Final Report

Virtual humans in games: realistic behavior and emotions for non-player characters

TI3800 Bachelorproject

July 2014

Authors:
Selman Ercan
Raoul Harel
Jeroen Peperkamp
Umit Yilmaz

Supervisors:
Rafael Bidarra
Joost Broekens

Coordinators:
Martha Larson
Feilienne Hermans
Foreword

This document is the final report for the bachelor project we performed for the TU Delft at the Computer Science faculty. The writers of this document are Selman Ercan, Jeroen Peperkamp, Raoul Harel and Umit Yilmaz.

We would like to thank Rafael Bidarra and Joost Broekens for their support and advice throughout the project, and for always being available to answer our questions. They gave us a lot of positive feedback and clearly showed us how to improve our product.
Summary

This project was commissioned by the Interactive Intelligence group and the Computer Graphics and Visualization group at Delft University of Technology.

In this project we have set out to promote behavioral-realism in non-player-characters (NPCs) on an emotional level. We do this as a step towards a tool that will allow developers to build flexible and scalable emotional behavior for NPCs and that could be used either in game-development applications or scientific psychological studies.

In order to achieve this goal, we make use of two existing applications: Entika, a semantic world editor and Gamygdala, an emotion appraisal engine. The hypothesis is that integrating these two applications into one will yield a system that is capable of maintaining the aforementioned emotional processes for NPCs.

The main questions we have set out to answer are the following:

1. What is the best way to integrate Entika and Gamygdala?

2. After integrating these tools, do NPCs display plausible emotional reactions under a variety of circumstances?

The end result was an extension of the Entika editor with new “emotionally sensitive” objects, and the corresponding extension of the semantic engine to support emotional annotations of events in the game world. Relevant information is then forwarded to Gamygdala in order to obtain the emotional state of NPCs as output.
## Contents

1 Introduction .................................................. 5

2 Overview ......................................................... 6
   2.1 Project description .......................................... 6
   2.2 Problem definition .......................................... 7
   2.3 Employed frameworks ....................................... 8
      2.3.1 Entika .................................................. 8
      2.3.2 Gamygdala ............................................. 8

3 Approach ......................................................... 9
   3.1 Orientation .................................................. 9
      3.1.1 Implementation options ................................. 9
      3.1.2 User interface ......................................... 10
   3.2 Planning .................................................... 12
   3.3 Technical overview ........................................ 13
   3.4 Quality assurance .......................................... 14
      3.4.1 Unit testing ........................................... 14
      3.4.2 Regression testing .................................... 14
      3.4.3 Integration testing .................................... 14
      3.4.4 External code review .................................. 14
      3.4.5 Manual tests .......................................... 14

4 Requirements analysis ......................................... 15
   4.1 Non-functional ............................................. 15
   4.2 Functional ................................................ 16

5 Implementation .................................................. 18
   5.1 Product design ............................................. 18
      5.1.1 Extending Entika ...................................... 18
      5.1.2 Integration with Gamygdala ............................ 20
   5.2 Technical details .......................................... 21
      5.2.1 Visualizer ............................................. 21
      5.2.2 Affect button ......................................... 21
   5.3 Testing ..................................................... 22
      5.3.1 Unit testing ........................................... 22
      5.3.2 Integration testing .................................... 22

6 Process .......................................................... 23
   6.1 Development methodology ................................. 23

7 Evaluation ....................................................... 24
   7.1 Requirements evaluation .................................. 24
      7.1.1 Non-functional ....................................... 24
      7.1.2 Functional ........................................... 25
   7.2 Results ..................................................... 26
      7.2.1 Entika editor ......................................... 26
      7.2.2 Demonstration applications ............................ 26
   7.3 Feedback SIG .............................................. 29
7.4 Recommendations .................................................. 30
  7.4.1 Entika .......................................................... 30
  7.4.2 Gamygdala ..................................................... 31

8 Reflection ......................................................... 32
  8.1 Project review .................................................. 32
  8.2 Personal reflections of project members ...................... 32
    8.2.1 Selman ..................................................... 32
    8.2.2 Raoul ...................................................... 33
    8.2.3 Jeroen ...................................................... 34
    8.2.4 Umit ......................................................... 35

9 Conclusion .......................................................... 36

A Original project description ........................................... 37

B Research Report ................................................... 39
  B.1 Introduction .................................................. 39
  B.2 Entika and Gamygdala ........................................ 39
  B.3 Entika-Gamygdala interface .................................... 40
    B.3.1 Implementation options ................................. 41
    B.3.2 Discussion ................................................ 45
  B.4 Tests ............................................................ 45
    B.4.1 Plausible reactions ..................................... 46
    B.4.2 Plausible state ............................................ 47
    B.4.3 Stress test ................................................ 47
  B.5 The graphical user interface .................................... 47
    B.5.1 Approach .................................................. 48
    B.5.2 Design ..................................................... 49
  B.6 Graphical environment ......................................... 53

C Feedback SIG ..................................................... 55
  C.1 First review .................................................. 55
  C.2 Sprint summaries ............................................. 56
  C.3 Meetings ....................................................... 57

D Bug report .......................................................... 60
1 Introduction

Video games have seen a tremendous increase in popularity over recent years. This recently found success has triggered an increase in the amount of resources invested in game development. As a result, aspects such as graphics, sound and gameplay have been pushed on to new heights time and again.

However, one aspect was left behind: behavior, to be specific: emotional behavior. Even today, NPCs are - in essence - handicapped. Currently, whenever a developer desires to implement an NPC’s internal emotional process, it is scripted by hand. This approach is not generic and hardly scales with a project’s size. The industry lacks the tools to build plausible, flexible and scalable emotional NPC behavior. This project aims to bring us a few steps closer to such a toolset.

Achieving a complete solution to this problem will have great effects - and these won’t be limited to the video game industry alone. Not only will it speed up the development process of games, it could also form a basis for a wide range of scientific studies in psychology and related fields.

The project is offered by Rafael Bidarra and Joost Broekens from the Delft University of Technology. Rafael Bidarra is associate professor in Game Technology and is with the Computer Graphics and Visualization group, where he works on topics in the fields of rendering, visualization of (scientific) data and 3D modeling. Joost Broekens is assistant professor and is with the Interactive Intelligence group. The group performs research in the fields of agent reasoning, computational and perceptual intelligence and user-centered design.

This report is set up as follows: first we establish an overview of the project’s mission (section 2) followed by an analysis of requirements (section 4). Then, a description of the approach taken follows with the orientation, planning and technical setup explained as well as our approach towards quality assurance (section 3). Next comes the implementation (section 5), where the program’s architecture is explained. Then comes the process (section 6), listing achievements, progress at various milestones and description of meetings. This is followed by an evaluation of the results achieved (section 7). Nearing the end we reflect back on the project (section 8). Finally there is the conclusion (section 9).
2 Overview

This section gives an overview of how the project has been set up, who was involved and what was to be accomplished. To this end it is divided in three subsections. First in line is the project description, followed by the problem definition and finally an overview of the frameworks used in the project.

2.1 Project description

The main goal of the Virtual humans in games project is to integrate the semantic world editor Entika and the emotion appraisal engine Gamygdala. In the words of the client (quoted from the original project description, see appendix A):

[Entika is] “... a semantic library and engine that enables the creation and use of semantic virtual worlds, in which objects have meaning and functions, assume roles, provide services and serve some purpose(s) while interacting with other objects and NPCs.”

[Gamygdala is] “... an emotion simulation engine that is computationally light-weight and can simulate emotions based on the meaning of events for individual NPCs, if the NPC has defined goals and the events are annotated such that it is clear which event influence what goal.”

The goal of the project, simply put, is to combine these two frameworks so that it becomes possible to simulate psychologically plausible emotional behavior of NPCs based not only on interactions with each other, but also with the world around them that is enriched with semantics.

The end result should be an integrated environment in which plausible emotions can be generated for entities defined in the semantic world, based on executed events during a simulation run in Entika.

The involved parties are as follows. There are the four authors of this report, Selman Ercan, Raoul Harel, Jeroen Peperkamp and Umit Yilmaz; they also form the team that implemented the integration solution. There are also two coaches that have helped guide the team (coincidentally, together they also form the client): Rafael Bidarra and Joost Broekens. They each contributed to the development of one of the two applications; Mr. Broekens to Gamygdala and Mr. Bidarra to Entika.
2.2 Problem definition

While the visual quality in games has improved a lot in the past decades, the gap between what the game world looks like and how it feels and reacts has grown as well. The better a game looks, the more obvious it becomes that most objects in it have no purpose. Imagine a room full of hundreds of books, while only one or just a couple are ones with which you can interact.

To solve this problem, the idea of defining semantics for objects has arisen, which has lead to the implementation of Entika[3].

The increase in visual quality of games not only reveals the lack of object interaction, but also highlights how implausible NPCs behavior tends to be. In order to address this, a black-box emotion appraisal engine was created in the form of Gamygdala, which is based upon the well-known OCC model of emotion appraisal[7].

Solving the problem introduced by this project comes down to answering two main questions:

1. What is the best way to integrate Entika and Gamygdala?
2. After integrating these tools, do NPCs display plausible emotional reactions under a variety of circumstances?

We will answer these questions in the remainder of this report.
2.3 Employed frameworks

To know what we are dealing with, we have investigated the tools we were given. There are two main pieces of software that we have to work with: Entika and Gamygdala. Each serves its own purpose and was designed with its own goals and perspective in mind. This section will describe both of them in more detail.

2.3.1 Entika

Entika is all about semantics, particularly enabling the addition and maintenance of semantics inside game worlds. These semantic definitions include objects, actions, interactions, events, effects and more. The framework consists of two separate modules, the semantics editor and the semantics engine. It was our focus to extend both of these modules so they could interact with Gamygdala.

The editor is the piece of the software that the user interacts with. It enables the specification of the semantics of the world, according to the model outlined in [3]. The semantics thus defined are stored in SQLite database files so that the engine will be able to load them consistently. This model of data storage also enables extension of the framework, since adding new tables does not interfere with the working of old ones.

The engine in its turn reads all the data about semantics from the database files. It makes objects in memory for everything it reads so that access to those will be faster. Once everything is loaded, the update loop of the engine is invoked, where events are fired and handled; such as objects moving around, appearing, disappearing or having an attribute change. The instruction to handle all these things has to be given from an external program, however; the engine itself does not have an ‘on switch’ after which processing can continue without the need for any interaction. In other words, the external program needs to call the update function at a certain frequency, since the engine does not keep itself running.

2.3.2 Gamygdala

Gamygdala[7] is an emotion appraisal engine. Basically what it does is given a set of goals of an entity and beliefs that affect those goals, it generates emotions befitting the entity’s current state of mind. In this context, a belief is an annotated event present in the game world; these annotations include the goals the event influences and whether this influence is positive or negative, the likelihood of this belief being true and the agent that has caused it.

The way the emotion generation works is as follows: for each agent that has emotions, a ‘brain’ is made. These brains each store their own set of goals and beliefs, based on which the updater function calculates the new emotional state. The calculation itself is based on beliefs and goals. From a given belief it follows which goals are affected, how they are affected (positively or negatively), with what likelihood and by whom. This data is then used to determine the correct emotion type and intensity, not only of the entity’s internal emotions, but also social ones. A decayer function could also be specified which controls the decay of emotion intensities over time.

The brains also have two ways in which they make the emotions available for querying. There are eight internal and eight social emotions that can be simulated, which can be queried directly. Alternatively, the emotional state can be mapped onto PAD space (Pleasure, Arousal, Dominance)[5]. Finally, similar to the semantics engine described in the previous section, the function that invokes the emotion recalculation procedure needs to be called from an external program.
3 Approach

3.1 Orientation

Before the work on the product could properly begin, the team went through an orientation phase. During this phase, research was done into what exactly was expected and how the goals envisioned could be realized. The main issue was how to link the two libraries together. Another issue was how to make a suitable user interface for the product that would enable game developers to define emotional agents with ease. The implementation options will be discussed in section 3.1.1, and the interface in section 3.1.2. For the full detailed discussion of the orientation phase, please refer to appendix B.

3.1.1 Implementation options

Originally the research phase has yielded three main options:

1. Using Gamygdala functionality in Entika;
2. Using Entika data in Gamygdala, or;
3. Making a dynamic mapping between the two.

We will now briefly summarize the pros and cons of each option. The enclosed research report (appendix B) can be referred to for further details.

Pros The first option would require only a single editor so there would be no need to go back and forth between an editor for the world and an editor for emotions. The second option would allow for the creation of a new, better thought-out user interface, and it would also better encourage modularity. The third option would take most of the work out of the hands of developers and make the product perform the work itself — figuring out the proper mapping between objects in Entika and the emotion processing units from Gamygdala.

Cons The first option sacrifices modularity and maintainability in favor of an easy (initial) implementation. It also introduces code duplication as it implies that similar functionality will be programmed twice - once in Entika’s domain and once in Gamygdala’s. The second option is the one the team had eventually chosen, though it too has a drawback: namely that it could be a potentially cumbersome process to define beliefs for all entities defined in Entika, given that there could be many hundreds of them. The third option causes an unwanted dependency in that the underlying data definitions of objects in Entika and Gamygdala had to conform strictly to an imposed standard. This hinders the maintainability as well as the flexibility of a project utilizing this approach.

Judging all pros and cons together, the team has eventually decided on the second option as the most desirable one.

Having chosen the preferred solution, it was time to implement it. However a short time after reaching this conclusion a meeting took place with the client, which have made us reconsider our conclusion. During this meeting it was made quite clear that versioning concerns were not deemed an issue for this project; it was perfectly acceptable for the product to be explicitly dependent on
a specific version of the libraries. With this new knowledge a different choice was natural and so the fourth option was arrived at, which was to simply extend the Entika functionality. This hadn’t come up before because of the tight interdependence it incurs, but since during that meeting it was pointed out that was not a problem, the advantage of being able to reuse a lot of existing code clearly won over the meager advantages of the other options.

### 3.1.2 User interface

While there was a new product to be made, it was a natural requirement that there be some kind of user interface for this product. The original idea was, in accordance with the second option mentioned under the implementation options (3.1.1), to build an interface in which the emotional aspects of agents could be defined, i.e. an interface for Gamygdala, with an import function that would enable the developer to import the necessary entities from Entika in order to link them to their emotional counterparts. As mentioned previously, the second option was dropped in favor of the option to extend the existing Entika interface, hence obviating the need for a separate user interface.

Despite the fact that a separate interface was not going to be built, it was still necessary to put some thought into the eventual layout of the product’s extension to the existing interface. The research report (appendix B) provides the full details, but we will mention the basic components here for completeness.

![Figure 1: The main window of Entika Game Edition. The main menu tabs are visible on the left, while the sub-tabs are visible at the top portion of the sidebar.](image)

The Entika interface consists of several main tabs on the left side of the editor window with each of them expanding into a list with more tabs on top (see figure 1). The extension that has been designed follows this pattern, introducing the new main tab named ‘Emotion objects’ that contains three sub-tabs for the specification of event filters, ‘utility goals’ (i.e. goals with an added utility
parameter for Gamygdala’s purposes) and so-called animate objects (a child of the existing tangible object class). The user could add goals only to animate objects, because it was determined that only these kind of objects can have goals. Inanimate objects such as abstractions (e.g. companies) might be said to have goals too, but in fact this is false: it is the living entities that create and maintain them that have goals, which may or may not be properly expressed or attained through inanimate entities.
3.2 Planning

In this section the planning of the project will be discussed in more detail. The two main phases will be discussed, but the most important will be the second phase and it will consequently receive the most attention.

As has been mentioned, the project is split into two main phases. A research phase and an implementation phase. During the first, research is done into what the best way would be to integrate Gamygdala and Entika. Other questions that are researched are what would be a suitable way of exposing the integration via the Entika editor, and what kind of testing methodology should be used to test the product. During the second phase the insights obtained from the research phase are made into a concrete implementation.

Since the implementation phase consists of the bulk of the project, we have decided to further split it into three parts:

1. **Extension of Entika**
   During the first part the extension of the Entika editor is implemented. This part consists of first and foremost a lot of research into understanding how Entika and Gamygdala work, without that knowledge it is impossible to extend the editor in a sensible way. After this stage, both the front-end as well as the back-end of the editor are extended to support the new functionality.

2. **Integration with Gamygdala**
   In the second part Entika is integrated with Gamygdala. This part mainly consists of extending Entika’s simulation-engine to support forwarding of relevant information to Gamygdala.

3. **Product demonstration**
   During this third and final part, the capabilities of the created solution will be demonstrated. This, by programming two-to-three small programs that will act as a proof-of-concept.
3.3 Technical overview

In this section we provide an overview of the frameworks and tools we have used during the project.

**C#** Programming language.

**Visual Studio** Integrated development environment for C#.

**XNA** The framework used to visualize the proof-of-concept demo.

**Git** Revision control system.

**Bitbucket** Source code, issue tracker and wiki hosting service.

**ShareLaTeX** Collaborative document authoring service.
3.4 Quality assurance

3.4.1 Unit testing

We implemented tests for each isolated, stand-alone piece of functionality. Within classes that depended on external sources (like a database), we isolated the data access layer (the database access code) from the class itself, so that within the unit tests we could exclusively test the functionality of the test-subject independent of the database.

3.4.2 Regression testing

After adding a new piece of functionality, we checked whether the additions did not introduce bugs in already existing code. To do this, we verified that existing tests were still passing before committing the new code.

3.4.3 Integration testing

Test combination of multiple modules, each of which was unit tested separately. We used integration tests mostly to test our Entika-Gamygdala interface in combination with the data access functionality. Given time constraints and the principle of favoring unit tests over integration tests, we tested only the ‘sunny day’ scenarios (where the system is tested under common-use circumstances).

3.4.4 External code review

During the project code will be sent twice for review by the consultants of SIG (Software Improvement Group). Once near the fourth sprint and once at the end. The team has used SIG’s feedback to improve the codebase (see section 7.3).

3.4.5 Manual tests

After the addition/change/removal of each piece of functionality it was necessary to verify that all functional requirements described in the requirements analysis section (4) were still being met. For this purpose, we navigated through the editor manually and checked whether all relevant parts of the application were functioning correctly. This tests also cover all components that could not be tested using the other methodologies; these were mostly graphical user interface related components.
4 Requirements analysis

This section includes the functional and non-functional requirements our project should accord to. These have been developed within the research phase of the project.

The functional requirements will be classified based on the MoSCoW method (must-, should-, could- and won’t-have)[6]. This method sorts a project’s requirement into four levels:

**Must:** A requirement that is critical for a project to be successful.

**Should:** Just as important as must-haves, but are not required to be implemented in the given time-frame in order for the project to be successful.

**Could:** A lower priority class of requirements that are nice to have, but that should not be heavily focused.

**Won’t:** Requirements that is declared as unnecessary at this time. No time should be dedicated to implementing those.

4.1 Non-functional

The end result should adhere to the following requirements:

**Usability**

- Understandability: The code should be easily understood by other programmers.
- Documentation: Should be complete, accurate and clear.
- User-friendliness: It should be easy to achieve basic and advanced functional tasks.

**Sustainability and maintainability**

- Testability: It should be easy to test the code.
- Changeability: It should be straightforward to address issues, add new functionality and modify existing functionality.
- Extensibility: It should be easy for future developers to build new functionality on top of existing code.
- Robustness: how well does the system deal with disturbances?

**Backward compatibility**

- Should be compatible with previous versions of Entika (Generic and Game editions).
4.2 Functional

Ultimately the end product should deliver our goal on two fronts: definition and simulation. Definition implies that a user would (using the integration of Entika and Gamygdala) be able to define a semantic world with emotional content. Simulation implies a structured algorithm that is responsible for processing the defined emotional content during runtime.

Emotional content definition encompasses the following:

- Marking of objects (NPCs) in the world as ones that should host an emotional state
- Events that are annotated in such a way that declares their emotional effect on NPCs.

These annotated events would then need to be processed during the simulation.

Since the emotion-appraisal part of the integration consists of Gamygdala and because Gamygdala’s black box model expects a certain format of input, this format should be mirrored in the extension of the Entika Editor.

Below we expand on these requirements and delve into some specifics using the MoSCoW scheme:

Must have

- **Entika editor extension**
  - Allow the marking of objects as “animate” (capable of emotion processing).
  - Allow the definition of Gamygdala-related concepts in the editor.
    - Allow definition of goals and their utility.
    - Allow assignment of goals to animate entities.
    - Allow definition of annotated events and their likelihood.
    - Allow definition of congruence between an event and an NPC’s goal.

- **Semantic engine extension**
  - Each animate entity that is spawned should be linked to an agent in Gamygdala.
  - Annotated events should be recognized during run-time and their effects forwarded to Gamygdala.
  - The engine’s performance should be good enough to be applied in real-time applications and programs.

Should have

- **Entika editor extension**
  - Extension built on top of the Game Edition of Entika.
  - Same graphical style as is convention in the Generic and Game editions.
  - Same user interface capabilities as is convention in the Generic and Game editions (drag & drop, input validation, etc.).
Could have

• Entika editor extension
  – Dynamic definition of certain properties (utility, likelihood) so that they will be calculated on-the-fly during a simulation.

Won’t have

• Integration of the emotional content extension into already existing objects in Entika.
• Changes made to existing code of the Generic or Game editions.
• An alternative way of storing world definition (other than the standard Entika database scheme).
5 Implementation

5.1 Product design

This section describes the technical designs we made prior to implementation and the accompanying class diagrams.

5.1.1 Extending Entika

Animate objects and utility goals  In Entika there was no class that was explicitly meant to represent living entities. We therefore created an extension of the existing *TangibleObject* class: *AnimateObject*.

Each animate object needs to be able to have goals. For this purpose, we created a *UtilityGoal* class, instances of which can be linked to animate objects. As the name implies, a *utility* value can be set for these goals, which indicates how desirable a goal is for an animate object. Utility goals can be added to animate objects either directly or indirectly through inheritance (i.e. a child animate object inherits all utility goals the parent has).

The relationship between these classes is illustrated in figure 2.

![Class diagram for AnimateObject and UtilityGoal](image)

Figure 2: Class diagram for *AnimateObject* and *UtilityGoal*

Event filters  As was discussed in the section about Gamygdala, Gamygdala needs beliefs in order to generate emotions. To recap, a belief is merely an event that is extended with information about which goals it affects and in what way. The fact that Entika already contained an *Event* class meant that introducing beliefs was a small step, at least on a conceptual level. In theory, this could even be done just by adding a list of goals to each event.

However, since ease of use for game developers using the editor was an important requirement, we decided to implement this in a more generic way. We wanted to be able to specify a list of affected goals per goal, and not just for a single event but for a range of events so that the workload of the user of the system would stay minimal. Therefore, we needed a way to indicate, for example, all events with a specific action regardless of the actor and target, or all events where the actor was a member of a particular group, and so on.
This brought us to the idea of event filters, in which constraints can be specified on a range of properties of an event: the action, actor and target. Event filters also contain a credibility field, which is the equivalent of likelihood in Gamygdala. In this setup, the EventFilter class represents a belief. This allows us to define beliefs on a higher level of abstraction and for multiple events at once.

An action in Entika is synonymous to what the event actually means (for example: ‘see’, ‘hear’, ‘jump’, ‘take’). Via this constraint only events triggered by certain actions can be selected.

We have also specified some constraints for the actor and target (who performed the action and on whom). You can sort these two using a name (including/excluding ‘self’ — the entity processing the belief) and presence/absence in a certain group and/or relationship. In order to illustrate the capabilities of these constraints, here are a couple of examples:

**Action** “All events triggered by action ‘throw’ ”.

**Actor and target name** “All events triggered by actor ‘person’ and target ‘food’ ”.

**Group/relationship** “All events triggered by actors that are ‘friendly’ and that were performed on a target that belongs to group ‘nice things’ ”.

Our event filter hierarchy is visualized in figure 3.

![Figure 3: Class diagram for the event filters](image)

Once an event filter is defined, it can be added to a utility goal and its goal compatibility may be specified. This is the equivalent of congruence in Gamygdala — a number that indicates how well a belief promotes a certain goal.
5.1.2 Integration with Gamygdala

**Bridging interface between Entika and Gamygdala**  With the Entika extension (animate objects, goals and event filters) in place, the next step was to integrate this with Gamygdala. What was necessary for this was an interface that would be active during a simulation run of Entika and would pass data from Entika to Gamygdala about each event that was triggered. Specifically, the entities involved in the event, their goals, and how these goals were affected by this event needed to be extracted during runtime. After having transformed this data into suitable input for Gamygdala, it could be passed to Gamygdala for the emotion appraisal process.

See figure 4 for an overview of the Entika-Gamygdala interface and for the relations between classes in Entika, Gamygdala and our interface.

![Figure 4: Class diagram for the Entika-Gamygdala interface](image)

---

20
5.2 Technical details

5.2.1 Visualizer

For the demonstration applications, we needed a visualizer - a piece of software we could use to display a simple scenario or two on the screen in a human readable format. We have gone through numerous frameworks and tools during our search for one, both external (Unity3D) as well as internal (the visualizer from the semantic crowds project) but all were discarded, either due to technical reasons (Unity did not support the version of C♯ in which Entika is being developed) or conceptual ones (the semantic crowds visualizer simply did not contain the functionality that was needed).

Ultimately we settled on an in-house visualizer made using Microsoft’s XNA framework.

5.2.2 Affect button

In order to help ourselves and others visualize the emotional state of animate objects, we have made use of the affect button[1]: it is a rendering of a human face that is capable of displaying a range of emotions based on user input. We have ported the button to C♯ and have made use of it in our demonstration applications (see section 7.2.2).
5.3 Testing

5.3.1 Unit testing

Ideally, a unit test method should fail only if the functionality being tested is faulty. This means that it is desirable to separate the data access logic from the class functionality, so that possible bugs in the data access layer don’t interfere with the tests.

To this end, we applied the principle of dependency injection to the classes that we created unit tests for, with respect to their data access code. This means that we separated the code used within the classes to read and write data from the class definitions, and put this in separate data provider classes. In other words, we injected a dependency on the (now external) data providers into the original classes.

One of the benefits of this approach is that changes to the data access logic of a class can be made without having to modify the class itself. One could even switch to a completely different database management system without the class having to know anything about this; all that would be needed is to set the data provider of the class to the new one.

The main benefit that we were after, however, is that of isolating the data access layer during unit testing. Because the data provider is external to the class itself, in the tests we could pass a mock data provider to the class under test. By configuring the mock to return fake test data when called, we succeeded in testing code that normally depends on a database without having to use a database.

5.3.2 Integration testing

Proper integration tests are complex and time consuming to set up. For such tests dependencies need to be created, test databases need to be populated and proper test scenarios need to be written. Additionally, when a unit test fails, we know that this is caused by a change in the code. When an integration test fails, however, the problem could be in any of the used dependencies.

Bearing this and our tight schedule in mind, we created integration tests for only the most critical parts of the program. This was the Entika-Gamygdala interface, since it makes use of all that we implemented. Within this test, we tested the whole ‘traject’ from defining the game world in Entika and starting a simulation, to passing the required data to Gamygdala and verifying that the proper emotions were generated based on the events triggered during the simulation.

The cost of setting up this integration test was more than compensated. Several critical bugs in the data providers were brought to light; it also helped us realize in what ways the interface still needed to be improved.
6 Process

6.1 Development methodology

In order to implement the product properly, a proper methodology was needed. This helps facilitate a good end result and should keep the workers focused on the goals envisioned at the start of the project. Furthermore, it must be easy to incorporate changes in requirements and new insights acquired during meetings with the coaches.

The methodology chosen that seems to meet these criteria most closely is the so-called Scrum method. In short, it works as follows. At the beginning there is a list of items made that need to be finished in order to completely implement the product. This list is called the Backlog, and it is maintained by the product owner. (Since we are a small team, members of the team sometimes had multiple roles). The time of the project is divided into sprints with a length of one week each. During a sprint, a single coherent set of features is implemented, leading to there being a working, if incomplete, product at the end of each sprint.

Together with Scrum, we have also incorporated test-driven development (TDD) into the implementation cycle. Although some components of the code were impossible to test (mostly graphical user interface related components) we did write over a 120 tests over the course of the project. Not only unit-tests, but integration tests as well.
7 Evaluation

7.1 Requirements evaluation

In this section we will evaluate the results achieved after implementation of the requirements mentioned in the requirements analysis (section 4).

7.1.1 Non-functional

Understandability: Great effort has been put into writing code that is readable by other programmers. The code has been put in regions and has well written comments. Keeping the format as clean as possible, by indenting nested blocks and paragraphing the code, is achieved. Suitable names for classes and methods were chosen. Considering the 4 stars we got from SIG, we are content with the achieved result.

Documentation: The documentation we have in our project is the final report, a research report and a wiki on Bitbucket. During this project it became clear that documenting a project is something that needs to be done on a daily basis. At the start of the project our aim was to have a document containing information on which steps another programmer should take to extend our product. At the same time, we wanted a document that would contain a guide on how to use our features within the Entika Editor. Unfortunately because of a lack of time as well as underestimating creating proper user documentation, we did not achieve a satisfying result on this requirement.

User-friendliness: It is a hard task to achieve user-friendliness while extending an existing product. The one thing we wanted to avoid was having an extension which had a totally different GUI. We remained true to the original GUI of Entika and made sure our extension had the same look and feel as the existing product. Within this context, we adhered to our requirement of being user-friendly. We will make some recommendations concerning user-friendliness and Entika later on in this chapter.

Testability: The testability of our code was promising according to SIG. We have over 150 test cases, which has saved us countless hours of debugging time.

Changeability: The way Entika is programmed makes it easy to extend and adjust current functionality. We have remained true to the way Entika is built. By remaining true, we have managed to keep adjusting code and adding new functionality a relatively easy task.

Extensibility: This is one of the requirements which we have put much effort in. Not a single line of existing code was altered. All existing code was extended with care. Within our Emotion Edition of Entika, the Generic as well as Gaming edition of Entika are still fully operational. By using the proper access modifiers, other programmers can also easily extend our product.

Robustness: By employing Test Driven Development, many possible failures were prevented. Exploratory testing was used to make sure that input from the Editor could not lead to failure.

Backward compatibility: Since Entika works with databases and our database has more entries, our product is not fully backwards compatible. Databases created in previous versions of Entika can be used, but the other way around, a database created with our version cannot be used in previous editions.
7.1.2 Functional

**Must have**  The two things that were essential to have were the extension of the Entika editor and of the semantic engine. As for the editor, the way we have enabled objects to be marked as animate is to create a new type of object that we have called an animate object. The details of this and other things we implemented can be found in section 5.

The Gamygdala-related concepts also have all found their place in the editor. Goals are defined as independent objects that need to be assigned to animate objects, with the utility defined at the animate object. Similarly events are bunched together in event filters that can have a certain likelihood and that need to be assigned to a goal. At the goal the congruence can be defined.

The semantic engine also has been fully implemented as desired. The agents have been linked in a special class called EmotionInterface that is used by our version of the semantics engine. This extended version of the semantics engine also automatically checks for animate objects that are involved in any events that occur so they can generate the right emotion. We have not done extensive performance testing yet because of a lack of time, however, the simulations all ran smoothly with several dozen animate objects in view at times.

**Should have**  Of course the editor extension had to be built on top of something, and it was not a big deal to build it on the game edition of Entika. Also the graphical styles and interface capabilities were easily implemented due to the extensible nature of the WPF components that were already present, and therefore ultimately the extensible design of Entika editor.

**Could have**  We had begun constructing a way to specify dynamic likelihood, however due to a lack of time we had to abandon the effort near the end of the project.

**Won’t have**  The problem with building the emotional capabilities into existing Entika objects turned out to be that those are not supposed to be modified. Entika was built to be extended, but also to make quite sure that existing objects would remain untouched. This is a good thing because otherwise the stability of the system might be impacted.

Clearly we could not modify existing code, already because we have no write access to the repository that contains it but also because it would be bad practice because of the aforementioned risk of instability.

Finally, to make an alternate way to store worlds would mean to change the heart of Entika. As we have seen in the beginning of the project on the code maps that Visual Studio is able to generate, almost everything is dependent on the database. Trying to implement a new storage format would amount to reimplementing almost the entire system, which we certainly have no time for.
7.2 Results

In this section we present the results achieved at the end of this project. As was mentioned in previous sections, our integration solution consists of the extension of two components: 1. The Entika editor and 2. Entika’s semantic engine. We demonstrate the new editor and engine capabilities in sections 7.2.1 and 7.2.2 respectively.

7.2.1 Entika editor

The Entika editor was extended per the design mentioned in chapter 5. A new ‘emotion objects’ menu tab was added, which hosts all emotion-related content under three sub-tabs: animate objects, event filters and utility goals (figure 5). The graphic style and user-interface has been designed to comply with the existing style and conventions of the other editions of Entika.

After creating a utility goal, the user may specify its utility value under the ‘Utility’ expander. This value serves only as a default, and may be overridden when adding a utility goal to an animate object (figure 6a and 6b).

Aside from utility, a user should also specify which beliefs affect the goal. This is done by adding an event filter to the goal, under the ‘Event filter’ expander (figure 6c). Once added, the goal compatibility may also be specified.

Defining an event filter is straightforward. First the user must create a new event filter node and then specify its constraints and credibility (figure 7).

7.2.2 Demonstration applications

In order to demonstrate the workings of the new engine, we have implemented two applications that make use of the integration solution: a simulation and a short single-player game.

The simulation consists of a busy street crossing. Pedestrians (animate objects) spawn randomly on the map and go about their business. They may choose to visit an ice-cream stand, a hot-dog stand or an ATM if they require more money. They may also wish to go and sit down on a nearby bench to rest in case they get tired and occasionally while crossing the road, a pedestrian may be ran over by a passing vehicle.

The pedestrians get happy when buying food and getting money, and become distressed when tired or while witnessing an accident. The emotional state of the pedestrians is rendered using the affect button (section 5.2). This application demonstrates the scalable nature of our solution.

In the game application, the player plays the part of Tommy an innocent young man who takes a walk around the park. On his journey, the player may choose to interact with any object passing his way. These could be trees, flowers, animals or other people. There is also a happiness bar that the player should make sure not reaches zero, otherwise the game is lost. This setup demonstrates the application of our solution to a real game prototype.
Figure 5: The three sub-tabs hosted under the ‘emotion objects’ menu option.

(a) A user may set the default utility of a goal.

(b) The utility of a goal can be overridden once added to an animate object.

(c) Event filters can be linked to goals.

Figure 6: Utility goals.
(a) An overview of event filter constraints.

(b) The credibility of a filter is defined much like the utility of a goal, but cannot be overridden.

Figure 7: Event filters.
7.3 Feedback SIG

The first batch of code was sent to SIG on the 13th of June. The code scored four stars on the maintainability category and was denied the fifth star due to duplication issues in one of the components. Another recommendation was to place test-code in the same project as the code that is being tested. Aside from that the reviewer was overall positive and our code was noted to be of above-average quality. The verbatim contents of the feedback can be seen in appendix C.

We have addressed the first recommendation by migrating the duplicated code into a common base-class. However, we have consciously chosen to not address the second recommendation, and here is why: Placing testing code in a separate project other than the production code is a common practice amongst C♯ developers — and rightly so. Having the test-code be in the same project as the production code would mean that it would be packed into the assembly that would eventually be used by consumers of the library. Since all of the test code is not relevant to these consumers, it would then be a manifestation of dead code.

Therefore, although we agree that putting test- and production code nearby each other promotes conciseness, we do not deem this practice useful enough to overpower the mentioned downside.
7.4 Recommendations

In this section we give some recommendations based on the experiences we’ve had with the project and the tools we worked with. We will start with a few general recommendations and then zoom in on both Entika and Gamygdala.

The team worked according to the TDD methodology, as has been described in the report (section 6.1). This way of working is highly recommendable in itself, since it really helped at least in this project to find some bugs right away without having to search for the cause of a new problem. Our main recommendation here, however, is that developers stick with the methodology. It is so easy to get distracted and just write code before tests, but it is important to always remember to first write tests and only then get to implementing the actual program.

During the project, we also held weekly meetings with the supervisors. These meetings were almost always enlightening and they helped us stay on course. Unfortunately there were some meetings at which only one of the supervisors could be present. That could be improved upon because it is important that the views of both of them be represented. It would therefore be recommendable that the supervisors make sure to schedule the regular meetings in advance so it becomes less likely that they miss one.

In the beginning of the project, during the research phase, it took a while before we got everything we needed to be able to find out what would be the best solution. At the end of the research phase we finally got a meeting with both supervisors where it turned out that the scruples we had had about depending too much on the existing tools were not necessary, so we basically took a different approach. This situation emphasizes a few points. Firstly, it would be better if we had gotten the necessary information, including more comprehensive documentation, a bit earlier. Secondly, it would have been very good to have had a meeting with both supervisors so that we had known earlier on that we could relax some of the requirements we were trying to adhere to.

7.4.1 Entika

The possibilities in the current versions of Entika are enormous. The idea of having objects contain their usefulness themselves is a strong solution for solving the problem of having to enable characters both real and computer simulated to interact with them. It would be such a massive task to have to make every object in the world interactive and natural by hand, but with Entika there is a hierarchical model that enables the specification of properties that apply to everything, or to certain categories, before filling in the specifics of each particular type of object.

Be that as it may, there are still some issues that we would recommend be addressed, to maximize the potential of this tool. The underlying semantic model is very sophisticated and it seems to have an analogue or some way simulate almost everything in the real world. One thing that is still missing, however, is the legal system, i.e. laws. Given that the purpose of Entika was to simulate game worlds, there may not be too much use for them currently, but one day it may come in handy in a more complete simulation of the world.

A more important issue is the way the editor is laid out, namely using a lot of nested tabs and expanders. This can be a significant hindrance for developers who want to become fluent in the use of the editor, because they are likely to have to do a lot of navigating before they get to the specific point where they want to be. This limits the efficiency that can be achieved, even if there are back and forward buttons. One specific point also about the user interface would be that it is
not entirely intuitive to get more options when items in a list box are clicked and different options when such items are expanded.

Finally, as was mentioned cursorily, it is highly recommended to get more comprehensive documentation for the editor as well as the code behind it. It took a lot of time to plough through the code, trying to find meaning, and the comments in the code weren’t exactly helpful most of the time either. Had these things been better, we might have been able to get more understanding a lot quicker and who knows how much more of Entika’s possibilities we would have been able to exploit.

### 7.4.2 Gamygdala

Gamygdala is a lot smaller, so even though it too lacked some documentation, it was still relatively easy to figure it out. Our understanding of it was also boosted by the fact that it was a fairly direct implementation of the paper that accompanies it. (While there was also a paper about Entika, the code for the latter was much more complex than could be conveyed in such a format.)

Unfortunately we also discovered that there was something off with it when we started experimenting with it. When an NPC was supposed to feel only negative emotions (when getting slapped), it also felt a positive emotion (gratitude). We discussed this issue with the relevant supervisor and clearly it is recommended that it be fixed, although it may not be easy because it seemed to be related to an error in the underlying model.

Perhaps this is what Gamygdala was made for, but it has also become apparent that it only supports achievement goals, i.e. goals that represent a single, discrete achievement, to generate emotions. It would therefore be very nice if there could also be an option to support state goals, i.e. goals to maintain a certain state, such as to keep a room warm or to keep the sharks wet, because clearly fluctuations in the relevant states will also cause emotion in natural situations.

Another recommendation we have found, with the help of one of the coaches, is to have an option for selecting the most plausible emotion to render or output, out of the combination of emotions that may have a nonzero intensity at any moment. This is the way the OCC model works, by having a tree and enabling multiple emotions in it to be active at the same time, specifying at each level another detail of the emotion. It would be good to have some way to have the lowest level, i.e. the one with the most details, be the one that’s picked automatically.

A final recommendation that actually goes for both tools that we would make is to write some comprehensive tests. In the case of Gamygdala, test cases were present, but it turned out that the test cases had been thought up with the same flawed mindset that had led to the bug, thus leaving it unnoticed. In Entika we also found a bug, but that was in a piece of code that was never used inside Entika itself. We have documented the bugs we found in appendix D.
8 Reflection

8.1 Project review

Looking back, this project has had a slow start and it took the team some time before they settled down onto the right rhythm. At the very beginning, there was a period of confusion and uncertainty, where the team was not entirely sure what kind of product is expected and in which way it should be achieved. This was caused mainly by conflicting schedules of the two clients, Mr. Bidarra and Mr. Broekens, that delayed the scheduling of a meeting where both would be present by about two or three weeks. Since each of them held key pieces of knowledge regarding the two main applications involved (Entika and Gamygdala) it was hard to get a complete picture of what is desired and what is not until after that period. Once it was managed to agree on a fixed meeting schedule with the clients, things got a lot clearer.

At that point, the team started focusing on the software itself, reviewing both applications’ code. Gamygdala, being a relatively small and compact piece of software, was straightforward to understand. Entika, however, is an entirely different story. It is an enormous and incredibly complex beast. The coming weeks we were struggling to navigate through its codebase and understand what each of the components’ task was. We would like to emphasize that a more comprehensive documentation would have gone a long way in helping an unfamiliar developer grasp the code.

Once we had Entika figured out, we transitioned into the implementation phase. All in all, this phase went rather smoothly. Our weekly meetings with the client were also always enlightening and team members got used to working with one another. Again, the toughest hurdle at this stage was Entika, whose code we have found to be too inflexible for our needs, which slowed down progress at times.

Near the end of the project, while preparing the final demos for the presentation, we have stumbled upon a couple of technical oversights in Gamygdala — bugs that caused illogical emotional responses to be generated under certain conditions. We have shared our findings with Mr. Broekens, and together we have implemented a temporary patch which addresses some of the problems.

8.2 Personal reflections of project members

8.2.1 Selman

I experienced this as a challenging and instructive project. It was quite different from the projects earlier in the curriculum in that it required a higher degree of independence. Whereas with other projects there was external guidance and structure, here it was pretty much a ‘blank canvas’ and the organization rested largely on ourselves.

At the same time I found it a ‘good’ and representative bachelor project. We had to show knowledge and insight in practically all aspects of software development; from communication with the client, design and documentation to testing and implementation.

One of the main challenges was the learning curve associated with the large codebases that had been developed over multiple years. The amount of work we had to do necessitated that we had to get up to speed quickly with the software. However, the sheer size and complexity of it meant that quite some time was needed before we were comfortable with it. Because of this our schedule got even tighter.
When looked at from this point of view, it becomes even more satisfying to realize that we succeeded in designing, solving and implementing all that we planned. Each of the three main phases felt more like a mini project on its own, rather than just a part of one project. Just when we had finished the integration with Gamygdala, for example, we immediately had to start with implementing a graphical visualizer, which was a completely different story altogether.

From an implementation point of view, I found the actual integration of the two frameworks to be the most difficult part. However, we managed to finish this right on time, mainly because everyone contributed to the thinking and design work.

All in all, I look back on it as a successful project. We managed to carry out all tasks that we had planned beforehand, and the final product met all important requirements.

8.2.2 Raoul

Coming into this project I was quite happy. I have always held a soft spot for video-games, as I grew up with them and I believe they influenced me greatly. That is why I was so excited that I got to work on a project that was involved in this topic.

I was also very excited to work with my fellow teammates. Jeroen and Selman I had already known prior to this project and I knew that I can expect only the best of work ethic and dedication from them. Umit also has left a distinctively positive impression on me the first time we have met and has lived up to expectations during the project.

The first few weeks were quite confusing for me. It seemed that every time we had something figured out, in the next meeting with the client it would turn out we had things thought out in the wrong direction. Nevertheless, with time I noticed we made less and less such mistakes. I was pleased with that.

That period was mostly during the research phase. During the implementation phase is where things started cooking. The hardest stage of the project were these first weeks of implementation, where we tried to figure out the inner workings of Entika and Gamygdala — but particularly Entika. At times I have gotten quite frustrated with the amount of code I had to go through in order to understand even the smallest of components.

It took us time, but eventually (I am glad to report) we have conquered Entika. Since then, things were much less stressed for me. The team and I have made extensive use of the issue-tracker and wiki services that Bitbucket offered which made our workflow go pretty smoothly. I have also enjoyed the fact that we kept (more or less) to our scrum approach. Holding those daily meetings was very helpful for me - as I have a tendency to lose track of what is going on.

I wish to also touch on the subject of the SIG review. I am not sure if that was expected, but out of nearly two thousand lines of code, we have gotten a feedback sheet no longer than a couple of paragraphs. I found that a bit lacking and simplistic (I refuse to believe our implementation was that good).

Near the end of the project it seemed we have returned to the stressful period again (in my opinion, at least). It turned out that Gamygdala has had some pretty nasty bugs hidden inside it, and I feared the presence of which would slow us down. Fortunately Mr. Broekens was able to help us to come a long way towards fixing those problems.

All in all, this project was a definite learning experience. Not only from a technical point of view, but also from a social one. I have learned a lot about the .Net framework (C#, WPF, XNA) and
even more about development methods, working in a team and interacting with clients.

8.2.3  Jeroen

Going into this project we were warned that this was an open project and it might give rise to a risk of trying to do too much. This seemed to be quite the way we were going at it at first, which had me worried a little. Especially after the first two meetings with the coaches, each of them separately, I felt that the project was very precarious, because the views that the coaches held seemed to diverge and we were kind of unsure which way to go.

Fortunately we reached more clarity after the meeting with both of them present, but unfortunately that was after quite some time had passed. In hindsight, it would seem the three weeks we were supposed to take to research instead of the regulation two weeks might have been more the result of delays in the readying of the tools we were supposed to use, as we got access to those only later.

Once everything was available to us we could actually start exploring and expanding, and it was quite a steep learning curve, with the need to learn both the new WPF concepts and the Entika concepts in tandem while at the same time working on our own new solution. The idea of pair programming was nice in principle, but in practice we both soon started working on our own parts, so it was more like a hierarchical distribution of the workload. That was fine with me to be honest because at first I felt like I was doing nothing while my partner was doing all the work, and the other day the roles were reversed and I suspect he might have felt the same way.

There were also other things that didn’t quite work as we had originally proposed, for instance the end-of-the-day meetings; they were seen as a spoiler for the next day’s Scrum meeting so they were soon abolished. Also the TDD approach was not always adhered to, particularly in the case of the GUI interaction code and the demonstration scenarios. It is always tempting to start small enough to give the illusion of being relieved of the need to test, but the code base almost always grows large enough for it pretty soon, as was quite evident in particular with the programming of the final simulation scenario.

The fact that we didn’t adhere too strictly to the rules we had set up beforehand kind of affirms to me something that I’ve seen going on in earlier projects that I’ve been a part of as well, namely that even if there are really nice principles stated at first and everything seems to look very nice and well organized in the beginning, the closer one gets to the end of the project, the dirtier the tricks get to keep things working properly, or at least well enough. In the words of the classic xkcd (349): as a project wears on, standards for success slip lower and lower.

On the one hand this is understandable, because something is better than nothing, but on the other hand it is also kind of regrettable that it is clearly very difficult to adhere to your own standards. The problem that leads to this phenomenon might be the fact that things do not always go according to plan, particularly when bugs show up that turn out to be hard to trace, and these things cause a certain delay, which means there’s no time to do everything according to the prespecified procedures. I suppose with TDD another problem could be that it is often tempting to start programming right away because you’ve got the idea of how you can transform your mental solution into code ready to be typed out, but of course the ‘TDD way’ to think about programming is to first figure out what you want the software to do, formulate that in tests and then write the software so that it passes the tests.

Ultimately I think it was a great project. We solved an interesting research problem, we were a great team and we had some good laughs, particularly once we started rendering the emotions that
Gamygdala generated and realizing that there was something off. It was also really nice to see the potential of Entika, although it took a while to get there because it is so poorly documented, and perhaps because the editor is kind of cumbersome.

### 8.2.4 Umit

Starting with the bachelors project was a very exciting event for me. Suddenly you have to employ everything you have learned in a single project, which needs to be finished in 10–12 weeks. This was a totally new experience and one where I was able to learn a lot from.

Right after the beginning I realized that it would be no easy task. Creating something from scratch is one thing, reading and understanding thousands of lines of code from other people as well as being able to understand what design intentions they had is another. On top of this, we had to extend existing code, while staying true to the original designs’ intentions.

Looking back, I realize how extremely important as well as really difficult the research phase is. Getting a good requirements analysis is no easy task and your entire project might as well depend on a correct requirements analysis.

Another lesson I learned is being able to make decisions without over thinking too much. Sometimes you just need to make quick decisions, especially when many decisions need to be taken and time is of essence.

I enjoyed working together with my team. They are a group of very disciplined and knowledgeable people, whom I learned a lot from. Working together is an important factor when having a team assignment. Programming in pairs, being able to ask questions and getting proper feedback is what makes your work increase in quality.
9 Conclusion

For this bachelor project we investigated the integration of a semantic world editor (Entika) and an emotion appraisal engine (Gamygdala). Our main goal was to use Gamygdala to generate emotions for entities defined in Entika.

We started our implementation by adding an extension to Entika to allow the definition of Gamygdala-related concepts. This brought us a step closer to enabling communication with Gamygdala. Then we proceeded with implementing an extension of the Entika engine to be able to keep track of all “emotion-related” aspects of a game world during simulation. We continued by building a bridging interface for the passing of data from Entika to Gamygdala. This enabled us to extract the necessary data from the simulation using the engine extension and input it to Gamygdala. In this way we were able to use Gamygdala to generate emotions for entities defined in Entika; this meant that the integration of the two libraries was successful.

Finally, we implemented two demonstration applications to visualize the functionality of the integrated application. Using a game scenario and a simulation scenario we assessed the plausibility of the generated emotions.

The main research questions that we had defined were:

1. What is the best way to integrate Entika and Gamygdala?
2. After integrating these tools, do NPCs display plausible emotional reactions under a variety of circumstances?

Our Entika extension and Entika-Gamygdala interface serve as answers to the first question. Our final product meets all requirements that were defined as must-haves and should-haves in section 4.

To answer our second question we have made use of our demonstration applications. We found that plausible emotional reactions were generated in many scenarios. The scenarios involving ‘state goals’ (where entities strive to maintain a state) rather than ‘achievement goals’ (where entities strive to realize a change) were the problematic ones, since Gamygdala was designed with the latter in mind. We therefore feel that additional value would be created if Gamygdala could be extended so as to be able to appraise emotions under a larger variety of circumstances; any research into the feasibility of this would certainly be useful.

Our final product serves as a proof of concept for the enrichment of semantic game worlds with emotional content. We wish our project coaches the best of luck in the application and extension of our integrated software.
A Original project description

In many computer games the Non Player Characters (NPCs) play an essential role in the gameplay. For example, in a Role Playing Game such as Skyrim, NPCs play semi-active roles in the storyline and some even have daily routines. In a social simulation game, such as The Sims, the characters in essence live their own lives and you as a gamer can influence what happens by giving orders to them or change their environment such as buying a novel piece of furniture or make them change jobs.

In The Sims, emotional reactions of NPCs to events play a major role, but this focus is quite rare in games, and there is no standard process or toolkit available to flexibly generate Emotional NPC behavior that is psychologically plausible and scales well to a large number of NPCs. In this project you will build and validate a system aimed at solving this limitation. For this goal, rather than starting from scratch, you will be able to integrate several advanced tools available, including:

- Entika[3][4], a semantic library and engine that enable the creation and use of semantic virtual worlds, in which objects have meaning and functions, assume roles, provide services and serve some purpose(s) while interacting with other objects and NPCs;
- a semantic crowd editor[4], that allows for the definition of complex demographics of NPCs;
- Gamygdala[7], an emotion simulation engine that is computationally light-weight and can simulate emotions based on the meaning of events for individual NPCs, if the NPC has defined goals and the events are annotated such that it is clear which event influence what goal;
- An emotional expression engine that enables 3D rendering of emotions on faces, which principles have been already described[2].

The characteristics of the final system will be such that given a semantic game world, in which objects have a particular meaning to each NPC, we will be able to simulate NPCs’ psychologically plausible emotional reactions to any relevant events and object interactions. In addition, those emotional reactions will use the emotional expression engine to visually render them on a NPC’s face. For example, if a thirsty NPC approaches a bar but has no cash, it’d likely become happy by finding an ATM on the way there, because the ATM enhances its chances of achieving its goal of drinking a beer.

The ultimate goal is to have a complete development suite for describing, ‘running’ and monitoring the behavior of NPCs in any environment. Among other tasks, this will involve

1. developing an interface between Entika and Gamygdala, such that relations and services offered by object classes in Entika can be used as basis for the events and goals needed by Gamygdala to simulate emotions; and
2. finding a simple way of coupling the resulting simulated emotions to the expression engine.

Finally, you will have to make an evaluation of the results achieved, by means of a proper user test on how plausible the simulated emotions are perceived. The focus during testing is explicitly NOT on the plausibility of the graphical rendering of the emotions, but on the plausibility of those emotions as related to the situations that triggered them.

It is expected that the demonstrator system and its results achieved at the end of this project will be used in a scientific publication.
For more information contact: Joost Broekens (joost.broekens@gmail.com), or Rafael Bidarra (R.Bidarra@tudelft.nl).
B Research Report

B.1 Introduction

In order to answer our main question *What is the best way to interface Entika and Gamygdala?* we have formulated several subquestions:

1. What is Entika?
2. What is Gamygdala?
3. What are the options for integration?
4. How can the interface be tested?
5. What are the options for a user interface?
6. How can the added functionality be visualized?

The ensuing sections have been devoted to answering these questions, starting with the first two. In the next section we will discuss the Entika-Gamygdala interface, which is the main focus of the project. We will present several alternative options for the interface and evaluate them. Then we will focus on the testing of the interface and the test cases to be used for this purpose.

The second important task in the project is the implementation of a graphical user interface for the combined configuration of Entika and Gamygdala, which is discussed in the next section. Given entities and the services provided by them in Entika, this interface would allow the user to create goals for entities and create beliefs in which the services can be included. The requirements of this GUI will depend on which option is chosen for the Entika-Gamygdala interface.

Last comes the third and final important implementation task in the project: the coupling of the Entika-Gamygdala combination to a graphical environment for testing and demo purposes. Initially a 2D environment will be used as a test case, and in the final phase of the project the application will be linked to a 3D environment.

B.2 Entika and Gamygdala

We will now describe shortly what the names Entika and Gamygdala mean. They refer to software libraries, each with its own purpose.

Entika is an engine that manages the semantics of objects in a given simulated world. To do this, it employs a hierarchical model. At the root are entities, which are things with a distinct existence. This is separated into physical entities, matter and abstract entities. The matter is further split up into elements, substances, mixtures and compounds. Among physical entities, spaces and objects are distinguished. Furthermore, relationships can be defined among various types of entities. Finally, services are provided, which basically give entities the ability to perform actions. Conditions can be involved in this, as well as artifacts, contexts and targets. Actions also have effects, which are split up as reaction, change, creation/deletion, transfer, transformation, relationship establishment.

Gamygdala is what is officially known as an emotion appraisal engine. This means that it takes as inputs a set of events, and based on predefined goals and beliefs, it generates an emotional state
befitting those events, goals and beliefs. The engine works using ‘brains’, and each brain has an emotional state and is linked to a specific agent with goals and beliefs.

**B.3 Entika-Gamygdala interface**

The main focus of the *Virtual humans in games* project is to implement an interface between Entika and Gamygdala, allowing the two frameworks to communicate with each other.

The purpose of this interface is as follows:

Within Entika a semantic world can be defined. When a simulation is run based on this definition, events are triggered in which entities can be involved. The other aspects that are relevant for our interface are the possibilities of defining entities and goals. However, as of yet there is no functionality to link goals to entities.

Within Gamygdala, goals can be defined for entities. Gamygdala also allows the definition of beliefs, which basically indicate which event has what type of effect on which goal.

A simplified version of this current situation, in which an interface is absent, is depicted in figure 8. Both Entika and Gamygdala contain more than what is shown, but only the for the interface relevant aspects are included in the figure.

![Figure 8: Current Entika and Gamygdala setup](image)

The interface to be implemented between Entika and Gamygdala will combine these two frameworks. When events are triggered in Entika and the involved entities are determined, the interface can pass this information to Gamygdala. Gamygdala can then, based on the beliefs that have been defined in Gamygdala, determine which goals of the given entities are affected in what way by the given events, after which the appropriate emotion for the entities can be computed.

The main challenge with implementing this interface is at the transition of relevant information from Entika to Gamygdala. We will now present four options we have devised to solve this challenge. Afterwards we will compare them and present our final conclusion.
B.3.1 Implementation options

1. **Extending Entika functionality** This option involves extending the existing Entika functionality so that it can be used as input for Gamygdala. Two parts of Entika would have to be extended for this option: goals and events.

Goals already exist in Entika but cannot be linked to an entity. They also cannot be given a utility value, which is required for the calculations in Gamygdala. Therefore, for the goals part, this option would involve extending the existing functionality so that goals can be added to entities and can be given utility values.

Beliefs, on the other hand, do not exist in Entika at all. However, in Gamygdala beliefs are merely annotated events, and not many attributes are required to transform an event into a belief. These attributes are the list of goals affected by the event and the likelihood and valence of the belief. This means that, in order to make events usable as input for Gamygdala, it would be sufficient to add these attributes to the already existing Event class in Entika.

This option is illustrated in figure 9.

![Figure 9: Transforming Entika data into Gamygdala suitable input (option 1)](image)

**Pros:**

- Especially because the Goal and Event classes and related functionality is so similar to what is used in Gamygdala, this option involves the least amount of development work.

- No separate editor needed. Since both the semantic and emotional data can be configured in the same application, developers can design the whole game world within one environment.

**Cons:**

- An extension implemented according to this option would work only with a specific version of Gamygdala. Since this option involves injecting a part of the Gamygdala class structure into Entika, any structural changes in Gamygdala would have consequences for the Entika implementation.
2. **Using Gamygdala functionality in Entika** This means that, within Entika, we make it possible to add goals to entities and to define beliefs (annotated events). This could be achieved by importing the ‘goals and beliefs’ functionality from Gamygdala and then using it in the Entika editor.

After the configuration has been set up in this way, the world simulation can be started in Entika. Whenever events are triggered, the goals of the NPCs involved in the event, the NPCs themselves and the beliefs in which the triggered event occurs can be passed to Gamygdala. The emotion simulation in Gamygdala can then be run based on this input. Figure 10 illustrates this option.

![Diagram](image)

**Figure 10: Using Gamygdala functionality in Entika (option 2)**

**Pros:**
- Only one editor would be sufficient to create a semantic game world with emotional data. There would be no need to set up the semantics in one view, the emotions in a different one and then look for a way to link these.

**Cons:**
- It sacrifices modularity, as it would be impossible to set up the emotions part of the world separately (outside of Entika) and then link it to the world that was defined in Entika.
- Code duplication, since beliefs and goals would be a part of two applications.
3. Using Entika data in Gamygdala This means that, when the user is defining annotations for events in Gamygdala, he can make use of events that have been defined in Entika.

What this could look like is as follows: the user first sets up the semantic world in Entika. Among other things, this leads to the creation of a collection of entities and a collection of possible events, based on the entities and the services provided by them.

When the user then starts the work in Gamygdala, these two collections can be imported from Entika. Goals can be defined for the imported entities and when defining beliefs, events that have been imported can be used. To make the mapping between Entika and Gamygdala complete, a reference to the original Entika object is included in these entities and events. After this, when the Entika simulator is run, the affected beliefs and NPCs can be easily identified in Gamygdala using these references.

This option is visualized in figure 11.

![Figure 11: Using Entika data in Gamygdala (option 3)](image)

Pros:

- Clear way of setting up Gamygdala. With the data imported from Entika visible, users would have control over the mapping.

- Does promote modularity. Both environments can still be used on their own. The coupling is not detrimental to the application if removed.

Cons:

- Setting up the beliefs can potentially be quite time consuming. If the semantic world defined within Entika consists of a large number of entities and services, finding the relevant ones when working in Gamygdala can be difficult.
4. **Mapping entities and events dynamically** The fourth option is to implement a bridging interface that, during a simulation run, dynamically maps events triggered in Entika to events defined within beliefs in Gamygdala. For this, it would be necessary to define events in both applications according to a common format.

For example, in Entika events already consist of a subject (an entity), a verb (the provided service) and possibly an object (also an entity). In Gamygdala, however, events are identified with just a string. If in Gamygdala a structured format similar to that in Entika were to be used, then during a simulation events triggered in Entika could be mapped to those in Gamygdala without the developers having to set this mapping up manually (as was the case with the second alternative presented above).

The impact of this option on Gamygdala would be that event names would have to adhere to a strict structure, wherein the subject, the action and the (optional) object would have to be specified.

This option is illustrated in figure 11.

**Figure 12: Mapping entities and events dynamically (option 4)**

**Pros:**

- Less effort for developers; since the matching between Entika and Gamygdala would happen automatically, there would be no need anymore to configure this manually.

**Cons:**

- Complex implementation and design challenges; bugs might be difficult to notice and resolve because of the dynamic and ‘out-of-sight’ matching during a simulation run.
- Need for defining exceptions. Since by default the interface will be trying to match
everything, separate functionality would be needed to indicate events that should be excluded from emotion simulation.

B.3.2 Discussion

We first evaluated these four options within our project team and then discussed it with our project coaches.

We discarded the third option (Mapping entities and events dynamically) because it involves the creation of a certain degree of unwanted dependency. This is because this option imposes the constraint of having to conform to a format dictated by another application (Entika) within the application the user is working in (Gamygdala). Events in Entika contain many attributes (actor, target, action, effect, context, time, etc.) while a user working in Gamygdala might want to specify only a short name. A user might also want to use different subsets of these attributes while defining different beliefs (omitting the context for one event, omitting the target and time for another event), which would be problematic in the presence of a common format that has to be adhered to.

We considered the second option (Using Gamygdala functionality in Entika) to be undesirable as well. Its one benefit is that no mapping between applications needs to be implemented anymore, since events in Entika would be linked to beliefs that are also in Entika.

However, the code duplication aspect of this approach is obvious; beliefs and goals would be present in both applications while belonging to only one (Gamygdala). The first option (Extending Entika functionality) we had eliminated for a similar reason, since this approach also involves copying functionality from one application into another.

We had therefore decided to implement the interface according to the third option (Using Entika data in Gamygdala). This approach would allow different development teams to simultaneously work with the two applications using different naming conventions, without either of them needing to know something about the structure or format used in the other. This also means that eventual changes in class structures in Entika would have no effect on Gamygdala and there would be no need to migrate existing mappings.

However, in our meeting with the project coaches a strong preference for the first option was expressed. We understood that the interface between Entika and Gamygdala needed to work only for the given versions of both applications; the fact that after changes to either application the interface would not be correct anymore was considered irrelevant. Therefore, after this meeting, we abandoned our initial choice and decided to implement the interface according to the first option.

B.4 Tests

To verify that the software created actually does simulate emotions reliably and plausibly, it is necessary to test this. Clearly, it is not necessary to test Gamygdala or Entika functionality, since these libraries are provided beforehand and can therefore be assumed to work. What needs to be tested is the link between the two libraries. To do this, two main functions must be tested. Firstly, the agents need to respond in a psychologically plausible way to events in their environment, and secondly, their emotional state must change naturally (e.g. it must not retain the same emotions indefinitely while adding new ones). Furthermore, the efficiency must be tested using a stress test.
B.4.1 Plausible reactions

The way we propose to test whether the agents respond plausibly is by recruiting a few people and showing them a demonstration of the system. After this demonstration they are required to fill in a form in which they can give feedback on the level of realism in the simulated emotions.

The way the demonstration will be set up is as follows. There will be several predefined situations in which certain emotions will have to occur. Depending on the number of people recruited, they can be given a random situation, or they will be asked to rate multiple situations. The situations are chosen such that all emotions will occur in at least some situation. Finally, this set of simulations is set up to take only a short time. We chose to test the emotional state separately, because this will take longer and recruits might want to opt out of a longer session.

Emotions are split in two groups in Gamygdala, internal emotions and social emotions. For the former we have derived the following simple tests, with the expected emotion in parentheses:

- An agent looking for a drink finds a glass of water. (Joy)
- An agent looking for a drink finds out there is a possibility to obtain soda from a vending machine. (Hope)
- An agent looking to avoid cacti finds out there are several cacti in its vicinity. (Fear)
- An agent that does not want to get killed, does. (Distress)
- An agent expecting to receive a soda from the vending machine, does. (Satisfaction)
- An agent that is worrying about hitting a cactus, does. (FearsConfirmed)
- An agent expecting to receive a soda from the vending machine is presented with an error telling it there will be no more sodas. (Disappointment)
- An agent traversing a hot floor and expecting to get burnt, doesn’t feel anything and emerges unharmed. (Relief)

For social emotions we have made the following tests, where A and B stand for distinct agents each time and the emotions are experienced by A towards B:

- A wants to be alone and finds out that B has entered the room. (Anger)
- A likes B, B wants to get a drink and A consumes the last drink it knows about. (Guilt)
- A wants to find some food and B hands A a plate of spaghetti. (Gratitude)
- A likes B, B wants to obtain a stethoscope and A hands B a stethoscope. (Gratification)
- A likes B, B wants to get an ice-cream cone and does. (HappyFor)
- A likes B and B wants to play a game but there are no games in the environment that A knows of. (Pity)
- A dislikes B, B does not want to die and B does. (Gloating)
- A dislikes B, B wants to get a beer and B does. (Resentment)

Note that like is not tested. This is because it is actually a simple difference between anger and gratitude toward a certain NPC, which means if those emotions work properly, Like can be assumed to as well.
B.4.2 Plausible state

Once it is established that the right emotions occur at the right times, it is necessary to verify that the emotional state of the agents develops in a natural way as well. This is more difficult to test, because there are clearly many ways the emotional state of an agent can develop. We therefore propose to set up a few tests with a fixed course of events, as with the previous tests, which ought to be easily recognizable for the participants. This way they can more readily relate to what the agents are going through, and report if the agents actually turn out to be developing their emotional state erratically. It is important that the particular sequence of events that caused the erratic development be stored, so that it can be used in debugging later on.

Of course, while there is a longer session with multiple events running anyway, there is another test that may be performed, namely that the right emotions are still exhibited even after the emotional state has been “polluted” with other emotions. This is different from the first test in that a different emotion might arise if something undesirable happens and something desirable happens, depending on the order in which these things happen. This is just one example, although of course it is not feasible to test all possible combinations of events; the likelihood of occurrence in the real world can be used to select the most important situations.

These things being premised, we propose the following testing scenarios. The scenarios are to be rated by the perceived realism of the simulation, so we choose not to include any fixed expected results. Again, A and B are NPCs.

- A wants to have some fun by playing a board game. A finds out there is a board game in its environment. B enters the room and invites A to play this board game. A wins the game.
- A is in an outside environment and wants to stay dry. It starts to rain. B also pushes A into a body of water. B subsequently slips and also falls into the water.
- A does not like B and A does not like touching plants. Both B and plants are present in the environment. A tries to walk away from B and walks into a plant. B subsequently walks towards A in order to help.
- A wants to get a drink and starts moving toward the vending machine. Meanwhile B takes out the power. A finds out that the machine doesn’t seem to be working.
- A likes B and B is struggling to survive. A finds B like this and uses a knife on the thing that’s causing B trouble. B subsequently snatches the knife from A and starts stabbing A.

B.4.3 Stress test

In order to verify that the added link between Gamygdala and Entika does not add excessive complexity, stress tests will be performed. First the most hardest original tests mentioned in the papers for both libraries will be performed. Second, a test with a moderate number of agents and a large number of beliefs, goals and semantic objects and services will be performed. If the system can cope with all these situations, it will be deemed to be efficient enough.

B.5 The graphical user interface

One of the most important aspects of the final product would be its user interface. The application we develop need not only provide a robust and yet flexible software library but also an environment
in which the user could utilize all of the available tools effortlessly. When it comes to the user interface, we had two options: 1) Make our own stand-alone application that will work in tandem with Entika or 2) Extend the editor already present in the Game Edition of Entika.

Because of time constraints and the wishes of the client, we have opted for the second option.

What follows is the description of our research with regard to the interface along three main facets: Approach, design and implementation.

B.5.1 Approach

Coming into this project we realize that our target user is a game developer. Bearing that in mind, we aim for our product to be as attractive as possible to game developers. Therefore, we have identified the following three requirements we must satisfy: Ease-of-use, functionality and flexibility.

Ease of use Game development is an iterative process. The interface should accommodate for this kind of working environment. It should be fast, intuitive and to the point. The design should make it so the user spends minimal time in the design environment and maximal time testing the created designs.

Functionality The interface should reveal as much of the underlying library’s functionality as possible. We realise that we may not be able to expose everything to the user, as that may lead to a conflict with the ease-of-use condition. Below we describe our specifications of the interface following the MoSCoW (must-, should-, could- and won’t-have) approach.

Must have

- Design:
  - Add entities, goals, and events to the design.
  - Link entities to goals.
  - Link events to goals.

Should have

- Design:
  - Specify likelihood on beliefs.
  - Specify utility on goals.
  - Specify valence on event-goal links.

Could have

- Design:
  - Link events to property-value change. Example: event triggers change in goal utility value.

Won’t have
A different style of GUI other than the one already present in the Entika editor.

**Flexibility** This requirement is based on two desired traits of the final product: Depth and scalability.

**Depth** We believe that in order to satisfy both the ease-of-use as well as the functionality requirements our interface should strive to imitate a “bigger than the sum of its parts” view. The developer should be able to insert intricacies to the semantical world even in the simplest of designs. At the same time, if the developer wishes to make a truly basic design, that should also be possible - the developer should not be forced to bother with details he is not interested in.

**Scalability** From Pack-Man to SimCity, the interface should reflect the underlying software’s capabilities to build small as well as large projects with the same relative effort.

**B.5.2 Design**

The approach described in the previous section implies the incorporation of the following guidelines in our design:

- Place components where they are intuitively expected.
- Make use of existing features of the editor (color-coding, expanders, search fields, etc.)

Below are some mock-ups demonstrating an implementation of the guidelines mentioned above.

![Mock-up](image)

Figure 13: The whole of our extension will be hosted under a single tab next to the regular tabs in the Entika editor.
Figure 14: Once in the main Emotion tab, the user can navigate to any of our extension’s objects by type: Animate objects for objects that are capable of being hooked to Gamygdala during simulation, Utility Goals to represent Gamygdala goals and Event Filters to define groups of events.
Figure 15: Animate objects can have utility goals added to them. They can inherit utility goals from their parents and they have the ability to override the utility property of those goals.

Figure 16: Utility goals have a utility property corresponding to Gamygdala’s property of the same name. They also contain a collection of event-filters (personal as well as inherited). During simulation, events that qualify all the filters will have an effect on all animate objects having the selected goal.
Figure 17: Event filter properties can be edited via their respective expanders: The action constraints expander (depicted above) and the actor, artifact and target expanders (seen in figure 18).
Figure 18: The actor, artifact and target expanders all share the same template seen above. Each of them imposes constraint on the actor, artifact and target properties of an event (as defined in Entika) respectively. Multiple constraints can be set: Fixed ID's, hierarchical, and the inclusion/exclusion of groups and/or relationships.

### B.6 Graphical environment

In the final two phases of the project, the combined Entika-Gamygdala framework will be coupled to a graphical environment. Initially this will be a 2D environment; this coupling will also serve as a test case for the Entika-Gamygdala interface that will have been developed. In the last phase of the project the framework will be coupled to a 3D environment. This coupling will also serve as a demo tool, since the working demo in the final presentation will be prepared in this graphical environment.

Several 2D/3D game engines have come to mind. The possibilities of Vizard, UnrealEngine and Unity3d have been explored. Bidarra et al. suggested the framework which made use of Entika had a Visualizer which was easily replaceable. The description of the visualizer is: "The visualizer is not strictly a component in the simulation system, since even without it the simulation will work. The simulator gives access to the semantic virtual world as well as the agents controlling the entities in that environment, which is all that is required to visualize what is happening in the simulation." The suggested game engines are very powerful tools which will be suited for testing purposes. There is a steep learning curve though, which needs to be taken into account.
From discussions with the project contacts we have understood that code already exists for both 2D and 3D purposes. A 2D environment has already been developed for demoing Entika and a 3D environment based on XNA also exists. Most of the work in this phase will therefore be directed towards reuse of implemented software, making existing functionality fit our implementation and preparing a demo.
C Feedback SIG

C.1 First review

De code van het systeem scoort 4 sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code bovengemiddeld onderhoudbaar is. De hoogste score is niet behaald door een lagere score voor Duplication.

Allereerst complimenten voor jullie code-indeling. De componenten zijn duidelijk in de directorystructuur te herkennen, wat het voor ons ook makkelijker maakt om te begrijpen hoe de code in elkaar zit. Nog een kleine suggestie: de meeste systemen die wij zien plaatsen de testcode binnen een component, terwijl jullie voor de componenten Aap en Noot ook AapTest en NootTest toevoegen. Het is natuurlijk niet de bedoeling om productie- en testcode door elkaar te zetten, maar je kunt dit onderscheid ook binnen een component (in plaats van erbuiten) maken.

Voor Duplication wordt er gekeken naar het percentage van de code welke redundant is, oftewel de code die meerdere keren in het systeem voorkomt en in principe verwijderd zou kunnen worden. Vanuit het oogpunt van onderhoudbaarheid is het wenselijk om een laag percentage redundantie te hebben omdat aanpassingen aan deze stukken code doorgaans op meerdere plaatsen moet gebeuren. In jullie geval zit de duplicatie vooral in de *EventFilterComponent.cs classes. In plaats van jezelf te herhalen zouden jullie de gedeelde code ook kunnen verplaatsen naar een gedeelde base class. Dat maakt het aanpassen van deze code ook makkelijker, want je hoeft dezelfde aanpassing dan niet drie keer te doen.

Over het algemeen scoort de code bovengemiddeld, hopelijk lukt het om dit niveau te behouden tijdens de rest van de ontwikkelfase. De aanwezigheid van test-code is in ieder geval veelbelovend, hopelijk zal het volume van de test-code ook groeien op het moment dat er nieuwe functionaliteit toegevoegd wordt.
C.2 Sprint summaries

There were a total of 8 sprints during the project. We will now list the results accomplished by the end of each one:

**Monday 12 – Sunday 18** The basic workspace (projects etc) was set up and the first elements of the extension of the editor were created: goals with a utility property and event filters. Unit tests were also made for these objects.

**Monday 19 – Sunday 25** The basic objects were implemented and various extensions to the editor were introduced: a tab was added for event filter definition, and one was added for the next extension, animate objects, and a way to upgrade projects of older versions. Finally, documentation was produced for the new extensions.

**Monday 26 – Sunday 01** Unit tests were added to test the utility property of goals. Furthermore, an expander was added to the animate object tab to enable the user to link goals to animate objects. Lastly, a few bugs were fixed.

**Monday 02 – Sunday 08** A few more bugs were fixed and small enhancements were made. The event filter definition tab was nearly done and a likelihood field was added. A start was made with designing some scenarios for the final demo. The unit tests were refactored to use the repository pattern and documentation for this pattern was added. Several questions of how to work with the semantics engine were addressed, which soon led to a basic implementation of an extension of the engine, along with unit tests.

**Tuesday 10 – Sunday 15** Implementation of the semantics engine counterparts of the new objects, and unit tests for them were added. The filter definition tab was completed with the addition of the last filter constraints. The interface to Gamygdala and its tests were made. Furthermore, a choice was made for a visualizer, namely XNA. Also, a few more bugs were found and fixed. Finally, the code was cleaned up and submitted to the SIG for the first time.

**Monday 16 – Sunday 22** Coach Broekens’ AffectButton was ported to C#. A bug that took a long time to trace was fixed, along with two other small bugs. Feedback on the code sent to SIG was received and used to improve some of the code.

**Monday 23 – Sunday 29** Implementation of the game and simulation demo-scenarios was started. A draft of the paper was also written. Another bug and a small issue were also fixed.

**Monday 30 – Sunday 06** The implementation of the scenarios was continued and the feedback received on the paper was processed. Furthermore, the report was completed and sent off.
C.3 Meetings

This section outlines the results and findings obtained from the various meetings that have been held with the coaches. The meetings are ordered chronologically.

2014-04-22

At this meeting the whole team was present but only coach Broekens was available. The meeting was meant as a kind of kickoff for the project, to give an overview of exactly which software was involved and what the basic idea would be behind the extension that was to be made.

After this meeting, the project was looking as follows. There were going to be three general phases: first a basic integration had to be constructed between Entika and Gamygdala, then focus would shift toward making a basic simulation, followed by a better, 3D solution to realistically render the emotions that Gamygdala calculates.

More concretely, a project plan had to be made, and in it the three phases had to be elaborated on. Other things to put in the plan was how the quality of the product would be assured, i.e. which criteria would be used to judge the quality of the software. Other than that some basic points about the approach followed, the parties involved and the organization of the project.

2014-04-25

This meeting, like the previous one, was during the research phase, and the main interest here was what kind of visualizer was supposed to be used in the end. After some discussion it was clear that there were not that many options for a 3D visualizer, and that a 2D visualizer ought to be relatively easy to implement. It was stressed, however, that although the underlying engine and connection that was to be made was not irrelevant, the key was to be able to convincingly demonstrate that the NPCs were actually exhibiting psychologically plausible emotions as a result of the addition of the emotion appraisal engine.

2014-05-01

This meeting was attended by coach Bidarra, since coach Broekens was away. The meeting was held to hear this coach’s side of the story. It turned out that he had a fairly different outlook on things. One of the things he stressed was that it was better to reuse existing code than to reinvent the wheel. It was also quite clear that he came at things from a scientific perspective, whereas Broekens had seemed to be rather more about the commercial side of things.

2014-05-12

This meeting was held with both coaches. There were several points to be discussed. There was a demo of the envisioned extension, there was a discussion on some of the tools to be used and some questions were answered.

The design of the extension was mostly scrapped in favor of building an extension directly into the existing editor, since there was no concern about the dependence this would cause. There was no feedback to speak of about the research report, possibly because the coaches thought it was just a
draft version. The question of whether Entika was to be influenced by Gamygdala was answered with a negative. Finally, some logistical issues were put to rest.

2014-05-22

Another meeting with both clients present, this time to discuss some small issues. The code of the new product must remain separated from that of the existing editor. It was stressed again that it is necessary to start thinking about scenarios to demonstrate both that the editor is capable of handling nontrivial use cases and that the resulting simulations will show the requisite psychologically plausible behavior. Finally, the request for more documentation was answered with a referral to a master’s thesis about a part of the existing software.

2014-06-05

Coach Broekens was the only one available. The new features of the editor were shown and he had some comments about certain complicated scenarios that ought to be made easier to construct in the editor. One notable thing was the idea to enable self-reference, for instance in filter constraints. This was an interesting new feature and it was taken up in the designs shortly after the meeting.

2014-06-12

This time coach Broekens was unavailable.

2014-06-19

Both clients were present to discuss the new demos and the basic layout for the paper that had been produced. The demos were well received; especially coach Broekens was pleased to find out how well the integration between Entika and Gamygdala was working. The basic visualizer that had been produced was also suitable enough for the purposes at hand.

The main concerns were with the scenarios and the paper. The scenarios were focusing too much on the interactions between NPCs and not enough on the influence of the semantic world on the NPCs’ emotions. The paper was too much focused on the software. It should talk in a more abstract way about the new approach developed in the project and only as a proof of concept can the software be mentioned.

2014-06-26

The meeting was held with both coaches in order to get some feedback on the first real draft of the paper, as well as to demonstrate some of the work that had been finished. The feedback took up almost the entire meeting, which meant that the agenda had to be shrunk somewhat.

The main feedback was as follows. The structure and style has improved, but the introduction section needs to focus on introducing the problem, not on what others have said about it, which is what the related research section is for. The latter could use some extra references, because it is kind of embarrassing how often the authors currently quote themselves. The basic approach section
needs to explain the ideas behind the solution, not the concrete implementation or anything like that. It is important in this section to really ferret out the root cause of the problem. For instance, there was talk about how it is hard for developers to deal with growing worlds because there were no tools, but the reason for there being no tools is that it is really hard to make good tools to solve this problem. The methodology section goes into too much detail; it merely needs to explain shortly how the basic approach was embodied. Finally, the results section could be expanded, which is understandable because the scenarios have not yet been fully implemented.

There was also a short demonstration of the game scenario. The meeting ended in a discussion about an issue that had arisen with Gamygdala, which could not be solved then and there. It was looked into in more detail the next afternoon and it turned out to be quite significant; see also the review (section 8.1).
In this section we report on some bugs we have found in Entika and Gamygdala, starting with Entika.

- The copy constructor(s) in the semantic model don’t work because `IdHolder` always sets the id to 0 first from the constructor that is called (i.e. `base()`). As was mentioned in the recommendations (section 7.4), this may have survived because they are never used.

- In the editor, if a property (e.g. attribute) is added to a child first, then also to a parent, it occurs in the child in both the personal and inherited boxes. This may not be a bug strictly speaking, but it is not said that this is desirable behavior.

- In the editor, if a tangible object A is parent of a tangible object B and a connection to tangible object C is added in A, it appears twice in B and cannot be removed from A.

In Gamygdala we found the bug that caused erroneous emotions to be produced to be caused by some of the relevant checks in the `Recalculate` method using the belief likelihood instead of the goal likelihood.
References


