DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

ConText – A Text/Choice Adventure Game Framework

Paul Preissner

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

ConText – A Text/Choice Adventure Game Framework

ConText – Ein Text/Choice Adventure Game Framework

Author:	Paul Preissner
Supervisor:	Prof. Gudrun Klinker
Advisor:	David Plecher
Submission Date:	15.08.2016

I confirm that this bachelor's thesis in informatics is my own work and I have documented all sources and material used.

Garching, 15.08.2016 Paul Preissner

Acknowledgments

Abstract

This thesis encompasses the development, documentation and evaluation of a framework for creating serious choice based text adventure games for mobile devices in Unity. The goal is to provide a tool and template that enables amateur users to create such educational games with ease as well as to provide experienced users with basic functionality that is expandable with individual modules. Text adventures have a unique way of eliciting the player's full imagination and mobile games manage to grasp a much wider audience than classic stationary systems thanks to much lesser complexity, so it seems logical to attempt to combine both. This paper documents the research and development that went into the project as well as evaluates the quality of the framework and success in reaching the project goals.

Contents

1. Introduction

1.1. Project description

This thesis encompasses the development, documentation and evaluation of the ConText framework. As the title suggests, ConText aims to be a tool that lets the user create text adventures for mobile devices in Unity Technologies' Unity game engine. Considering the ideal usage principles of smartphones and tablets, however, the term "text adventure" needs to be somewhat modified here. The simplest and most intuitive way to navigate interfaces on smartphones is to tap and swipe, as such the games created with ConText fall more in line with choice based adventure games or gamebooks in how they are controlled. The main interest of the chair for Augmented Reality is the usage of this framework for serious, specifically educational games. Additionally, ConText is intended to provide both simplicity so inexperienced users can create a game without the need for specialized knowledge as well as expandability, so developers with existing Unity, C#, JavaScript or otherwise relevant knowledge can use the framework as a baseline and only need to add whichever additional parts they require. While similar solutions do exist already, the author determined that most of them that were readily and freely available either only offered simplicity or expandability, or were only available for programming languages or barebones tools, thus lacking the immediate potential functionality provided through Unity, such as efficient and extensive 2D and 3D rendering, mobile platform integration and active community support. The framework is a plugin made in and made for the Unity game engine in version 5.x. Development first required research on what existing solutions offered in possibilities, on what existing mobile text adventure games offered in features as well as specialist literature and proceedings on the topics of interactive storytelling, serious games and usability. Development included working out an initial concept, implementing it in Unity with C# and continuously reiterating and adapting the concept to stepping stones that turned up along the way. A later stage was user studies, which consists of two separate user studies and their respective evaluation regarding the user experience and usability of ConText. The sections of this thesis appear in the order this introduction set up, first covering research, followed by development and user studies and concluding with an outlook and evaluation of the thesis results.

1.2. Project goal

In the spirit of the project description, the goal of this thesis was to ideally end up with a tool that provides a sufficient baseline for creating mobile text adventures, providing both a simple and intuitive interface for amateur users and a flexible, modular and expandable structure that allows experienced users to add their specific functionality relatively hassle-free. The user studies should provide feedback on what features are prime candidates for included or future implementation and how successful the project was in reaching this goal. Sufficiently reaching the goal would set ConText up for actual public or prolonged use as well as future expansion into a more refined and feature rich framework, topics covered in chapters 4 and 5. Content quality and gameplay in terms of the use of provided mechanics of the created games are not considered since this thesis focuses on the development and evaluation of the framework alone. Specifically the provided gameplay mechanics are not regarded as they only provide atomic functionality and their effect in a game highly depends on the specific use and implementation for the game.

2. Research

2.1. Inspiration

The concept of text adventures, choice based games or gamebooks is by far not new, and dates back long before electrical computers were even invented. They are a type of game that work almost inherently well on any type of device capable of displaying text and accepting input, as that in its most basic form is all they need. However, the question with every type of device remains how that is done intuitively and most true to the user's preferred consumption method with that device. With modern smartphones and tablets, the most popular input methods are tapping and swiping, and extensions of those to perform more complex actions. For a text adventure, this is contrast to the classic physical typing on a keyboard or even the pointing and clicking with a mouse. As such, it is clear a mobile text adventure ideally adapts and adopts tapping and swiping. In the spirit of the aforementioned game genres, that means swiping through texts or other forms of content and tapping items, replies or similar. Another aspect of these types of games has always been the deliberate lack of visual stimuli. Much like the classic understanding of books, they aim to elicit purely imaginary experiences. The user should mentally picture the events, draw their own version of the world, and not be bound to an artist's specific interpretation. One game in particular that we think captures the essence of these two sides very well is the Lifeline series of games developed by Three Minute Games in 2015 and 2015 for Apple iOS and Google Android. The player is confronted with an at the beginning unknown world and one character. Descriptions of the world and events are delivered through the eyes of that character, and all the player can sometimes do is choose between two replies to a prompt by the character. Despite this simplicity, through the course of the game, the player envisions the entire world, the looks, the events and everything as they see fit. At the same time, the game is simple enough to be played even while being mobile with a smartphone in hand and does not require constant attention. This sparked the question about what options are available nowadays for creating this type of game. As looking at related work showed, the readily available options are often lackluster, but also provide valuable insight into what features need to be combined into one tool to provide satisfying overall functionality to both inexperienced and experienced user groups.

2.2. Related work

When analyzing related work, several categories were sampled. For one other popular text adventure games, both mobile and web based, to see what resonates well with audiences so far. Secondly other frameworks designed for the same or similar purposes, creating text adventures, to see what feature sets are available and seen as worthwhile by developers. Lastly, several specialist literary works on interactive storytelling, serious games and usability. Additionally, some functionality was deduced through logical inference and, frankly, common sense, since some basic features are simply necessary for such a framework to function at all. Whenever a game was analyzed, a list of relevant features was extracted and matched up with the existing feature list in order to determine which additions or changes may need be made.

2.2.1. Games

Lifeline The first game analyzed was Lifeline by Three Minute Games on iOS, as it also posed as inspiration for the thesis. What can be seen in the game is that simplicity is key when aiming for accessibility with a large user group of smartphone users of all skill levels. Lifeline offers a simple structure of 1:1 communication between the player and one game character through a sequential text message stream. The interactive part for the player is reduced to the ability of choosing one of two replies when prompted by the game character every few messages. The story is acted out in real time, or rather with 1:1 time scaling. When the character claims to need several hours for an action, the game will accurately remain inactive for that long and only notify the player once as much time has passed. A brief settings menu allows trivial changes such as toggling sounds and music, changing language, and more relevant to the game, to rewind the story to a previously reached decision and to speed up the gameplay by way of reducing UI effects and animation. The player can always scroll back through the entire story and through these settings choose a particular previous state once they have completed one story branch. What could be gathered from Lifeline is that ConText would need at least a basic dialogue system with multiple reply options offered to the user and possibly timed firing, as well as a mechanism to save new and load existing story progress. Additionally, it would need to provide at least a simple UI with potential customization for the user, as well as possibly settings for language support, sounds and the UI.

Storynexus.com Another sample was taken from a web based service called storynexus.com. Storynexus is a website where users can create and more prominently play text adventures through their web browser. The story "The Thirst Frontier" was

chosen for analysis. It is mostly reminiscent of choice adventures or choose-your-own-adventure stories in that the user is guided through a story with partially branching story arcs and with every new page is asked to choose from available options that can include virtual actions or dialogue replies. The story is segmented into locations or chapters of sorts. While it does not offer 1:1 time scaled playback as Lifeline, it does stand out through a more complex item and character quality system that can additionally influence the success of attempted actions or their mere existence. This analysis expands the list of viable features for ConText by a system for splitting the story into chapters, which may not reflect in the created App but for clarity in the framework interface. As a possible secondary feature it promotes extended player properties that influence and are influenced by how the story unfolds, as well as a dedicated inventory for items to be used in gameplay actions.

Other (?) TODO. e.g. Moritz Becher's Ludus? prime reference actually

2.2.2. Frameworks

Quest TODO.

WibbleQuest TODO.

2.2.3. Literature

On E-Learning, Edutainment and Interactive Storytelling technology TODO.

On usability and user experience TODO.

3. Development

With research done, the list of proposed features as well as the basic requirements and constraints of the tool had to be compiled into a sound implementation concept. As is usual and likely inevitable for development, changes had to be made over the course of the project and certain obstacles were hit, both covered in the latter two subsections of this chapter.

3.1. Concept

Technical constraints Constraints presented through the platform the framework is to be built on are mostly bound to Unity engine. Since the Unity toolset only works on x86-based computers running Windows 7 or up or Mac OS X 10.8 or up and requires graphics hardware supporting DirectX 9 with shader model 3.0 or up [Uni16], the same applies for ConText. Since the source code for ConText is entirely written in C#, code modification on Mac OS requires an IDE capable of handling C# such as MonoDevelop. As Unity Technologies put it, "The rest mostly depends on the complexity of your projects", at least for the development system. Depending on the target platform the resulting game is ported to, additional constraints are the availability of the respective devices with sufficiently recent operating system versions.

Layers With these underlying system requirements given, the ConText framework itself was envisioned as a layered construct with modular structure both internally and externally. Ideally, this would provide a high degree modularity and flexibility. The layered construct borrows an idea similar to that found in standardized system layer definitions with categories of actions and objects in the framework assigned to different layers, with communication or interfacing function calls happening between the layers. This would encapsulate and decouple similar functionality and make it easier to identify and modify or expand specific parts. The following layers are part of the concept:

1. The **Manager Layer** consists of all managers, which essentially represent the central logic handling the communication, tracking and updating of the game elements. The following managers are part of the Manager Layer:

a) The **Module Manager** which would keep track of all story modules in the storyline, provide functions for loading modules in or out, prompting the Log Manager to trigger log entries and the UI Manager to display modules in the game UI.

b) The **UI Manager** would keep track of the UI objects for instanced story modules, handle scaling of the UI on the playing device and provide functions for displaying story modules, log entries and other info.

c) The **Input Manager**, intended to handle (touch) input and redirect or trigger the according reactions to the respective affected managers.

d) The **State Manager** which would handle saving and loading of story progress, keep track of the current game state and generally juggle the bits of data that don't fit into any of the other managers.

e) The **Log Manager**, to keep track of all log entries and provide functions for loading and updating entries.

The Input Manager would directly communicate with the Input Layer, the UI Manager with the UI Layer and the Module Manager with the Module Layer.

2. The **Input Layer** is very compact and only contains immediate input handling, basically entirely computed by Unity's internal input handler. As such it might be seen as only a theoretical layer, or at least not an actual layer of ConText itself.

3. The **UI Layer** which contains the UI settings, visual ingame representations of the text/story module stream, the log list and a UI wrapper to channel calls and communication of the former to the UI Manager.

4. The **Modules Layer** is essentially the whole collection of modules and module instances and directly communicating the the Module Manager to exchange information about next modules and the storyline's elements.

Where the layer concept actually differs somewhat from the usual definition of layers is that they don't communicate exclusively vertically. The Modules, Input and UI Layers all communicate with and via the Manager Layer.

Modules The initial concept did not contain specific implementation details for the inner logic of individual parts, however the internal messages were defined as far as that they were to be designed as modules, with each module encapsulating one message and containing pointers to both the previous and next modules, akin to a

double-linked list, the message data such as the respective sending character, the message content and a unique message ID and functions interfacing in a given way with the managers.

As such, the user would be able to plug together the story with a list of these modules, or even expand the system with their own modules if they want specific functionality. It would only require them to implement the specific details of the interfacing functions to work in line with the module system.

3.2. Final structure

Regarding the final structure of the ConText framework an overview with consideration of the differences is presented first, followed by a more detailed look at perhaps the most central part, the message module system.

Overview For the development system used, the same technical constraints as with the concept are present, as Unity is still used for the tool. The layer structure is mostly retained with Input, Manager, Modules and UI Layers still in place, but communication between the layers is not as channeled as in the concept. Most calls from Input, Modules and UI still happen with and via the Managers layer, but both Input and UI layers also directly communicate with parts of the Modules layer. Thus there is no real distinction between horizontal and vertical layers and the structure is more akin to components in a graph than strict layers. As such one proposed change for a future revision would be to assess and rework the irregular communication between the components and decouple them in a more organized manner.

The Manager Layer has seen the most drastic changes since the initial concept compared to the other layers. The Input Manager is removed entirely as the input is redirected straight to the UI manager and UI objects and state manager changes related to UI actions as well. The State manager concept was essentially split up into the State manager keeping track of the current game state and the Unify class holding references to all managers. It is proposed the two are merged into one class. Additionally, a Story Settings part was split out designed to hold data related to the story properties of the game, namely a list of characters with their respective properties as well as the default message sound and background track for the game. The Module layer calls directly to the Story settings to get relevant data instead of requesting the information through the Module manager. The Module manager retained most of its concept design with the only differences being that module loading and storing is now managed by itself entirely and does not communicate with the State manager and that the UI layer can restart the module stream once it is stopped (while the concept envisioned

the Module manager to keep exclusive control over the module stream).

The Input layer is even more of a theoretical construct than in the concept, as it turned out that all relevant input handling is done entirely by Unity's internal systems. As such it is still a layer of the structure but not in any way part of the ConText code base. In the UI layer the text stream and log reference are stripped from the final implementation as the UI manager accesses the objects contained within the two parts in the concept are now directly accessed by the UI manager. What remained are the UI wrapper which keeps references to some of the UI menu's controls and parts and handles transitions between the three main UI views, as well as the UI settings which adhering to its title keeps track of settings such as the mapping of message modules with their UI templates and of visual properties of the UI views and modules.

The Modules layer is implemented almost entirely as envisioned in the concept, with the small difference that the unique message ID is constructed from multiple partial IDs instead of one single number.

Module system The module system is by far not the only complex part of ConText, but it is certainly the most central part. As mentioned before, the story is made up of modules, which each encapsulate a message of some sort. As the game should be able to offer not just text, but perhaps also images or sounds to keep up a sense of variety, so the message modules have the following structure:

1. The **previous module**, which will always have been fired right before the current
2. The **message ID**, made up of four parts
   a) A *sequential part*, which denotes the message's sequential ID within its branch,
   b) A *branch part* denoting the branch the message is in, respective to the latest prior module with multiple outgoing connections,
   c) A *hierarchy part* representing the hierarchy level of the message, with the hierarchy increasing for each branching module,
   d) A *subpart ID* which is only calculated internally when the message has to be split up into multiple partial instances.

   With this structure, each message has its unique identifier. The hierarchy part first specifies which level the message is on, after which branch split it follows, the branch part specifies which of the split branches it lies in, the sequential part then specifies which in line it is in that branch, and the subpart part specifies another order when split partial messages between two consecutive sequential messages happen.

3. The **sending character** as in the character the message belongs to. The character does not need to be an actual person of the story if the message is not to be sent as a message in the visually classical sense, for example one might call the character "Narrative" or not give it any name and set the message property to disregard alignment in order to display it more like a simple text field.

4. The **message send delay** or **delay before send** in seconds which specifies how long the automatic stream should wait before deploying the message.

5. The **message sound**, not to be mistaken with the default message sound in the Story Settings. The latter is a sound played every time a message is triggered, the former is only played for this specific message and could for example be used to imitate a voice message.

6. The **message content** which is fully dependent on the message or rather module type. Default included in the framework are modules of with the content types **Text**, **Image**, both self-explanatory, as well as **Tic Tac Toe** to show how a mini game may be used to provide more involved gameplay to the user and **Reply** which provides the player with several choices to reply to a message and can be used to query knowledge, opinions an direct the story along different paths. These types specify which form of content can be set.

7. The **log entry** which may contain additional information not fit for the message itself and which can then be viewed in the Log view.

8. The **next module** analogously to the previous module specifying which module comes next. Depending on the module type, there can be multiple next modules, such as with the Reply module. To note here is that implementation is subpar as a single next module is part of the basic module type, but multiple next modules are additions made only to the code of branching modules. A proposed change is that next modules be a list with variable length in order to enable better compatibility with calls working with next module data.

The communication procedure for message modules is best explained through an example. Assumed be the following parameters:

1. The framework is correctly configured.

2. The story consists of the modules
   a) M1-T of type Text Module,
   b) M2-R of type Reply Module and

c) three Mi-T of type Text Module with 3 <= i <= 5,

with M2-R being the successor to M1-T and M3-T, M4-T and M5-T as the replies outgoing from M2-R.

3. Playback starts with M1-T as the first story module.

Once the story playback is started, the module manager will initiate the automatic text stream. As the first module is specified as M1-T, the manager will access M1-T to grab the sending delay and message sounds to delay the message firing accordingly and trigger then sound. Then it will pass the module object to the UI manager for instancing. The UI manager attempts to find the correct UI template mapping for M1-T's type and instance a GameObject using the template. Next the module's setContent function will be called with the template instance as a parameter in order to input its specific message info into the UI instance. The details of this are down to the specific type, but for the case of M1-T, the setContent function will put the message's sender and message send time at the top of the message bubble, the message text as the main text portion and adjust width and side padding for correct alignment, quite like a text bubble in a common messenger app. Once the content is set, the UI manager takes over again and adds the instance into the Text View's scroll area, scrolls to the bottom to make the new module the latest displayed and finally adds the instance to the module manager's list of instanced messages. At this point, control is returned to the module manager. With M1-T's instancing completed, the automatic stream attempts to grab the next module in line via the getNextPart function implemented in each module type's class. This will return whichever module is set as the next module or a null return. When null is returned, the automatic stream will stop and be halted until something starts it again, such as specific user input. In M1-T's case, the call will return M2-R. The automatic text stream's loop starts from the top again, computing M2-R's UI template mapping, waiting the send delay, playing the message sound and handing over the module to the UI manager. Once the content is set for M2-R, the module manager will again request the next module. This time the call will return null as the Reply module offers reply choices to the player and does not know the next module until the player has made a choice. Once the player chooses an option, the corresponding option will call the restart function of the module manager which will continue the automatic stream with the now known next module. As such, the stream loop will again start from the top with M3-T, M4-T or M5-T. A similar system is used when loading previously saved story progress, excluding any sending delays or user input interrupts.

3.3. Stepping stones

Along the way, development of ConText stumbled across a few difficulties, posed by discrepancies between the concept and expected implementation versus the viable possibilities offered through Unity as well as general concept insufficiency.

Game UI The UI for games created with the framework use the "new" UI system introduced with Unity 4.6 and Unity 5.0, which is based on canvas-object relationships, full material supported texture drawing and internally managed screen space dependent scaling and alignment instead of the previously used low level primitive drawing functions. This allows for easier creation of complex and detailed user interfaces but also poses new constraints when using higher level features of the system such as automated alignment or ordering, the latter of which turned out to be a longer running hassle to get working right. Specifically, the module stream should eventually automatically display the modules in correct order, correct scaling and allow the user to scroll through them, with correct scaling being the crux here. The UI system offers settings and components for measurement limits and dependencies, but at first sight is inconsistent in how it applies them across GameObjects and even object types. Hierarchical structures need to be set up in a more complex and unintuitive way to get the scaling of low level items working right and based on their content. A solution was found on the Unity3D forums eventually, along with a reply by a Unity Technologies employee indicating the UI system was designed with certain such drawbacks known but accepted for performance and simplicity concerns [run16]. The alternative to this solution would have been to create a custom approximation for message module scaling, requiring a lot more time and likely increasing complexity of module code.

Inspector heritage Another problem encountered is mostly to be attributed to subpar planning and vague concept design. The initial concept did not consider the number of interfacing functions the module classes and their editor inspectors would require, and as such first implementation only had a select few calls integrated into the parent class to all module inspector types. It was not until close to the project deadline that a major refactoring attempt was made and many parts of the respective classes were centralized into the highest parent, eliminating the majority of duplicated module inspector code and reducing the respective code files' footprint by more than half. Similar happened on a smaller scale to the module classes throughout the project as well. It is proposed a more ideal modification would be to use a custom defined interface for both module types and module inspectors instead of the simple parent structure employed in the current revision of the framework. This would make sure

experienced users implement all necessary functions for custom module types and would also simplify possible future extensions.

User study When conducting the two user studies of this thesis, there were severe problems with finding participants and receiving sufficient feedback to the surveys. This problem is discussed in more detail in sections 4.2 and 4.3.

4. Accessibility and user perception

In order to determine the quality of the ConText framework, the accessibility for both experienced and inexperienced users and the quality of the general user experience, user studies needed to be conducted. The studies did not conform to a given standard, however we do not consider this an issue for two reasons, 1) the sample size is too small for numeric statistics to provide meaningful accurate insight, 2) they were not intended to provide definite statistics to compare the framework to other solutions on the market that would use standardized evaluation formats, but instead to get an idea of whether the framework works as designed, more akin to a focus group survey sans active monitoring. A larger and standardized study would be appropriate for a more polished version of the framework in the future.

4.1. Study procedure

The quality evaluation for ConText consisted of two consecutive user studies, the procedure for which was essentially identical. Participants were selected and invited to take part in the study, with a small number planned for the first study to get an initial idea of the user perception and qualitative direction the framework was headed at that point in development and a larger number of participants intended for the second study that would evaluate the progress and changes made between both studies and provide more differentiated feedback. Due to time constraints and unfortunate circumstances, however, the second user study did not work out as intended and only returned very lackluster results. A more elaborate rundown of the issue is provided in section 4.3.

In order to ensure the participants would see all relevant parts of the framework, they were given an orientation sheet and a tutorial to guide them through the creation of a simple game. Additionally included was a documentation explaining remaining and individual features of the framework, followed by a survey sheet - either as a PDF or a Google Forms form to be filled out online - to rate their experience and give feedback on certain aspects. All four documents are included in Appendix section A.1.

Orientation The orientation sheet presents an overview of what the study task is and what each of the other documents is intended to be as well as informs the participant on what the study and its results will be used for. The participant is guaranteed that their creation will not be used unless explicitly permitted and that the results of the study will only be used and evaluated for the thesis.

Tutorial The tutorial is a two-part guide through the framework. Part 1 explains the installation and setup process, from installing Unity through importing the framework files to setting up the main screen. Part 2 is a step by step walkthrough to creating a sample game with the framework, including creating and configuring characters, creating, configuring and linking different types of modules and finally how to test the game within the editor. A final section explains the process of importing own files into Unity.

Documentation The documentation details the various aspects of the framework in order to possibly explain functionality not covered in the tutorial but of interest to the user in further usage. It covers the module system and its structure, the managers and their functionality, the characters and their purpose, the UI settings and theirs, the custom inspectors and their heritage setup as well as what to watch out for when creating custom modules.

Survey The survey sheet queries the participant on their experience with the framework and opinions and feedback on certain aspects. The participant is first to describe the computer system they are using for development and their preexisting skills regarding game development and story writing. The further parts of the survey prompt the participant to rate and if possible explain their rating for the intuitiveness, complexity and arrangement of the interface, the perceived performance and responsiveness of the tools, the quality and detail of the documentation and their overall experience using the framework.

Study files Accompanying the four above documents the participants get access to a Unity Package containing all technical parts of the framework such as the code base and the initially set up scene, while the last file is a Unity layout file.

4.2. First user study

The first user study had five participants out of seven invited. All participants were given four days as well as all aforementioned files to create a small game, fill out the

survey and send their feedback. The statistics and results of the study are discussed in the following paragraphs, sorted by their order within the survey sheet.

Basics TODO.

Interface TODO.

Performance TODO.

Documentation/Tutorial TODO.

General impression TODO.

Additional TODO.

4.3. Second user study

Changes With the results of the first user study evaluated, several changes and adaptions were made in the framework, mostly related to usability improvements. For one, the various segments representing module data in the module inspectors were put into foldouts so users would not be overwhelmed by the plethora of options at first sight but could instead choose step by step what they configure along with controls for opening and closing all open or closed foldouts at once. Additionally, more hints were added to mark critical missing info. A major refactor attempt highly improved the custom module inspector heritage structuring and made it more accessible to programmers. The ConText Overview window was renamed as ConText Options and the various options in renamed and reordered to be grouped according to functional purpose and given updated optional hints so their purpose would be more easily identified at first sight. Game sound settings were added to the game, consisting of a default message sound to be played for each message fired, a unique sound a message can play on its own as well as a background track to be played in loop. Lastly, a test run on Android was done as in a short story was set up and ported to an Android device in development mode to verify that the existing systems would work on the mobile platform as the second user study would allow the participants to test on Android as well.

Basics TODO.

Interface TODO.

Performance TODO.

Documentation/Tutorial TODO.

General impression TODO.

Additional TODO.

After the second user study One often requested and initially planned feature was a visual representation of the entire story within the editor. The intuitive ideal representation is a visual graph with story modules as nodes and their connections as edges. Thus such a system was put in place towards the end of the project, unfortunately after the second user study though. The node viewer added to the framework does not offer a lot of interactivity yet, upon opening it the user is presented with an image as described with the message modules shown as graph nodes displaying the content and character and an edge connecting it to its previous and next modules' nodes. A button within each node lets the user jump straight to that module in the inspector window. A small control in the upper left of the node viewer lets the user pan left, right, up and down as well as reset to the origin, a control at the top makes the viewer reset and redraw the graph.

5. Closing words

TODO.

5.1. Outlook and possible future work

TODO.

5.2. Conclusions from surveys

TODO.

5.3. Project conclusion

TODO.

5.3.1. Summary and accuracy to goal (del)

5.3.2. what else? (del)

A. Appendix

A.1. User study documents

A.1.1. Survey form

TODO. include (Rev 2?)

A.1.2. Tutorial

TODO. include (Rev 2?)

A.1.3. Documentation

TODO. include (Rev 2?) (included but commented out)

A.2. Other (?)

TODO.

List of Figures

List of Tables

Bibliography

[run16] runevision. *Why doesn't Layout Element have Max values?* 2014-2016. URL: http://forum.unity3d.com/threads/why-doesnt-layout-element-have-max-values.274221/#post-1816127.

[Uni16] Unity Technologies. *SYSTEM REQUIREMENTS FOR UNITY*. 2016. URL: https://unity3d.com/unity/system-requirements.