Artificial Insemination with OpenEngine

Christian P. V. Christoffersen
cpvc@openengine.dk

Carsten Nørby
tic@openengine.dk

Ian Zerny
zerny@openengine.dk

Dept. of Computer Science
Aarhus University
Denmark

April 18, 2008

Abstract

This report summarizes the process of developing an interactive illustration of artificial insemination within the OpenEngine framework. The report describes our methods, observations and reflections while developing a specific product, intended for actual use. The report will give the necessary knowledge to start maintaining the actual product. However, the reader should be able to use this report in the context of evaluating the framework or developing their own product within it.

Contents

List of Figures

1 Project Description

The goal of the project was to build an application that, at an educational level, would give viewers an impression of how artificial insemination is done. We have named this application the Inseminator throughout this report. The application is built around the story of a couple that go through the procedure of artificial insemination. The audience is taken through this process step by step. Each step is communicated to the audience through pictures, movies and simulations By simulation, we mean that the audience is interacting with the application, and performing tasks that are part of the insemination process. The simulations are not in any way representive of the actual process, but try to be believable and give the audience an impression of what the process is like in reality.

The application was developed for use by the Steno Museum and the project parties included The Steno Institute, The Department of Computer Science (DAIMI) and Institute of Information and Media Studies (IMV) at The University of Aarhus.

The following list enumerates the people that were involved in the project.

- Mette Kia Krebbe Meyer from the Steno Institute was in charge of building the special exhibition: "Ægløsninger" that this project was part of.
- Ole Caprani from DAIMI was the main coordinator of the project. He is also part of the "Natur i teltet" event where the project application was displayed and tested.
- Jesper Mosegaard og Karsten Noe from IMV provided the Mediotic Physics Engine, and helped integrate it into the OpenEngine system.
- Iben West created all media in the form of pictures and videos.
- Christian Esbo Agergaard provided models and textures for all the simulations.

The project process started with some small technical prototypes and ideas for what would be interesting to display at the museum as part of their exhibition. Several meetings helped to define the goals of the project and what needed to be developed. The process is ruffly sketched below.

1. 2-3 meetings with all involved parties. Development of the story board.
2. Fieldwork at the Maigaard clinic
3. Story board reevaluation and revision.
4. User test of prototype in "Natur i teltet"
5. Finalized story board.
6. Lab testing of the application.
7. Final approval of the application.
8. On-site setup of hardware and software.

The system was developed at DAIMI, close to all involved participants. DAIMI supplied the necessary development environment and tools. The final story board can be found on the CD and is the basis for the implementation.

1.1 Inseminator Stages

Each individual stage has been developed in accordance with the storyboard. However, a few extra video stages were added after the final revision. The following is a complete description, in chronological order, of all the stages:

Intro Picture.
A static start up picture.

Intro State.
Video with scroll text introducing the user to the story of Morten and Mette who wants to get pregnant.

Donate Text.
Video with additional explanation of the first part of the procedure.

Donate.
Video showing next part of the story.

Hit The Little Guy Text 1.
Video describing first part of the procedure where the spermatozoa is hit.

Hit The Little Guy Text 2.
Video describing concrete what to do in the upcoming simulator.

Hit The Little Guy.
Simulator where the user are supposed to hit one of the spermatozoa. When the spermatozoa i hit correctly it will lie still and the procedure is successfully completed.

Hit The Little Guy Success.
Video congratulating on success.

Select The Little Guy Text.
Video describing how to suck the chosen spermatozoa into the needle.

Select The Little Guy.
Simulator where the user must select the marked spermatozoa and suck it up into the needle. Selecting the wrong spermatozoa is an error and the stage must be repeated.

Select The Little Guy Success.
Video confirming the successful procedure.

Turn The Egg Text.
Video text describing how the egg should be prepared.

Turn The Egg.
Simulator where the user must turn the egg into the correct position.

Turn The Egg Success.
Video text saying it went well.

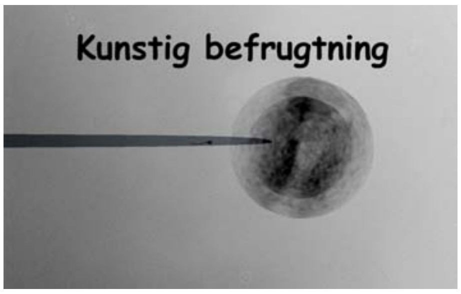
Insemination Text.
Video text describing how to do the actual insemination.

Insemination.
Simulator where the user must perform the actual insemination carefully. Releasing the spermatozoa outside the egg or between the two spheres is an error requiring the user to repeat the stage.

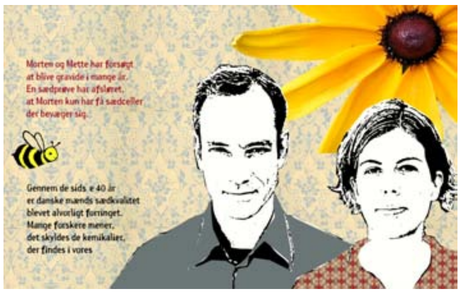
Insemination Success.
Video text congratulating on success.

Outro.
Video ending the story of Morten and Mette.

Figure 1: Screenshots of the final Inseminator application.

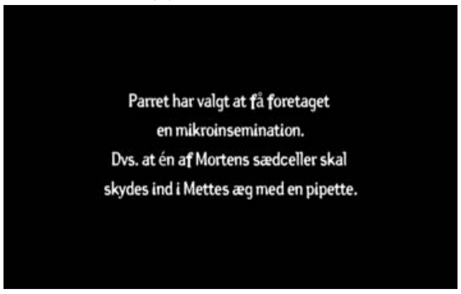


(a) Intro Picture

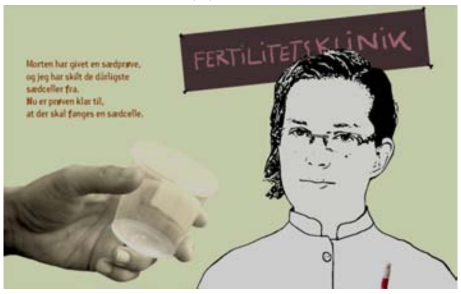

Morten og Mette har forsøgt
at blive gravide i mange år.
En sædprøve har afsløret,
at Morten kun har få sædceller
der bevæger sig.

Gennem de sidste 40 år
er danske mænds sædkvalitet
blevet alvorligt forringet.
Mange forskere mener,
det skyldes de kemikalier,
der findes i vores

(b) Intro

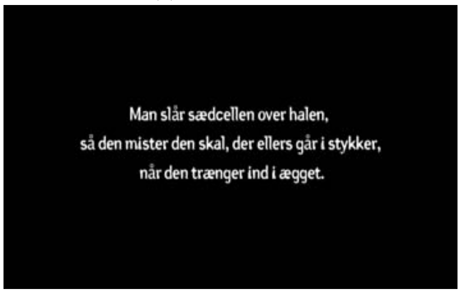

Parret har valgt at få foretaget
en mikroinsemination.
Dvs. at én af Mortens sædceller skal
skydes ind i Mettes æg med en pipette.

(c) Donate Text

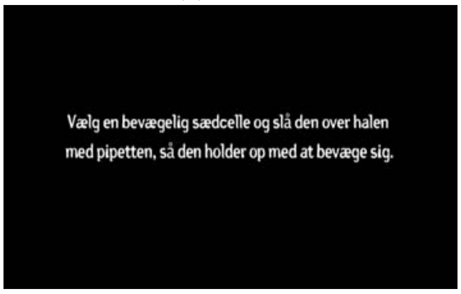


Morten har givet en sædprøve,
og jeg har skilt de dårligste
sædceller fra.
Nu er prøven klar til,
at der skal fanges en sædcelle.

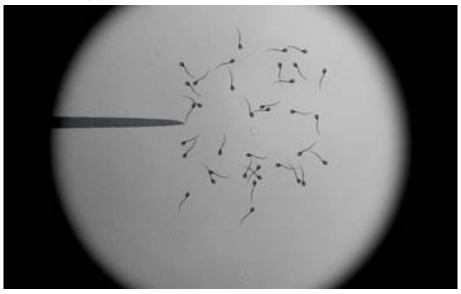
(d) Donate


Man slår sædcellen over halen,
så den mister den skal, der ellers går i stykker,
når den trænger ind i ægget.

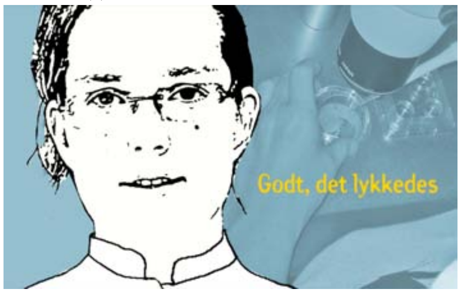
(e) Hit The Little Guy Text 1


Vælg en bevægelig sædcelle og slå den over halen
med pipetten, så den holder op med at bevæge sig.

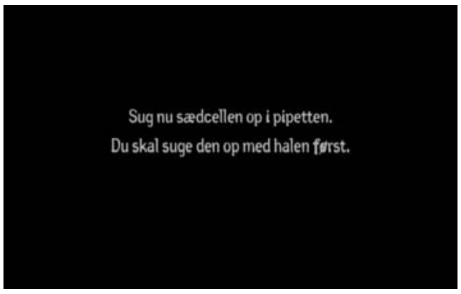
(f) Hit The Little Guy Text 2



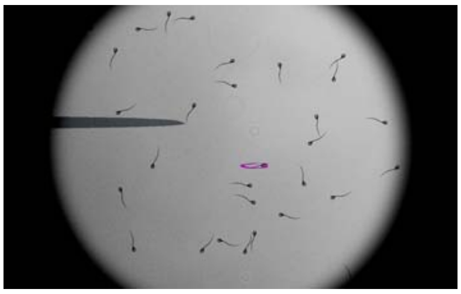
(g) Hit The Little Guy

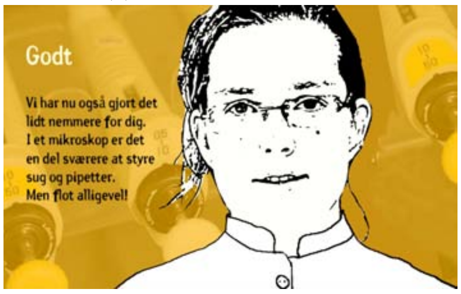


(h) Hit The Little Guy Success


Sug nu sædcellen op i pipetten.
Du skal suge den op med halen først.

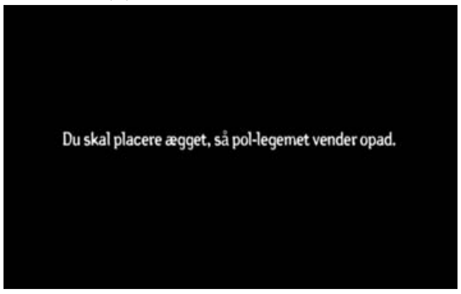
(a) Select The Little Guy


(b) Select The Little Guy

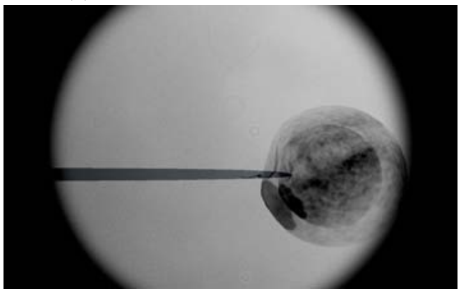

Godt

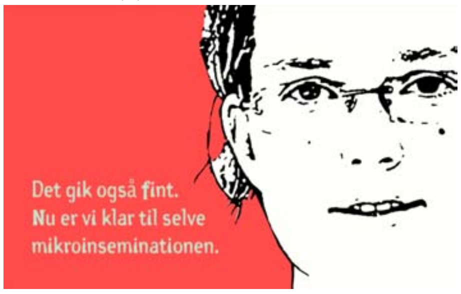
Vi har nu også gjort det
lidt nemmere for dig.
I et mikroskop er det
en del sværere at styre
sug og pipetter.
Men flot alligevel!

(c) Select The Little Guy Success

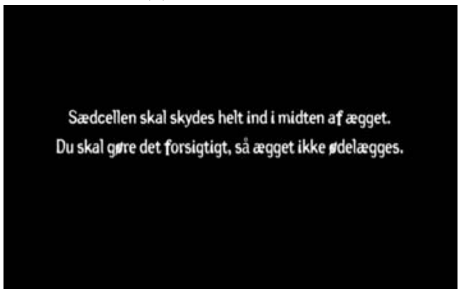

Du skal placere ægget, så pol-legemet vender opad.

(d) Turn The Egg Text

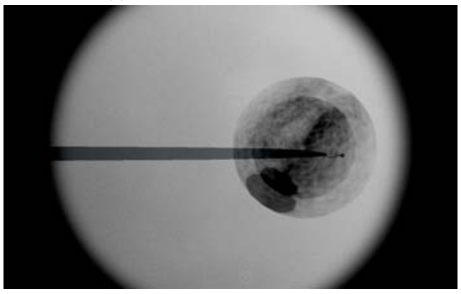

(e) Turn The Egg

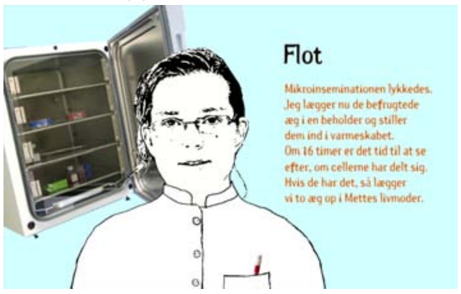

Det gik også fint.
Nu er vi klar til selve
mikroinseminationen.

(f) Turn The Egg Success


Sædcellen skal skydes helt ind i midten af ægget.
Du skal gøre det forsigtigt, så ægget ikke ødelægges.

(g) Insemination Text


(h) Insemination


Flot

Mikroinseminationen lykkedes.
Jeg lægger nu de befrugtede
æg i en beholder og stiller
dem ind i varmeskabet.
Om 16 timer er det tid til at se
efter, om cellerne har delt sig.
Hvis de har det, så lægger
vi to æg op i Mettes livmoder.

(i) Insemination Success

(j) Outro

2 Software Framework

The primary reason for developing a software system as part of the solution is to support the interactive elements. Several parts of the product are interactive and supporting this type of system is best done with software. The interactive sequences should be believable and give the user the impression of looking through a microscope and performing the actual procedure. The interaction rules out using just video. A system supporting both video and interactive elements would be beneficial.

Software Base The requirements of this project mandate a fairly large code base when all the required features are taken into account. The main reason why this project was completed before the deadline was due to the use of an existing framework and supporting libraries. When it comes to frameworks capable of handling user input and drawing something responsive on the screen, existing possibilities are endless. Lots of commercial frameworks can be tweaked into fulfilling the requests of this project. The drawback of acquiring an external framework is that it can take a lot of time to learn how to use the system and non-obvious problems can arise late in the project phase. We have chosen to use a game engine framework developed at DAIMI. We, the authors, are some of the main contributors to this framework and therefore do not need additional training to start developing within it. The engine is intended for developing games and learning about how to create an engine for such work. By using the framework as base for this project we hope to show that it can easily be set to good use for other tasks than games. The framework has the other benifit that it is free software, comes at no cost and supports a wide range of operating systems.

Existing Functionality OpenEngine is an ongoing project under constant development. At the time this project started out all basic functionality such as loading 3D models with textures, handling input from keyboard and mouse, drawing the scene on the screen etc. Besides the basic functionality OpenEngine also provides a consistent software design easing the task of organizing the domain specific code. Our project was developed alongside the OpenEngine project. Many missing features were therefore added to OpenEngine as we progressed. Features added to the OpenEngine project were done with general use in mind, hereby improving the OpenEngine frameworks functionality.

2.1 Extending the Framework

The OpenEngine framework is split into a base and several extensions. Non essential features are placed in OpenEngine extensions and all the main structures, such as interfaces and core components are placed in the OpenEngine base. This also provides us with a way to structure our software closely related components are grouped together but isolated from the non-relating ones. The following subsections describes the extensions we created and used as part of the Inseminator.

2.1.1 Multimedia Abstraction

A multimedia abstraction helps to support working on several operating systems. By abstracting the mechanisms for creating frames and receiving input from devices we can ignore the many different ways that system choose to handle this. We use SDL for handling user events from devices such as mice, keyboards and joysticks. SDL also handles the window and 3D canvas creation. SDL is open source and cross platform. It is a widely used framework which has been around for quite a while, and is therefore considered very stable. It is tested with OpenEngine on Linux, Mac OS X and Windows.

2.1.2 Physics

To handle the physics calculations on the egg, a module based on particle manipulation was needed. Creating a spring and constraint based physic library would require a substantial amount of work and we settled upon using an existing project. Jesper Mosegaard provided us with the Mediotic library which is a particle physics framework based on verlet integration and has the ability to handle springs and vector constraints. The library was developed on Microsoft Windows using Visual Studio which does not comply with the ISO/IEC C++ language standard. To keep the application platform independent we needed to port the library. The Mediotic Physics library was wrapped as an extension. The main module file is implemented as a time dependent module and simply added to the engine module performs best at a tick count of 20 while running on the hardware chosen for the project.

2.1.3 Displaying Movies

The request for decoding a video stream and showing it in full screen demanded the use of an existing library as video decoding requires a substantial amount of work. The video extension is based on the ffmpeg library, which is cross platform and open source. Interleaving video and interactive 3D scenes without any flickering on the screen was a must. Our solution was to display the movie inside the 3D canvas as texture on a 3D object. Thus, we avoided the need for changing the canvas when switching between video and 3D scenes. Changing the canvas could easily have caused unwanted artifacts and would make it difficult to smoothly blend between the scenes. This work is made available as an OpenEngine extension named `FFMPEGResource`. The `FFMPEGResource` extension implements the `ITextureResource` interface and the engine `IModule` interface. By implementing the module interface we can add the resource to the engine loop so it receives processing time. This processing time is used to decode the movie and bind the frames as textures in the same way as a texture resource is bound. The module keeps track of time and implements synchronisation functionality, so the correct picture is bound to provide the frame rate dictated by the movie. All the functionality is decoupled from the other parts of the system, and can be run by itself. Because of this we have made a project `GLTexturePlayer` which was used to display the functionality of the `FFMPEGResource` extension.

2.1.4 Application Stages

Throughout the application execution the scene changes seventeen times. This consists of fourteen video stages and four interactive simulations. To encapsulate each of the stages we have created an abstraction over an engine state and each stage implements this abstraction as a separate state object. The states are managed by a central state manager and processed in sequence. Different kinds of states are provided depending on the features of the specific stage. All states implement fading in and out in the `BaseState` class that forms the root of the state hierarchy. Several special stats are then derived from this base for specific tasks as show in figure 2.1.4.

`PictureState` is used for displaying a single picture. This is used for the start-up screen and can render pictures in HUD[1] mode.

`MovieState` is similar with the additional feature of rendering a movie and switching state when the movie ends.

[1]`http://en.wikipedia.org/wiki/HUD_(computer_gaming)`

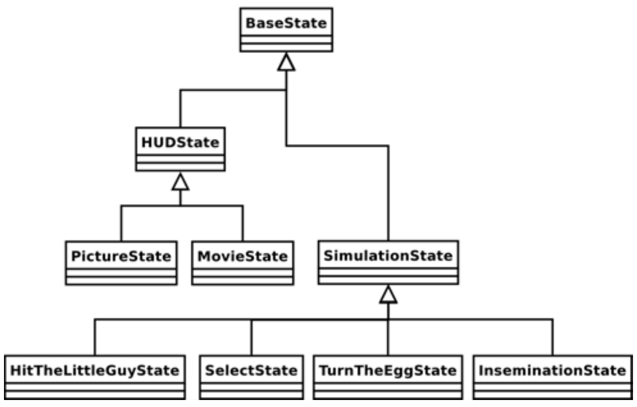


Figure 2: Hierarchy of States

`SimulationState` displays a custom scene and activates interaction capabilities.

The application is implemented as a OpenEngine project called `Inseminator`. In this project the `Factory` class is the key component of the project. The factory is responsible for creating all components and binding them together inside the engine.

2.2 Interactive States

The four interactive stages, are derived from `SimulationState`. This class includes code for the needle, the background settings and a transparent microscope foreground in HUD mode. `HitTheLittleGuyState` and `SelectState` are fairly similar and contains the following components:

- Spermatozoa cells made of a single semi-transparent texture. As seen on figure 2.2.
- Randomized logic for the cells movement based on Bezier. This makes the cells move randomly around just the way you would expect them to do if you where looking into an actual microscope.
- Collision detection in the form of a simple distance function use to check if the needle actually hit the spermatozoa.
- If an error occurs a texture is used to illustrate that the procedure was performed incorrectly and require restarting the simulation step.

Also `TurnTheEggState` and the `InseminationState` are interactive steps with similar elements.

- The egg is built by two spheres. The inner is the core of the egg and is connected to a larger outer shell. The two spheres are connected by spring constraints making them capable of moving independently while kept in place. Each sphere is textured and transparent to give them a noisy surface.
- Collision Detection is used to identify when and where the needle is touching the egg. Constraints are added to simulate friction and a bit of resistance when the needle penetrates the egg surface.

Figure 3: Spermatozoa Billboard.

- Turning Logic is used when the needle is used as a tool for turning the egg. The tip of the needle must stick to the surface of the egg so both the inner and outer egg shell is affected by movement of the needle. An invisible line goes through the egg to identify the degree of rotation. When the dark spot is pointing upwards or downwards the rotation degree exceeds 45 making the operation successful.
- Centering The Egg is needed to keep the egg within the visual area of the screen. This makes it impossible to push the egg out of the screen. The Egg center is calculated and slightly dragged towards a fixed point in the middle of the scene. This way the egg can be moved and turned around while its nice and slowly seeking towards staying inside the scene.
- Reset Logic is used if the user fails to successfully complete the operation, triggering a restart of the stage.

3 Platform

The system was deployed on a machine with the following hardware configuration.

- Intel Core 2 Duo 2,66 GHz E6750.
- 2048 MB (2x1024) DDR2 667 MHz.
- MSI P35 Neo-F LGA 775 motherboard.
- Onboard sound: Realtek High Definition.
- Onboard net: Realtek RTL8168/8111 GB PCI-E.
- SATA: 160 GB Samsung HD161HJ SATA2.
- IDE: TSST DVD+RW DL Lightscribe, SH-S182M.
- Inno3D nVidia GeForce 8800 GTX-768 768 MB PCI-E.
- Monitor: Dell 2407WFP 24", running native at 1680x1050.
- Apple Pro Mouse.
- Dell usb keyboard and a custom made two button ps2 keyboard.
- Unknown projector.

The bios has been set to start up the system as soon as the machine is connected to a power outlet. No effort has been take to hide bios or OS start-up screens, as we believe the on-site staff could benefit from access to the machines bios on startup.

3.1 Operating system

The machine is running Ubuntu 7.10, which is a freely available distribution of GNU/Linux. We encountered a few difficulties while installing the basic operating system at the time. This was mainly because the hardware was very new and some of it could not be identified by the operating system. The installer could not identify drivers for the CD drive and a few other functions provided by the motherboard. To enable hardware rendering and support for external monitors (or projectors) we needed to install NVidias proprietary graphics driver. After installing the NVidia driver running nvidia-settings will allow you to enable antialiased fonts and setup external screen properties. To compile and run OpenEngine several tools and development libraries need to be installed. All of this can be installed with the package system included with the system. An up-to-date description of the dependency list on vairous systems can be found at `http://openengine.dk/trac/wiki/Building`. The building guides also cover how to build the system. Make sure to place the required texture and video resources in correct location.

After a successful build, the program can be run as: `./build/Inseminator/Inseminator`

3.2 Kiosk Mode with X11

The system setup needs to be reliable and secure. This includes automatic recovery from errors and making access to the system unavailable for untrusted parties. We wanted the system to run the application at system startup, and to have it restart in the event crashes of either the application or the windowing system it runs in.

3.2.1 Users and Privileges

The system consisted of two user accounts. One privileged user, called `admin`, and one unprivileged user, called `simulator`. All system files are owned and only editable by the `admin` user. Only the inseminator binary is then executable by the `simulator` user. Only the system developers have login permissions to the system. The on-location staff would not have the necessary knowledge to recover from a system error, and as so, have no need for the login information.

3.2.2 Automatic Startup and Recovery

To start the system we first need to identify when the system has been brought up. This can be done by attaching a process to a system terminal. The code is shown in listings 1 and 2. This automatically performs a login for the unprivileged user and will restart the process on exits.

Listing 1: Content of `/etc/event.d/tty1`

```
# /etc/event.d/tty1
# This service maintains a getty on tty1 that does an autologin of
# some user from the point the system is started until it is shut down
# again.

start on runlevel 2
start on runlevel 3
start on runlevel 4
start on runlevel 5
stop on runlevel 0
stop on runlevel 1
stop on runlevel 6

respawn
exec /sbin/getty -n -l /usr/local/sbin/autologin 38400 tty1
```

Listing 2: Content of `/usr/local/sbin/autologin`

```
#!/bin/sh
# /usr/local/sbin/autologin
# automatically login the simulator user
login -f simulator
```

To start the X server and run the inseminator program we check for logins on `tty1` without a running X server. If found we invoke the `xinit` program as shown in listing 4. This starts up the X server and runs the `.xinitrc` shown in 3. Here we run the inseminator program in a loop in order to restart it on errors or completion.

Listing 3: Content of `/home/simulator/.xinitrc`

```
# ~/.xinitrc
# set some basic X settings
xrdb -load $HOME/.Xresources
xsetroot -solid gray &

# cd to the OpenEngine directory and continuously restart the
cd openengine
```

```
while (true); do
  ./build/Inseminator/Inseminator
done
```

Listing 4: Content of `/home/simulator/.bashrc`

```
# ~/.bashrc
# if we are not running X and we are on tty1 startx
if [ -z "$DISPLAY" ] && [ $(tty) == /dev/tty1 ]; then
  xinit
  logout
fi
```

4 Reflection

At the time of writing the system has been in use for half a year and no problems have been reported during this period. A few issues were solved in the days immidiatly following deployment in regard to setting up the projector. Apart from the few problems with cables and projectors we have barely heard from the users. A recent inspection of the system showed no problems. This is a good indication that the system is performing and that the product for fills its purpose. We believe that a structured development and continuous testing is in part responsible for this success.

5 CD Content

The CD supplied with this report contains the following.

- This report.
- A preliminary version of the *OpenEngine Technical Report*.
- Source code of the deployed, but now outdated, version of the application.
- Source code of the application running the on the current version of OpenEngine.
- API documentation for OpenEngine and the Inseminatior components.
- All videos, images and other content that is part of the Inseminatior application.
- A screencast of the running application.
- Screenshots of developing versions of the application.
- Additional information used in the development process, such as the product specification and the story board.

6 Future Work

After a couple of months we received feedback on the simulator from Mette Kia and Ole Caprani. The list includes some issues and a request for minor changes based upon feedback received from museum guests.

Make it possible to skip a video sequence.
In order to proceed to the next stage without watching the video sequence to the end, a skip feature should be available. Some of the video sequences showing informative text are fairly long and it would be beneficial for viewers to proceed at wish to avoid making the process boring. The simulator hardware does not include a keyboard so either a third button could be added or a combination of existing buttons could be used. The actual skip feature is already implemented in the software but not made available for users.

The chosen spermatozoa can be pushed outside the screen area.
This is considered a bug because it prevents the user from completing the stage. A simple bounding mechanism restricting the spermatozoa to the visual screen area could be implemented to solve the problem.

When the egg genome points downwards the simulator proceeds to next stage.
This is not a bug but part of the first system specification. When the genome points away from the needle, making it possible to penetrate the egg without touching the genome, the egg turning stage is completed successfully.

Draw a hit spermatozoa with a red color instead of drawing a circle around it.
This could be fixed with a small change of the used textures. The spermatozoa with the red circle around it, is just a texture named `SpermatozoaMarked.tga`. Replacing this texture with one showing a red spermatozoa would fix this.

The screen built into the table clones the projector output.
If this is not intentional it is clearly a mishap in the communication from all along. Removing the screen or the projector would not cause any problems in regards to the software.

The projector stretches the picture due to wrong resolution.
This is due the to fact that the simulator is designed to run in wide-screen mode. This choice was made early in the developing process before anyone knew that the output was supposed to be projected up on a wall. The graphical content could be changed so the simulator would look nice in 4:3 format like the projector. This requires all videos and textures to be converted. Alternatively the projector could be replaced by one supporting wide-screen format, probably a more expensive solution.

Video sequences are too long.
Many of the videos displaying text are simply too long. Users can get bored and leave the stand prematurely. A possible solution could be to allow drawing movies faster. This might be problematic depending on the hardware. The video skip feature mentions above implemented might show this to be a non-problem. That way people could simply press a button when they were done reading the text.

Decrease the inactivity timeout for restarting the.
If a user leaves the simulator before the insemination has been completed, the simulator waits too long before it resets and starts allover. To adjust this change `RESET_TIME` in `Factory.h`.

Go to the next stage if a user repeatedly fails a stage.
This feature is necessary since some users simply can't perform the operation correctly. This however, should not keep them from completing the overall procedure of artificial insemination.

Adjust mouse speed so hitting the spermatozoa becomes easier.
This is just a small adjustment easily done by changing `MAX_MOUSE_SPEED` in `NeedleHandler.h`. Currently this is hardware dependent and must be adjusted specifically for the target machine.

References

Christian P. V. Christoffersen, Carsten Norby, and Ian Zerny. *OpenEngine Technical Report*. URL http://www.openengine.dk/oetech.pdf.