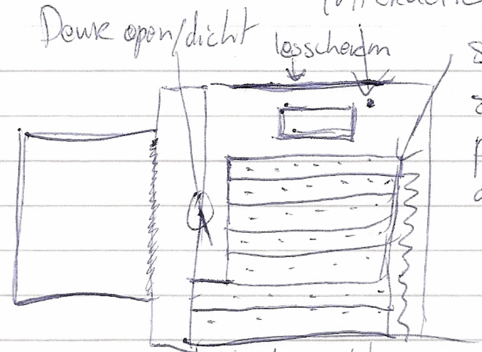
B3. Actuators

Name	Figure	Description
Visibility		Show and hide the current object.
Game		Restart and Quit the current level. Can also load a new scene.
CD		Allows for control over CD music tracks.
Message		Send a message to all objects, or to a certain object. This message will trigger the Message sensor.
Random		Sets a random value into a property of the object.
Motion		Allows control over the motion of the objects. This includes direct positioning and rotating of the object (dLoc and dRot), as well as applying forces to a physical object to move it (Force and Torque).
Edit Object		Allows for control over adding, editing and deleting objects within the scene at run-time.

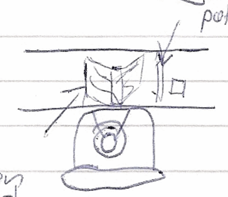
Property	 Property act5 Assign Prop: Value:	Sets the property value of the object.
Sound	 Sound act4 Use Sound window (F10) to oad samples	Allows you to control sounds from within Blender (Only sounds into Blender will be accessible).
Camera	 Camera act3 OB: Height: 0.00 Min: 0.00 X Y Max: 0.00	Allows the camera to track an object. The camera can be placed behind the object within a certain distance (min and max) and height.
IPO	 Ipo act2 Play Child Sta 0 End 0 Force	Allows control over playing object animations.
Constraint	 Constraint act1 Damp: 0 Min Max None 0.00 0.00	Constrains the objects position.
Scene	 Scene act7 Restart	Allows for control over scenes (loading, playing, suspending etc.).

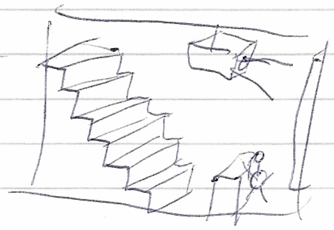
C. Sketches

interactief met leraar.



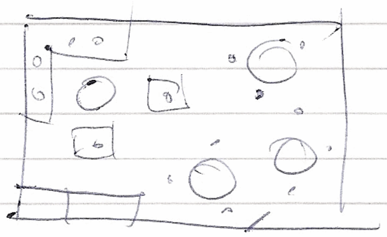
Stoelen moeten ook bezet zijn door agents. Als je probeert zitten bij agent, agen reageert.



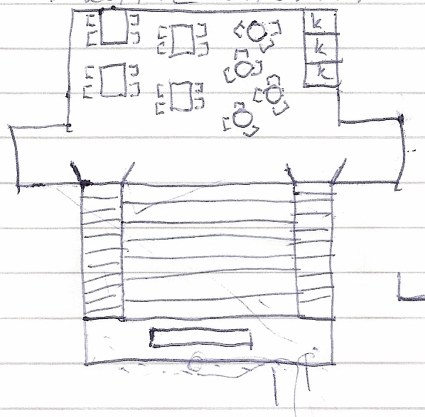


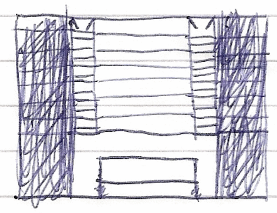
Bladeren in schrift (1 ~ 2 blz)

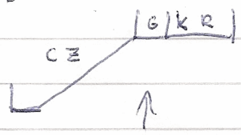
lesscherm = video draaiend en/of powerpoint



koffieruimte
- interactief met elkaar
- ? koffieautomaat.







niveau Hoogte.

D1. Screenshot lecture room

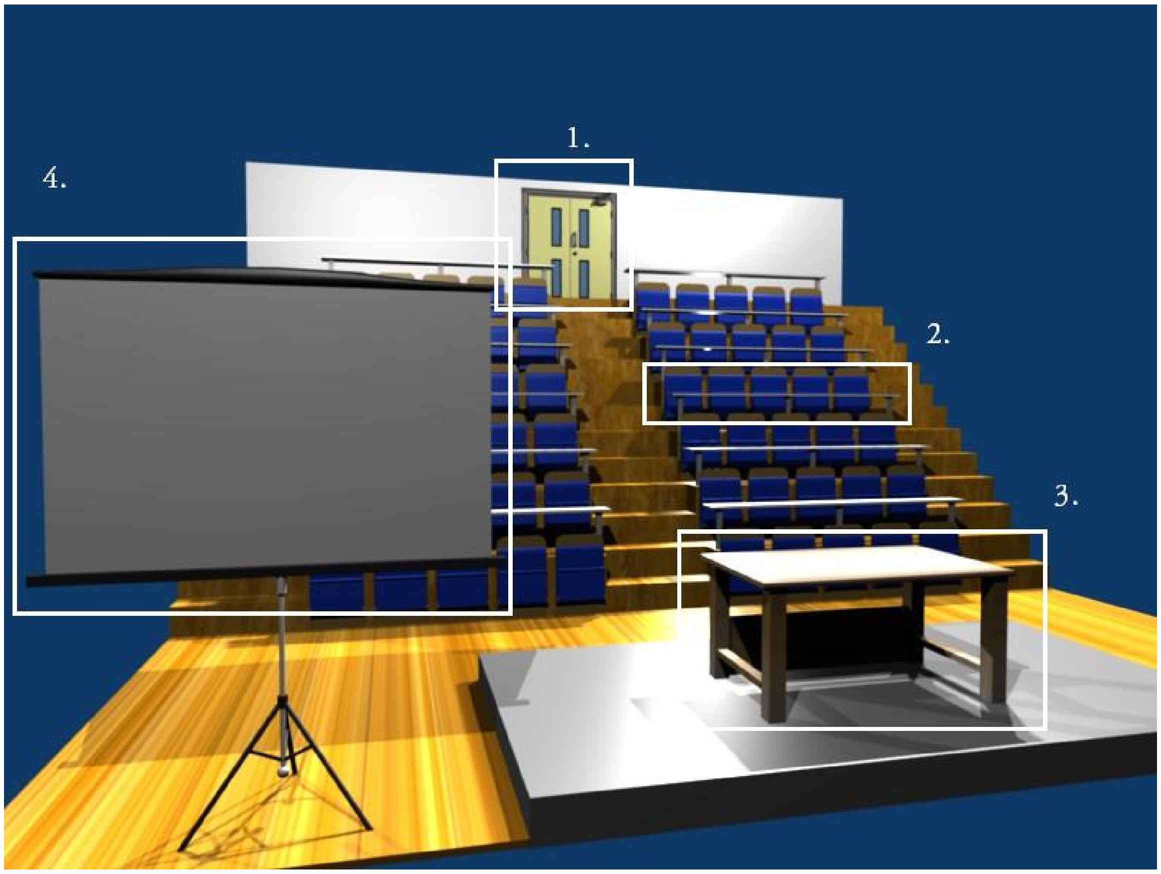


Figure D1.1 Lecture room

1. Door to the corridor
2. Lecture room seating's and desk
3. Teacher's desk
4. Projector screen

D2. Screenshot corridor

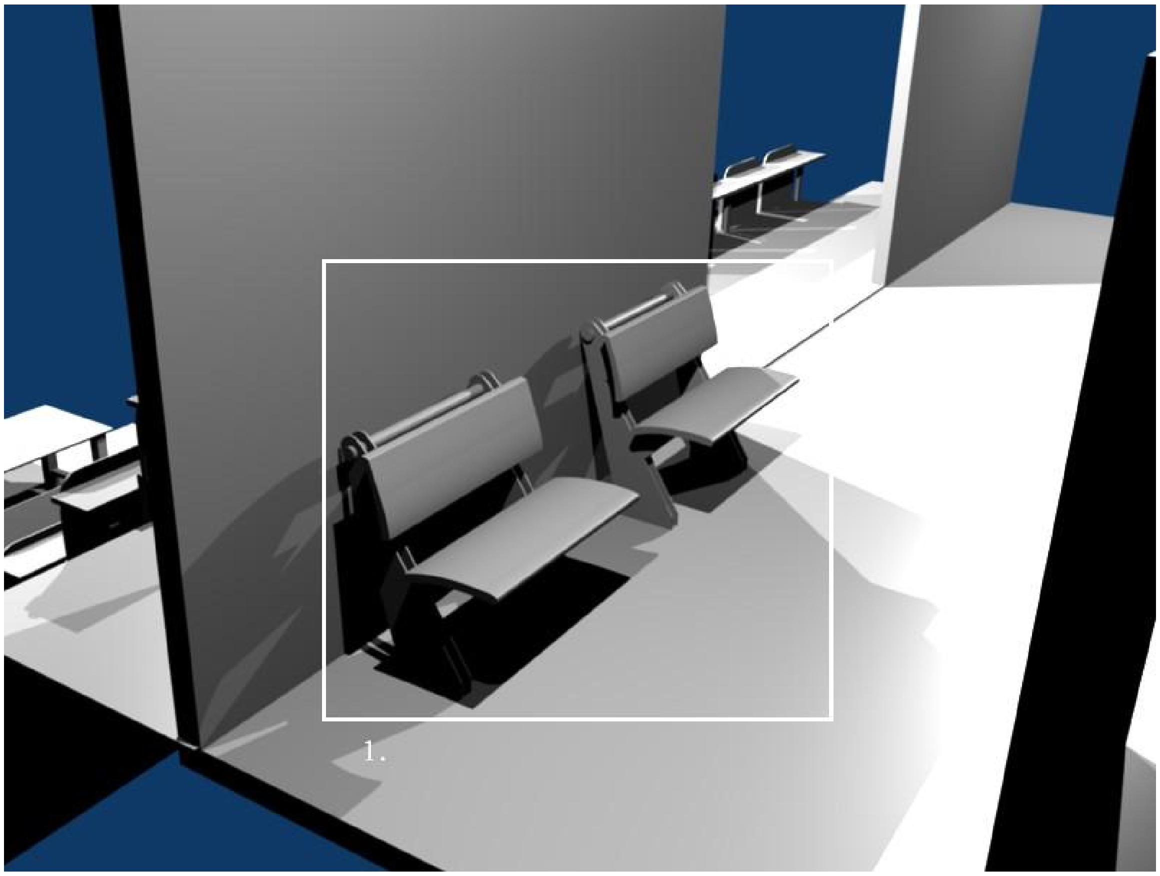


Figure D2.1 Corridor

1. The two benches in the corridor, where users can relax, rest or have a chat.

D3. Screenshot meeting place

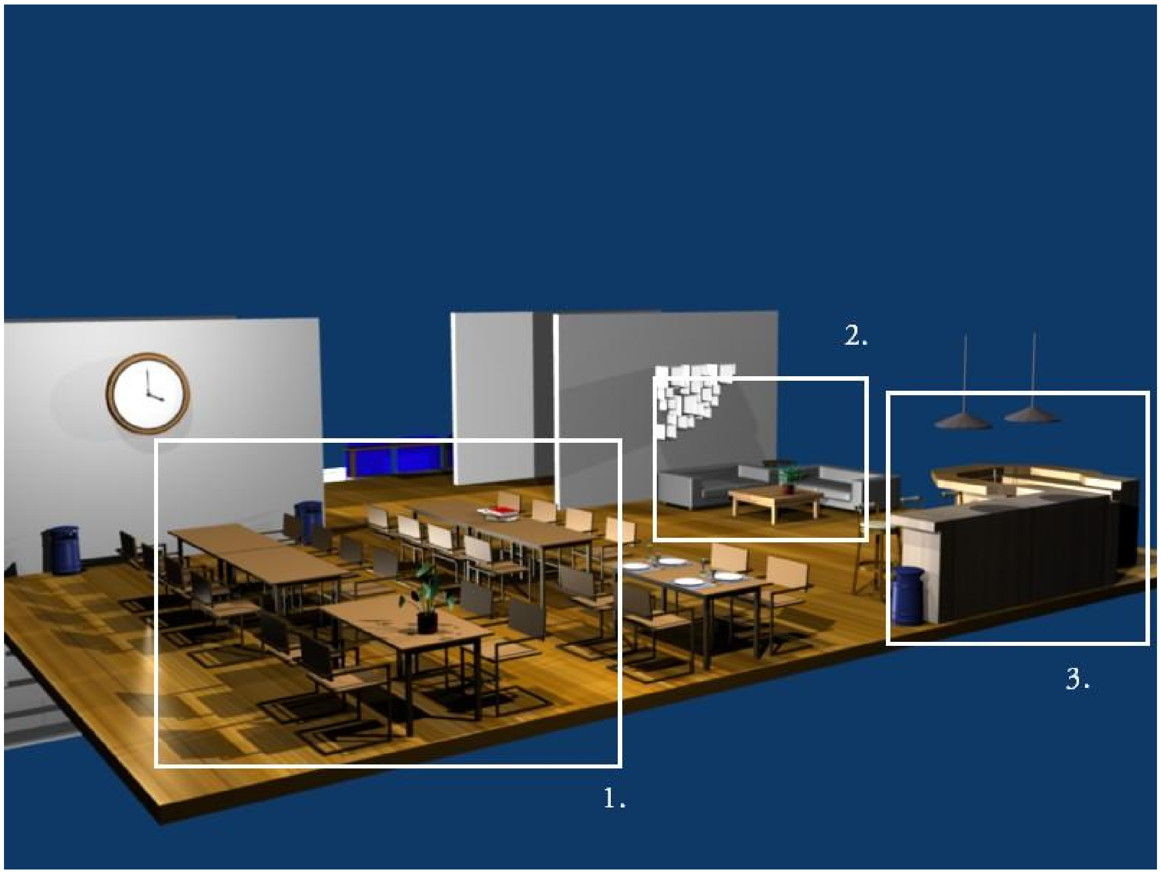


Figure D3.1 Meeting place

1. Dining/studying area
   Area of the room filled with tables and comfortable chairs where the users can study, talk or lunch. Users can also meet new people while studying or lunching.

2. Couch Corner
   A little corner with 2 couches a coffee table and a side table. The users can relax or study in this corner of the room. The comfortable couches together with the wall decoration and the coffee table make this corner a cozy area of the room.

3. Bar
   The bar makes sure everyone can buy something to drink. While having a lunch or while studying people can buy a drink at the bar or have a chat with the people hanging around.
   A couple of bar stools is placed in front of the bar.

E. Building and animating the characters

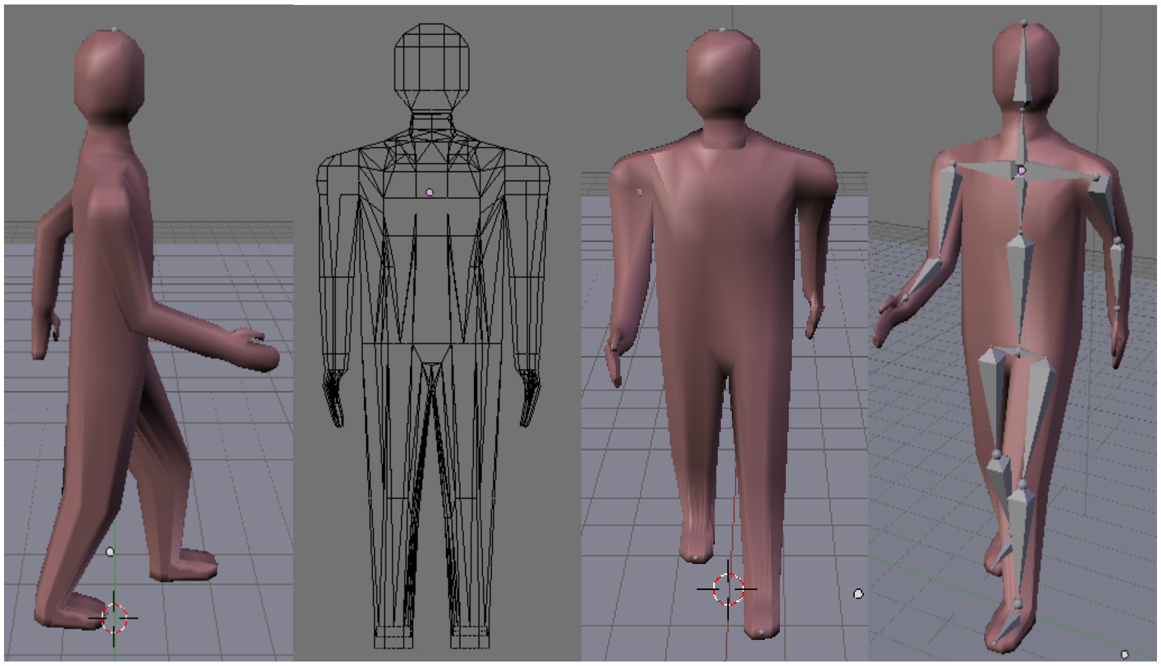


Figure E1. 3D Character side view
Figure E2. 3D Character wireframe
Figure E3. 3D Character front view
Figure E4. 3D Character with armatures

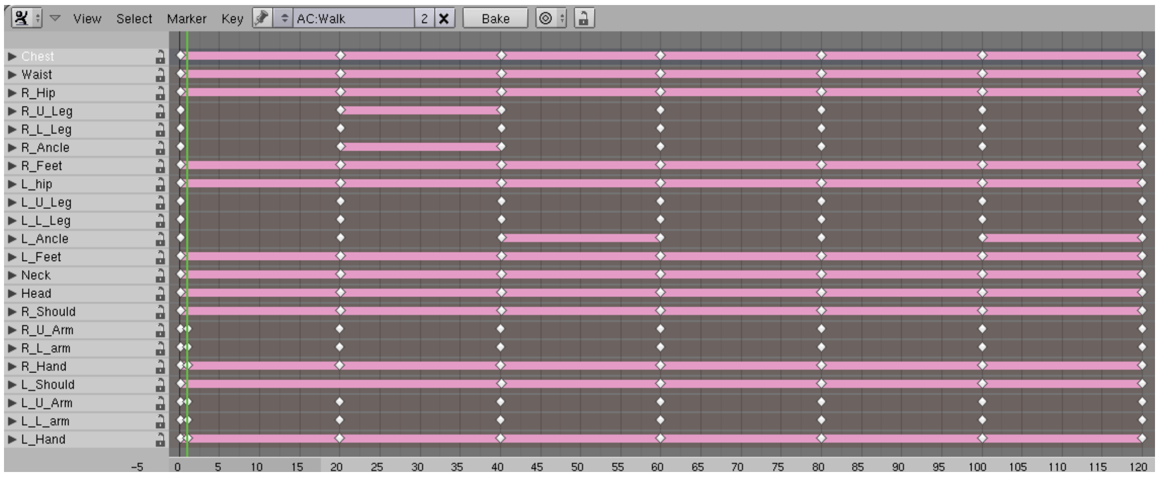
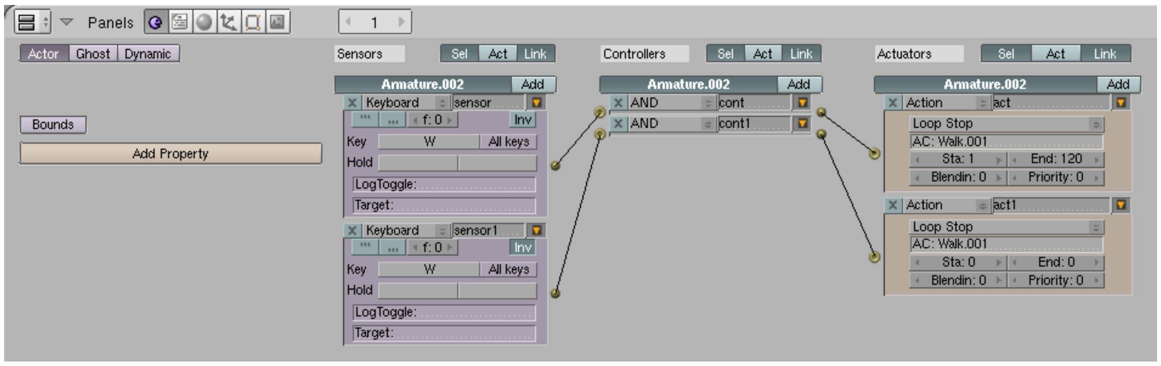


Figure E5. Animation window (with key frames and "timeline")

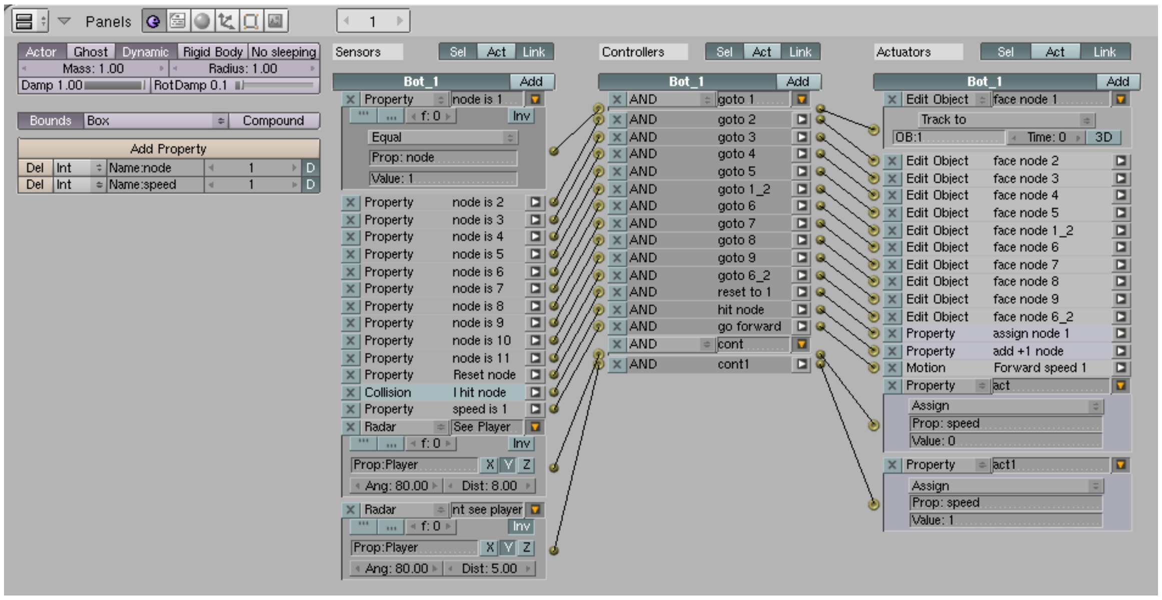

Panels 1
Actor Ghost Dynamic
Bounds
Add Property

Sensors Sel Act Link
Armature.002 Add
Keyboard sensor
f: 0 Inv
Key W All keys
Hold
LogToggle:
Target:
Keyboard sensor1
f: 0 Inv
Key W All keys
Hold
LogToggle:
Target:

Controllers Sel Act Link
Armature.002 Add
AND cont
AND cont1

Actuators Sel Act Link
Armature.002 Add
Action act
Loop Stop
AC: Walk.001
Sta: 1 End: 120
Blendin: 0 Priority: 0
Action act1
Loop Stop
AC: Walk.001
Sta: 0 End: 0
Blendin: 0 Priority: 0

Figure E6. Armature's logic bricks for triggering the animation

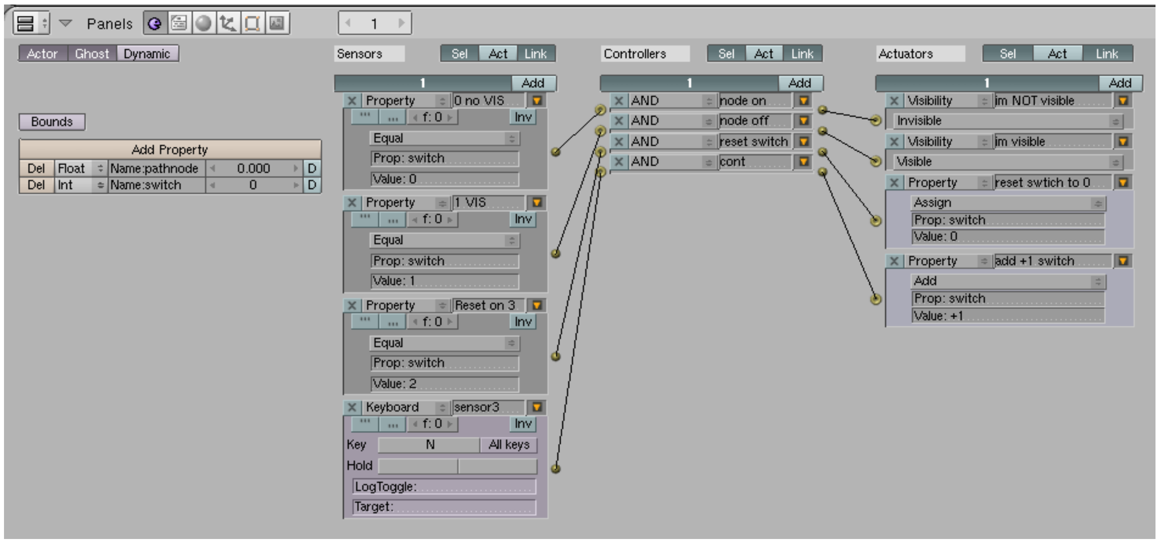

Panels 1
Actor Ghost Dynamic Rigid Body No sleeping
Mass: 1.00 Radius: 1.00
Damp 1.00 RotDamp 0.1
Bounds Box Compound
Add Property
Del Int Name:node 1 D
Del Int Name:speed 1 D

Sensors Sel Act Link
Bot_1 Add
Property node is 1
f: 0 Inv
Equal
Prop: node
Value: 1
Property node is 2
Property node is 3
Property node is 4
Property node is 5
Property node is 6
Property node is 7
Property node is 8
Property node is 9
Property node is 10
Property node is 11
Property Reset node
Collision I hit node
Property speed is 1
Radar See Player
f: 0 Inv
Prop:Player X Y Z
Ang: 80.00 Dist: 8.00
Radar nt see player
f: 0 Inv
Prop:Player X Y Z
Ang: 80.00 Dist: 5.00

Controllers Sel Act Link
Bot_1 Add
AND goto 1
AND goto 2
AND goto 3
AND goto 4
AND goto 5
AND goto 1_2
AND goto 6
AND goto 7
AND goto 8
AND goto 9
AND goto 6_2
AND reset to 1
AND hit node
AND go forward
AND cont
AND cont1

Actuators Sel Act Link
Bot_1 Add
Edit Object face node 1
Track to
OB:1 Time: 0 3D
Edit Object face node 2
Edit Object face node 3
Edit Object face node 4
Edit Object face node 5
Edit Object face node 1_2
Edit Object face node 6
Edit Object face node 7
Edit Object face node 8
Edit Object face node 9
Edit Object face node 6_2
Property assign node 1
Property add +1 node
Motion Forward speed 1
Property act
Assign
Prop: speed
Value: 0
Property act1
Assign
Prop: speed
Value: 1

Figure E7. Bot's logic bricks

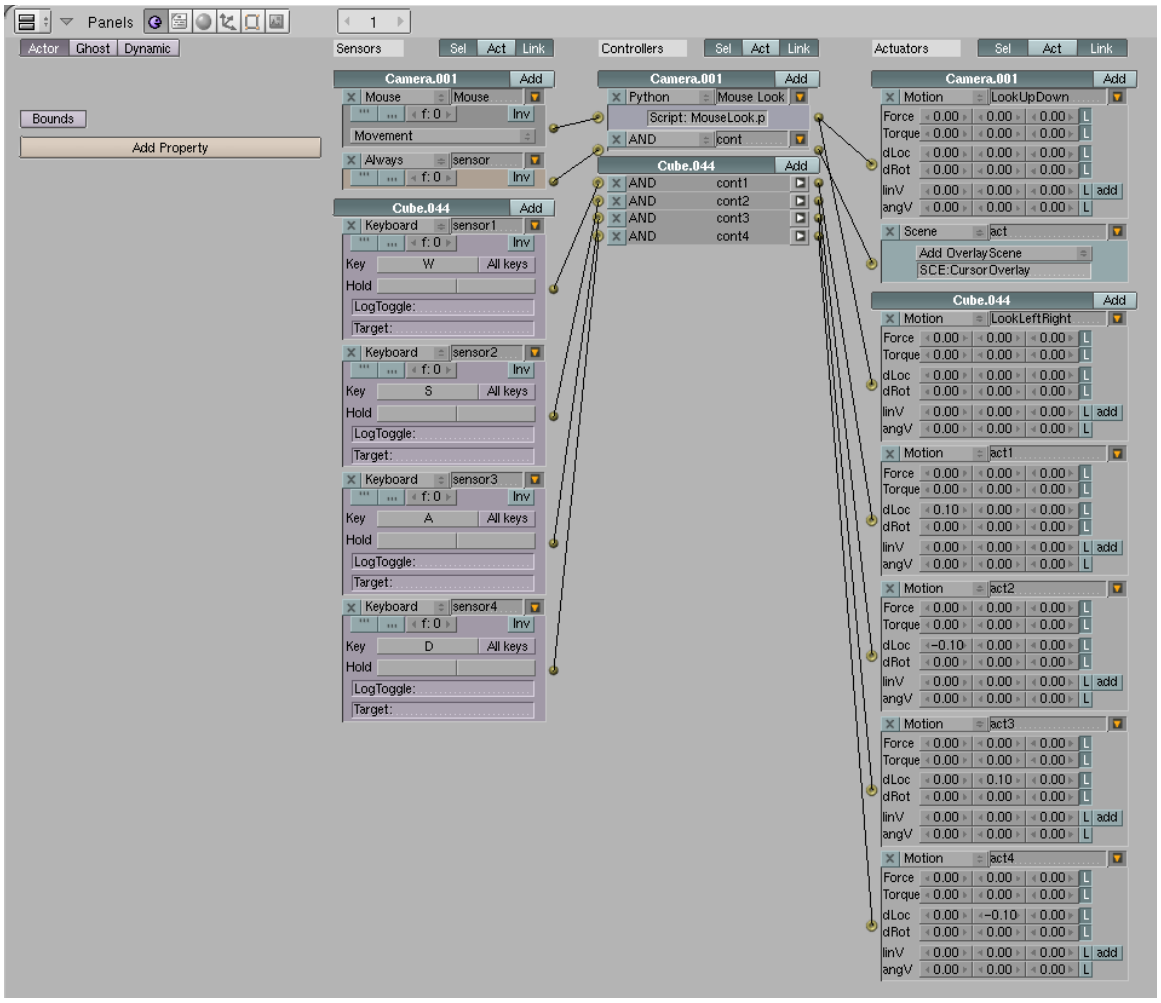

Panels 1
Actor Ghost Dynamic
Bounds
Add Property
Del Float Name:pathnode 0.000 D
Del Int Name:switch 0 D

Sensors Sel Act Link
1 Add
Property 0 no VIS
f: 0 Inv
Equal
Prop: switch
Value: 0
Property 1 VIS
f: 0 Inv
Equal
Prop: switch
Value: 1
Property Reset on 3
f: 0 Inv
Equal
Prop: switch
Value: 2
Keyboard sensor3
f: 0 Inv
Key N All keys
Hold
LogToggle:
Target:

Controllers Sel Act Link
1 Add
AND node on
AND node off
AND reset switch
AND cont

Actuators Sel Act Link
1 Add
Visibility im NOT visible
Invisible
Visibility im visible
Visible
Property reset swtich to 0
Assign
Prop: switch
Value: 0
Property add +1 switch
Add
Prop: switch
Value: +1

Figure E8. Waypoints logic bricks



Sensors

Camera.001
Mouse – Mouse – f: 0 – Inv – Movement
Always – sensor – f: 0 – Inv

Cube.044
Keyboard – sensor1 – f: 0 – Inv – Key: W – All keys – Hold – LogToggle: – Target:
Keyboard – sensor2 – f: 0 – Inv – Key: S – All keys – Hold – LogToggle: – Target:
Keyboard – sensor3 – f: 0 – Inv – Key: A – All keys – Hold – LogToggle: – Target:
Keyboard – sensor4 – f: 0 – Inv – Key: D – All keys – Hold – LogToggle: – Target:

Controllers

Camera.001
Python – Mouse Look – Script: MouseLook.p
AND – cont

Cube.044
AND – cont1
AND – cont2
AND – cont3
AND – cont4

Actuators

Camera.001
Motion – LookUpDown
Force 0.00 0.00 0.00 L
Torque 0.00 0.00 0.00 L
dLoc 0.00 0.00 0.00 L
dRot 0.00 0.00 0.00 L
linV 0.00 0.00 0.00 L add
angV 0.00 0.00 0.00 L
Scene – act – Add Overlay Scene – SCE:CursorOverlay

Cube.044
Motion – LookLeftRight
Force 0.00 0.00 0.00 L
Torque 0.00 0.00 0.00 L
dLoc 0.00 0.00 0.00 L
dRot 0.00 0.00 0.00 L
linV 0.00 0.00 0.00 L add
angV 0.00 0.00 0.00 L

Motion – act1
Force 0.00 0.00 0.00 L
Torque 0.00 0.00 0.00 L
dLoc 0.10 0.00 0.00 L
dRot 0.00 0.00 0.00 L
linV 0.00 0.00 0.00 L add
angV 0.00 0.00 0.00 L

Motion – act2
Force 0.00 0.00 0.00 L
Torque 0.00 0.00 0.00 L
dLoc –0.10 0.00 0.00 L
dRot 0.00 0.00 0.00 L
linV 0.00 0.00 0.00 L add
angV 0.00 0.00 0.00 L

Motion – act3
Force 0.00 0.00 0.00 L
Torque 0.00 0.00 0.00 L
dLoc 0.00 0.10 0.00 L
dRot 0.00 0.00 0.00 L
linV 0.00 0.00 0.00 L add
angV 0.00 0.00 0.00 L

Motion – act4
Force 0.00 0.00 0.00 L
Torque 0.00 0.00 0.00 L
dLoc 0.00 –0.10 0.00 L
dRot 0.00 0.00 0.00 L
linV 0.00 0.00 0.00 L add
angV 0.00 0.00 0.00 L

Figure E9. Player’s logic bricks

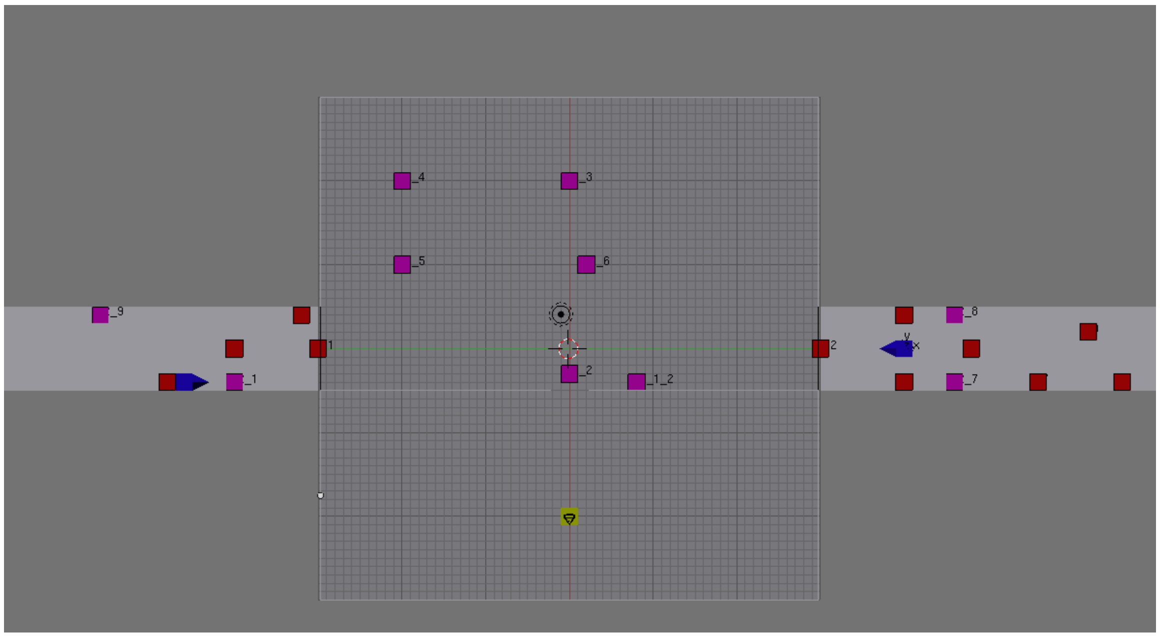


Figure E10. Path Node (All waypoints + bots and player – Old version)

F1. MouseLook.py

```
#####################################################
#
#    MouseLook.py        Blender 2.45
#
#    Clark R Thames
#    Released under Creative Commons Attribution License
#
#    Tutorial for using MouseLook.py can be found at
#
#    www.tutorialsforblender3D.com
#

#####################################################


###########################  Logic Bricks

# Get controller
controller = GameLogic.getCurrentController()

# Get sensor named Mouse
mouse = controller.getSensor("Mouse")

# Get the actuators
rotLeftRight = controller.getActuator("LookLeftRight")
rotUpDown = controller.getActuator("LookUpDown")


############################## Need the size of the game window

import Rasterizer

width = Rasterizer.getWindowWidth()
height = Rasterizer.getWindowHeight()


############################### Get the mouse movement

def mouseMove():

    # distance moved from screen center
    x = width/2 - mouse.getXPosition()
    y = height/2 - mouse.getYPosition()

    # intialize mouse so it doesn't jerk first time
    if hasattr(GameLogic, 'init') == False:
        x = 0
        y = 0
        GameLogic.init = True

    return (x, y)

pos = mouseMove()


######## Figure out how much to rotate camera and player ########

# Mouse sensitivity
sensitivity = 0.0015

# Amount, direction and sensitivity
leftRight = pos[0] * sensitivity
```

```
upDown = pos[1] * sensitivity

# invert upDown
upDown = -upDown

######### Use actuators to rotate camera and player #############

# Set the rotation values
rotLeftRight.setDRot( 0.0, 0.0, leftRight, False)
rotUpDown.setDRot( upDown, 0.0, 0.0, True)

# Use them
GameLogic.addActiveActuator(rotLeftRight, True)
GameLogic.addActiveActuator(rotUpDown, True)


############# Center mouse pointer in game window ###############

# Center mouse in game window
Rasterizer.setMousePosition(width/2, height/2)
```

F2. os.py

```
# os.py -- either mac, dos or posix depending on what system we're on.

# This exports:
# - all functions from either posix or mac, e.g., os.unlink, os.stat, etc.
# - os.path is either module posixpath or macpath
# - os.name is either 'posix' or 'mac'
# - os.curdir is a string representing the current directory ('.' or ':')
# - os.pardir is a string representing the parent directory ('..' or '::')
# - os.sep is the (or a most common) pathname separator ('/' or ':' or '\\')
# - os.altsep is the alternatte pathname separator (None or '/')
# - os.pathsep is the component separator used in $PATH etc
# - os.defpath is the default search path for executables

# Programs that import and use 'os' stand a better chance of being
# portable between different platforms.  Of course, they must then
# only use functions that are defined by all platforms (e.g., unlink
# and opendir), and leave all pathname manipulation to os.path
# (e.g., split and join).

import sys

_names = sys.builtin_module_names

altsep = None

if 'posix' in _names:
    name = 'posix'
    linesep = '\n'
    curdir = '.'; pardir = '..'; sep = '/'; pathsep = ':'
    defpath = ':/bin:/usr/bin'
    from posix import *
    try:
        from posix import _exit
    except ImportError:
        pass
    import posixpath
    path = posixpath
    del posixpath
elif 'nt' in _names:
    name = 'nt'
    linesep = '\r\n'
    curdir = '.'; pardir = '..'; sep = '\\'; pathsep = ';'
    defpath = '.;C:\\bin'
    from nt import *
    for i in ['_exit']:
        try:
            exec "from nt import " + i
        except ImportError:
            pass
    import ntpath
    path = ntpath
    del ntpath
elif 'dos' in _names:
    name = 'dos'
    linesep = '\r\n'
    curdir = '.'; pardir = '..'; sep = '\\'; pathsep = ';'
    defpath = '.;C:\\bin'
    from dos import *
    try:
        from dos import _exit
    except ImportError:
        pass
    import dospath
    path = dospath
    del dospath
elif 'os2' in _names:
    name = 'os2'
    linesep = '\r\n'
    curdir = '.'; pardir = '..'; sep = '\\'; pathsep = ';'
    defpath = '.;C:\\bin'
    from os2 import *
    try:
        from os2 import _exit
    except ImportError:
        pass
    import ntpath
    path = ntpath
    del ntpath
elif 'mac' in _names:
    name = 'mac'
    linesep = '\r'
    curdir = ':'; pardir = '::'; sep = ':'; pathsep = '\n'
    defpath = ':'
    from mac import *
    try:
        from mac import _exit
    except ImportError:
        pass
    import macpath
    path = macpath
    del macpath
else:
    raise ImportError, 'no os specific module found'

del _names

sys.modules['os.path'] = path

# Super directory utilities.
# (Inspired by Eric Raymond; the doc strings are mostly his)

def makedirs(name, mode=0777):
    """makedirs(path [, mode=0777]) -> None

    Super-mkdir; create a leaf directory and all intermediate ones.
    Works like mkdir, except that any intermediate path segment (not
    just the rightmost) will be created if it does not exist.  This is
    recursive.

    """
    head, tail = path.split(name)
    if head and tail and not path.exists(head):
        makedirs(head, mode)
    mkdir(name, mode)

def removedirs(name):
    """removedirs(path) -> None

    Super-rmdir; remove a leaf directory and empty all intermediate
    ones.  Works like rmdir except that, if the leaf directory is
    successfully removed, directories corresponding to rightmost path
    segments will be pruned way until either the whole path is
    consumed or an error occurs.  Errors during this latter phase are
    ignored -- they generally mean that a directory was not empty.
    """
    rmdir(name)
    head, tail = path.split(name)
    while head and tail:
        try:
            rmdir(head)
        except error:
            break
        head, tail = path.split(head)

def renames(old, new):
    """renames(old, new) -> None

    Super-rename; create directories as necessary and delete any left
    empty.  Works like rename, except creation of any intermediate
    directories needed to make the new pathname good is attempted
    first.  After the rename, directories corresponding to rightmost
    path segments of the old name will be pruned way until either the
    whole path is consumed or a nonempty directory is found.

    Note: this function can fail with the new directory structure made
    if you lack permissions needed to unlink the leaf directory or
    file.

    """
    head, tail = path.split(new)
    if head and tail and not path.exists(head):
        makedirs(head)
    rename(old, new)
    head, tail = path.split(old)
    if head and tail:
        try:
            removedirs(head)
        except error:
            pass

# Make sure os.environ exists, at least
try:
    environ
except NameError:
    environ = {}

def execl(file, *args):
    execv(file, args)

def execle(file, *args):
    env = args[-1]
    execve(file, args[:-1], env)

def execlp(file, *args):
    execvp(file, args)

def execlpe(file, *args):
    env = args[-1]
    execvpe(file, args[:-1], env)

def execvp(file, args):
    _execvpe(file, args)

def execvpe(file, args, env):
    _execvpe(file, args, env)
```

```
_notfound = None
def _execvpe(file, args, env = None):
    if env:
        func = execve
        argrest = (args, env)
    else:
        func = execv
        argrest = (args,)
        env = environ
    global _notfound
    head, tail = path.split(file)
    if head:
        apply(func, (file,) + argrest)
        return
    if env.has_key('PATH'):
        envpath = env['PATH']
    else:
        envpath = defpath
    import string
    PATH = string.splitfields(envpath, pathsep)
    if not _notfound:
        import tempfile
        # Exec a file that is guaranteed not to exist
        try: execv(tempfile.mktemp(), ())
        except error, _notfound: pass
    exc, arg = error, _notfound
    for dir in PATH:
        fullname = path.join(dir, file)
        try:
            apply(func, (fullname,) + argrest)
        except error, (errno, msg):
            if errno != arg[0]:
                exc, arg = error, (errno, msg)
    raise exc, arg

# Change environ to automatically call putenv() if it exists
try:
    # This will fail if there's no putenv
    putenv
except NameError:
    pass
else:
    import UserDict

    if name in ('os2', 'nt', 'dos'):  # Where Env Var Names Must Be UPPERCASE
        # But we store them as upper case
        import string
        class _Environ(UserDict.UserDict):
            def __init__(self, environ):
                UserDict.UserDict.__init__(self)
                data = self.data
                upper = string.upper
                for k, v in environ.items():
                    data[upper(k)] = v
            def __setitem__(self, key, item):
                putenv(key, item)
                key = string.upper(key)
                self.data[key] = item
            def __getitem__(self, key):
                return self.data[string.upper(key)]

    else:  # Where Env Var Names Can Be Mixed Case
```

```
        class _Environ(UserDict.UserDict):
            def __init__(self, environ):
                UserDict.UserDict.__init__(self)
                self.data = environ
            def __setitem__(self, key, item):
                putenv(key, item)
                self.data[key] = item

    environ = _Environ(environ)
```

F3. ntpath.py

```
# Module 'ntpath' -- common operations on WinNT/Win95 pathnames
"""Common pathname manipulations, WindowsNT/95 version.

Instead of importing this module directly, import os and refer to this
module as os.path.
"""

import os
import stat
import string


# Normalize the case of a pathname and map slashes to backslashes.
# Other normalizations (such as optimizing '../' away) are not done
# (this is done by normpath).

def normcase(s):
    """Normalize case of pathname.

    Makes all characters lowercase and all slashes into backslashes."""
    return string.lower(string.replace(s, "/", "\\"))


# Return wheter a path is absolute.
# Trivial in Posix, harder on the Mac or MS-DOS.
# For DOS it is absolute if it starts with a slash or backslash (current
# volume), or if a pathname after the volume letter and colon / UNC resource
# starts with a slash or backslash.

def isabs(s):
    """Test whether a path is absolute"""
    s = splitdrive(s)[1]
    return s != '' and s[:1] in '/\\'


# Join two (or more) paths.

def join(a, *p):
    """Join two or more pathname components, inserting "\\" as needed"""
    path = a
    for b in p:
        if isabs(b):
            path = b
        elif path == '' or path[-1:] in '/\\':
            path = path + b
        else:
            path = path + os.sep + b
    return path


# Split a path in a drive specification (a drive letter followed by a
# colon) and the path specification.
# It is always true that drivespec + pathspec == p
def splitdrive(p):
    """Split a pathname into drive and path specifiers. Returns a 2-tuple
"(drive,path)";  either part may be empty"""
    if p[1:2] == ':':
        return p[0:2], p[2:]
    return '', p


# Parse UNC paths
```

```
def splitunc(p):
    """Split a pathname into UNC mount point and relative path specifiers.

    Return a 2-tuple (unc, rest); either part may be empty.
    If unc is not empty, it has the form '//host/mount' (or similar
    using backslashes).  unc+rest is always the input path.
    Paths containing drive letters never have an UNC part.
    """
    if p[1:2] == ':':
        return '', p # Drive letter present
    firstTwo = p[0:2]
    if firstTwo == '//' or firstTwo == '\\\\':
        # is a UNC path:
        # vvvvvvvvvvvvvvvvvvvv equivalent to drive letter
        # \\machine\mountpoint\directories...
        #           directory ^^^^^^^^^^^^^^^
        normp = normcase(p)
        index = string.find(normp, '\\', 2)
        if index == -1:
            ##raise RuntimeError, 'illegal UNC path: "' + p + '"'
            return ("", p)
        index = string.find(normp, '\\', index + 1)
        if index == -1:
            index = len(p)
        return p[:index], p[index:]
    return '', p


# Split a path in head (everything up to the last '/') and tail (the
# rest).  After the trailing '/' is stripped, the invariant
# join(head, tail) == p holds.
# The resulting head won't end in '/' unless it is the root.

def split(p):
    """Split a pathname.

    Return tuple (head, tail) where tail is everything after the final slash.
    Either part may be empty."""

    d, p = splitdrive(p)
    # set i to index beyond p's last slash
    i = len(p)
    while i and p[i-1] not in '/\\':
        i = i - 1
    head, tail = p[:i], p[i:]  # now tail has no slashes
    # remove trailing slashes from head, unless it's all slashes
    head2 = head
    while head2 and head2[-1] in '/\\':
        head2 = head2[:-1]
    head = head2 or head
    return d + head, tail


# Split a path in root and extension.
# The extension is everything starting at the last dot in the last
# pathname component; the root is everything before that.
# It is always true that root + ext == p.

def splitext(p):
    """Split the extension from a pathname.

    Extension is everything from the last dot to the end.
    Return (root, ext), either part may be empty."""
    root, ext = '', ''
    for c in p:
        if c in ['/','\\']:
            root, ext = root + ext + c, ''
        elif c == '.':
            if ext:
                root, ext = root + ext, c
            else:
                ext = c
        elif ext:
            ext = ext + c
        else:
            root = root + c
    return root, ext


# Return the tail (basename) part of a path.

def basename(p):
    """Returns the final component of a pathname"""
    return split(p)[1]


# Return the head (dirname) part of a path.

def dirname(p):
    """Returns the directory component of a pathname"""
    return split(p)[0]


# Return the longest prefix of all list elements.

def commonprefix(m):
    "Given a list of pathnames, returns the longest common leading component"
    if not m: return ''
    prefix = m[0]
    for item in m:
        for i in range(len(prefix)):
            if prefix[:i+1] <> item[:i+1]:
                prefix = prefix[:i]
                if i == 0: return ''
                break
    return prefix


# Get size, mtime, atime of files.

def getsize(filename):
    """Return the size of a file, reported by os.stat()"""
    st = os.stat(filename)
    return st[stat.ST_SIZE]

def getmtime(filename):
    """Return the last modification time of a file, reported by os.stat()"""
    st = os.stat(filename)
    return st[stat.ST_MTIME]

def getatime(filename):
    """Return the last access time of a file, reported by os.stat()"""
    st = os.stat(filename)
    return st[stat.ST_MTIME]
```

```
# Is a path a symbolic link?
# This will always return false on systems where posix.lstat doesn't exist.

def islink(path):
    """Test for symbolic link.  On WindowsNT/95 always returns false"""
    return 0


# Does a path exist?
# This is false for dangling symbolic links.

def exists(path):
    """Test whether a path exists"""
    try:
        st = os.stat(path)
    except os.error:
        return 0
    return 1


# Is a path a dos directory?
# This follows symbolic links, so both islink() and isdir() can be true
# for the same path.

def isdir(path):
    """Test whether a path is a directory"""
    try:
        st = os.stat(path)
    except os.error:
        return 0
    return stat.S_ISDIR(st[stat.ST_MODE])


# Is a path a regular file?
# This follows symbolic links, so both islink() and isdir() can be true
# for the same path.

def isfile(path):
    """Test whether a path is a regular file"""
    try:
        st = os.stat(path)
    except os.error:
        return 0
    return stat.S_ISREG(st[stat.ST_MODE])


# Is a path a mount point?  Either a root (with or without drive letter)
# or an UNC path with at most a / or \ after the mount point.

def ismount(path):
    """Test whether a path is a mount point (defined as root of drive)"""
    unc, rest = splitunc(path)
    if unc:
        return rest in ("", "/", "\\")
    p = splitdrive(path)[1]
    return len(p)==1 and p[0] in '/\\'


# Directory tree walk.
# For each directory under top (including top itself, but excluding
# '.' and '..'), func(arg, dirname, filenames) is called, where
# dirname is the name of the directory and filenames is the list
# files files (and subdirectories etc.) in the directory.
# The func may modify the filenames list, to implement a filter,
# or to impose a different order of visiting.

def walk(top, func, arg):
    """Directory tree walk whth callback function.

    walk(top, func, args) calls func(arg, d, files) for each directory d
    in the tree rooted at top (including top itself); files is a list
    of all the files and subdirs in directory d."""
    try:
        names = os.listdir(top)
    except os.error:
        return
    func(arg, top, names)
    exceptions = ('.', '..')
    for name in names:
        if name not in exceptions:
            name = join(top, name)
            if isdir(name):
                walk(name, func, arg)


# Expand paths beginning with '~' or '~user'.
# '~' means $HOME; '~user' means that user's home directory.
# If the path doesn't begin with '~', or if the user or $HOME is unknown,
# the path is returned unchanged (leaving error reporting to whatever
# function is called with the expanded path as argument).
# See also module 'glob' for expansion of *, ? and [...] in pathnames.
# (A function should also be defined to do full *sh-style environment
# variable expansion.)

def expanduser(path):
    """Expand ~ and ~user constructs.

    If user or $HOME is unknown, do nothing."""
    if path[:1] <> '~':
        return path
    i, n = 1, len(path)
    while i < n and path[i] not in '/\\':
        i = i+1
    if i == 1:
        if os.environ.has_key('HOME'):
            userhome = os.environ['HOME']
        elif not os.environ.has_key('HOMEPATH'):
            return path
        else:
            try:
                drive=os.environ['HOMEDRIVE']
            except KeyError:
                drive = ''
            userhome = join(drive, os.environ['HOMEPATH'])
    else:
        return path
    return userhome + path[i:]


# Expand paths containing shell variable substitutions.
# The following rules apply:
#       - no expansion within single quotes
#       - no escape character, except for '$$' which is translated into '$'
#       - ${varname} is accepted.
#       - varnames can be made out of letters, digits and the character '_'
# XXX With COMMAND.COM you can use any characters in a variable name,
# XXX except '^|<>='.

varchars = string.letters + string.digits + '_-'

def expandvars(path):
    """Expand shell variables of form $var and ${var}.

    Unknown variables are left unchanged."""
    if '$' not in path:
        return path
    res = ''
    index = 0
    pathlen = len(path)
    while index < pathlen:
        c = path[index]
        if c == '\'':   # no expansion within single quotes
            path = path[index + 1:]
            pathlen = len(path)
            try:
                index = string.index(path, '\'')
                res = res + '\'' + path[:index + 1]
            except string.index_error:
                res = res + path
                index = pathlen -1
        elif c == '$':  # variable or '$$'
            if path[index + 1:index + 2] == '$':
                res = res + c
                index = index + 1
            elif path[index + 1:index + 2] == '{':
                path = path[index+2:]
                pathlen = len(path)
                try:
                    index = string.index(path, '}')
                    var = path[:index]
                    if os.environ.has_key(var):
                        res = res + os.environ[var]
                except string.index_error:
                    res = res + path
                    index = pathlen - 1
            else:
                var = ''
                index = index + 1
                c = path[index:index + 1]
                while c != '' and c in varchars:
                    var = var + c
                    index = index + 1
                    c = path[index:index + 1]
                if os.environ.has_key(var):
                    res = res + os.environ[var]
                if c != '':
                    res = res + c
        else:
            res = res + c
        index = index + 1
    return res


# Normalize a path, e.g. A//B, A/./B and A/foo/../B all become A/B.
# Previously, this function also truncated pathnames to 8+3 format,
# but as this module is called "ntpath", that's obviously wrong!
```

```
def normpath(path):
    """Normalize path, eliminating double slashes, etc."""
    path = string.replace(path, "/", "\\")
    prefix, path = splitdrive(path)
    while path[:1] == os.sep:
        prefix = prefix + os.sep
        path = path[1:]
    comps = string.splitfields(path, os.sep)
    i = 0
    while i < len(comps):
        if comps[i] == '.':
            del comps[i]
        elif comps[i] == '..' and i > 0 and comps[i-1] not in ('', '..'):
            del comps[i-1:i+1]
            i = i-1
        elif comps[i] == '' and i > 0 and comps[i-1] <> '':
            del comps[i]
        else:
            i = i+1
    # If the path is now empty, substitute '.'
    if not prefix and not comps:
        comps.append('.')
    return prefix + string.joinfields(comps, os.sep)


# Return an absolute path.
def abspath(path):
    """Return the absolute version of a path"""
    try:
        import win32api
        try:
            return win32api.GetFullPathName(path)
        except win32api.error:
            return path # Bad path - return unchanged.
    except ImportError:
        if not isabs(path):
            path = join(os.getcwd(), path)
        return normpath(path)
```

F4. stat.py

```
# Module 'stat'
#
# Defines constants and functions for interpreting stat/lstat struct
# as returned by os.stat() and os.lstat() (if it exists).
#
# Suggested usage: from stat import *
#
# XXX Strictly spoken, this module may have to be adapted for each POSIX
# implementation; in practice, however, the numeric constants used by
# stat() are almost universal (even for stat() emulations on non-UNIX
# systems like MS-DOS).

# Indices for stat struct members in tuple returned by os.stat()

ST_MODE  = 0
ST_INO   = 1
ST_DEV   = 2
ST_NLINK = 3
ST_UID   = 4
ST_GID   = 5
ST_SIZE  = 6
ST_ATIME = 7
ST_MTIME = 8
ST_CTIME = 9

# Extract bits from the mode

def S_IMODE(mode):
   return mode & 07777

def S_IFMT(mode):
   return mode & 0170000

# Constants used as S_IFMT() for various file types
# (not all are implemented on all systems)

S_IFDIR  = 0040000
S_IFCHR  = 0020000
S_IFBLK  = 0060000
S_IFREG  = 0100000
S_IFIFO  = 0010000
S_IFLNK  = 0120000
S_IFSOCK = 0140000

# Functions to test for each file type

def S_ISDIR(mode):
   return S_IFMT(mode) == S_IFDIR

def S_ISCHR(mode):
   return S_IFMT(mode) == S_IFCHR

def S_ISBLK(mode):
   return S_IFMT(mode) == S_IFBLK

def S_ISREG(mode):
   return S_IFMT(mode) == S_IFREG

def S_ISFIFO(mode):
   return S_IFMT(mode) == S_IFIFO

def S_ISLNK(mode):
    return S_IFMT(mode) == S_IFLNK

def S_ISSOCK(mode):
    return S_IFMT(mode) == S_IFSOCK

# Names for permission bits

S_ISUID = 04000
S_ISGID = 02000
S_ENFMT = S_ISGID
S_ISVTX = 01000
S_IREAD = 00400
S_IWRITE = 00200
S_IEXEC = 00100
S_IRWXU = 00700
S_IRUSR = 00400
S_IWUSR = 00200
S_IXUSR = 00100
S_IRWXG = 00070
S_IRGRP = 00040
S_IWGRP = 00020
S_IXGRP = 00010
S_IRWXO = 00007
S_IROTH = 00004
S_IWOTH = 00002
S_IXOTH = 00001
```

F5. UserDict.py

```
# A more or less complete user-defined wrapper around dictionary objects

class UserDict:
    def __init__(self, dict=None):
        self.data = {}
        if dict is not None: self.update(dict)
    def __repr__(self): return repr(self.data)
    def __cmp__(self, dict):
        if isinstance(dict, UserDict):
            return cmp(self.data, dict.data)
        else:
            return cmp(self.data, dict)
    def __len__(self): return len(self.data)
    def __getitem__(self, key): return self.data[key]
    def __setitem__(self, key, item): self.data[key] = item
    def __delitem__(self, key): del self.data[key]
    def clear(self): self.data.clear()
    def copy(self):
        if self.__class__ is UserDict:
            return UserDict(self.data)
        import copy
        return copy.copy(self)
    def keys(self): return self.data.keys()
    def items(self): return self.data.items()
    def values(self): return self.data.values()
    def has_key(self, key): return self.data.has_key(key)
    def update(self, dict):
        if isinstance(dict, UserDict):
            self.data.update(dict.data)
        elif isinstance(dict, type(self.data)):
            self.data.update(dict)
        else:
            for k, v in dict.items():
                self.data[k] = v
    def get(self, key, failobj=None):
        return self.data.get(key, failobj)
```

F6. callFMV

```
# callFMV:

import os
if hasattr(GameLogic, "fmvName"):

  filename = GameLogic.fmvName

  f = open("fmv-ed.in", "w")
  f.write(filename)
  f.close()
  s = os.system("fmv-ed")
```

F7. hasFMVfinished

```
# hasFMVfinished:

import os

f = open("fmv-ed.out", "r")
in1 = f.read()
f.close()

if in1 == "ok":
  f = open("fmv-ed.out", "w")
  f.write("")
  f.close()
  ownr = GameLogic.getCurrentController().getOwner()
  ownr.unpause = 1
```

F8. playFMV

```
# playFMV:

ownr = GameLogic.getCurrentController().getOwner()
GameLogic.fmvName = ownr.fmvName
ownr.playFMV = 1
```

Appendix G. Internship Report – Digital University