MPP405 Logbook

The Creation of Three Cross-Platform Casual Games

Submitted in partial fulfilment of the Master of Arts/Master of Science Professional Practice at SAE Institute.

Student name:	Nicolas Zanotti-Schudel
Date of submission:	January 2013
Academic advisor:	Dr. Fares Kayali
Word count:	6812

December 10th 2011

The topic of my research will be the process of creating cross-platform casual games. My choice of tools to use within this process has its importance. If I were to research the process of building a wooden bird house for instance, choosing an axe over a saw could likely change the outcome.

Preliminary investigations for the research proposal have shown that *Adobe Flash Professional* content can be compiled to multiple devices using *Adobe AIR*. I have considered the use of the following software technologies as alternatives for producing casual games:

- *Corona* (2011)
- *haXe NME* (2011)
- *HTML5 EaselJS* (2011)
- *Google PlayN* (2011)
- Moai (2011)
- MonoGame (2011)
- *Unity* (2011)

All these technologies appear to have their own strengths and weaknesses. Testing each of them in detail could be an interesting research project in itself. For this research, I have chosen to use *AIR* for three reasons: the maturity of the tools, my existing experience, and the reach of the platform. Working as a Flash Developer for a cross-media advertising agency, I have previously created multiple web games professionally with the toolset. Without a knowledge-gap, I can avoid making 'beginners mistakes' and concentrate more on the process. I will, however, stay aware of the current trends and technologies in the gaming software development field and consider switching if appropriate.

December 12th 2011

I have been investigating the development of *AIR* games for home entertainment systems and have found three possibilities so far:

1. It is possible to develop *AIR* applications [apps] for the television. However, since the app will be steered using a remote control typical for TV sets, game interaction would be severely limited.
2. *Scaleform* would allow the embedding of Flash into *Sony Playstation*, *Microsoft Xbox* and *Nintendo Wii* games. Sadly, buying such software is not in my budget by far.
3. The *Nintendo Wii* and *Sony Playstation* support old versions of the Flash Player inside their browsers.

For the purpose of the research paper, I will compile to a native OSX app and run it on a computer attached to the television. This way, I can test the games using the game controllers of the console and on a television screen, without having to pay any additional licensing costs. The in-game interaction is the same as with the console.

December 13th 2011

During the writing of the paper leading up to this research, “A Comparison of Action Research and Scrum” (Zanotti-Schudel 2011), I have put a lot of thought into the design of the research. I have documented the design in my paper and started putting together the necessary background information.

December 20th 2011

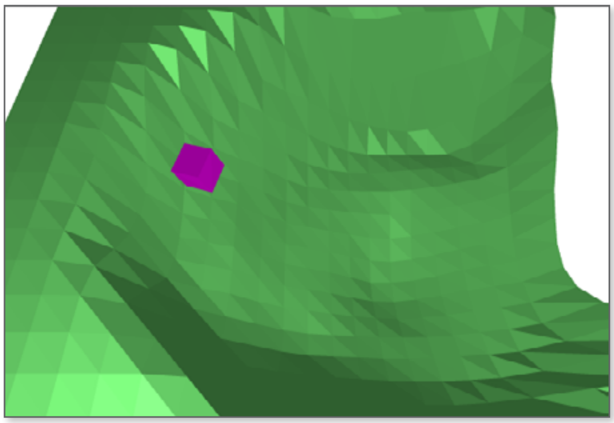
I have created the concept for my first game:

Title:	Proximity
Goal:	Place miniature black holes in order to attract all the needed items and advance to the next level.
Backstory:	The player is in possession of a device that can generate short-lived miniature black holes. Using this device, the player can fill the black hole with items gathered from the surface of a planet.
Rules:	The black hole is open for 12 seconds, during which it has a constant pull on items. Items that touch the event horizon are instantly pulled into the black hole. All the required items need to be gathered. Some items are harmful, so picking one up will result in failing the level. Some required items are translucent and cannot be pulled by the black hole. They can, however, be pushed along with other items.

The targeted device platforms for this game are the web, the iPad and the Mac desktop.

December 22nd 2011

I have created a proof of concept [POC] for the basic gameplay. This version runs using the Open Source ActionScript 3D engine *Away3D* (2011). I tested

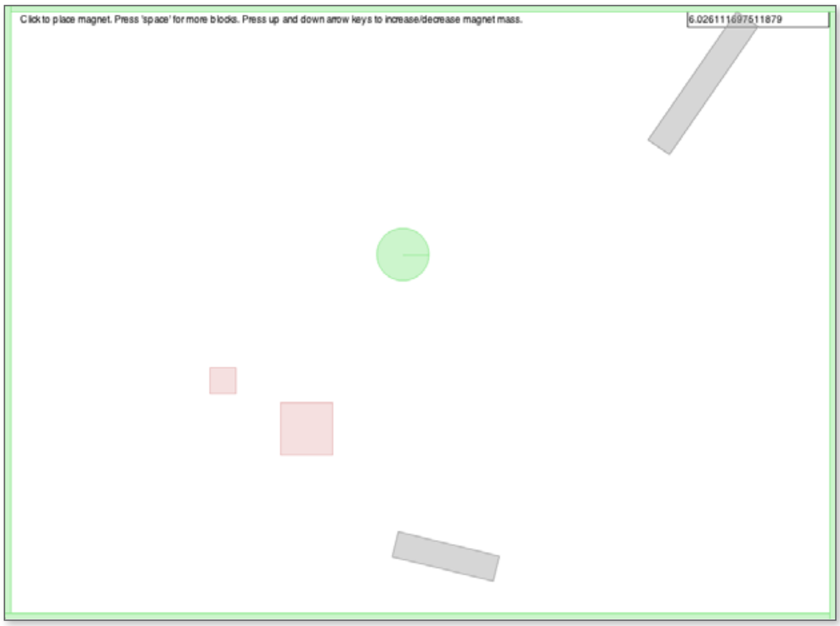

A box falling into a warped 3D plane.

creating the black hole by warping a flat plane and having a box element fall into it. The physics engine I used for this POC is a framework called *Jiglib* (2011). Since Adobe is still implementing their support for 3D-Hardware for mobile devices, I will most likely run into performance issues when using a 3D framework with 3D physics.

December 25th 2011

The second POC is using only *Box2D* (2011) and simple rendering within Flash. I went a step further and implemented the mathematics for pulling objects towards a point.

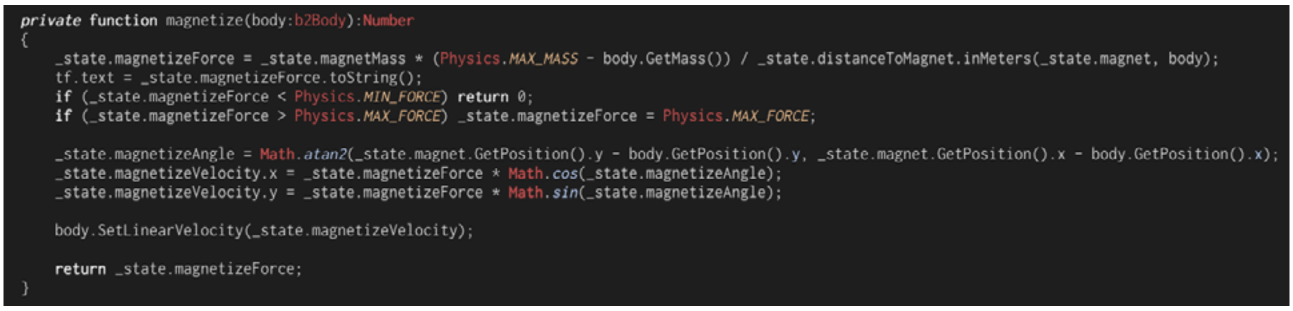
The squares are being attracted to the green circle:



Click to place magnet. Press 'space' for more blocks. Press up and down arrow keys to increase/decrease magnet mass.

6.026111697511879

The Code for 'magnetizing' the elements turned out to be rather complex, since I needed to adapt the math to the physics framework: rather than pulling an object I needed to find out how much force (and from what angle) an object will be pushed. Since the magnetize function is being called



```
private function magnetize(body:b2Body):Number
{
    _state.magnetizeForce = _state.magnetMass * (Physics.MAX_MASS - body.GetMass()) / _state.distanceToMagnet.inMeters(_state.magnet, body);
    tf.text = _state.magnetizeForce.toString();
    if (_state.magnetizeForce < Physics.MIN_FORCE) return 0;
    if (_state.magnetizeForce > Physics.MAX_FORCE) _state.magnetizeForce = Physics.MAX_FORCE;

    _state.magnetizeAngle = Math.atan2(_state.magnet.GetPosition().y - body.GetPosition().y, _state.magnet.GetPosition().x - body.GetPosition().x);
    _state.magnetizeVelocity.x = _state.magnetizeForce * Math.cos(_state.magnetizeAngle);
    _state.magnetizeVelocity.y = _state.magnetizeForce * Math.sin(_state.magnetizeAngle);

    body.SetLinearVelocity(_state.magnetizeVelocity);

    return _state.magnetizeForce;
}
```

The magnetize function adapted for the physics framework.

30 times a second for each element, I reuse every variable by wrapping it into a state object. This way, the garbage collector of the virtual machine won't have to deal with thousands of variables that are used only once.

December 27th 2011

I have taken two days time to test the gaming framework *Unity* (2011 a).

They have a Flash export feature that should be released in the next few months. With the web included, *Unity* games can now target browsers, mobile devices, desktop computers and gaming consoles (2011 b). Even with my brief tests during these two days, it was clear to see that *Unity* is the better tool than Flash for creating 3D games.

December 29th 2011

After *Unity*, I have checked *Haxe NME* (2011) for creating games. It can target mobile devices, desktops, and the web with both HTML5 and the Flash Player. I was able to get started quickly. The programming language was easy to comprehend with my existing knowledge of *ActionScript 3*. It seems that *Haxe* has a reverse approach to *Unity* for creating games. With *Haxe*, most things are done with code. Whereas with *Unity*, most things are done visually within the development environment.

Any deeper comparisons aside, learning the basics of both *Unity* and *Haxe NME* helped me understand different approaches to producing games. The conceptual model [CM] of my research should be tool agnostic. Knowing about the different tools will help me pinpoint any part of the workflow that is targeted to much towards Flash.

January 4th 2012

The releases of new devices and development tools since I proposed this research has been great. This leads me to believe that technologies must be reassessed at a faster pace. I will thus include such assessments at the beginning of the model, similar to the initial planning phase of Action Research [AR]. This way, with each new game, the current state of devices and tools can be checked and compared to previous choices.

Scrum has the concept of the backlog in order to adapt the product to current business circumstances:

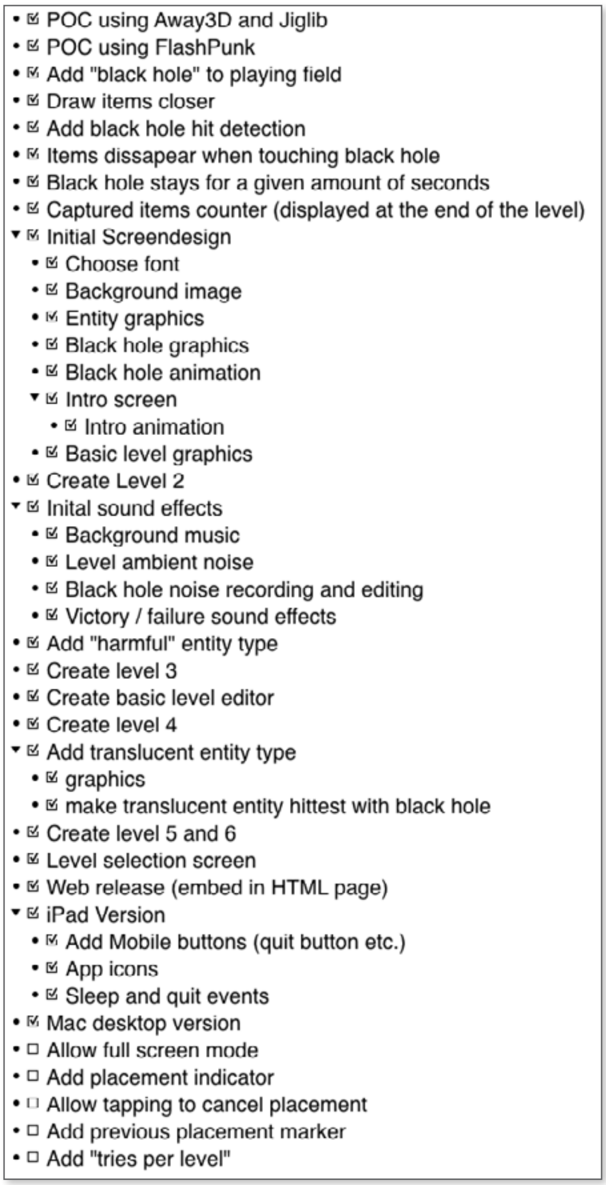
> *"The Product Backlog evolves as the product and the environment in which it will be used evolves."* (Schwaber & Sutherland, 2011, p. 12)

Development tools may also evolve during the course of production. However, switching tools may lead to unnecessary expenditures. Like having to recreate specific graphics assets in a different file-format for instance.

My current stance is to chose the correct tools for the job with each new game, but then to stick with those tools during the course of production. This is in contrast with my original research proposal. At that point, I put more emphasis on the toolset. I even added the technology, the Flash Platform, in the original title of the paper. Now that I am focusing on the 'soft' side of the problem area, like AR and Scrum, the technical 'hard' side has less influence. I will try and be as technology agnostic as possible within the model, but will remain clear on what tools I used for the practical component of the AR cycles.

February 16th 2012

After choosing to base the game on my second POC, I have been producing the game during the past weeks. I decide to use *Flash Punk* for rendering the game, since it promises better performance than the classic Flash display list. Similar to Scrum, I made a project backlog that is ordered by relevance.



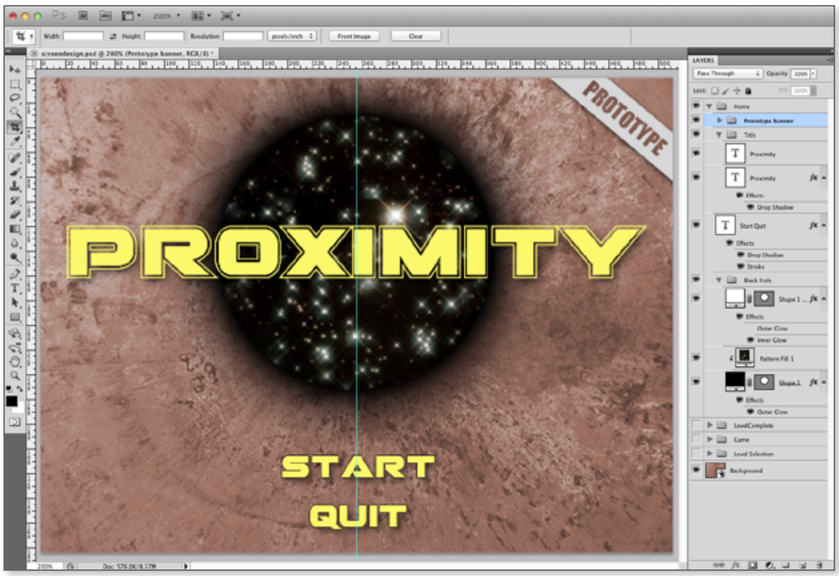
- ☑ POC using Away3D and Jiglib
- ☑ POC using FlashPunk
- ☑ Add "black hole" to playing field
- ☑ Draw items closer
- ☑ Add black hole hit detection
- ☑ Items dissapear when touching black hole
- ☑ Black hole stays for a given amount of seconds
- ☑ Captured items counter (displayed at the end of the level)
- ☑ Initial Screendesign
  - ☑ Choose font
  - ☑ Background image
  - ☑ Entity graphics
  - ☑ Black hole graphics
  - ☑ Black hole animation
  - ☑ Intro screen
    - ☑ Intro animation
  - ☑ Basic level graphics
- ☑ Create Level 2
- ☑ Inital sound effects
  - ☑ Background music
  - ☑ Level ambient noise
  - ☑ Black hole noise recording and editing
  - ☑ Victory / failure sound effects
- ☑ Add "harmful" entity type
- ☑ Create level 3
- ☑ Create basic level editor
- ☑ Create level 4
- ☑ Add translucent entity type
  - ☑ graphics
  - ☑ make translucent entity hittest with black hole
- ☑ Create level 5 and 6
- ☑ Level selection screen
- ☑ Web release (embed in HTML page)
- ☑ iPad Version
  - ☑ Add Mobile buttons (quit button etc.)
  - ☑ App icons
  - ☑ Sleep and quit events
- ☑ Mac desktop version
- ☐ Allow full screen mode
- ☐ Add placement indicator
- ☐ Allow tapping to cancel placement
- ☐ Add previous placement marker
- ☐ Add "tries per level"

The product backlog for Proximity

Besides the previously described code for pulling objects towards the black hole, further highlights during the game production process were: creating screen designs, programming with a graphics engine, recording the sound of the black hole, animating the black hole using bitmaps, and editing the levels. I will now describe these parts with more detail.

Screen Design

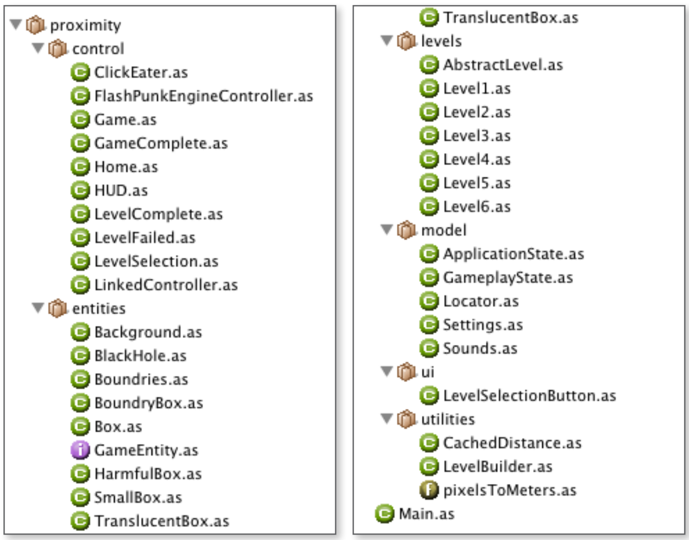
Each screen was designed in *Adobe Photoshop*. The graphics were then exported in the PNG file format to be placed in *Flash Professional*.



The screen design in *Adobe Photoshop*.

Programming with a Graphics Engine

Rather than using the Flash display list, all the rendering is handled by *Flash Punk*. Since it is based on bitmaps rather than vectors, the performance is better, allowing me to place more elements in the game. In hindsight, *Starling* (2012) would have been the better option, because it utilizes the graphics processor rather than the main processor. However, at the time I made the decision, *AIR* did not yet support this feature on mobile devices. The trick with *Flash Punk,* was to draw the game elements with the data coming from the physics engine. I found an Open Source library named *Box2PF* (2012) that helped make this possible. Also with the physics engine, it may have been better in hindsight to use *Nape* (2012) because is built for Flash rather than being a C++ port. This again shows how rapidly frameworks and tools for creating games are evolving.

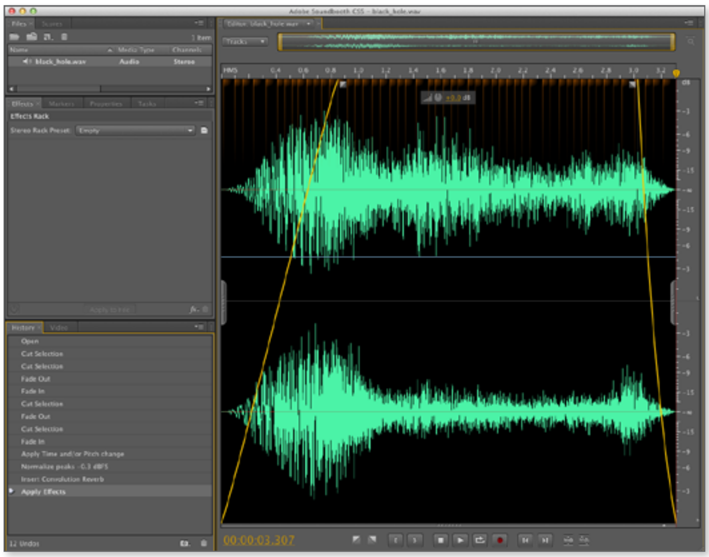


```
▼ proximity
  ▼ control
      ClickEater.as
      FlashPunkEngineController.as
      Game.as
      GameComplete.as
      Home.as
      HUD.as
      LevelComplete.as
      LevelFailed.as
      LevelSelection.as
      LinkedController.as
  ▼ entities
      Background.as
      BlackHole.as
      Boundries.as
      BoundryBox.as
      Box.as
      GameEntity.as
      HarmfulBox.as
      SmallBox.as
      TranslucentBox.as
  ▼ levels
      AbstractLevel.as
      Level1.as
      Level2.as
      Level3.as
      Level4.as
      Level5.as
      Level6.as
  ▼ model
      ApplicationState.as
      GameplayState.as
      Locator.as
      Settings.as
      Sounds.as
  ▼ ui
      LevelSelectionButton.as
  ▼ utilities
      CachedDistance.as
      LevelBuilder.as
      pixelsToMeters.as
  Main.as
```

Code structure of *Proximity*.

Sound Recording

To create the effect of opening a black hole, I recorded a plastic box being pushed along the floor and then added effects in *Adobe Sound Booth*. While



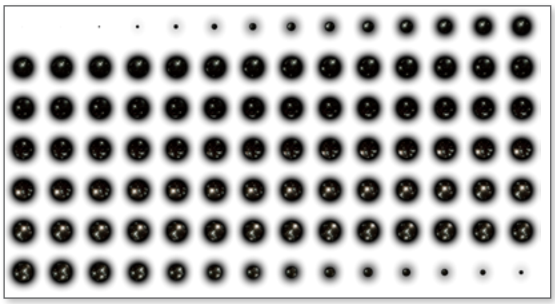
Open
Cut Selection
Cut Selection
Fade Out
Fade In
Cut Selection
Fade Out
Cut Selection
Fade In
Apply Time and/or Pitch change
Normalize peaks -0.3 dBFS
Insert: Convolution Reverb
Apply Effects

Applying effects to the recorded audio using Adobe Soundbooth

this would be an easy feat for an Audio Engineer, I added this highlight to show that a single person can also cover this part of game production.

Animation

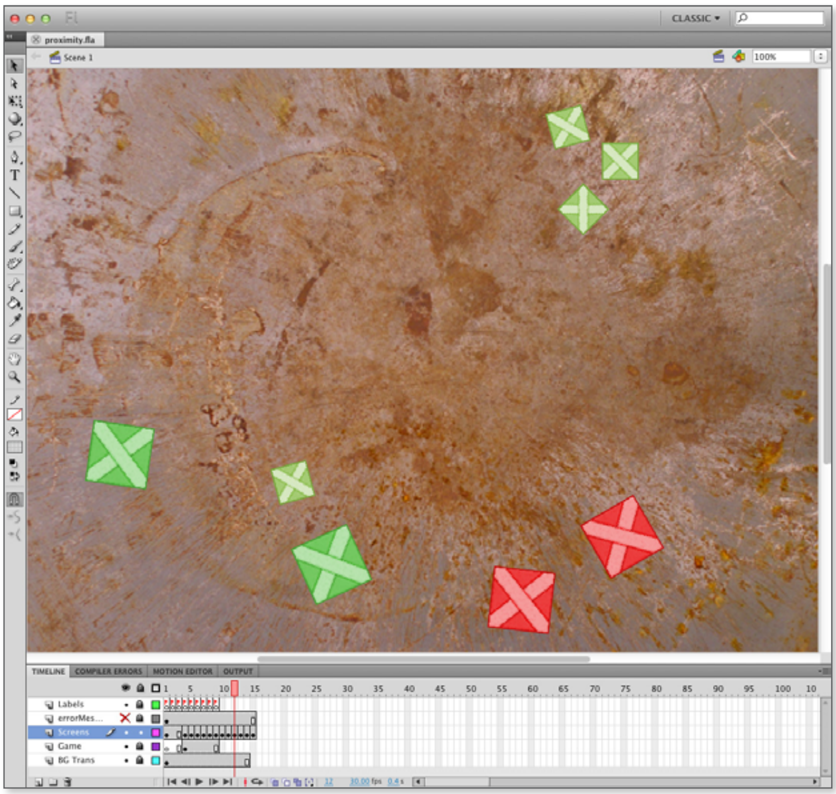
The opening and closing of the black hole was first animated in Flash and then exported as a sprite sheet using a tool called *Zoë* (2012).



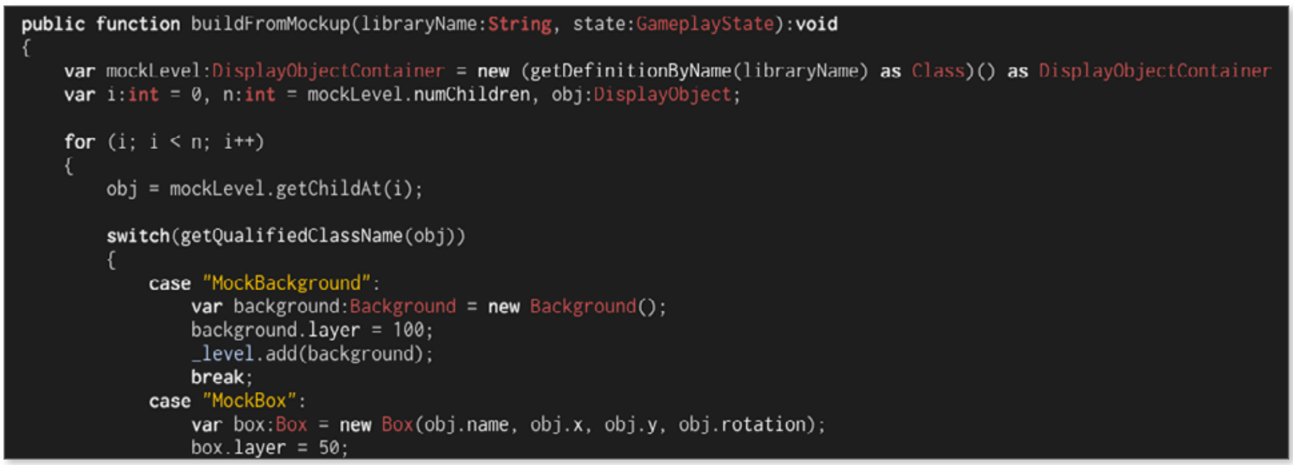
The sprite sheet for the black hole animation.

Level Editing

After creating the first few levels using code, I soon noticed that it would be very time consuming to make every level in this manner. For example, I would need to edit the x, y, and rotation values of a square and compile the game in order to see it visually. I then decided to use *Flash Professional* to place mock-ups of the game objects and create a movieclip[1] for each new level. Then, I created a utility class that reads these movieclips while the new level is being loaded (during runtime) and copies the positioning data to the physics and rendering engines.



Editing levels in *Flash Professional*



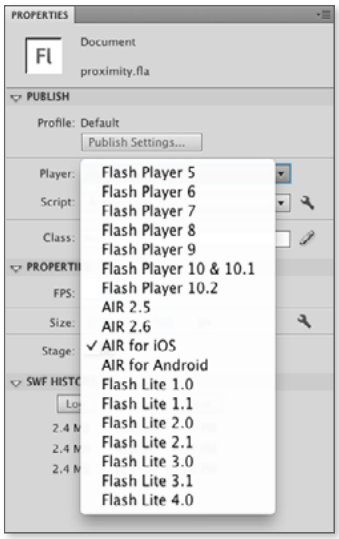
```
public function buildFromMockup(libraryName:String, state:GameplayState):void
{
    var mockLevel:DisplayObjectContainer = new (getDefinitionByName(libraryName) as Class)() as DisplayObjectContainer
    var i:int = 0, n:int = mockLevel.numChildren, obj:DisplayObject;

    for (i; i < n; i++)
    {
        obj = mockLevel.getChildAt(i);

        switch(getQualifiedClassName(obj))
        {
            case "MockBackground":
                var background:Background = new Background();
                background.layer = 100;
                _level.add(background);
                break;
            case "MockBox":
                var box:Box = new Box(obj.name, obj.x, obj.y, obj.rotation);
                box.layer = 50;
```

Part of the code that converts a movieclip to a game level.

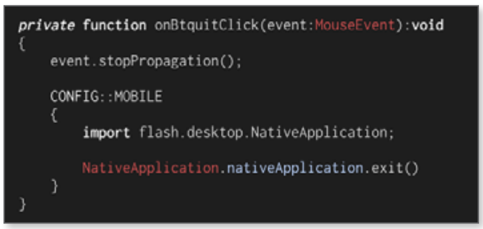
1 Movieclip: A container for graphics with its own time-line in Flash Proffesional.

April 2nd 2012

Creating a release for a specific platform in *Flash* is basically as easy as selecting the target version in the publishing settings. There are, however, device specific additions that need to be made. The web-browser, for instance, does not require an option for quitting the game. A way to make this work is to define constants in the publishing settings and have the compiler check for these during compilation.



The publishing options in *Flash CS5.5*



```
private function onBtquitClick(event:MouseEvent):void
{
    event.stopPropagation();

    CONFIG::MOBILE
    {
        import flash.desktop.NativeApplication;

        NativeApplication.nativeApplication.exit()
    }
}
```

The code within the CONFIG::MOBILE statement only gets compiled when a mobile application is being published.

April 4th 2012

Since I was targeting the web, iPad, and desktop, I created the game using a fixed ratio of 4 x 3. If I were to port the game to Android, I would need to adapt this ratio.

Proximity running on an Android tablet.

To target Android and other devices, I could make the layout adjust dynamically to the screen size. Since this is only a prototype, I will currently not develop this extra feature. However, I should put more emphasis on dynamic layouts in my next game.

April 10th 2012

The prototype of *Proximity* is complete. While there are still many features that I would like to implement, I need to proceed with the next game in order to meet the submission deadline.

The Proximity prototype running on a first generation iPad.

The web version of the game prototype can be played at:
http://nicolas.zanotti.me/proximity/

The source files can be downloaded from:
http://nicolas.zanotti.me/sae/proximity.zip

The Readme.md file in the source ZIP contains credits to all contributing frameworks and resources.

April 23rd 2012

Today I visited the office of *DigiDingo* to interview Jann Sigrist and Jörg Sigrist. The questions were about the game development process, the issues I faced during the production of *Proximity*, and the first conceptual model. I did not present the model until after the interview in order not to persuade their answers. The recorded talk was a little over 30 minutes in duration.

We decided to talk in Swiss German instead of English, so the interviewees could concentrate on the topic of game creation without needing to concentrate on speaking another language. I later translated the interview to English by myself.

May 20th 2012

Upon translating the interview and completing the first AR cycle, I submitted the first draft of the research project to my supervisor.

June 1st 2012

Now that I completed the first AR cylcle, I can advance to the creation of the second game. At this point I can implement the CM, starting with the target audience, target domains and game concept:

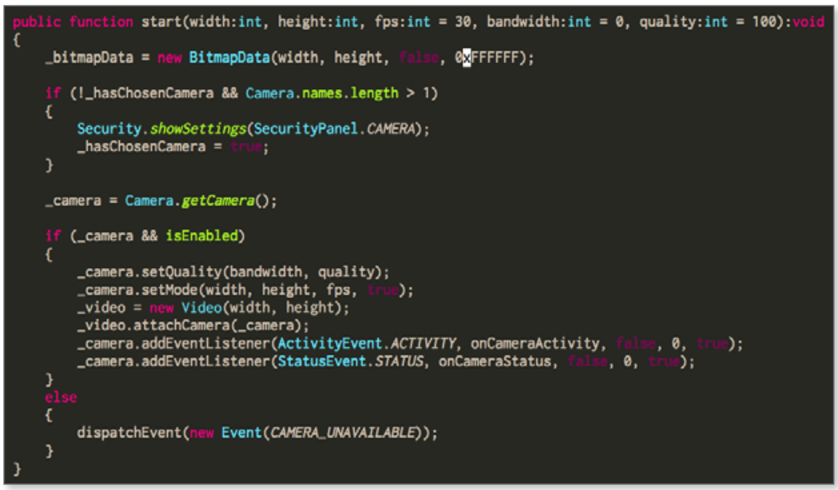
Target Audience:	Designers and Photographers, ages 12 and up.
Target Domain:	Mobile or workplace setting.
Title:	King of the Hex
Goal:	Identify the hexadecimal values of a color as precisely as possible. For instance "0xFFFFFF" is white, "0x000000" is black.
Rules:	Choose from 5 different hex code pairs in order to define the hexadecimal value of a displayed color. The end of each round shows the chosen color next to the displayed color, with an accuracy reading. After 10 rounds a percentage is shown with 100% being all color guessed correctly and 0% being no colors guessed correctly.
Multiplayer Rules:	In a two player game, the first player chooses a color using the devices camera. The second player tries to guess that color. Then, the players switch: Player 2 chooses the color with the camera and so on. After a few rounds the player with the most accurate answers wins.

The targeted device platforms for this game are Android smartphones, the web and desktop computers. The primary computing platform is the Android smartphone. For the prototype, two players play with the same device. Playing each other via social media, e.g. Facebook, would be the preferred method for the full game. Due to time constraints, however, no Facebook integration will be developed.

June 12th 2012

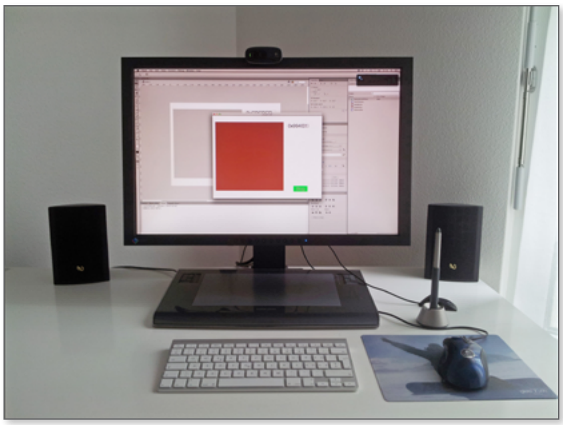
Unlike *Proximity*, *King of the Hex* is a tool that aids in playing a real-world game – the game does not take place solely on the device. Not many special effects are needed. I can, therefore, develop the game entirely using Flash and *AIR*. I do, however, need to test some of the technical possibilities.

The first POC is for selecting a color using a camera. Getting an input stream from a mobile camera or a webcam was fairly easy, because *AIR* provides a unified API that handles this. The next problem was to pick just a single color from the input stream. I first tried an algorithm that extracts the primary color from an image. However, running this on a high resolution picture at least once per second on a mobile device is far too processor intensive. Instead I configured the camera to only record a resolution of 1 x 1 pixels at the very center of the sensor.



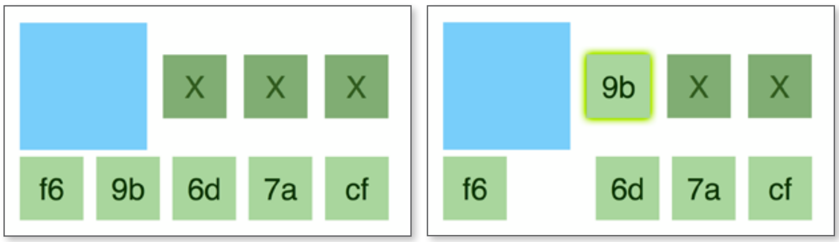
```
public function start(width:int, height:int, fps:int = 30, bandwidth:int = 0, quality:int = 100):void
{
    _bitmapData = new BitmapData(width, height, false, 0xFFFFFF);

    if (!_hasChosenCamera && Camera.names.length > 1)
    {
        Security.showSettings(SecurityPanel.CAMERA);
        _hasChosenCamera = true;
    }

    _camera = Camera.getCamera();

    if (_camera && isEnabled)
    {
        _camera.setQuality(bandwidth, quality);
        _camera.setMode(width, height, fps, true);
        _video = new Video(width, height);
        _video.attachCamera(_camera);
        _camera.addEventListener(ActivityEvent.ACTIVITY, onCameraActivity, false, 0, true);
        _camera.addEventListener(StatusEvent.STATUS, onCameraStatus, false, 0, true);
    }
    else
    {
        dispatchEvent(new Event(CAMERA_UNAVAILABLE));
    }
}
```

Start method from the model class that handles the input stream of the camera.



Recording the color red using the webcam attached to the top of the screen.

The second POC is the user interface for guessing a color. Rather than typing in values, I used drag & drop with predefined color codes (amongst which, the correct values are placed). With this POC, I could test the usability of the screen on target devices.



Drag and drop interface for choosing the hexadecimal values for the color blue.

June 26th 2012

Developing the game based on the two POC demos was straight forward. The single player mode has a predefined set of color values that the player needs to guess. At the end of the game, the percentage of correctly chosen values is displayed. The multiplayer mode has one player choose the color, then the opponent needs to guess it. After a few rounds the game is over and the winning player is displayed.

Both single player and multiplayer modes where created using mock-up graphics. This way, I could focus completely on the user interface and the game flow, before optimizing the visuals. The general flow of iterations was as follows:

Create single player game screens and game logic → Create multiplayer screens and game logic → Refactor the code to unify the game logic → Design the screens.

June 27th 2012

For the second game, I did not make use of a backlog. As mentioned in the research report, the backlog is used in Scrum to order the priorities of the products features. Also, it acts as a communication device between the different parties involved in the project, e.g. the stakeholders and the production team. Making a game in a small group or by myself, this document is optional. I already know what generally needs to be done. Also, since I am adding features in short iterations and the playability is reassessed each time, a main benefit of having a backlog is already covered.

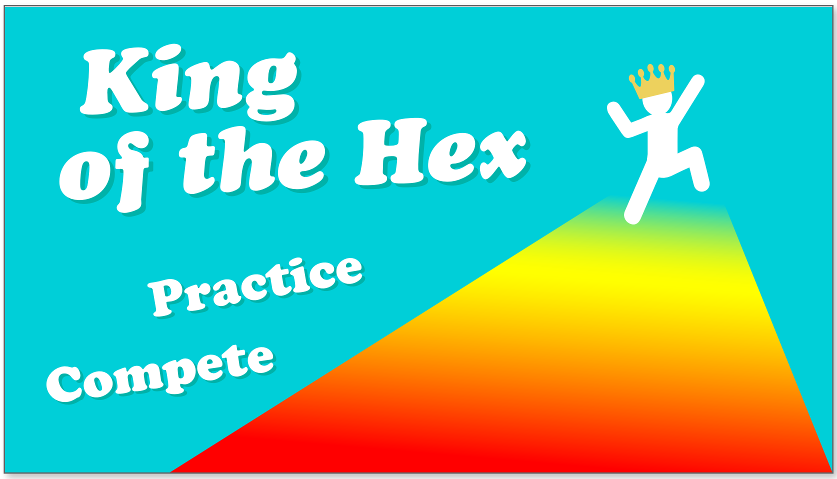
This is different to my previous thoughts on creating documentation. Depending on the size of the group, writing extra documents could be a waste of time. That time could be better spent with better comments in the code base, for instance.

A flaw in the design of my AR cycles, is that I am producing each game alone. I cannot explore the possibilities of communication between the people involved in creating the game. However, with the interviews, I am able to ask the necessary questions to fill that void.

For now, however, my stance is that there does not need to be separate documents describing the state of game development, but rather cleaner code and logically categorized game assets.

June 29th 2012

I am using bright, saturated colors for the screen design of the game. The designs were first made in *Adobe Illustrator*, then exported over to *Flash Professional*. For the start screen, I created an animation of a color gradient using the built in tools of *Flash Professional* (Movieclips and the 3D-Transform tool).



The start screen for King of the Hex

I applied the design and rearranged the mock-up graphics in Flash. The next iterations will be:

- Change the rules so that choosing a similar color also gets points. Otherwise, the game is too hard with just 'right' and 'wrong' answers.
- Adapt the layout for different screen sizes.
- Adapt the layout for an orientation change on smartphones (if the device is held horizontally or vertically).
- Include sound effects.
- Include animations for the 'Win' and 'Loose' screens.

June 30th 2012

I followed the process I laid out as part of the CM from the first AR cycle. There were no major deviations from it, mainly due to the fact that the game was not very complex. There was a clear concept, target audience, and target domains. The tools were an easy choice, since no specific frameworks other than *AIR* were needed. Two proofs of concept allowed me to test the technical feasibility of core parts of the game before commencing the production phase. A first working version was reached quickly, due to the use of mock-ups and wireframes. New functionality, such as the multiplayer mode, and new designs, such as the start screen animation, were added in short iterations. Marketing materials, such as the game icons, were created as an iteration. Two alternate computing platforms (web and PC desktop) , were targeted after completing the first release of the primary computing platform (Android smartphone).

The development time was much shorter than the first game. Mostly, I would credit this to the reduced complexity of the game. However, having a clearer goal in mind also contributed to the increase of productivity. Like with *Proximity*, there are still many more things that need to be completed before having a finished game that can be released to the public. In this case, it would be better graphics and animations, sound effects, and a social media element. This type of game would work well when playing with friends on *Facebook*. The remaining features can be included as iterations. The unknown element is the marketing of the game. This part is not covered by the CM, as it is a different problem and can be decoupled from the release phase of the game's creation (except for the creation of the marketing material).

July 3rd 2012

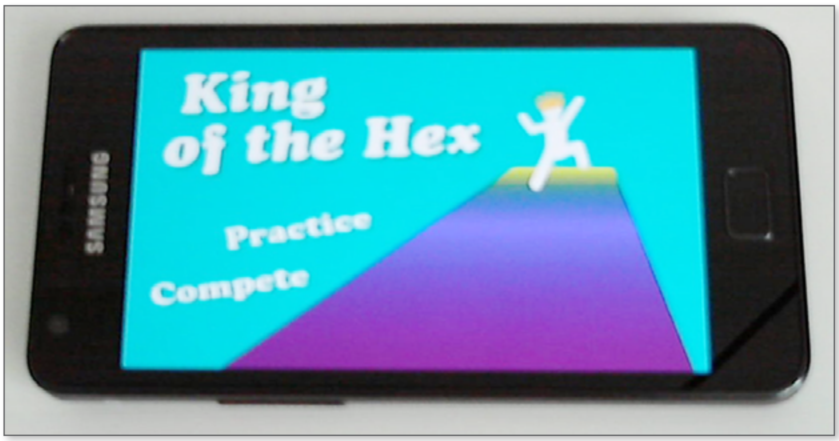
My employer asked me today if I could create a demonstration app for our clients. In a little over a month, it would be the company's turn to host a meeting for a large business network in the region. My second game prototype has the core features in place, so I am able to halt production at this point. Instead, I will use this chance to develop my third game during business hours, with extra documentation for my research in the evenings.

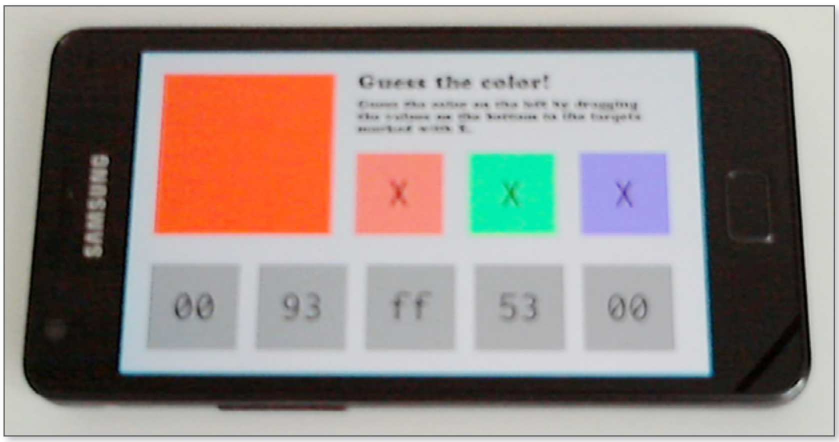
Before starting the 3rd AR cycle, I need to wrap up the current state of the *King of the Hex* prototype:

- Minor changes were made to make the interface more usable.
- A bug was fixed where the camera feed would flicker when recording on an Android phone.
- The icons and materials needed for the Android release were created.
- Credit to the authors of the used frameworks were gathered and added to a readme file.
- The web release was built and published on my website.
- The desktop version was compiled and tested.

These changes were made in one workday. This shows the benefit of working in iterations that always outputs a functioning version of the game. Even after cutting the production time of the game short, I was able to release a stable version. Were this not the case, and the game would have been in the middle of development with many open bugs and incomplete graphics, it would be much harder to continue its creation at a later date. An unfinished mess could even have caused the entire project to fail, because of the extra effort that would be needed for its comprehension.

The game is not yet as fun as I wish it to be. Since I went into overtime with *Proximity*, I still think that my work is good overall, considering the time investment. If I get the chance to invest more time into *King of the Hex* before the research is due, I will try and implement some of the missing features.





King of the Hex running on an Android smartphone.

The web version of the game prototype can be played at:
http://nicolas.zanotti.me/king-of-the-hex/

The source files can be downloaded from:
http://nicolas.zanotti.me/sae/king-of-the-hex.zip

The Readme.md file in the source ZIP contains credits to all contributing frameworks and resources.

July 4th 2012

The entire production code for *King of the Hex* was saved to a Git repository (2012) in incremental steps during its production. This allowed me to revert to a prior version if necessary.

July 5th 2012

Using the CM created in the second AR cycle, the game concept will be:

Target Audience:	Company clients, ages 25 to 65.
Target Domain:	Office or party setting. Fixed installation for soccer goal, one handheld device per soccer player.
Title:	Flick Kick Soccer
Goal:	Shoot as many soccer goals as possible.
Rules:	Kick a soccer ball from one device and attempt to make a goal on another.

July 6th 2012

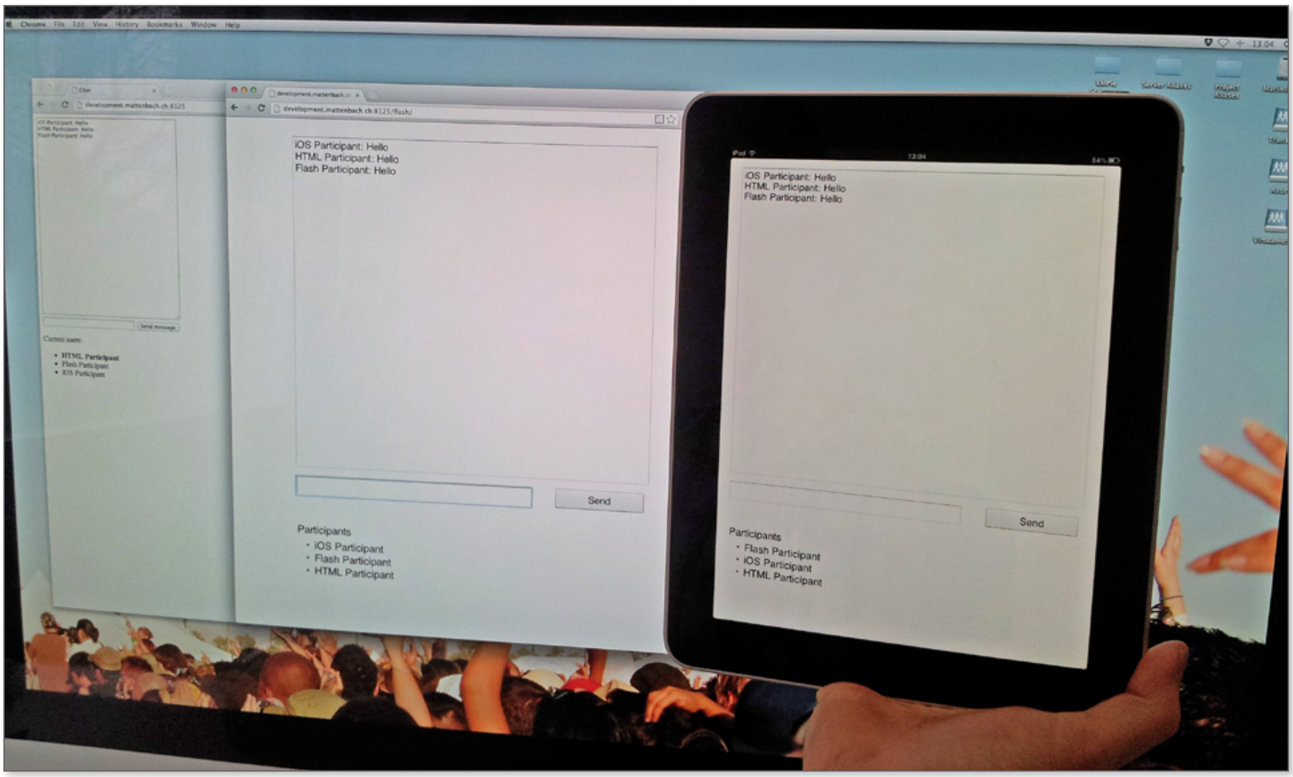
In the case of this game, there are three different elements that will function together to create the game experience:

Soccer Goal: iOS	The soccer goal will run as an iOS application. Using a technology called AirPlay, the screen of an iPad device can be mirrored to a video projector. Also, a Mac application will be created that could be run from a desktop or laptop computer.
Soccer Ball: HTML5	In order for soccer players to join the game using their own device, without the need for installing an app, the second platform will be a mobile browser. Flash is not supported by many mobile browsers. Thus, I must create this part of the game using HTML.
Communication: Node.js	A server is needed to connect the devices together. *Node.js* can stream data live between multiple platforms (Node.js, 2012).

I will need to create a POC for each of the three technologies. In the following text, the applications will be referred to as the *soccer ball app, soccer goal app,* and *server.*

July 8th 2012

The server should pass messages between the soccer ball app and the soccer goal app. At this stage I need to know if a message can be sent from a web page to a compiled app. In order to test the communication possibilities, I will create a crude chat application and have HTML, Flash and iOS clients connect to it. I found a library for *Node.js* called *Socket.io* that handles the low-level functionality of receiving and broadcasting messages between clients (Socket.io 2012). The HTML5 chat application was created in a few hours and was able to connect to my development server without any problems.


iOS Participant: Hello
HTML Participant: Hello
Flash Participant: Hello

Participants
- iOS Participant
- Flash Participant
- HTML Participant

iOS Participant: Hello
HTML Participant: Hello
Flash Participant: Hello

Participants
- Flash Participant
- iOS Participant
- HTML Participant

Three versions of the chat application communicating via a Node.js server. From right to left: HTML5, Flash, iOS.

The Flash application was similarly easy to set up. However, security settings in Flash Player did not allow me to access the server. After many hours of testing, I finally was able to configure the security settings on my development server correctly, so that the Flash application could join the chat server and I could send messages back and forth with the HTML5 version. The iOS version worked immediately. I assume the communication from the app to the server is the same as Flash running in a browser. This POC was a success, since I could prove that the technology functions in a development environment and that I did not have any overhead of game code (and misleading error messages).

July 10th 2012

The soccer goal app will run in 3D and will require a 3D physics library. I will use *Away3D* as the rendering engine, since I already have some experience with the framework (it was considered for use in *Proximity*). Also, in version 4 of *Away3D*, the framework has been updated to support the 3D graphics acceleration API provided by *AIR* (Away3D, 2012). The physics engine will either be the *ActionScript* port of *Jiglib Flash* (2011) or *AwayPhysics*, an *ActionScript* port of the *Bullet Physics Engine* (Away3D, 2012). I cannot use *Box2D* again, because, as the name suggests, it only works in two dimensions.

The first test is mainly to check if I can get a rotating cube object running in accelerated 3D on an iPad. This was a rather lengthy process, taking over 8 hours. I had issues with the Apple provisioning profiles, security settings for apps running on Apple devices, my Apple developers account, and the *AIR* compiler settings for 3D hardware. However, I did finally manage to compile from Flash and get that rotating cube running on the device.

July 15th 2012

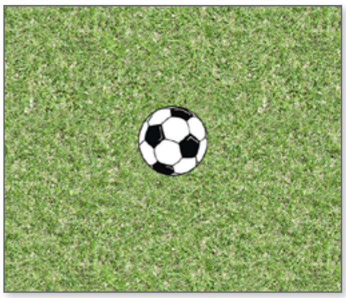
With the 3D engine running smoothly, I needed to check if I could combine it with the physics engine. While I could run a POC of *AwayPhysics* locally, the library would crash on the iPad. After spending four hours on this issue, I decided to switch to *Jiglib Flash*. Luckily the library worked on the iPad without complications.

A real issue with *Jiglib Flash*, is that there is very little documentation for it. I had to learn the library by checking the source code of what little examples exist. Learning the framework in this manner was a waist of precious production time. I will push ahead none the less, since the game only needs a sphere for the ball, a plane for the ground and some cubes for the goal posts. Also, there are currently no viable alternatives for the Flash Platform, to my knowledge.

Unity would most likely be the better tool for the job. Especially with version 4, due for release in November, where *Unity* can export content to the Flash Player. For this game prototype, I will stick to using the software available to me, but will most likely switch for future games.

July 20th 2012

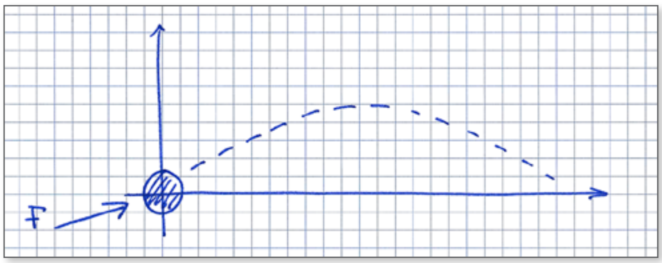
The soccer ball app should run as a full-screen web app so that players can join using their own devices. It is a top down view of a soccer ball, that the player can flick with their finger. The screen size adapts to the dimensions of the device it is running on. I started developing using a library called *CreateJS* (2012).Interaction with the ball was easy to program using *CreateJS*, since the *JavaScript* library uses



many of the conventions known from Flash. I had a working version within a few hours.

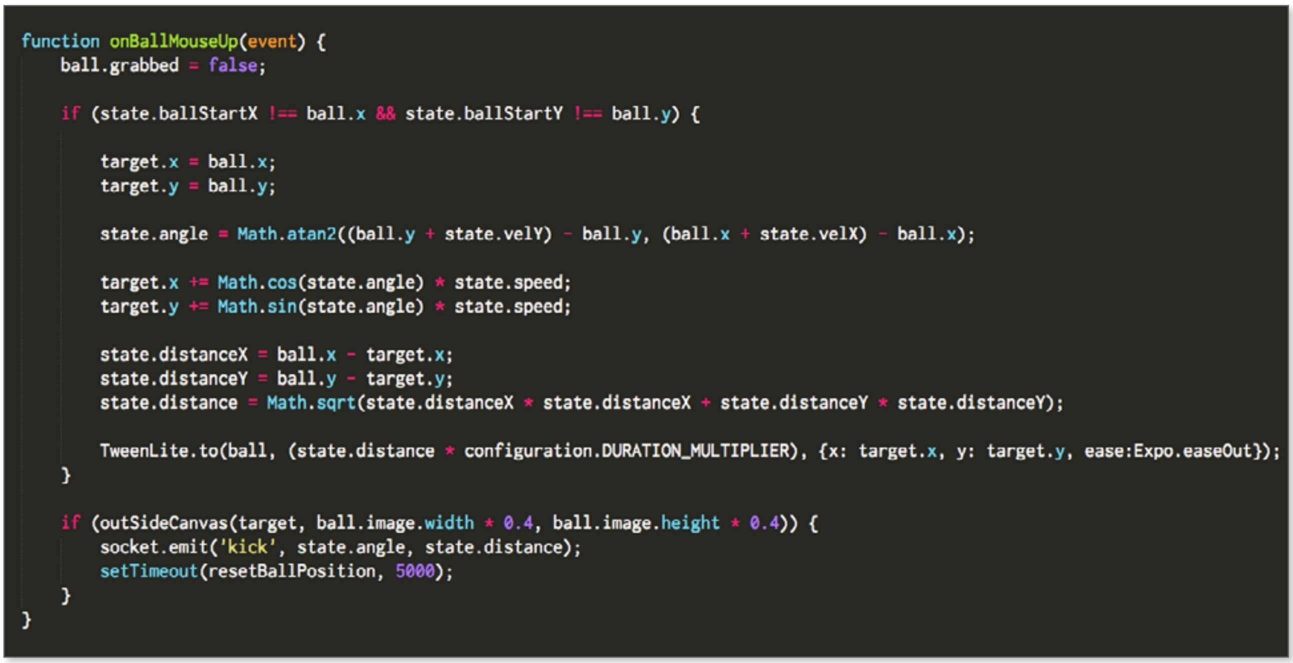
July 21st 2012

A more complicated task was to figuring out the math for transforming a ball being ‘flicked’ in two-dimensional space to a ball being ‘kicked’ in three-dimensional space. The missing z-axis in the third dimension can be assumed, based on the force in the x-axis. For instance, if I kicked a real soccer ball straight forward, with the intention to make it fly a certain distance, I would have force applied forwards and upwards in an angle.



In the game, I assume that each kick is executed perfectly and the forces are applied to the ball in the same angle.

The soccer ball app sends two parameters to the soccer goal app: angle and distance. This is done by recording the velocity and angle of the ball movement while it is being dragged by the finger, until it is released. On the soccer ball app’s screen, I need to know the angle and the distance so I can animate the ball to the projected point.

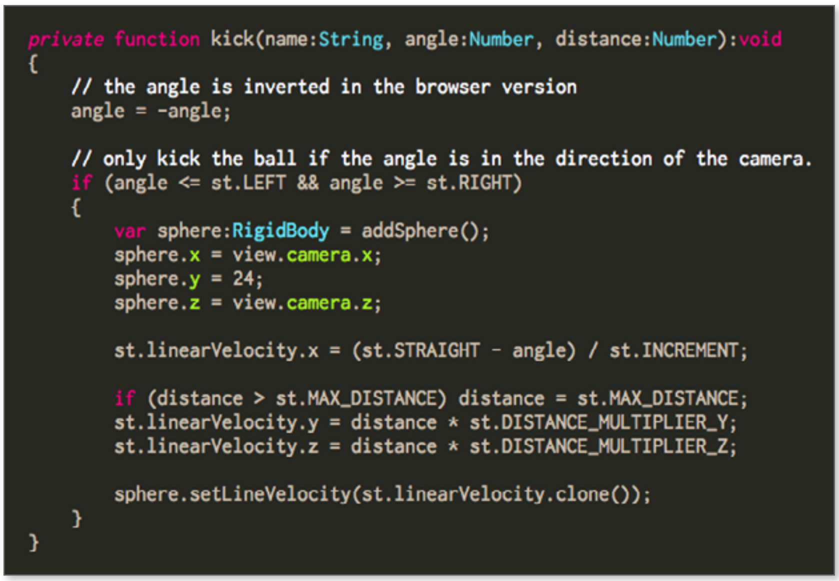


```
function onBallMouseUp(event) {
    ball.grabbed = false;

    if (state.ballStartX !== ball.x && state.ballStartY !== ball.y) {

        target.x = ball.x;
        target.y = ball.y;

        state.angle = Math.atan2((ball.y + state.velY) - ball.y, (ball.x + state.velX) - ball.x);

        target.x += Math.cos(state.angle) * state.speed;
        target.y += Math.sin(state.angle) * state.speed;

        state.distanceX = ball.x - target.x;
        state.distanceY = ball.y - target.y;
        state.distance = Math.sqrt(state.distanceX * state.distanceX + state.distanceY * state.distanceY);

        TweenLite.to(ball, (state.distance * configuration.DURATION_MULTIPLIER), {x: target.x, y: target.y, ease:Expo.easeOut});
    }

    if (outSideCanvas(target, ball.image.width * 0.4, ball.image.height * 0.4)) {
        socket.emit('kick', state.angle, state.distance);
        setTimeout(resetBallPosition, 5000);
    }
}
```

Calculating the angle and the distance of the soccer ball in the 2d world.

In order to test the correct mathematical equation, I created a simple Flash application. The debugging in *JavaScript* was cumbersome, because the language and the tools did not give me enough support. For instance, my Flash development tool of choice, the FDT plug-in for *Eclipse*, gives me code completion, a debugger and a performance profiler (2012). Also, I could never be certain if I was dealing with a limitation in the browser, or if my calculations were incorrect. Copying code back and forth between *JavaScript* and *ActionScript* is fairly easy, so this turned out to be a productive way of testing geometry.

On the other side, the soccer goal app, I calculated a *force* based on the two parameters *angle* and *distance*. After scribbling calculations on paper for roughly two hours, I noticed that the solution is actually quite concise. Some constants were necessary to convert the differing distances (scale) between the 2d and 3d worlds.



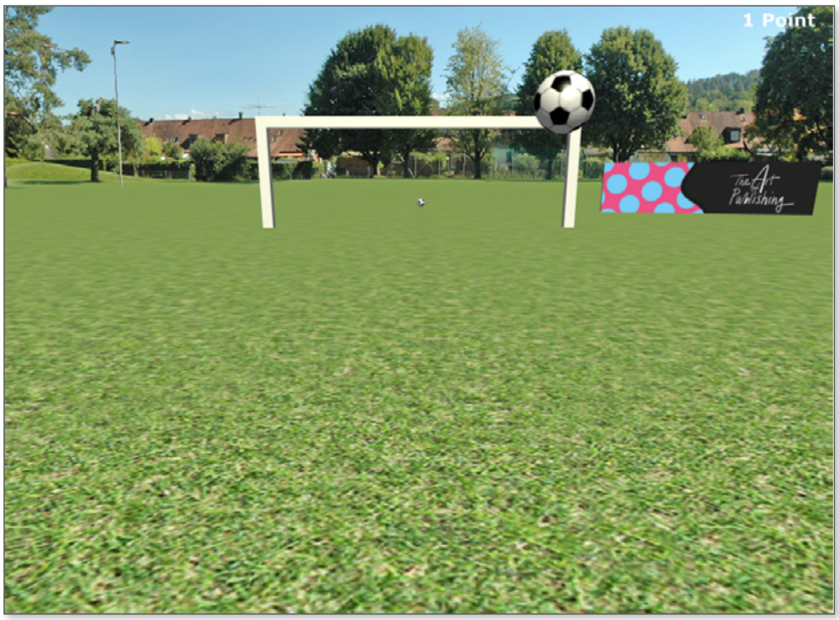
```
private function kick(name:String, angle:Number, distance:Number):void
{
    // the angle is inverted in the browser version
    angle = -angle;

    // only kick the ball if the angle is in the direction of the camera.
    if (angle <= st.LEFT && angle >= st.RIGHT)
    {
        var sphere:RigidBody = addSphere();
        sphere.x = view.camera.x;
        sphere.y = 24;
        sphere.z = view.camera.z;

        st.linearVelocity.x = (st.STRAIGHT - angle) / st.INCREMENT;

        if (distance > st.MAX_DISTANCE) distance = st.MAX_DISTANCE;
        st.linearVelocity.y = distance * st.DISTANCE_MULTIPLIER_Y;
        st.linearVelocity.z = distance * st.DISTANCE_MULTIPLIER_Z;

        sphere.setLineVelocity(st.linearVelocity.clone());
    }
}
```

Calculating the force that needs to be applied to the soccer ball in the 3d world.

July 25th 2012

The apps will start up directly without an introduction screen, so no screen designs are necessary. I visited a nearby soccer field to take a picture of the background image for the game. Further assets, like the icons and textures

were created in *Photoshop*. Sound effects, like the kicking of the ball and cheering, will be added in later iterations.



The soccer goal application.

August 4rd 2012

After creating a POC for each element of the game it was time to put it all together. Instead of starting fresh, as with the previous two games, I used the existing code for the soccer ball app, soccer goal app and server as a basis. After reaching the first working version of the game, the rest of the production was iterative with working versions at the end of each workday.

Some of the added functionality was a goal counter, a branded banner that falls over when hit by a ball, and that a new soccer ball appears for kicking after a few seconds.

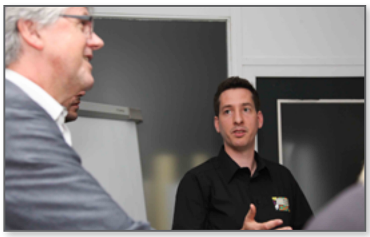
For the presentation to the business network, a high quality projector was set up in the meeting hall of the company. The soccer goal app was running on an iPad that connected to the projector over a wireless network using Airplay and an Apple TV. Printed QR codes[2] in the room allowed for the visitors to take a picture with their smartphones and open the soccer ball app.

The deadline was met and the prototype was presented at my workplace to the participants of the business network. I was not a part of the presentation, this was handled by the company management. They did, however, congratulate me the next day, saying that the demo was a tremendous hit. Keeping the game simple had the benefit that everybody could pick it up and play without much explanation. According to my superior, many of the visitors from this business network kept playing the game until 9 o’clock at night – instead of taking a tour of the company. They only stopped because there was a brief power outage from a thunderstorm and the projector and wireless router got switched off.

2A quick response code, or QR code, is a barcode containing data that can be interpreted by a device when photographed. (Denso Wave 2012)

August 24th 2012

After the success of presenting to the business network, I demonstrated *Flick Kick Soccer* to 50 of my employers clients. They visited my office during a tour of the company in groups of 10 people. I handed out various tablets and smartphones running the soccer ball app. This was a good opportunity to test the usability of the game with a large audience. Some of the older aged clients had problems orienting the tablet towards the screen. One even played the wrong way around. To enhance the usability I should add an arrow pointing forwards. Generally, there was some amazement that the ball could be seen instantly on a separate screen. Some already started playing against each other, commenting on the other persons shot. Some clients asked what the use is of the game, or what benefit they would have from such technology. My answer was that I wanted to demonstrate how data can be transferred live across multiple devices using *AIR* and web technologies.



The game can be played at:
http://nicolas.zanotti.me/flick-kick-soccer/ball/
http://nicolas.zanotti.me/flick-kick-soccer/goal/

Please note that the game needs to be played in separate browser instances (a new tab in Chrome or a new window in Firefox). Otherwise the server cannot recognize the different web clients. Preferably, the soccer ball app should be opened on a smartphone or tablet.

The source files can be downloaded from:
http://nicolas.zanotti.me/sae/flick-kick-soccer.zip

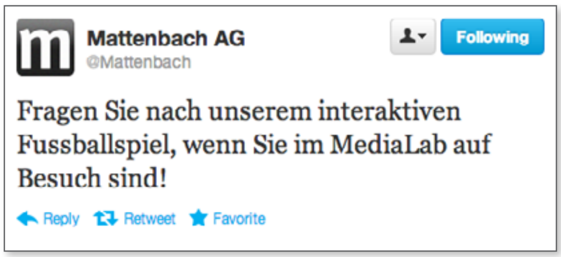
The Readme.md file in the source ZIP contains credits to all contributing frameworks and resources.

September 5th 2012

In hindsight, I would think twice before using *AIR* for producing another 3D game. The tools and libraries are not yet as good as competing products. I think I would have been able to produce faster and achieve a nicer looking game by using *Unity*. I may return to create the soccer goal in *Unity* at a later date, but for now the *AIR* version should suffice.

On the other hand, I still think Flash is superior for creating 2D games. The language and the tools have gone through years of improvements. I felt more comfortable create a POC for the soccer ball in Flash first, where I had a strictly typed language, advanced code completion, a debugger and a performance profiler. Then, porting the code over to JavaScript.

Producing the third game for work had the benefit that I could complete the process, including a release phase. There was a release deadline that needed to be met and marketing was done to promote the game via Twitter and the company's website.


Mattenbach AG
@Mattenbach

Fragen Sie nach unserem interaktiven Fussballspiel, wenn Sie im MediaLab auf Besuch sind!

A tweet about Flick Kick Soccer. Translates to: "Ask about our interactive soccer game when visiting the MediaLab!"

The benefit of playing the game after each feature implementation, was that I had a working version of the game ready ahead of the deadline. Rather than publishing to the *Apple App Store*, I installed the soccer goal app directly (add-hoc) on the iPads of the company's employees.

September 7th 2012

The web release of the game was much more time-consuming than expected. The original version of the game was run from a staging server in the company network. In order to run the game on my own website I needed to have full control of the server (this is usually not the case with most shared hosting providers). A brief test using the *Node.js* service *Nodejistu* (2012) was a problem because of the Flash security settings – the same problem I had when running the app on the staging server. I thus decided to rent my own Linux web server in the cloud and move my existing website there. Multiple days were spent learning Linux server administration and configuring my machine, before having a version of the game running on the open web.

A further issue was that many firewalls block ports by default. Thus, a web socket connection, needed to transfer data between clients, is blocked, making the game unplayable. A workaround for this problem was to use a port that is usually left open for secure data transfers.

Even late in the process of creating a game, unexpected technical difficulties can be encountered. It would be wise to allow enough time when planing the release phase of a game. Presumably, there may also be delays when releasing to an app store such as the *Play Store* or the *Apple App Store*.

October 10th 2012

Today I drove Basel, a city on the Swiss-German border, in order to interview the Jomoho team. This interview would have ideally taken place after the second AR cycle back in June. Since I had the opportunity to produce the third game at my workplace, I decided to instead concentrate my efforts on meeting that deadline instead.

The interview took place in a "startup incubator": special office spaces partially funded by the city for new businesses. Moritz Laass was working on his latest game together with his colleague Simon Siegenthaler. Although the interview was only being held with the game's developer, a mistake I my behalf since I did not ask to interview the entire team, the graphics artist would sometimes yell comments form the neighboring office cubicle. The three of us later went for lunch together and had a very interesting discussion about game design and development. Some of the points were:

- They (Laass and Siegenthaler) believe in targeting a broad audience, from children to seniors.
  Research in the sales of games targeting broad audiences versus targeting specific audiences would be an interesting and useful topic to cover.
- Their efforts to make an anthropomorphic game character to appeal to the general audience.
- Their motivation is not strictly commercial. The also benefit by learning and consider the creation of a game to be fun.

This interview was held in English and transcribed in a little under three hours.

The interview recording can be accessed at:
http://nicolas.zanotti.me/sae/interview_jomoho.mp3

The transcription is included in the appendixes of the research report.

October 12th 2012

I traveled to the Swiss capitol city Bern for an interview with Bastiaan van Rooden and Mark Gruber of Nothing Interactive. The agency has been producing games for many years and is larger and more advanced than the previously interviewed companies. I believe this contrast is to my advantage, since I can see how different the thinking is between startups and established companies.

The interview took just under an hour and was held in German. The translation was done by a professional translator during two workdays and I corrected some parts due to the technical context.

The interview recording can be accessed at:
http://nicolas.zanotti.me/sae/interview_nothing_interactive.mp3

The transcription is included in the appendixes of the research report.

October 15th

With all three games and all three interviews, I have completed most of the data gathering for this research paper. I will continue to read new material and follow the independent game development scene up until the completion of the research paper.

My current work is on the analysis of the interviews. It is important for me as a researcher not to misinterpret any of the acquired data though translation. I have sent the interviewees their transcriptions and translations back and have yet to hear about any mistakes.

December 3rd 2012

Adobe released the *Adobe Gaming SDK* that includes *Away3D* and *Starling* amongst other tools (2012). It is interesting to see, that even during the course of my research, the technology has advanced in great strides. This solidifies my argument, that the tools need to be reevaluated with each new game production. Looking back at the very first entry of this research logbook a year ago, my concerns were the same. In the third game, *Flick Kick Soccer*, I did employ other technologies as predicted.

References

- Adobe (2012), *Adobe Gaming SDK*. Available at: http://gaming.adobe.com/technologies/gamingsdk/
- Away3D (2011), *Away3D, 3D Framework for Flash*. Available at: http://away3d.com [Accessed September 28th, 2012]
- Away3D (2012), *Away3D 4.0 Features*. Available at: http://away3d.com/features/ [Accessed 4th July 2012]
- Box2D (2011), *Physics Engine Box2D*. Available at: http://box2d.org [Accessed February 29th, 2012]
- Box2FP (2012), *Flashpunk/Box2D Integration*. Available at: https://github.com/pdyxs/Box2FP [Accessed April 4th, 2012]
- Corona (2011), *Corona, Ansca Mobile's cross-platform mobile app development tool*. Available at: http://www.anscamobile.com/corona/ [Accessed February 14th, 2012]
- CreateJS (2012), *CreateJS, A suite of Javascript libraries & tools for building rich, interactive experiences with HTML5*. Available at: http://www.createjs.com/ [Accessed July 28th, 2012]
- Denso Wave (2012), *QR Code Features*. Available at: http://www.qrcode.com/en/qrfeature.html [Accessed December 12th, 2012]
- FDT (2012), *FDT is a flexible development toolkit in Eclipse for interactive developers*. Availbale at: http://fdt.powerflasher.com/ [Accessed September 22nd 2012]
- FlashPunk (2012), *FlashPunk - Free Open Source 2D Game Engine for Flash Developers*. Available at: http://flashpunk.net/ [Accessed February 22nd, 2012]
- Git (2012), *Open Source Distributed Version Control System*. Available at: http://git-scm.com/ [Accessed 4th July 2012]
- haXe NME (2011), *NME :: Create high-performance Windows, Mac, Linux, iOS, Android, webOS, Flash and HTML5 applications, written with Haxe*. Available at: http://www.haxenme.org/ [Accessed February 14th, 2012]
- HTML 5 EaselJS (2011), *Easel, a javascript library for working with the html5 canvas element*. Available at: http://easeljs.com/ [Accessed February 14th, 2012]
- Jiglib (2011), *JiglibFlash - 3D Physics Engine AS3*. Available at: http://www.jiglibflash.com/ [Accessed September 25th 2011]
- Google PlayN (2011), *PlayN Cross platform game library for N≥4 platforms*. Available at: http://code.google.com/p/playn/ [Accessed February 14th, 2012]
- Moai (2011), *The mobile platform for pro game developers*. Available at: http://getmoai.com/ [Accessed February 15th, 2012]
- MonoGame (2011), *MonoGame – Write Once, Play Everywhere*. Available at: http://monogame.codeplex.com/ [Accessed 18 February 2012]
- Nape (2012), *haXe/AS3 Physics Engine*. Available at: https://github.com/deltaluca/nape [Accessed February 5th 2012]
- Node.js (2012), *Node.js, platform for easily building fast, scalable network applications*. Available at: http://nodejs.org/ [Accessed July 4th 2012]

- Nodejistu (2012), *Node.js clouds. Simple. Scalable. Enterprise-ready.*
  Available at: http://nodejitsu.com/ [Accessed September 28th 2012]

- Socket.io (2012), *Socket.IO: the cross-browser WebSocket for realtime apps*.
  Available at: http://socket.io [Accessed July 10th 2012]

- Starling (2012), *Starling Framework - The Open Source Game Engine for Flash*. Available at: http://gamua.com/starling/ [Accessed April 4th 2012]

- Unity (2011) a. *UNITY: Game Development Tool*.
  Available at: http://unity3d.com/ [Accessed February 14th, 2012]

- Unity (2011) b. *Publishing* Available at: http://unity3d.com/unity/publishing/ [Accessed February 14th, 2012]

- Zanotti-Schudel N. (2011), *A Comparison of Action Research and Scrum*. Available at: http://nicolas.zanotti.me/sae/cmp-402_theoretical_perspectives_schudel.pdf [Accessed September 9th, 2011]

- Zoë (2012), *a tool for exporting swf animations as EaselJS sprite sheets*. Available at: http://easeljs.com/zoe.html [Accessed April 5th, 2012]