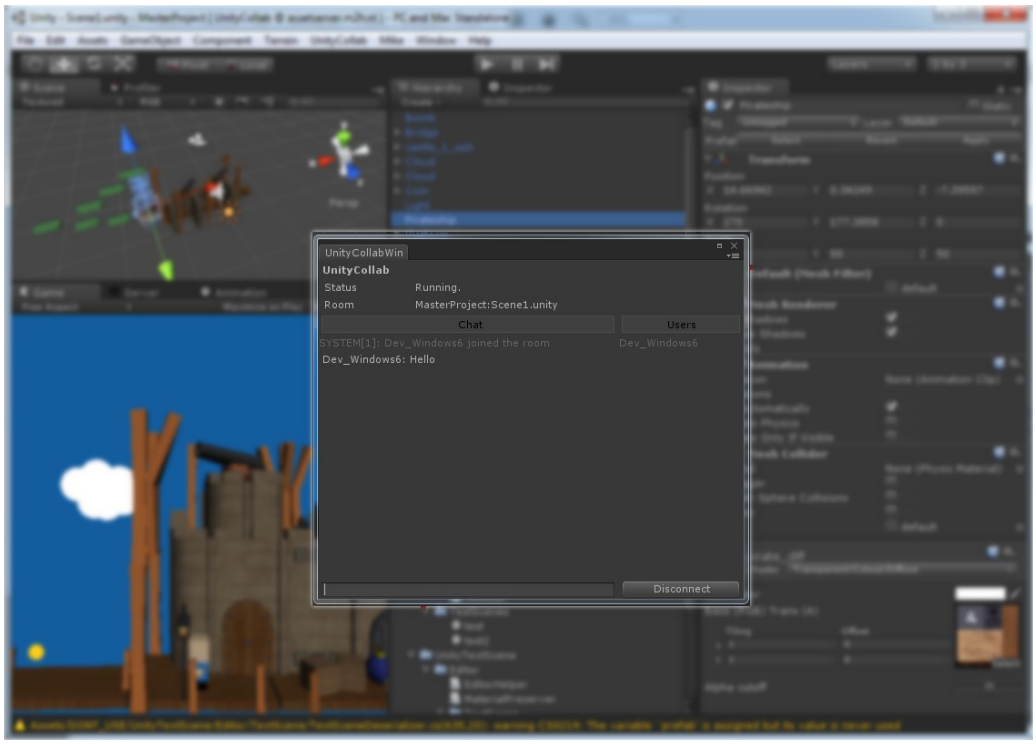
# Collaborative game development




Master thesis - June 2011
Mike Hergaarden - mhn212@few.vu.nl
Vrije Universiteit, Amsterdam
Supervisor: Prof. A. Eliëns
Co-supervisor: Dr. M. Hildebrand

**Abstract**
Collaboration in game development can be a painful process. When several people in a development team depend on each other, miscommunication can often lead to missing deadlines. This thesis discusses collaboration in game development with the purpose of streamlining it. The first chapters provide a general overview of (multiplayer) game development. Lessons learned in multiplayer game development can be extrapolated to collaboration and vice versa. We will discuss the issues involved in collaboration in more detail, and provide solutions to closely-coupled, real-time, collaboration between actual game developers. This thesis is accompanied by a practical implementation of a tool that is aimed at improving collaboration, namely UnityCollab. UnityCollab is a proposed solution that supports collaboration in game development projects. UnityCollab has not been used in production environments yet, but the "lab results" are very promising. UnityCollab will therefore be developed further to be made suitable for production environments.

# Table of contents

# 1. Introduction

The game industry is still a relatively new field with many movements with regards to organization and structure. Lately the *independent* game developers have been attracting quite a bit of attention. These independent developers work in small and, usually, highly effective teams which are not obstructed by bureaucratic systems. Taking this efficiency to the bigger development studios is hard; the more people working together on a project, the more likely it is for problems to arise with respect to collaboration. Collaboration does not seem to scale well. Without proper tooling it is hard to work together efficiently. Tools such as SVN[1] have been around for a while now, these tools support a team in working on the same project semi-simultaneously but do not specifically support real-time collaboration; it Is  asynchronous. Collaboration in the scope of this thesis is defined as working together, synchronous, in real-time. The notion of real-time is important since it is what makes this thesis' research stand out from excisting collaboration methods: Adding a real-time element to collaboration in game development has significant advantages as Chapter  3.4 and 5.1 will make clear. Throughout this thesis the notion 'collaboration' should be interpreted as 'real-time collaboration'.

There are no tools specifically tailored at supporting real-time collaboration in a game engine, e.g. the Unity game engine[2]. Having such a tool specifically made for Unity offers the advantage of being able to utilize domain knowledge to allow for even tighter collaboration support. The practical side of this thesis is concerned with such a tool, named UnityCollab.

This thesis researches collaboration, and more specifically, if (and how) specific tooling could support collaboration. Possibly, the efficiency that larger development teams often tend to lose, can be solved by making use of such tooling.
The following research questions are answered throughout this thesis:

General
These research questions  lay a foundation for understanding game development. This helps to identify the game development process to be able to create support.

- What are the characteristics of the game development process?
- How different is the development of multiplayer games compared to singleplayer games?

Teamwork

From a team perspective, when does it make sense to support collaboration? It is useful to question whether and when collaboration will not have a negative impact on the development output.

- Can (and how does) collaborative editing support development?

To answer this question it is important to clarify the following two definitions:

1. Collaboration can be synchronous(e.g. speech) or asynchronous(e.g. mail). This thesis focuses on a synchronous collaboration solution.
2. The definition of “real time” is relative; it depends on its application. In our collaboration environmental we perceive “real time” with a margin of error of a few miliseconds up to a second. As long as expectations are being met regarding a deadline; the outcome of an action must be perceived as swift and efficient.

The goal of this thesis is to research supporting real time, synchronous, collaboration. In other words: Working together on developing games where we perceive all actions as happening instantly.

Technical

For the implementation of UnityCollab, some technical hurdles will have to be taken. By experience I know it could be possible the Unity API does not expose all required functionality to integrate collaboration support. Furthermore, from a first evaluation it became clear that synchronization conflicts and performance will become the most important technical problems to solve.

- What are the technical limitations and obstacles for collaborative support in the Unity Editor?
- How can synchronization conflicts be addressed?
- How can synchronization performance be guaranteed?

To address the general and teamwork research questions both my own experience and literature research. To answer the technical research questions UnityCollab is developed. Development and testing of this tool will form the answers to these questions. By answering the research questions, the tool to support collaboration is shaped. Firstly the game development process is explored in chapters two to four. This thesis starts with background information about game development and collaboration in game development, after which multiplayer games will be discussed to widen our scope. This helps to identify if multiplayer games can benefit from additional collaboration. Throughout Chapter 5 it will become clear in what regards the UnityCollab tool meets its requirements and what can still be improved upon. This answers the technical questions.  Especially performance and serialization still need to be improved. The

UnityCollab tool is already fully functional for a more limited environment as is shown in the *Lab tool* chapter.

Throughout this thesis the term “Game development” can be interpreted more broadly since the specifics of collaboration when building a 3D game are not as different as e.g. building an architectural visualization tool and other (2D/3D) multimedia applications.

To help put my thesis into context it can be useful to provide a background about myself. I have been creating games since 2005, with a big passion for co-operative multiplayer games. In 2005 I launched my first browser based RPG game. A big change occurred when I discovered Unity in 2008. My interests and activities have shifted from AJAX/PHP/MYSQL based browser games to Unity based games on the web/mobile and standalone. In this process I have founded my own “indie” game company M2H[3]. In 2009 I graduated for a Bachelors degree Computer Science at the Vrije Universiteit Amsterdam. By completing this thesis I will graduate for the Master’s degree Multimedia (June 2011). After graduation I will be able to fully focus on my business in the game development.

# 2. My background as game developer

From 2004 to 2008 game development was purely a hobby to me. By registering the game development company “M2H” in January 2009 it became more serious and I started to make a living with it. After graduating for my Masters, I will probably begin to realize that this hobby has turned into my full time job. M2H is run together with my brother, Matt Hergaarden. He specialized in art and I focus on programming.

This chapter features a chronological list of all notable projects I’ve undertaken. For each game characteristics are listed: date, title, technology, platforms, collaboration and a summary of the project.


Figure 1: Some logos of our projects.

June 2004 - June 2005
**WarriorFields** (http://www.WarriorFields.com)
Technology: MYSQL, PHP, AJAX, HTML, CSS.
Platforms: Web
Collaboration:

> Solo project. Some of the ideas and art were made in collaboration with players and friends.

Summary:

> A text based medieval game. Every player has a spot on the worldmap with it’s castle and units. Players can attack eachother for loot and build up their army and castle.

December  2004 - December 2005
**RobotRevolutions** (http://www.RobotRevolutions.com)
Technology: MYSQL, PHP, AJAX, HTML, CSS.
Platforms: Web
Collaboration:

Solo project.

Summary:

A text based futuristic co-operational game. All players must work together to hold off the alien invasion on the human settlements. They have 30 days to work together and hold of the big invasion.

March 2005 - Now
**Syrnia** (http://www.Syrnia.com)
Technology: MYSQL, PHP, AJAX, HTML, CSS.
Platforms: Web
Collaboration:

Started as solo project. Later on lots of ideas were introduced by the community. Matt Hergaarden made art for the items and work places.

Summary:

Text based role playing game. You are living in a medieval fantasy world. The selling point of this game is having total freedom: specialize in whatever skill you want without the game enforcing restrictions on your character.

July 2006 - Now
**SlaveHack** (http://www.SlaveHack.com)
Used technology: MYSQL, PHP, AJAX, HTML, CSS.
Platforms: Web
Collaboration:

Solo project. Attempts were made to bring more programmers aboard but this never worked out.

Summary:

Text based hacking simulation. Every player owns a virtual computer and tries to collect the best software and make money. Players can hack each others computers to loot software and money or to simply obstruct other players.

February 2007 - Now
**Verdun Online** (http://www.Verdun-Online.com)
Used technology: Unity
Platforms: Mac & Windows standalone
Collaboration:

Out of all the games listed in this chapter, this game is the most interesting in regards of collaboration since it is one of the few games where Matt and me work together with other developers.
The game has been in development for a long time for three reason;
Firstly we've shifted technology a few times. This stopped after switching to Unity in 2009. Secondly it's quite an ambitious project, mostly in regards of requiring many art assets. Lastly collaboration has been troublesome. At the start of the project we've tried to gather lots of volunteers to help create the art assets. In hindsight recruiting and managing those volunteers cost a lot of time only paid off for a few of them. Probably even worse, collaboration in the core team was not properly defined either: Bad communication resulted in low motivation and productivity. There is good hope that all issues have now been addressed properly for a smooth release in summer 2011.

Summary:

Multiplayer FPS game set in the world war one. It's unique setting and game modes should make it stand out from today's first person shooter games.

November 2008

**Crate mania**

Used technology: Unity

Platforms: Web, Mac dashboard widget

Collaboration:

Art was Matt's responsibility and programming mine. As for the ideas it was a mix of both of us. Collaboration went very smooth due to the clear distinction between tasks. This being one of our first Unite games, we both had a lot to learn. Collaboration was not a noticeable hurdle for this project as we were both exploring Unity for our own expertise.

Summary:

Simple multiplayer FPS game with a twist: All players look like crates, in the middle of a crate warehouse, thereby adding a hide&seek element.

November 2008

**Coinpusher**

Used technology: Unity

Platforms: Web, Mac dashboard widget, iOS

Collaboration:

*Same as Crate mania.*

Summary:

A carnival coin pusher machine.

December 2008 - July 2011
**Crashdrive** (http://crashdrive.M2H.nl)
Used technology: Unity
Platforms: Web, Standalone, Mac dashboard widget, iOS
Collaboration:

Same as Crate mania. However, this was a far bigger project, therefore cooperation was more serious here. With the experience of the previous minigames we formed a good team.

Summary:

Multiplayer freeroam racing game. This game was improved in 2009 and 2010 for launches on Shockwave, Netlog, the Mac store and iOS.

April 2009
**Domino creator**
Used technology: Unity
Platforms: Web, Mac dashboard widget
Collaboration:

*Same as Crate mania.*

Summary:

Domino builder minigame.

April 2009
**Surrounded by Death**
Used technology: Unity
Platforms: Web, Mac dashboard widget
Collaboration:

This project was made together with Jos Hoebe. The project was an entry for a game development contest with a one month deadline. Due to this being rush work, collaboration did not stand out very well; Jos made the player character, Matt made the rest of the art and I did the programming.

Summary:

Multiplayer FPS game. Play co-operative with friends and defend yourselves against invading zombies.

April 2009 - Now
**Wooglie.com**
Used technology: MYSQL, PHP, AJAX, HTML, CSS.
Platforms: Web
Collaboration:

Layout design by Matt, implementation by myself.

Summary:

Unity game portal. All unity game developers are welcome to submit their games to this portal and gain advertisement revenue share.

August 2009
**Paradudes**
Used technology: Unity
Platforms: iOS, Web, Mac dashboard widget
Collaboration:
*Same as Crate mania*
Summary:
Simple iPhone minigame, ported to several paltforms.

September 2009
**Bomb factory**
Used technology: Unity
Platforms: iOS, Web, Mac dashboard widget
Collaboration:
*Same as Crate mania*
Summary:
Bomb sorter. Simple iPhone minigame, ported to several platforms.

August 2010
**Cubelands**
Used technology: Unity
Platforms: iOS, Web, Mac dashboard widget
Collaboration:
This game requires far more code than art. So it started as a solo project, after a few weeks Matt added character art and textures for the cubes in the world.
Summary:
Social building game. Build worlds together with your friends and share screenshots on your website profile by the press of a button.

March 2011
**Stacking Island**
Used technology: Unity
Platforms: iOS, Web
Collaboration:
This game was made at the Global Game Jam 2011 in Antwerp by five developers: Jan Boon(Art), Jos Hoebe(Art), Matt Hergaarden(Art), Pieter Hoste(Programming) and Mike Hergaarden(Programming). The programming collaboration went very smooth, I was responsible for leading the programming side. On the art side there

was no clear lead, but art was divided amongst the three artists (Matt the characters, Jos put together houses, Jan made objects for Jos)

Summary:

Strategic building game. Construct your village by stacking houses on top of each other to escape the rising water level.

# 3. Game development

This chapter characterizes game development and outlines how game development is different from regular software development. For those who are unfamiliar with game development this chapter will provide a solid base for the rest of this thesis. The game industry is a fascinating business regarding the different organizations. On one side there is the single developer creating a small game within a few weeks and at the other side there are the big corporate studios making an AAA-class game in 3 years with 100 full-time employees.

## 3.1 What is game development

To discuss game development, we first need to introduce the concept of a game. Definitions of games vary widely. But they often include; rules, challenge, interaction, fun, practice and mental/physical stimulation. As for the technical side, a game engine roughly consists of[4]:

- Rendering system (2D/3D)
- An input system for user interaction
- Sound (effects and music)
- Physics
- Animation system
- Artificial intelligence

To discuss game development, we do not need to go into details about games itself. Game development is the process of making games. It is somewhat similar to the movie industry and 'regular' software development; a balance of art and business. The result of a game does not provide measurable standards. 'Fun' and 'experience' cannot be measured. This can be traced back to the art side of games which makes for sometimes unpredictable results and deadlines. This is just one of the features that make game industry stand out from other industries.

In the next section all involved parties will be outlined in detail, but briefly put; there is the producer that creates the concept for a game, a game studio then develops that game. The entire process is being funded by a publisher. Finally after development a marketing team will promote the game. However, this traditional setup is slowly changing. In the game industry there is a clear distinction between two kinds of organizations. At one side there are the big budget game developers which are funded by mainstream publishers, and on the far other side there are indie (independent) game developers. Indie development teams range in size from one single developer to a small team and are

not (financially) supported by any big company. Thanks to the rise of more easily accessible platforms, such as the iOS and Xbox Arcade platforms, several indie developers have been able to achieve remarkable successes. As a recent example the sole developer of Minecraft has earned millions of euros in 2010 . Like in the movie industry, the big game publishers usually go for the concepts that have proven themselves. Therefore it's usually the indie developers that  innovate. Furthermore, the required tooling has become a lot more indie-friendly as well; Game engines such as Unity and Unreal Engine have special indie licenses that greatly reduce the (initial) investment for beginning game developers [5][6].

When developing a game there are a few key concepts that you need to keep in mind at all times. A list of these concepts is provided below to clarify their importance in this field.

**Funding and Organization** - What kind of organization will develop the game and how is it funded?
The type of funding and organization have a big influence on a game and it's development process. There are the 'traditional' publisher funded games; big companies supporting a large development team. On the other side the indie teams are on the rise[7]; one person, or a handful of people, creating and funding a game independently from bigger organizations. The game budget is of big influence of the type of game produced. This restriction has encouraged some indie developers to innovate by using procedural content, just to save on the art budget. Good examples of such innovative games can be found on the Independant Games Festival website[8].

**Platforms** - What platforms will you be targeting?
Options include consoles and handhelds (Xbox, PS3, Wii, Nintendo DS etc.), phones and tablets (Android, iOS), web (Flash, HTML5, Unity), standalone (Mac/Window/Linux). An engine such as the Unity engine allows you to target almost all platforms, but even then you need to keep in mind that every platform still has it's own specific needs regarding performance, gameplay, marketing, monetization etc.

**Monetization** - Is the game commercially viable?
Assuming the game itself is of sufficient quality, there is still the big question whether it is monetizable. Keep in mind that certain distribution platforms require different monetization techniques. Users of Android are much less likely to purchase games then iOS users are[9]. Therefore currently it's best to monetize Android applications via advertisements whereas the iOS market might perform best using in-app purchases. The gameplay should support and encourage the games monetization method[10].

**People** - Do you have the right people?
In this industry there are quite a few specializations and you're not likely to find a network programmer that can also animate 3D models. When it might be too expensive to have all expertises in-house, contracting/outsourcing is a great option.

**The game concept itself** - Will it be any good?
There must be a strong initial game concept that everyone believes in, possibly even a prototype. Furthermore, all before mentioned items are intended to support the game, but might change it as well. The quality of the final game must never be overlooked. A very important method to ensure the quality of the game is to make sure there is always a playable version. In the concept stage this would be a prototype. In production phase this can be realized by gearing the development towards this goal by developing step by step; Hereby reducing the chance of major overhauls that break the game for weeks.

## 3.2 Involved parties

In the actual game development process we can distinguish roughly four trades. Each of them are specialisms. You will rarely find a person that is skilled in more than one specialism since it is more worthwhile to get the most out of one field. Even so, all of these people do need basic level knowledge about each others trades since close cooperation is required; Programmers needs to implement art and sounds made by their colleagues. Designers need to have know-how about the technical possibilities etc.

Designers
Designs the game, its gameplay, story, dialogue and game mechanics. The designer has the responsibility to create the initial concepts, but also to test and possibly adapt them during the development process.
Examples are:

- Story designers
- Game mechanic designers

Programmers
Working on the actual software and tooling for the game.
Examples are:

- Network programmers
- AI programmers

Artists
Responsible for all the games visuals. Ranging from menu art to 3D models.
Examples are:

- Level artists
- Animators
- Texturers
- 3D Modellers

Audio engineers
Creating sound effects, music and voice overs.
Examples are:

- Musicians
- Sound effect designer

In the bigger organizations it is common practice to structure the teams[11]. All of the above mentioned usually have a better defined structure, with for example one lead member per specialization which directs the specialists. Of course this can be structured

as much as required since every party has it's own sub level specialities, e.g. the artists team could have a lead level designer, a lead animator etc.

There are more parties involved, but these are not directly involved with the development process. For the purpose of this thesis they can be neglected as a collaboration tool would not affect them in the first place. Therefore, these are summed up only briefly:

Management/Business
This party should speak for itself. Depending on the type and size of a company there can be many departments, managers etc.

Marketing
You could have made the next best game, but lacking a lot of luck you will not earn back any investments. Marketing is key to a games success. Marketing starts weeks/months before launch depending on the size of the project. It is important to build a hype before launch as the initial reception is a vital indication for future sales[12].

Testing
Testing is very important. Depending on your deployment platform bugfixes might not easily be possible after launch and this will greatly impact sales. Testing starts as soon as there is a basic playable version and gets more intensive towards the date of release. For a specific period before launch, no new features will be added and testing and bugfixing is top priority.

## 3.3 Phases of game development

This section highlights the three most important phases of game development. Three phases can be distinguished: pre-production, production and post-production[13].

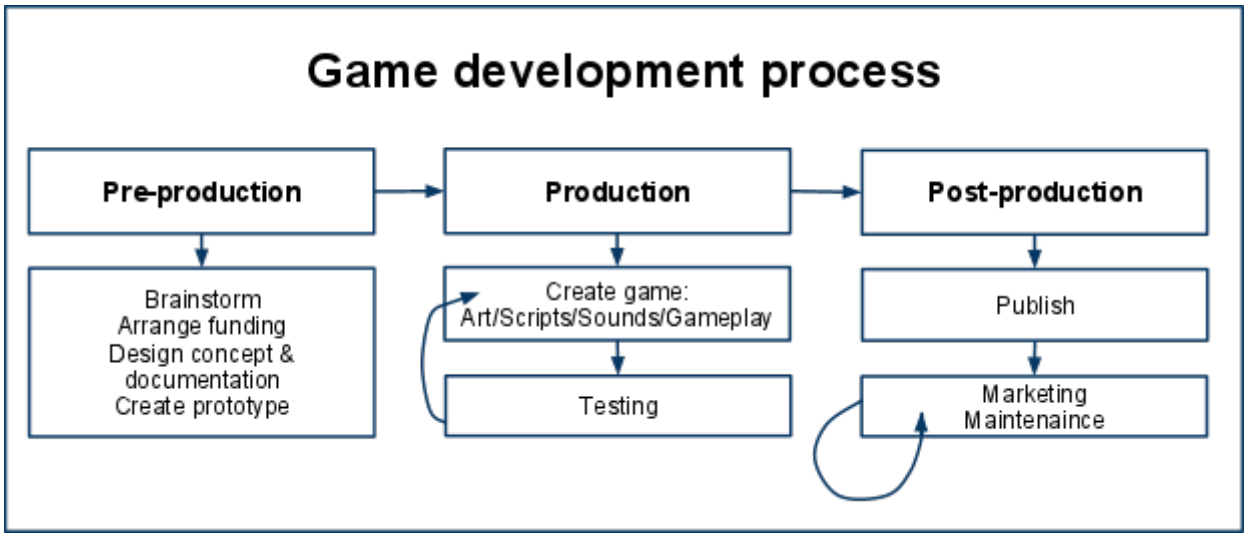



Figure 2: Abstract view of the game development process

**Pre-production:**

Concepts and a game design document are made, the funding and legal issues are taken care of.
Planning/ideas.
Possibly a prototype will be made.

This phase is concerned about shaping the idea for the game and making sure it is feasible. This phase is complete once all required foundations have been covered: financials, planning, game concept.

**Production**

Alpha, Beta, Gold master[13].
Testing.

This is the actual development phase. The game is made incrementally and once the foundation of the game has been laid out testing can begin. An Alpha defines a version that roughly works, but certain features are still missing. A beta is nearly complete, but still requires a lot of testing . A gold master is the final product. This phase ends with the public launch of the game.

**Post-Production**

Launch
Marketing
Maintenance bugfixes/updates

Most of a games success is defined by good marketing. This is the most important post-production task. Depending on the game, bugfixes might be required after launch as well and certain game communities even require regular updates/expansions after the first public launch.

## 3.4 Collaboration in game development

This section points out where, why and how collaboration is visible in game development. Collaboration is most visible between artists and programmers since they greatly depend on each other; Together they need to make sure a game performs well, which is a balance of optimizing art assets and programming. Besides that, because programmers implement art assets, they are dependant on the assets that they are delivered. One example of this are *pivots* (the center for positioning/rotation/scaling) , which need to be set correctly and another example is that animations must meet expectations. These two groups already work according to the designers plans, but when working on their individual tasks there is often no direct communication until an end result is presented. The following graph can be constructed [14]:

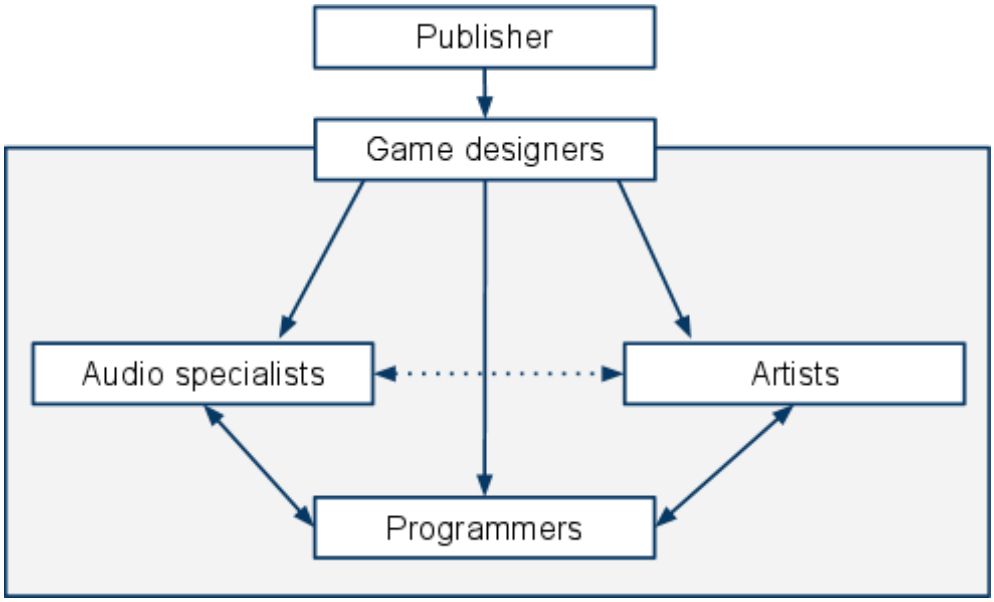



Figure 3: Collaboration in game development

As a note on the figure above, there is obviously some more collaborations between parties then that outlined above i.e. two way collaboration instead of single arrows. However only the most important *dependencies* are shown here.

Let us return to the problem where the programmers have art requirements regarding optimalisation, pivots etc. In a team that has been working together for a while, problems like these disappear in time: everyone will get used to working with each other and their workflow. However, there is a lot of collaboration with external teams because

of outsourcing that takes place for both art and programming, hence why you cannot rely on working with the same parties all the time.

A party that has not been mentioned yet are the audio specialists. Their sound also needs to be placed inside a game and called from the code. While the importance of the sound field cannot be underestimated, the collaboration with audio engineers is not as present as with artists because sound does not imply as many requirements on other fields as art does.

The need for collaboration originates from requiring work from a different specialization. These requirements must match expectation and must be tested for errors. Testing is especially important if it involves cooperation between two parties that have not yet worked together before (remember the outsourcing example). Next to these returning requirements, there are also always newly introduced requirements that are specific for a certain game. These requirements themselves need to be tested. Think of a new use of A.I. for generative sounds. You can send audio engineers a list of requirements but the results of the final product are unknown, this depends on the A.I. system that the programmers create. Thus, collaboration is a required and recurring phenomena.

Examples of collaboration in game projects[15];

- An artists creates models, which need to comply with performance.
- An artists creates models that need to be implemented in game, with various rotations (not via the normal animations). The *pivot* point need to be set correctly.
- Animations; length, speed and naming need to be dicsussed between artists and programmers.
- Art aesthetics (looks of art, but also look of animations) need to match.
- Level design has a big impact on game play; level designers need to collaborate with game designers.
- Level artist need to use the tools that are created by programmers. Some tools will have inevitable requirements/limitations that need to be kept in mind.

The reason for collaboration has been answered in the previous paragraphs. How to collaborate is a different question. There are stories of the early days of game development where post-it notes were physically attached to developer monitors. You were only allowed to touch certain parts of the code or art if you had the right post-it notes attached to your screen. This system was in place to prevent conflicts. To actually share work, floppy drives were used. Technology has improved and nowadays network sharing tools are a standard. Solutions like Dropbox and SVN help developers share and synchronize a project. Certain parts of a project can be locked and conflicts can be merged quite easily. The floppy disk synchronization cost quite some effort and time to apply every time, nowadays tools make synchronization far more real time and

synchronous. UnityCollab, which will be described in a later chapter, is aimed at making collaboration even more real time by allowing truly synchronous development.

# 4. Multiplayer games

While one might not see the connection immediately, multiplayer games are related to the topic of collaboration. Collaboration can be seen as a game on itself. Therefore there are also lessons to be learned the other way around. This chapter outlines the definition of multiplayer games, evaluates the rise of multiplayer games and researches the differences between single- and multiplayer games in regards of gameplay and development.

## 4.1 The shift from singleplayer to multiplayer games

The first multiplayer game was made in 1973. But it was only after the broad adaption of the internet that multiplayer games really took off. However, even recently we see more and more multiplayer content applied to gaming. This trend goes hand in hand with the trend of social games, which work best in a (true) multiplayer environment. Milestones in this area are 'Second Life', 'World of Warcraft' and more recently the social games such as 'Farmville'.

The difference between single- and multiplayer games is that a multiplayer game always has a connection with at least one other player. The ability to play a game simultaneously with other people radically changes the game. All concepts of a game need to be adapted for multiplayer: From preventing cheating to changing the game rules so that players do not need to endlessly wait on each other. With multiple players, infinite combinations of all actions are possible, thus the output is very unpredictable. When integrating interaction with other players in the gameplay, it needs to be considered whether the interaction is beneficial for both parties as otherwise there is a chance that the interactions will fail. Interaction with unknown people over the internet tends to invite more egoistic behaviour.

It is not rare for players to buy a game that contain both a multiplayer and singleplayer mode, but never touch the singleplayer mode. More and more people prefer multiplayer modes. It is more rewarding to most players thanks to achievements, levelling and statistics. Developers notice this and assign a high priority to adding a multiplayer feature to their games, even if this comes at the cost of a lower quality singleplayer mode. Singleplayer games have become shorter (4-6 hours of gameplay)[16], instead multiplayer is added to add more replay value to a game. Unfortunately, with the loss of

singleplayer mode games become less immersive. The reason for this is that storylines are rarely implemented in multiplayer games, and if at all, very limited. Then again, the slow retract of singleplayer games is compensated by the rise of more and more co-operative enabled games where two or more players play a story mode together, which is basically the old singleplayer mode but multiplayer enabled.

## 4.2 Development of multiplayer games

As single- and multiplayer games are very different, so is their development process. Ignoring obvious differences, e.g. adjustments to a storyline, the main multiplayer related differences are the requirements of security and synchronization in multiplayer games. Synchronization is important because the game needs to appear to be running exactly the same for all clients.

The main differences:

- Synchronization
- Security
- Debugging
- Multiple codebases (Client, Server, ..)
- Gameplay

Since there is an inevitable delay in communication between a server and it's clients, there is also a delay in  synchronization. By the time messages arrive at a client they can already be inaccurate. Clients need to have prediction (both interpolation and extrapolation) build in to hide this problem. Furthermore, in the development process it also needs to be decided what information is vital to synchronize over the network to make sure all game states remain the same. This state information should be as minimal as possible to save bandwidth.

Since only vital information is transferred between server and clients, the server can not easily check a clients exact state completely. A client could simulate the network traffic without even running a game. To ensure security, exact the right amount of information needs to be synchronized so that the server can be sure that the client is not cheating while not wasting too much bandwidth.

Debugging multiplayer games is harder than singleplayer games. Both the server and client need to be in a specific state for a bug to be tested. Every bug costs a bit more time to track down than usual.

Multiplayer games require two independent code bases, at least one for the server(s) and one for the clients. There can be multiple types of servers, each an independent application.

There are many gameplay related differences. Those of importance here are the changes that are the result of adding multiple players in the same world. Player behaviour affects other players. If players are able to lit fireworks, it could be possible for players to flock a certain area and fire of a lot of fireworks; this can also cost a lot of performance on the client computers, unless all art is optimized for flocking. Furthermore an important question is whether to allow collisions between players. This may sound trivial but examples like these can be very bothersome to the gameplay.

## 4.3 Multiplayer implementation

This section will provide a brief introduction to multiplayer implementations in Unity. First of all it is important to distinguish the different available networking solutions, as each have their own use cases.

Simple games (1-32 players)

1. Built-in Unity networking solution[17]

Server based solutions

2. Photon http://www.exitgames.com
3. ElectroTank http://www.electrotank.com
4. Smartfox http://www.smartfoxserver.com

MMO solutions:

5. Badumna http://www.badumna.com
6. Netdog http://www.netdognetworks.com

Network frameworks

7. RakNet http://www.jenkingssoftware.com
8. Lidgren http://code.google.com/p/lidgren-network-gen3/

Which of the above solutions is the best to use depends on the target game; Is an authorative server required, how many players need to be able to play together?
If your target game is a simple web/minigame, Unity networking works well enough. If more players must be able to play simultaneously and/or connectivity needs to be guaranteed, any of the other options needs to be chosen. Photon, ElectroTank and Smartfox are all quite similar, supporting social games up to full scale MMO games. Badumna and Netdog are purely MMO networking solutions. Lastly RakNet and Lidgren are basic frameworks for networking with which any game could be developed. However, with solutions one to six available, there is usually no need to reinvent the wheel.

For UnityCollab Photon was chosen. Unity's own networking implementation only works in Unity builds and cannot run inside the editor. Furthermore a dedicated server was required, which narrowed the choice down between Photon, ElectroTank and Smartfox. All of these would have worked well. I ultimately chose Photon because it can be extended in C#, whereas Smartfox is Java based on the server side. As for the last choice, I preferred Photon over ElectroTank because I already had good contacts with the Photon developers.

# 5. UnityCollab: Collaborative game development

In this chapter the UnityCollab tool is discussed in detail. We begin with discussing the goal and purpose of this tool, after which a technical background is provided in sections 5.2 and 5.3. In the last section of this chapter a “Lab tool” implementation of UnityCollab is discussed, which displays the use UnityCollab in a limited and predefined environment.

## 5.1 Introduction

The practical side of this master thesis is concerned with creating and researching a tool that can support collaboration in game development.  The purpose is to create software that allows for real time synchronization of the Unity editor (http://www.Unity3D.com), hereby making possible collaborative development in Unity. This can critically change game development thanks to faster iterations, real time cooperation and easier accessibility.

An example would be, as an artist, opening the Unity editor and start adding creatures to a game level. At the same time the teams programmer notices the new creatures and starts adding code to move the creatures. Both team members will see the same content and see it come to life in their editor without requiring any manual saving/uploading/downloading.

The goal of this project was to research how such a tool can be realized and to what extent. After realizing a basic implementation the consequences for game development have been evaluated. The main research issues are focused on synchronization performance and synchronization conflicts. Furthermore the limitations of the Unity editor are another research issue

### Why?

The first question to answer is why such close collaboration could be useful. On several occasions, even after showcasing the workings of the tool, this question has been raised. There are three main benefits of real time collaboration in game development.

1. **Specializations and their dependencies: Direct feedback**
   In game development there are several trades, all specializations. These specializations bring dependencies since people rely on each others work. UnityCollab supports this by creating an effort-free synchronization. Dependencies will be smoothed out more easily since direct feedback on assets are possible. This feedback is at an earlier stage than without UnityCollab since usually synchronization costs little effort and it is therefore done in bulks only. While a team could try to enforce synchronization more often, the power of UnityCollab here is that it does the work for you automatically.
2. **Speed up communication**
   Communication in teams easily introduces bottlenecks[18]. UnityCollab encourages the communication between different professions. Furthermore this communication can be more specific and quicker since everyone can see the current state of each others assets in real-time.
3. **Fun/motivational**
   In game projects motivation tends to drop at the end of the development cycle. Very often members of the team have made something they are proud of, but it takes weeks for other parties to implement it in the game. With UnityCollab assets can be integrated in the game quicker and easier. The assets are already synchronized out of the box making implementation much easier. Overall progress of the project is also visible in real-time which gives a motivational boost for the entire project.

Since the introduction of real time collaboration introduces a new way of working, it also has it's own unique pitfalls:

1. **Focus and complexity**
   Real-time synchronization can cause distractions since users can be tempted to keep checking the latest status of certain progress. Furthermore, work in progress assets are synchronized automatically which might not be desired.
   Implemented solution(s): Synchronization is only applied per-scene (level), no real time notifications of events are sent to prevent distraction. There is an option to disable updates temporarily.
2. **Conflicts**
   The conflicts that are possible in the traditional asynchronous development methods are still present here, even though perhaps more limited. Assigning ownership/locking of assets is even more important than before since developers can work with any asset they like. UnityCollab does not impose strict agreements about ownership as before.
   Implemented solution(s): None

### What it is not

UnityCollab is not a tool for user generated content, it does not offer advantages for brainstorming, neither for project management. You could argue that the tool has overlaps in these areas and it is true that plugins can be made to make it compatible for these purposes. However, none of these uses have been considered in making this tool. The tool is targeted at collaboration for game developers only; Only for the developers who directly work with any kind of assets within the game engine.

## 5.2 Requirements

The requirements for the UnityCollab tool can be derived from the previous sections that describe the purpose of the tool. The key requirements are listed below.

1. Run inside the Unity editor
   The tool needs to run inside the Unity editor to have access to the Unity API to be able to access all assets.
   Solution(s): Use Unity editor scripting to create an editor application.

2. Embed a networking solution for real time synchronization
   For the synchronization an efficient networking library was required. Furthermore this library should be easy to implement as it is no use to waste time by reinventing a networking library.
   Solution(s): Photon, by ExitGames, was used since it is one of the main networking solutions supporting Unity. Furthermore ExitGames showed interest in this projected and offered hands on support to integrate their solution, making it the most feasable option.

3. Should require no manual work
   The tool should work in the background and require no manual input. The idea behind this is that anyone should be able to use the tool, and that the tool should not require more time to use(and learn) than that it actually saves.
   Solution(s): Start automatically with Unity, automatically detect scene changes, automatically resolve conflicts best effort base.

4. Speed
   The tool should not noticeably affect system performance.
   Solution(s): Limit synchronisation to e.g. 10 times a second to apply changes in bulk only. Furthermore only actual change-sets should be transferred.

## Similar solutions

Next to the “traditional” synchronization tools like SVN, CVS and GIT there is only one product that is similar to UnityCollab[19], namely the HeroEngine[20]. HeroEngine is a full fledged (MMO) game engine featuring built in collaborative world editing.

> *“HeroBlade features an unprecedented toolset fully integrated into a massive, collaborative, real-time development environment. World building, particle effects, lighting, audio, animation wiring, and even game mechanics and core systems can all be collaboratively worked on within the robust HeroBlade client.*
>
> *Massively Collaborative*
> *Imagine entire teams from many geographical locations with the ability to simultaneously work on nearly any aspect of your game’s development. Built from the ground up with collaborative development as its goal, HeroBlade makes this dream a reality. This revolutionary paradigm makes building huge MMO worlds lightning fast!*
>
> *Work from any geographic location.*
> *- Outsource any aspect of the project and remotely monitor its progress in real-time.*
> *- Easily localize the project working in tandem with regional partners and offsite teams.*
> *- Integrated project management and whiteboarding linked to 3D environment and game scripts.”*
> Source: http://www.heroengine.com/features/heroblade-all-in-one-development-environment/, 11 June 2011

HeroEngine’s commercial model is Software as a Service (SaaS), they keep control of the engine and don’t allow customers to customize it. While the lack of soure code access is not a problem per se, the hero engine consists of a broad range of tools fully fledged at supporting MMO games. Since the entire engine is geared towards MMO’s,  the Hero engine is not useful for any other type of games.

# 5.3 Technical implementation

The implementation of UnityCollab was done between mid-January and mid-April. The milestones as described in the master thesis plan(Appendix A) were respected: The first goal was to implement a tool inside the Unity editor that runs in the background. Then this tool was extended with Photon which connected the  editor to the Photon server. Finally, with networking in place assets could be synchronized. This was done using the UnityTextScene project.

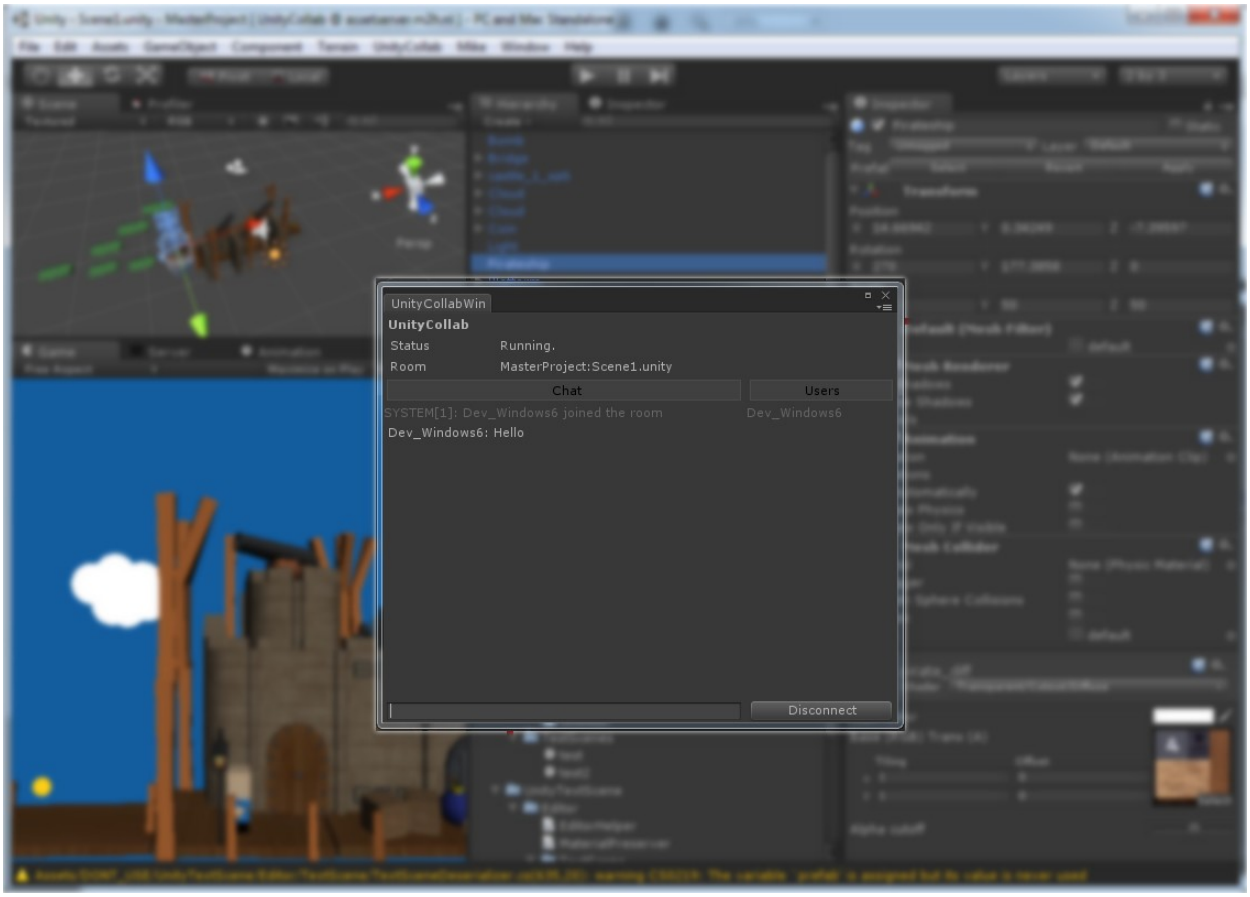



Figure 4: The final result of UnityCollab running inside the Unity editor.

## Unity Editor API

UnityCollab is running inside the Unity editor as a regular editor script. Unity provides an editor API that allows developers to create their own custom behaviours inside the editor. Using this API UnityCollab is called 100 times per second inside the editor. By

making use of this same API UnityCollab is also notified every time a game starts from inside the editor. Because if this happens, UnityCollab needs to stop synchronizing and instead buffer incoming changes. While a game is playing inside the Unity editor, it's level will change, but these changes can not be made persistent; think of picking up coins in a game, these coins would be deleted for this game instance. For this same reason, any incoming changes would also not make sense to apply to the temporary game state.
The UnityCollab tool spawns a EditorWindow and registers delegate calls to *EditorApplication.playmodeStateChanged* and *EditorApplication.update* to realize it's functionality.

## Networking

With the editor extension functioning, the next step was to integrate Photon. For this a Photon example project was used, namely the LiteLobby[21] example. The LiteLobby uses a room based networking logic. Network clients can connect to certain rooms and network communication is limited to the same room only. Using the room logic, the UnityCollab clients are automatically joining a room with as title the currently open scene. This ensures that clients are in the room for the scene they are currently editing. Therefor, networking is per scene only. For UnityCollab the server side of the LiteLobby base did not require any modifications, instead I chose to deploy all logic on the clients themselves, with the first client in a room being a master client. The master client can be useful in case one, and only one, entity is required to make authoritative decisions. In the final implementation making use of the master client has not been required yet.

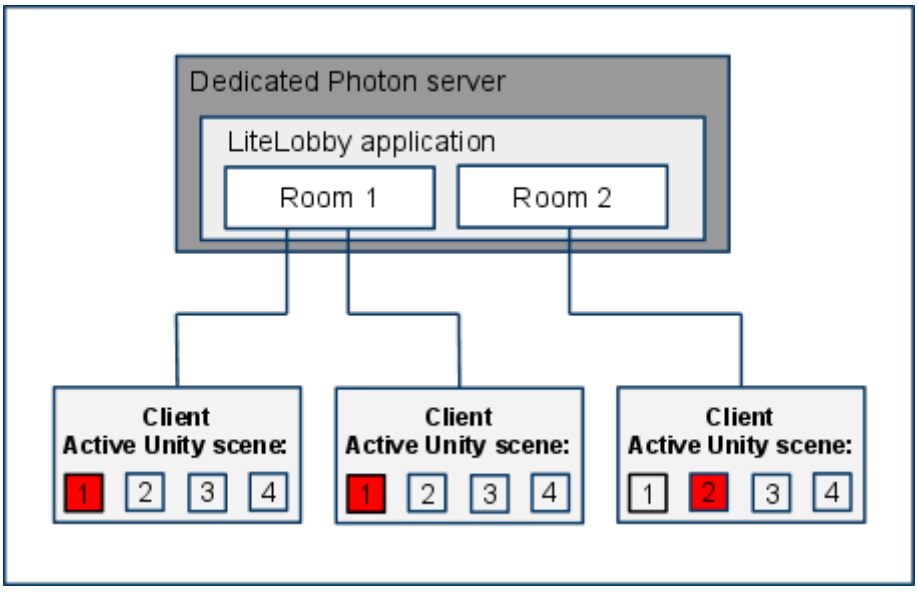


Figure 5: The network architecture

## Synchronization

Synchronization is ensured by checking the state of all local assets in intervals. Every interval the scene is exported to a text variant, then checking whether the text changed. If it has changed, new scene data is send to other clients. When a user is moving an object in the 3D space it is useless to send the changes every interval; this wastes performance. Therefore after having detected a change, it will only send out the changes once one interval remains unchanged. Most likely behaviours, like dragging an object, have finished at such an interval.

The current UnityCollab implementation uses the UnityTextScene project, which is freely available via GitHub[22]. UnityTextScene is a toolset for working with text-based scene files in Unity instead of the built-in binary format. It was made to address the problem that the Unity scenes are saved in binary, which is not mergable when working with SVN and similar solutions.

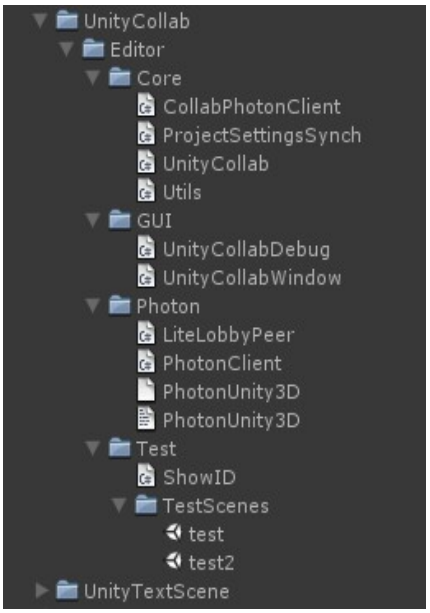
An example of a text representation of a Unity scene is displayed below. This snippet only displays three objects, for the full representation see Appendix C.

```
gameobject Plane
tag Untagged layer 0
0.00000 -1.80522 0.00000
0.00000 0.00000 0.00000 1.00000
1.00000 1.00000 1.00000
children 0
components 3
  UnityEngine.MeshFilter 1
    property sharedMesh builtinmesh UnityEngine.Mesh = Plane
  UnityEngine.MeshCollider 4
    property sharedMesh builtinmesh UnityEngine.Mesh = Plane
    property convex primitive System.Boolean = False
    property smoothSphereCollisions primitive System.Boolean = False
    property isTrigger primitive System.Boolean = False
  UnityEngine.MeshRenderer 5
    property castShadows primitive System.Boolean = True
    property receiveShadows primitive System.Boolean = True
    property sharedMaterials array UnityEngine.Material[] = 1
       builtinmaterial UnityEngine.Material = Default-Diffuse
    property lightmapIndex primitive System.Int32 = -1
    property lightmapTilingOffset primitive UnityEngine.Vector4 = (1.0, 1.0, 0.0, 0.0)

prefab Sculture_02
assetpath Assets/Art/Sculptures/Prefabs/Sculture_02.prefab, a32de86777619244b971db05aae055a7
1.42753 -1.80522 -0.10234
0.00000 0.00000 0.00000 1.00000
1.00000 1.00000 1.00000

prefab wallBrickWet
assetpath Assets/Art/wallBrickWet/wallBrickWet.fbx, adbbf4bb96ccf4c088bf0b8ef30a9001
2.28652 -2.20994 5.65659
0.00000 0.00000 0.00000 1.00000
8.00000 1.00000 1.00000
```

## Files




To get a rough understanding of the workings of UnityCollab, the file structure is laid out here.

UnityCollab/Editor/Core

CollabPhotonClient is the implementation of the Photon networking library.
ProjectSettingsSynch is a placeholder for synchronization of project settings. Not all of these settings are exposed in Unity, therefore it is not implemented yet.
UnityCollab is the main script. It contains all listeners and synchronization calls.
Utils holds various handy functions.

UnityCollab/Editor/GUI

UnityCollabDebug A GUI for debug information.
UnityCollabWindow the main window that starts UnityCollab.

UnityCollab/Editor/Photon

All files in this folder are default Photon files.

UnityCollab/Editor/Test

The files in this folder are used for testing only. I.e. ShowID is an editorscript to print the ID of a selected asset.

UnityTextScene

The publicly available UnityTextScene project. Source code is available at GitHub[22].

# 5.4 Implementation issues

During the development of UnityCollab several issues surfaced. For some of them suitable workarounds have been found, others still need to be addressed properly. The most critical issues were caused by Unity bugs and limitations.

## Unity specific limitations and bugs

Unity Editor bugs:
The first hurdle to take was to ensure UnityCollab is running from inside the editor at all times. Furthermore, it must pause it's functionality when the editor is in Play-state; there cannot be any incoming or outgoing changes here since the scene changes while playing.

It is possible to register a delegate to listen for PlayMode state changes, but this was bugged in Unity at the initial UnityCollab version. After filing a bugreport this was fixed a few days before a new Unity release, otherwise it would have cost several months for a fix. A second issue with this delegate is that it will only properly work when called from an EditorWindow's OnEnable function (EditorApplication.playModeStateChanged). This workaround was pointed out early in the development. To ensure UnityCollab is running all the time, the EditorApplication.update delegate can be used.

API exposure:
Not all of Unity's functionality is exposed, therefore limiting the possibilities. At several stages in the project this posed a big threat. Main issues were whether it would be possible to have a passive application running from inside the editor. And lastly whether it was possible to serialize all assets.

Synchronization still needs to be addressed: It fully works for prefabs (which is i.e. all the lab tool needs) but for any other random asset it does not work yet. UnityCollab started out using the UnityTextScene project for a convenient text based format to exchange through Photon. The UnityTextScene project was limited though; some Unity objects would not fully work since they were not exposed by Unity. To completely solve the synchronization of any arbitrary assets, serialization can be used. This is not yet implemented in UnityCollab.

Besides synchronization of scenes in Unity, projects settings are also important. In the project settings some general settings such as level order is saved. However, Unity exposes only about 50% of these settings. One possible alternative is to try synchronizing the files in which these settings are saved.

## Networking

Networking API's take some time getting used to, and can be quite cumbersome to implement. The network setup of UnityCollab requires a dedicated, authorative, server which could make things even more complicated. However, thanks to the direct support from ExitGames, Photon was quickly implemented. Without this hands-on support the networking implementation could have cost too much time. This issue was evaded.

## Synchronization

Performance of synchronization is still an open issue. The problem is the balance between swift synchronization and the cost of performance; frequency of the updates are in relation with the performance of the tool.

During the lab tool user testing (see 6.4), only two users tested simultaneously since more users would greatly slow each others systems down. The bottleneck here is not the dedicated server but the current synchronization implementation which is very blunt. When a local change is detected, it's entire scene is transferred to other clients. Changing this to a small change set that contains only deleted, added and/or modified assets will dramatically increase performance.

Scaling of synchronization is also a performance issue, since every user will add more frequent synchronizations. Possible solutions to this are lowering the update rate, or sending only synchronizations to a neighbour client, so that all updates are send around in a circle. This will help distribute the load as updates are bulked.

# 6 Evaluation

In this chapter the UnityCollab tool is evaluated. First, UnityCollab is evaluated using the lab tool, this is a limited implementation of UnityCollab for focused testing. This chapter will conclude with overall results of  UnityCollab and collaboration supporting tools in general.

## 6.1 Lab tool

As a proof of concept, the first version of UnityCollab was geared towards only supporting a limited environment. The idea of this lab tool was to allow multiple people to develop a simple game using UnityCollab and a set of predefined assets. Imagine a group of students given the assignment to create a game. Basic objects such as a player, an enemy, obstacles and environment are provided. All they need to do is to combine the desired objects in the scene to create a game and it's gameplay.


Figure 6: A Lab tool game made out of the predefined assets.

The reason for this lab tool setting was to make the tool feasible to implement, since this is a much more limited environment. Quite some UnityCollab features can be ignored at this stage. For the lab tool the base assets can never change and only prefabs need to be synchronized instead of *any* arbitrary object. Next to making the project more feasible, another reason for the lab tool setting was that this use of UnityCollab itself is also quite interesting: It allows for research in behaviour and collaboration in a group of

developers. For example, to research feedback behaviour an add-on can be written to allow instant feedback on changes, rating of each others work, etc. However, this is a completely different research field on its own and has not been explored in this thesis.

The lab tool works as follows. At the far right of the figure above is the project view. This view contains all the predefined assets. From here users can drag the default assets into the Hierarchy or the game view (both represent the games level/scene). As for the basic assets there is all you need for a simple platformer: A player, a coin (can be collected), a bomb (must be avoided) and the rest of the objects are used to make up the level. Users can position, rotate, scale, duplicate and remove objects.

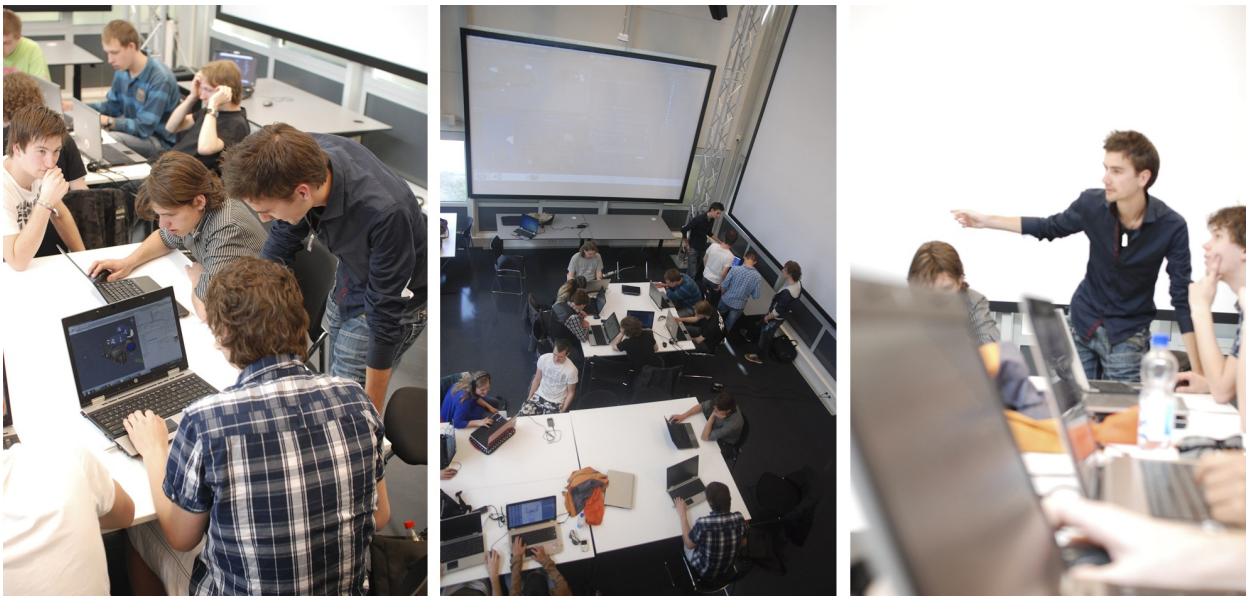


Figure 7: First user test of UnityCollab

On the 18th of April 2011, UnityCollab was tested for the first time, using the lab environment (namely: predefined assets and prefabs only). For this test two computers were used. The results were clear: the tool worked, but as expected beforehand, synchronization was still quite heavy. This was also the reason why only two computers were used. This version of UnityCollab still synchronized the entire scene for every little change, instead it should only synchronize the changes itself: the set of added, removed or changed objects.

Other than the performance issue, the tool itself was a good proof of concept and explained the notion of collaborative editing well. Users did not need to set up anything and no introduction was required since the concept of automatized synchronization 'makes sense' and works in the background. The only skill required for the lab tool is to have a feeling for 3D space navigation such as used in Unity and other 3D applications, but this is unrelated to UnityCollab itself.

Most interesting finding of this test session was to discover how the tool is received: before any explanations people were in doubt whether the tool could be of value to game development and especially how. This does not seem to speak for itself. The participants here were no game developers themselves though, in the game development world itself there is anticipation for the tool[19]. After a short discussion explaining game development workflow everyone was convinced that this tool provided added value because of the advantages as listed in chapter 5.1.

## 6.2 Preliminary results

The current state of the UnityCollab tool is clear; UnityCollab is not yet ready for deployment in production environments. For a lab environment the tool does already work though.

UnityCollab allows multiple developers to work together on a project without having to set up anything: the networked part of the tool is taken care of automatically by making sure that only developers that work on the same scene are put together in a multiplayer room. Users do not have to think about using the tool, the tool automatically handles everything for the user. This includes conflicts: the most recent changeset will simply override earlier changes. In favor of usability behaviour like this is, on purpose, not verified with the user. Tests have shown users had no problem grasping the concept of networked development. The easy of usability of the tool is therefore fully realized.

Functionality and performance wise the tool has two main issues:

1. The tool is too heavy on performance. After every change, an entire scene's data is sent to all other clients, instead of only the small change set. Processing big change sets freezes the receivers computer for an instant. This gets worse with every connected client.
2. Synchronization is limited: serialization is only compatible with a limited set of objects due to the implementation of UnityTextScene.

For the lab tool setting the UnityCollab is already useful as is. Thanks to the use of prefabs synchronization is fully supported. Furthermore there are some optimizations in place to reduce the rate of updates sent to other clients. However, using more then two clients (per scene) will noticeably choke a client.

Solutions for the listed issues are all but complicated. The next chapter discusses the solutions for these limitations and the (possible) future for UnityCollab and similar tools.

## 6.3 Future work

The next steps for UnityCollab as a tool are quite straightforward. UnityCollab needs it's synchronization implementation improved and extended. Performance is an issue and synchronization is limited to a subset of all possible assets. While the performance can be fixed quite easily by improving the UnityTextScene implementation, it is even better to solve both problems at once. This can be done by removing UnityTextScene and replacing it with direct serialization of the Unity assets. It is hard to estimate how much more work that will cost as there is a chance of new issues being introduced. After properly implementing serialization the tool should be ready for deployment.

Having addressed the two standing issues, the following possible additions come to mind:

1. Synchronize the asset folder. This extends synchronization from using only pre-defined assets to changing the projects assets at real time. This removes the need for version control like the asset server or SVN.
2. Synchronization of project settings (Build settings, product name, etc.). The tool is hereby extended from just synchronizing scenes to synchronizing entire projects.
3. Real-time locking of assets in the project and in the scene. Real time synchronization of the assets can cause conflicts, therefore it is required to add the option of applying locks in real time. Both to parts of a scene, as to editing assets.
4. Extensions for project management: From statistics to collaboration enhancements. An example of an enhancements would be to allow users to attach feedback to scene objects to improve level design collaboration.

As for deployment of the UnityCollab tool there is a rough idea of what would make sense for this product. It could be possible to sell this as a service (or product), depending on whether a polished product can be realized and if there is enough time and priority to do so. With the tool this close to being a finished product, it is very appealing to continue development. First of all, the current caveats will have to be addressed. Adding the first three additional features of the list above would be a logical next step.

Deployment itself could be one of two forms: The software could be delivered as Software as a Service (SaaS) or sold as a standalone product. When running UnityCollab as SaaS there would be a monthly fee to cover the server and maintenance costs. The advantage is that this keeps the server 'assets' in own control, furthermore the users don't need to manage a server and can be using the tool right away. Another option would be to supply the server and client code to end-users. An advantage for the users would be that they are responsible and in control of their own data and privacy.

For distribution there is only one logical option; to deploy it in the Unity asset store. This way all Unity developers can buy and update the tool from inside the Unity Editor itself.

A possible roadmap for the UnityCollab deployment:
August 2011: Addressed performance and synchronization (serialization) problems
September 2011: First beta version ready for deployment. Possible extra functionality implemented.
December 2011: First finished version of the product available on the Unity Asset Store.

Taking a broader look at collaborative development, all signs indicate that there is future for these types of products. We see it in all social and collaborative products, from Youtube to collaborative document editing. Collaborative editing will probably gain more grounds in software in the next few years. In the field of office tooling collaboration is getting adopted quite rapidly (Etherpad, Google Docs). I expect the same to happen for game engines, albeit more slowly. From direct contact with one of the founders of Unity it became clear that as of April 2011 they did not have any plans to work on a collaborative tool themselves. They have discussed this at the highest level though. Unity usually makes plans for up to 6 months, so who knows what the future brings. I do believe Unity and other game engine companies are all very aware of the phenomena, it is very likely there will be more collaborative-enabled game engines like Hero Engine in the next few years.

# 7. Conclusion

Game development can still benefit from improving collaboration in the development process. Tooling such as UnityCollab can fill the gap and help remove (communication) barriers. While the UnityCollab tool is not fully finished as of yet, the lab tool has proven it’s use. UnityCollab helps faster-iterations by allowing real-time synchronization and collaboration. A side effect is the motivational and fun factor of being able to work together so tightly. Collaboration is improved upon not by hiding/abstracting details from one others work, but by allowing everyone to see each others work right away and collaborate based on the progress.

By examining game development the general and teamwork research questions were answered in chapters 3, 4 and 5. The characteristics of game development have been laid out by defining it's phases, roles, and general workflow. In addition to that, the development of multiplayer games has been examined in more specific detail. Having characterized game development, an evaluation on supporting development trough collaboration could be made in chapter 5.1. Using these findings, the requirements for UnityCollab have been derived. The technical research questions were answered by the development of UnityCollab: During development several obstacles became clear. the Unity Editor API had it's flaws for which several workarounds had to be found. For all of the Unity limitations workarounds have been found and several bugs have been reported to Unity. Synchronization conflicts did not prove to be much of an important issue for this first version of UnityCollab: in real-time synchronization conflicts have a different nature. There is a smaller chance on conflicts, and if there are conflicts they are noticeable right away which gives both parties the opportunity to repair immediately. Performance still is an issue in the current version of the tool, but solutions for that have been documented in 6.3. These remaining limitations will be addressed in further development, most likely this summer (2011). In hindsight it would’ve been better to implement the serialization of objects manually instead of making use of UnityTextScene. On the other hand, using UnityTextScene was a big time saver and made sure the thesis would not get stuck on the practical side of work and made room to focus on the theory instead.

This thesis shared my experiences in the research and development of a collaborative game development tool. With the game industry now being bigger than the movie industry, there is a lot of interest for middle ware solutions that aid development in any way.  But right now there is only one solution similar to UnityCollab, being Hero engine. It is very likely that collaboration tools will become more popular over the next few years.

Next to UnityCollab still being open for more development, UnityCollab brings to mind new research in various other fields. Not only in the field of collaborative editing, but also research that can be based on this new form of development; such as group behaviour and feedback in “social” game development. While I have made no attempt to research these fields, UnityCollab could be used as a tool for this research.

# References

1: Apache, Homepage of SVN, 2011, http://subversion.apache.org/
2: Unity Technologies, Homepage of the Unity game engine, 2011, http://www.Unity3d.com
3: M2H, Homepage of M2H game studio, 2011, http://www.M2H.nl
4: Prof. A. Eliëns, Constructing a game, 2011, http://www.cs.vu.nl/~eliens/media/11-1.html
5: Unity Technologies, Unity (indie) license information, 2011,
http://unity3d.com/unity/licenses.html
6: Epic Games, Inc., UDK indie license information, 2011, http://www.udk.com/news-udk-licensing
7: Raz Cunningham, The Rise of the Indie Game Developer, 2011,
http://weatherlightblog.blogspot.com/2011/03/rise-of-indie-game-developer.html
8: IGF, Independent games festival, 2011, http://www.igf.com/
9: Kim-Mai Cutler, Let There Be Monetization: Android's In-App Billing Is Out to Consumers, 2011,
http://www.insidemobileapps.com/2011/03/29/android-in-app-billing
10: Jesse Schell, The art of game design, 2008
11: Clinton Keith, Cross Dicipline Collaboration, 2007,
http://www.agilegamedevelopment.com/Articles/Cross-Discipline%20Collaboration.ppt
12: Paul Taylor, Building Buzz for Indie Games, 2009,
http://www.gamasutra.com/view/feature/4117/building_buzz_for_indie_games.php
13: Skylla Janssen, Micha van der Meer, Proffesional playground, 2010
14: Minh Quang Tran, Robert Biddle, Collaboration in serious game development: a case study,
15: Minh Quang Tran, Robert Biddle, Collaboration in serious game development: a case study,
16: Rich Carlson , Making a Case for Short Games, 2005,
http://www.gamasutra.com/view/feature/2292/making_a_case_for_short_games.php
17: Unity Technologies, Unity Networking, 2011, http://unity3d.com/unity/features/networking
18: Claudia Eckert, The communication bottleneck in knitwear design: Analysis and computing solutions, 1998
19: Erik Harg, Collaboration using unity, 2011
20: Idea Fabrik Plc., Hero Engine homepage, 2011, http://www.HeroEngine.com
21: ExitGames, Photon LiteLobby example, 2011,
http://developer.exitgames.com/liteandlitelobbyaddon/litelobbyconcepts
22: Terravision, GIT repository of the UnityTextScene project, 2010,
https://github.com/terravision/UnityTextScene

# Appendices

**Appendix A: Original master thesis plan**
The original master thesis plan; listing various milestones, goals and planning.

**Appendix B: Unity February Newsletter**
The UnityCollab project was officially mentioned by Unity as one of the community projects.

**Appendix C: Textual representation of a Unity scene**
An example showing the textual representation of Unity scenes using UnityTextScene.

**Appendix D: Unity game development resources**
Highlighting various Unity resources that I've made on the last few years.

# Appendix A: Original master thesis plan

## Introduction

This document outlines the goals and planning for my master thesis “Collaborative game development”.

## Thesis synopsis

This master thesis will describe my experiences in game development in general and in particular highlight collaboration in game development. Furthermore it will research the results of a practical implementation in this area. The practical side of my master project consists of creating software that allows for real time synchronization of the Unity editor (http://www.Unity3D.com), hereby making possible collaborative development in Unity. This will truly back up Unity’s latest slogan "democratizing game development" by critically changing game development; faster iterations, real time cooperation, easier accessibility.
An example would be, as an artist, opening the Unity editor and start adding static creatures to a game level. At the same time the teams programmer opens the editor and starts adding code to move the creatures. Both team members will see the same content and see it come to life in their editor without requiring any manual saving/uploading/downloading. To my knowledge there is no comparable software on the market.

The goal of this project is to research how such a tool can be realized and to what extent. After realizing an implementation the consequences for game development will be evaluated. The main research issues are focused on synchronization performance and synchronization conflicts. Furthermore the limitations of the Unity editor functions will very likely be another major research issue.

## Research problem

Collaboration in multimedia application development is quite static; in a team with several artists and coders changes are only synchronized in ‘batches’ when a member of the team decides to manually upload his/her changes to a central server (SVN/Asset server/GIT/etc.). To improve this my master thesis researches the possibility of real time synchronization of multimedia applications. This should allow for faster iterations thanks to real-time and solid collaboration.

# Research questions

**General**

- How can one characterize the game development process?
- How different is the development of multiplayer games?

**Teamwork**

- Can collaborative editing change development?
    - How does a collaborative editor change development?

**Technical**

- What are the technical limitations and obstacles for a collaborative editor?
    - Is the Unity Editor ready for such a tool?
- How can synchronization conflicts be handled?
- How can synchronization performance be met?

# Research methods

**Research game development characterization**

Research the typical game development process. Own and external developer experiences will be used.

**Create a collaboration tool for the Unity development environment**

By creating the actual collaboration tool, it's implementation feasibility and limitations can be discovered.

**Gather feedback from game developers and other experts**

Game developers from the Unity community have already shown a lot of interest in this project, by having several developers actually use the collaboration tool it's use fullness will be evaluated from various angles.

# Initial goals

As the the practical implementation of this thesis is quite a lot of work, it has to be broken down in several subgoals to make the project feasible. My goal is to realize at least the first two goals. Furthermore, limitations of the Unity Editor could constrain the project if the Unity Editor does not expose sufficient functions. This last problem will influence whether goals 3 and 4 are realizable at all.

The following assumptions are used:

A1 No project assets are added/removed.
A2 All persons start from the same version of the project (up to date on asset server or SVN).

The listed assumptions can be removed in later goals, increasing difficulty. However, by design there are the following additional assumption(s):
A3 Synchronization only applies when the editor is not in "play" mode.
A4 Synchronization can be disabled at will.

**Goal 1: Simple editing information**
Allowing for multiple persons to connect to each other from within the Unity editor: See who is working on what scene. No real synchronization as of yet. This lays the foundation for networking inside the Unity Editor.
**Assumptions:** A1, A2

**Goal 2: Simple scene editing**
Allowing for synchronization of position,scale,rotation of existing objects in a scene.
**Assumptions:** A1, A2

**Goal 3: Extend scene editing**
Allow synchronization of new objects and attributes of objects in a scene.
**Assumptions:** A2

**Goal 4: Project asset synchronization**
Allow synchronization of all of a projects assets in near real-time.
**Assumptions:** none

# Plan of action

The master thesis exists of general research and documentation in relation to (multiplayer) game development with next to that the practical implementation of the collaboration tool in Unity. To ensure both subjects are completed successfully documentation is written simultaneously with the development of the collaboration tool.

| | |
|---|---|
| February 14 | Project: Researched networking libraries and completed goal 1. |
| February 28 | Documented section 2: "Game design and development" (see thesis outline) |
| March 14 | Project: Completed goal 2 |
| March 28 | Documented section 3: "Multiplayer game development" (see thesis outline) |

| | |
|---|---|
| April 11 | Project: Explored the feasibility of goal 3, executed if possible. |
| April 25 | Project: Refine goal 3, start goal 4 |
| May 9 | Project: Explored the feasibility of goal 4, executed if possible. |
| May 23 | Documented section 4: “A collaborative Unity editor” (see thesis outline) |
| June 6 | Documented section 5 & 6: “Evaluation” and “Observations” (see thesis outline) |
| ... | Finish remaining documentation and research and polishment |
| June 30 | Finished master thesis. |

# Thesis outline

The following list shows a draft outline of the thesis and is subject to change depending on research/implementational results.

Abstract
1. Introduction
2. Game design and development
    - 2.1 Describe different phases of development
    - 2.2 Collaboration in game development
3. Multiplayer game development
    - 3.1 Multiplayer game worlds
    - 3.2 Multiplayer experiences
    - 3.3 Implementations in Unity
4. A collaborative Unity editor
    - 4.1 Requirements and Design
    - 4.2 Technical infrastructure and constraints
    - 4.3 Implementation issues
5. Evaluation
6. Observations on game development work flow
7. Conclusions

References
Appendices

# Appendix B: Unity February Newsletter

*This newsletter, for the first time, introduces the UnityCollab project to all Unity developers.*

# Unity February Newsletter

## See you at GDC!

**Dear Unity Developers,**
2011 is off to an exciting start – we've been expanding our team (we now are 100 people!), released Unity 3.2 and finalized some exciting partnerships.

Game Developer Conference (GDC)
We hope to see many of you at GDC! On March 2 GDC will be holding a full Unity Track Day, with topics ranging from how to make money with Union to how Electronic Arts uses Unity. We'll also be at our booth (#1416) March 3-5 showcasing our latest demo and products and have a booth theater and Partner Pavilion. For a full list of the Unity Track sessions and the booth theater info, visit our blog.

Unity 3.2 Is Out
Our massive Unity 3.2 release is now available as a free upgrade! This is a really big release, and we've been hard at work ensuring that it is the fastest and most stable Unity has ever been. Some of the highlights include high-performance mobile optimized shaders, beautiful image effects such as a new depth of field with bokeh and improved bloom, numerous tweaks and fixes to existing effects and a new pro water prefab with waves and more. Oh, and bugfixes galore! Check out the release notes for a full listing of improvements and fixes here.

Our Partnership with Sony Ericsson
We're very excited to be teaming with Sony Ericsson to bring Unity-based games to their new PlayStation-certified, Android-based Xperia PLAY smartphone which is expected to be released next month. All features required for Xperia controls are coming to Unity 3.3. It's the first time that Unity has partnered with a platform manufacturer and is great news for the Unity community! Find out more.

### In the Spotlight

Unity was used to create a fantastic web app for the launch of this season's newMcLaren F1 car - the app reveals the new car and allows you to take it around a test track for a spin. Check it out.

Smuggle Truck is a 2D iOS game made with Unity. What makes it stand out is the level editor.

A cool Kinect hack developed to allow you to play a game of Dodgeball using iPhone and iPad.
.

Asset Store Update
We now have more than 10,000 registered Asset Store users, all with our repository right at their fingertips! There's some great new content to check out - slice your meshes procedurally in runtime using Object Cutter, get a head start making an MMO with M2H's Ultimate Networking Project, and splatter blood, gore and bullet-holes everywhere using the Frameshift Decal Framework! Don't forget if you are a content creator, you can sell your work on the Asset Store. Submit now!

Community News
Design3.com is now offering "Android Development with Unity" tutorials, which will teach you how to build and distribute games and applications for Android devices.

**As part of his Master thesis project highlighting collaboration in game development, Mike Hergaarden has been working on realtime synchronization of the Unity editor by integrating Photon. Take a look at the video featuring some of his exciting progress with "UnityCollab" on the M2H blog.**

The Unity community on Facebook has been growing fast - we now have more than 16,000 fans! Like us on Facebook to stay updated on Unity news and interact with fellow fans from around the world.

Create your avatar and travel back in time to live the life of an ancient Singaporean citizen with MMO 'World of Temasek' from Magma Studio's – launching March 3, 2011 at the Singapore Media Fiesta 2011. Born of a cooperation between professional game developers, the National Heritage Board of Singapore and top academics in the field, World of Temase is a non-fiction multiplayer 3D game world that recreates life in the 14th Century kingdom of Temasek located on the island which today is the nation of Singapore.

That's all for now. Stay updated on Unity news on our blog, follow us on Twitter and Facebook.

Lots of Love,
The Unity Team

# Appendix C: UnityTextScene

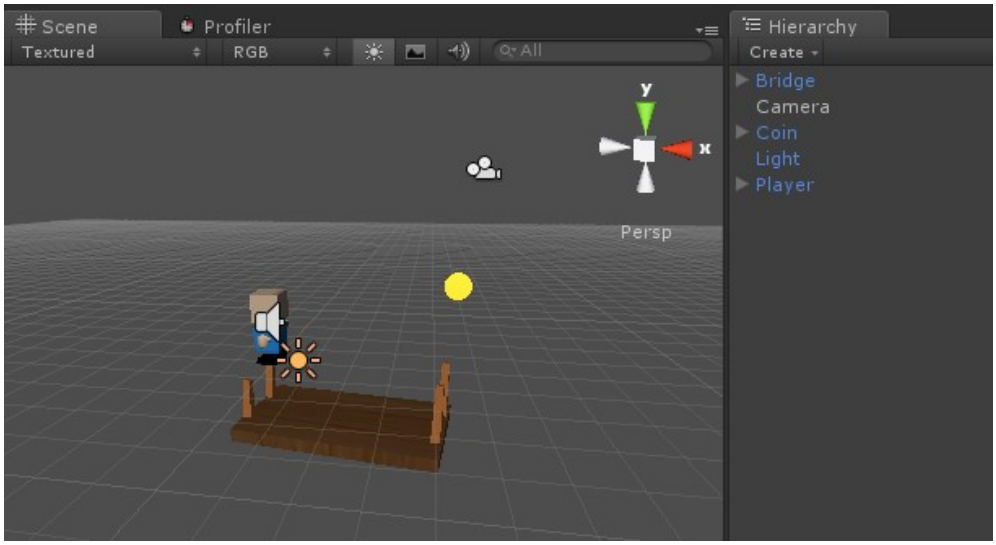


Figure X: The visual and hierarchy contents of the example scene.

Using the scene that is shown in the figure above, UnityTextScene is used to generate a textual representation of this scene. For UnityCollab I have made few changes to UnityTextScene to be able to process and detect changes. As for the output of scenes, the only thing that has changed is that the output is now alphabetically sorted (Bridge, Camera, Coin, …).

As a reminder, the use of UnityTextScene was originally created to introduce a mergable format for Unity scenes, as Unity scenes are saved in binary format by default, which does not allow merging of changes. Using a mergable format multiple persons can work on different parts of a scene and merge the results afterwords. The output of the UnityTextScene tool for the scene shown in the previous figure is shown at the end of this appendix. Note that this output is not so large. However, that's only because this is such a small scene and thanks to the use of so called prefabs (the blue objects in the Hierarchy view).

> *"A **Prefab** is a type of asset -- a reusable **GameObject** stored in **Project View**. Prefabs can be inserted into any number of scenes, multiple times per scene. When you add a Prefab to a scene, you create an **instance** of it. All Prefab instances are linked to the original Prefab and are essentially clones of it. No matter how many instances exist in your project, when you make any changes to the Prefab you will see the change applied to all instances."*
> http://unity3d.com/support/documentation/Manual/Prefabs.html

To summarize, you can define a prefab once, then place instances of it in your project a hundred of times. If you suddently need to change all of the instances, you can simply edit the prefab, save it, and then all the hundred instances in your project as are updated

automatically. However, you can also modify properties of an instance of a single prefab, without affecting the prefab or other instances. If you want to, you could save the changed of this instance and apply it to the prefab (and therefore to all other instances).

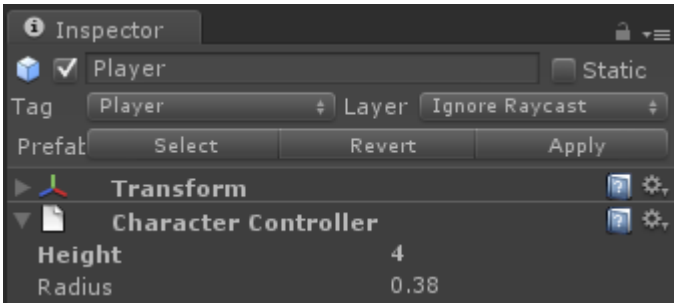



Figure X: A prefab instance with modfied 'Height'

When changing an instance of a prefab, it does not affect any other instances as the change is not applied to the prefab itself directly. As shown in the figure above, the 'Height' setting of a script in this prefab instance has been changed (changes are bold).

Now, the reason for this detour to prefabs is to explain the limitations of prefabs in UnityTextScene. UnityTextScene does serialize prefab instances, but it does not (yet) take care of modified prefab instances, instead it only saves the originating prefab. This was implemented like this to address problems of certain assets that could not be properly serialized. The use of prefabs hides this problem.

**UnityTextScene output of the example scene:**

```
prefab Bridge
assetpath Assets/_PREFABS/Bridge.prefab, c4d3ad65664753a43b6f4a520bcf9417
-0.45909 -1.69337 -3.52566
0.00000 0.75137 0.00000 0.65988
1.00000 1.00000 1.00000

gameobject Camera
tag Untagged layer 0 static False
5.09375 4.55615 -2.68794
0.00000 0.00000 0.00000 1.00000
1.00000 1.00000 1.00000
children 0
components 4
  UnityEngine.GUILayer 0
  UnityEngine.AudioListener 1
    property velocityUpdateMode primitive UnityEngine.AudioVelocityUpdateMode = Dynamic
  UnityEngine.Camera 16
    property fov primitive System.Single = 60
    property near primitive System.Single = 0.3
    property far primitive System.Single = 1000
    property fieldOfView primitive System.Single = 60
    property nearClipPlane primitive System.Single = 0.3
    property farClipPlane primitive System.Single = 1000
    property renderingPath primitive UnityEngine.RenderingPath = UsePlayerSettings
    property orthographicSize primitive System.Single = 100
    property orthographic primitive System.Boolean = False
    property isOrthoGraphic primitive System.Boolean = False
    property depth primitive System.Single = 0
```

```
    property cullingMask primitive System.Int32 = -1
    property backgroundColor primitive UnityEngine.Color = (0.192, 0.302, 0.475, 0.020)
    property rect primitive UnityEngine.Rect = (0.00, 0.00, 1.00, 1.00)
    property clearFlags primitive UnityEngine.CameraClearFlags = Skybox
    property depthTextureMode primitive UnityEngine.DepthTextureMode = None
  UnityEngine.Behaviour 0

prefab Coin
assetpath Assets/_PREFABS/Coin.prefab, 25d10ea2f4d3bab4ba34babd669ddc06
8.65496 7.06987 -5.70229
0.00000 0.00000 0.00000 1.00000
0.65586 0.65586 0.65586

prefab Light
assetpath Assets/_PREFABS/Light.prefab, c2498de037357da4abf671451b293228
1.59966 1.25848 -8.18498
0.30819 -0.06846 -0.02230 0.94860
1.00000 1.00000 1.00000

prefab Player
assetpath Assets/_PREFABS/Player.prefab, 1157d723205776941b9920337e714c61
0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 1.00000
1.00000 1.00000 1.00000
```

# Appendix D: Unity game development resources

A listing of all the Unity game development resources that I've made in the last few years. Most of them were made as part of my Bachelor and Master degrees curriculum. All of the following resources are available via http://www.M2H.nl/Unity.

**Replay project in Unity**
A Unity implementation of in-game replay functionality.
Documentation : http://m2h.nl/files/replay/replay.pdf
Webplayer demo: http://m2h.nl/files/replay/

**M2HCulling: an occlusion culling system**
A document describing Unity game optimization combined with a free culling system that automatically culls objects. While Unity has a built in occlusion system since 3.0, M2HCulling is sometimes still outperforming the built in solution since it's much more simplistic and controllable.
Download: http://u3d.as/content/m2h/m2h-culling/1ui
Documentation: http://m2h.nl/files/Unity_occlusion_system_by_M2H.pdf
Forum: http://forum.unity3d.com/viewtopic.php?t=55686

**Unity game examples in C#**
5 unity minigames as introduction to Unity scripting.
Download: http://u3d.as/content/m2h/c-game-examples/1sG
Forum: http://forum.unity3d.com/viewtopic.php?p=331188

**User customizable virtual worlds**
A document describing the technical and community aspects of Cubelands.
Documentation: http://www.m2h.nl/files/UserCustomizableVirtualWorlds.pdf

**Introduction to (Unity) shaders**
A beginners manual explaining shaders. Contains all pointers you need to get started with shaders in Unity.
Documentation: http://www.m2h.nl/files/LiteraturestudyShaders.pdf

**House of the Future example**
This example project showcases some multimedia features integrated in a single Unity project. Streaming images and video, Wii mote input, music, webcam input.
Webplayer: http://www.m2h.nl/simpleprojects/House.html
Download: http://www.m2h.nl/files/HouseOfTheFuture.zip
Forum: http://forum.unity3d.com/viewtopic.php?p=258336

**Domino builder example**
This example contains a building/simulation game. It allows the player to place domino blocks and start/stop/pause the domino simulation.
Documentation: http://www.m2h.nl/files/ConstructorExample.pdf
Download: http://www.m2h.nl/files/ConstructorExample.zip
Forum: http://forum.unity3d.com/viewtopic.php?t=33577

**Detect hardware**
Quickly check hardware statistics or let a client copy&paste it to you.
Webplayer: http://www.m2h.nl/files/HWstats.html

The following projects are sold (10 Euros and up). They are only available via the Unity asset store.

**Ultimate Unity Networking project**
A complete guide for Unity Networking, from the very basics to advanced features. Also contains code that makes developing new multiplayer games a lot.
Download: http://u3d.as/content/m2h/ultimate-networking-project/1ut
Forum: http://forum.unity3d.com/threads/75385

**M2HPatcher**
An patch solution for Unity games.
A greatly improved version will be launched on the asset server July 2011.
Documentation: http://m2h.nl/files/M2HPatcher.pdf
Download: http://www.m2h.nl/files/M2HPatcher.zip
Forum: http://forum.unity3d.com/threads/62326

**UnityScript/Javascript to C# converter**
A tool to help quickly converting your scripts from Javascript to C#.
Download: http://u3d.as/content/m2h/js-to-c-script-converter/1tH

**C# to Online UnityScript/Javascript converter**
A tool to aid you in converting your scripts from C# to Javascript.
Download: http://u3d.as/content/m2h/c-to-js-converter/1tG

**Unity code obfuscator**
Obfuscate your code thereby making code extraction pretty much useless. Do mind that you can never completely secure your code in .NET.
Download: http://u3d.as/content/m2h/obfuscator/1sA