ESSAY CA3 - Have Fun And Play

Jan Kolkmeier

THE COURSE

At Have Fun And Play we put our focus on working on something that could be presented at the Gogbot-festival. So we had two requirement for the result of the project: It should be "fun and [let] play" and deal with the *technological singularity*, as described by Ray Kurzweil, because it is the theme of this years Gogbot-festival.
Some brainstorms were held to find our position/attitude towards "The Singularity" as Creative Technology students. The result was that we see the singularity both as chance and menace. A chance for people (like us) that know how to use technology and can keep up with its innovations and rapid development. A menace for people that fear technologies because they don't know how they work, or because they simply lost track of it once.
The conclusion of this was that, as Creative Technology students, we should be able to make people aware of the technologies that surround us and explain how they work, helping them to understand and handle them with responsibility. This position opposes to the pessimistic slogan of this years Gogbot: "The Singularity is Near: Resistance is Futile". Resistance is NOT futile and we can "CreaTe resistance".
How could this statement be turned into an installation for Gogbot? Since the singularity is a broad topic, we focused on a more specific part: information trails that people generate of themselves in a world of digital surveillance and monitoring. So in the last two weeks we came up with the idea of a "canvas" that reflects the visitors actions of a room into virtual space. We started preparing for a small event to present a prototype of this installation. The result looked as following:
The "visitors actions" mentioned before were present through a number of “stations”. The first station was a **QR-Code scanner** at which people could "buy" things from a shelf. Digital representations of the things that were bought dropped on the canvas immediately. A similar idea was present at the **shape-drawing station**. Here people could draw shapes on a tablet. Here three dimensional representations of these shapes would be dropped on the canvas. There was a **webcam** installed that observed the room. By moving through the room patterns of your movement are generated and sent to the canvas as images. Another webcam was installed in a **photo booth**, triggering a flash every time a visitor enters it, uploading his face to the canvas as well. Other installations were a **hacking terminal** that allowed the visitor to "hack" the system by using a unix-style terminal (this installation actually did not make it to the presentation), a **targeting game** that could be used to launch a bomb to wipe away the elements from the canvas and a **profile generator** that mimes a registration form of a social community, submitting your personal data to the canvas. In the end any action that was taken immediately reflected in a virtual world, making the visitor aware of the digital traces he leaves every time he surfs the Internet or buys something with a membership card at their local supermarket. These ideas were written down in the Creative Technology Manifesto, a

document to go along this event.

MY PART

My own contributions to the project were a number of demos, like my version of Draw & Walk, or the prototypes of the UnityCanvas, which is the network enabled 3D drawing application we use for the installation at Gogbot 2010. I delivered a processing "library" for the UnityCanvas to the other students so that they could easily connect their installations to it. Also I am responsible for our presence at the Gogbot festival itself, leaving me a lot of organizational tasks as well. Further I helped making the design of the Creative Technology Manifesto.

CONNECTING EXTERNAL APPS VIA SOCKETS TO UNITY

In the following a little example on how to include a socket server to unity, and how to connect to it via external applications.

Including the socket server

The easiest and fastest way to get a socket server running in Unity3D is by copy-pasting this example in C# from the unifycommunity wiki [1] to a file named Server.cs. Of course *any* socket server written in C# will work, since there is nothing Unity3D-specific about it. The server will automatically start as soon as it is added as a component to a GameObject. This can be done either through the editor or from a script:

```
Server server;
server = gameObject.AddComponent<Server>();
```

The next part is getting the incoming messages and processing them. The server from the wiki fills an array named "m_rBuffer" with the incoming messages. We just need to read them out, and send them to another script to process them. This can be done by adding a function like the following to the Server.cs:

```
void Update() {
  if (m_rBuffer!=null && m_rBuffer.Count>0) {
    Array stack = m_rBuffer.ToArray();
    foreach (String msg in stack) {
      protocol.runCmd(msg);
      m_rBuffer.Remove(msg);
    }
  }
}
```

Unity will call the “Update()” function once per frame. It will read out the array, passing each messages to a function "runCmd" in the "protcol" object before removing it from the buffer. Here is a basic example for a Protocol class:

```
using UnityEngine;
using System.Collections;

public class Protocol : MonoBehaviour {

   /* Process commands from a message string */
  public void runCmd(string s) {
    string[] commands = s.Split('\n');
    for(int i = 0; i<commands.Length; i++) {
      commands[i] = commands[i].Trim('\n');
      string[] args = commands[i].Split(' ');
      if (args[0] == "line" && args.Length==5) {
        Vector3 a = new Vector3(int.Parse(args[1]), 0,
                                int.Parse(args[2]));
        Vector3 b = new Vector3(int.Parse(args[2]), 0,
                                int.Parse(args[3]));
        DrawLine(a, b);
      }
    }
  }

  /* Draw a line on the screen from a to b */
  void DrawLine(Vector3 a, Vector3 b) {
    // create linerenderer
    GameObject line = new GameObject("line");
    line.transform.position = Vector3.zero;
    line.AddComponent<LineRenderer>();
    LineRenderer lr = line.GetComponent<LineRenderer>();

    // set start/endpoint
    lr.SetPosition(0, a);
    lr.SetPosition(1, b);
  }
}
```

At first note that the message passed to “runCmd()” is first split at a delimiter, in this case '\n'. Delimiters are used to divide multiple commands sent in a single message. In this protocol, we have a “line” command implemented. It requires four parameters. Four integer-values, two for each of two Vectors. These get passed to a function “DrawLine()” that will draw a line between these two vectors in Unity3D using the LineRenderer. To test this, start up putty [2] or any other telnet client and test if sending "line 0 0 5 5" creates a line from the origin, (0,0), to (5,5).
Other commands can be implemented to the protocol by extending the conditional statements in the “runCmd()” function.

Writing a processing client

The last part of this tutorial shows a simple processing client. Connecting processing to a socket server is trivial thanks to the Client library [3]. Here is an example sketch:

```
import processing.net.*;
```

```
Client myClient;

int x1;
int x2;
int y1;
int y2;

void setup() {
  size(200, 200);
  myClient = new Client(this, "127.0.0.1", 9349);
}

void draw() {
  background(0);
  stroke(255);
  line(x1,y1,x2,y2);
}

void mousePressed() {
  x1 = mouseX;
  y1 = mouseY;
}

void mouseDragged() {
  x2 = mouseX;
  y2 = mouseY;
}

void mouseReleased() {
  sendLine(x1, y1, x2, y2);
}

void sendLine(int x1, int y1, int x2, int y2) {
  myClient.write("line"+" "+(x1-100)+" "+(y1-100)+
                      " "+(x2-100)+" "+(y2-100)+'\n')
}
```

There is nothing exiting here, the connection is initiated once passing the IP and port number. Then, by pressing, dragging and releasing the mouse a line can be drawn. The "line" command is formatted appropriately and gets sent through the socket.

[1] http://unifycommunity.com/wiki/index.php?title=Server
[2] http://www.chiark.greenend.org.uk/~sgtatham/putty/
[3] http://processing.org/reference/libraries/net/Client.html