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 main 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.

Ludus A prime candidate for consideration was TUM student Moritz Becher's *Ludus*, another Bachelor's Thesis building on a very similar topic as this one. Ludus is a mobile text adventure game set in ancient Rome and designed to teach the player historical facts about the setting. It was actually developed under the same supervisor and for the same overall topic of educational serious games. The game itself was not directly analyzed, instead Becher was contacted and asked about his ideas on what features this framework would need and which features he deemed useful for his project. As a result, some of his concepts for internal structures such as the decoupling and communication of story and user interface served as inspiration for some of ConText's systems. Over the course of development the framework actually diverged from these concepts again, however. Additionally, Becher participated in the second user study given his thesis essentially qualifies him as an expert on the subject.

2.2.2. Frameworks

Quest & Squiffy Both Quest and Squiffy are web based frameworks for creating text adventures, gamebooks and interactive fiction. The focus seems to lie on straight forward creation of these types of games with a high level of compatibility with devices due to the reliance on entirely web based technology like HTML and JavaScript. With Quest, a simple result is easy to achieve, with scripting language integration available should users want to add specific functionality. In these aspects, it is very akin to what ConText aims to be. The downside is, however, that the feature expansion is limited to what HTML and JavaScript can do within the confines of the tools and should the user want to integrate functionality outside of the typical scope of a website, they are confronted with a large amount of custom scripting. Squiffy is locked down to the specific markup and scripting language used in the tool, but also promises

quick results. Both tools need additional tools to encapsulate a story into a standalone app. Experimenting with Quest and Squiffy did provide a better understanding of the properties the module types would need such as names, types, content, linking options and possible actions as well as a better idea of the scope of the framework that is needed for a basic system and viable for the temporal terms of this thesis.

Other Other tools such as Inform, a natural language based system, WibbleQuest, a library for developing iOS text adventures or ADRIFT, a graph based text adventure creator were all briefly considered but not analyzed in more detail as the already listed sources of inspiration as well as some specialist literature and aforementioned logical inference provided enough aspects and features to define ConText's core concept and structure design. One downside that was common to all regarded toolkits was the comparably lackluster expansion capabilities, with some supporting extenions within the limits of their base systems, but none as potentially diverse and simultaneously viable solutions as Unity engine's.

2.2.3. Literature

Specialist literature was used to help determine what other international projects and papers deem necessary or beneficial to watch out for when creating tools designed to create text adventures.

On E-Learning and Interactive Storytelling technology In order to get such an idea specifically related to educational and serious games, proceedings from several conferences on the topic were consulted. In one such volume from [Ste12], Danu Pranantha et al. while developing a HTML5 framework for creating serious games suggest categorization of serious game creation tools and their development into „1) the development of platform for ease to use game creation using pre-created game props 2) the use of advanced web technologies to develop browser games which offers cross platform capability"(sic!)[Dan12]. Additionally, they acknowledge in the prior sentence that game engines exist and can serve as a base platform for such tools. Thus, ConText could be categorized as following approach 1 while using Unity engine as a base. In that same sentence, Pranantha et al. also raise concerns about the cross platform capabilities and issues of using an existing engine, which we deem outdated at this point, however, as in the four years between the proceedings and this thesis, cross platform capabilities of the Unity engine have been greatly improved and verified to work well in conjunction with ConText's features and a deployed game created with it in an internal test. Overall, similar categorization is also suggested by various other efforts in the consulted material and ConText fits as a plugin system for a widely available existing

game engine. The advantages of this design are that the low level components such as rendering, input handling, etc. are covered, that a large community as well as detailed documentation are available and that through this community, some potential users are easier to reach. Diving into proceedings on interactive storytelling also yielded some helpful directions, in the case of [Ale14] and [Har14] specifically on user experience testing. While the paper itself actually targets methods for successful story authoring, one key point is user experience, that "The User Experience is crucial". In it, Koenitz writes that user studies should deliberately "cast the net as wide as possible and try to avoid techno-savvy users"[Ale14, p. 135] as those do not already appreciate the studied artifact for technical aspects but instead judge from the viewpoint of someone without preexisting knowledge and as such more akin to the common average user. This also promises to bring more attention to lackluster UI simplicity as these users would not be as used to complex interfaces.

The overall experience when looking for adequate literature touching the topic of (serious) text adventures, frameworks for the creation of such and the requirements and best practices proved lackluster and suggests that the subject has been somewhat unpopular until now and a lot of groundwork has yet to be covered or publicly and widely shared. As described earlier, plenty of functionality of ConText has been designed based on logical needs and common sense dictated by the planned feature set.

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.

Editor interface While the previous paragraphs specify the structure of the framework's systems and offerings, there was also considerations to be made for the interface provided to the developers and users. As ConText is a Unity plugin/package, it is informally bound to the established design used by Unity in order to be more accessible and seamlessly integrated into the engine during use. This mostly targets spatial arrangement and sizing of windows such as the inspectors and the ConText Options, as well to a degree the arrangement and design of controls within each window. For the initial concept, however, no specific constraints were yet set for the detailed presentation of information and as such started out as a vaguely ordered list of options and data controls.

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.

Editor interface Through reiteration, user feedback and user experience and usability research, the editor interface was transformed from the continuously grown messy list of data controls and options in some of the ConText windows to a more clean looking appearance with logical order and structuring and visual decoupling as well as hiding and layered display to reduce the visible clutter and complexity.

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. The participants were recruited from other students at TUM, students at Ludwig-Maximilian-University (LMU) in Munich and personal acquaintances. 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. For each segment when numerical rating data is used, the arithmetic average and standard deviation were calculated and rounded to the second increment using Excel 2016 with STDEV.P for the deviation as the entire population is known and given.

Basics Starting out with the Basics segment, the noteworthy data is about age, time spent with the framework, preexisting development and writing skills and to a degree the hardware used by the participants. User names are omitted for privacy reasons. As there were only five participants in this first study, the numbers do not have a high statistical value, but give an idea of the study audience which can help interpret the qualitative feedback. The average age was 30.75 years with a standard deviation of 18.84 years. Except for one outlier, all participants were in their early twenties. The average time spent with the framework was 1.38 hours (std. dev. 0.99 hours), indicating most participants stopped using ConText after finishing the study tutorial. Two participants claimed existing and recent game development skills, one other

	age (yrs)	**time** (hrs)
user 1	59	3
user 2	22	2
user 3	21	1
user 4	21	0.5
user 5	-	0.416
avg	30.75	1.38
std. deviation	16.32	0.99

Table 4.1.: User age and framework usage duration: avg. and std. deviation for user study 1.

claimed "solid knowledge about story writing", otherwise all participants denied further knowledge. The computers used by the participants mostly used modern desktop Intel Core i5 and i7 processors with at least 16 gigabytes of memory and graphics cards equal to or stronger than an Nvidia GeForce GTX 970, with one outlier using a low power Intel Core i5 mobile SKU, 4 gigabytes of memory and integrated graphics. All systems used Windows 8.1 or 10 as well as Unity engine in version 5.3.4f1.

Interface The interface segment consisted of four questions about the user interface quality, intuitiveness and spatial arrangement that prompted the participants to rate the queried aspect on a scale from 1 to 7 and possibly elaborate on reasons for their

rating. A fifth task asked the participants to describe the workflow they experienced. The questions were:

I1 On a scale from 1 (worst) to 7 (best), how intuitive did you find the framework to be? (e.g. immediate understanding of features, functions, processes)

I2 On a scale from 1 (worst) to 7 (best), rate the spatial arrangement of framework items (i.e. the placement of the different windows within the tool; e.g. Overview Window, Inspector/Settings Window, preview, Hierarchy window, Project/Asset folder view)

I3 On a scale from 1 (worst) to 7 (best), rate the naming of framework items (Overview Window, Inspector/Settings Window, preview, Hierarchy window, Project/Asset folder view)

I4 On a scale from 1 (worst) to 7 (best), rate the complexity of the framework items (e.g. bad if windows were overloaded, oddly sorted, etc.)

The ratings given for these four questions were: Looking at these ratings, one can

	I1 intuitiveness	**I2** spatial arr.	**I3** naming	**I4** complexity
user 1	3	7	7	7
user 2	2	5	3	3
user 3	4	4	6	2
user 4	5	6	4	3
user 5	7	4	5	6
avg	**4.2**	**5.2**	**5**	**4.2**
std. deviation	1.72	1.17	1.41	1.94

Table 4.2.: Interface aspect ratings (in x out of 7)

tell that perception of the framework's interface was mediocre to moderately positive. Among the qualitative responses explaining the ratings, common opinions and criticism on the interface were that

- Intuitiveness was hurt by an overload of information presented at first sight and confusing structure
- The framework windows and controls are neatly arranged and most are appropriately named

- It was not fully clear at first sight what each window's purpose or relevance is
- A (graph) visualization of the storylines would greatly improve the user's overview of the creation

Performance Performance ratings were divided into

P1 On a scale from 1 (far too long) to 7 (fine), rate the saving and loading times of the frameworkitself(e.g. perceived hickups/stuttering when creating modules, changing settings, etc.)

P2 On a scale from 1 (very stuttery) to 7 (fully smooth), rate the framerate of the game (in previewmode)(i.e. how visually smooth did the game run)

P3 On a scale from 1 (very slow) to 7 (very quick), rate the responsiveness of the interface(e.g. how quickly did the framework react to your input)

	P1 save/load	**P2** game framerate	**P3** response time
user 1	7	7	7
user 2	5	6	6
user 3	7	7	7
user 4	7	7	6
user 5	7	7	7
avg	**6.6**	**6.8**	**6.6**
std. deviation	0.8	0.4	0.49

Table 4.3.: Performance ratings (in x out of 7)

Given these values, it can be assumed the framework did not show any significant bottlenecks, and matching the data with the used computers, user 2's comparatively lower ratings can be attributed to them using the aforementioned low power mobile machine. However even with such a low performance machine, the experience was still rated acceptable. The only common complaint named the initial loading process duration of the Unity engine as an issue. On a sidenote, one participant responded that they did not understand they were creating a game, indicating the first revision of the tutorial did not make this information entirely clear.

Documentation/Tutorial The documentation and tutorial segment only contained one rating, namely

D1 On a scale from 1 (very vague) to 7 (very detailed), rate the detail of the provided documentation

	D1 detail
user 1	5
user 2	7
user 3	7
user 4	4
user 5	6
avg	**5.8**
std. deviation	1.17

Table 4.4.: Documentation and tutorials ratings (in x out of 7)

The given ratings suggest adequate detail was provided in the documentation and tutorial, though considering the average time spent with the framework, it stands to reason that none of the participants delved deep enough into ConText to need much documentation. Qualitative assessments include, among others, that

- A schematic overview of the framework's structure and principle is missing
- Some steps used language too technical for the common user
- The sequential nature of the tutorial fits very well, but some steps need more atomic explanation

When asked about what type of tutorial the users would prefer - such as textual, videos, in-tool walkthrough or mixed - two each responded they would like video instructions and walkthroughs within the framework. Unfortunately, time limitations did not allow for either of these to be created.

General impression & Additional The second to last and last segment asked the participants to freely describe the overall experience they had using the framework, to which the responses essentially summarized the individual criticism raised in prior segments. The documentation and tutorial lacked a schematic overview, the individual steps proved too technical and complex in some cases and the interface too cluttered to be intuitive, eventually leading to a degree of frustration.

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. Both the tutorial and documentation received an overhaul with several tutorial steps split up into smaller steps with more accompanying images and explanation and the documentation going into far more detail on each core system. 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. Participants for the second user study were recruited primarily from attendees of the regularly held FAR Story Stammtisch event at TUM, other students at TUM and eventually in a last effort from online communities.

Lack of participants Despite continued efforts, repeated polite contact and larger number of invitees, only a unsuitable fraction of participants followed through and provided feedback. Out of approximately fifteen directly invited, half of which initially confirmed their participation and an unknown number of potential users reached online, only four gave signs of participation after the initial agreement, only two of which actually completed the survey and submitted their feedback. While it was unfortunate that the second user study was hosted in the active examination period of the summer semester 2016, thereby losing several potential candidates, it remains inexplicable how almost all participants eventually declined. We believe enough time until the due date was given, including an extension to a total of at least two weeks - noting the first user study successfully completed with five out of seven users within four days - and the invitees were treated politely, with respect and given sufficient and detailed information on the procedure. Given these indoubtedly lackluster participation

statistics, the following feedback evaluation is to be taken with skepticism and not to be mistaken with a well averaged and allround profound depiction of user perception. The non-numerical feedback can however be examined within its limitations and with the first study's results kept in mind, to very vaguely determine achieved progress.

Basics Again starting with the basic information provided, the average age was 25 years (standard deviation SD = 2.00), the average time spent with the framework 2.25 hours (SD 0.75). One participant used a system in line with the average development computer in the first study, with a modern desktop Intel Core i7 SKU, 8 gigabytes of memory, an Nvidia GeForce GTX 970, running Windows 10, the second participant used a 2012 Apple MacBook Pro with a 2.9GHz Intel Core i7 mobile SKU, 8 gigabytes of memory and integrated graphics, OSX version unknown. The first participant claimed prior game development and writing experience, the second denied any relevant experience.

Interface The interface intuitiveness ratings diverged too much between the two participants with one giving almost perfect scores, the other mediocre. The qualitative comments prove more helpful here, indicating that the interface at first sight still offers too many possibilities at once but can be understood well after a short period, with the inexperienced user being most overwhelmed by the tool. It still stands to be determined whether the main confusion stems from ConText's options or Unity's. It was suggested the modules be displayed more hierarchically and the amount of windows open at once reduced, as well as the reoccurring suggestion of a story graph visualization. One thing to note is that the interfaces was described as scaling unfavorably with low or very high resolutions. Low resolutions (below 1600x900 pixels) or positive desktop scaling lead to the individual windows requiring scrolling to see even small amounts of information while very high resolutions (above 2560x1440 pixels) or negative desktop scaling lead to content inside windows becoming small and hard to read.

Performance Performance was rated excellent by both participants, falling in line with observations made in the first user study (avg. 7.00, SD 0.00).

Documentation/Tutorial The documentation and tutorial likewise received high scores (avg. 7.00, SD 0.00) and were described as "rich of details" (sic), the tutorial "absolutely useful" yet still also rather complex in parts. Textual, video and in-framework walkthroughs were all likewise named as suitable methods for the tutorial.

General impression & Additional In summary, the framework was described as "a great tool to create message based interactive fiction", "very flexible", and "surprising to achieve such quick results [even for a beginner]". One additional suggestion were code examples in the documentation for custom classes.

Statistical trend While the numerical ratings cannot be used to make definitive statements as the sample size is insignificantly small, a vague trend can be seen suggesting the interface has improved over the first revision as has the documentation quality. Further quantitative studies with a much larger sample size would need to be conducted to gain statistically significant data.

After the second user study Through both studies, 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

The following four documents are the files that were provided in the respective user study as is

- Orientation document revision 1 (for user study #1) and revision 2 (for user study #2)
- Survey form revision 1 and revision 2
- Tutorial document revision 1 and revision 2
- Documentation revision 1 and revision 2

A.1.1. Orientation

DOCUMENT: Orientation **revision 1**

User Study #1 for *ConText – A Text/Choice Adventure Game Framework*

This is an orientation sheet for this user study you are participating in. ConText is a framework intended for the creation of text/choice based adventure games for mobile devices. The framework is currently work in progress and as such, missing features and bugs are to be expected. This user study does not represent the final product.

It is advised you first complete the basic tutorial (Tutorial_UserStudy1_English.pdf).

After that, you should implement a short story of your choice as well as fill in the survey (ConText_UserStudy1_English.pdf) and submit the survey (and if possible, your Unity project folder as well) to me by Friday, June 1, 2016. The story does not need to be complex or clever. If you have troubles using the framework, please first consult the documentation (Documentation_UserStudy1_English.pdf), contacting me should be a last resort.

Your story will not be used in the thesis paper unless you explicitly grant permission. This user study primarily serves the purpose to find out whether the framework works as intended and how it resonates with various user types. There will be a second user study examining the changes suggested through this one as well as with a larger user group.

The results of this survey will not be shared with third parties, and only be used and evaluated by the author of the thesis. If any specific individual answers are used, they will be anonymous unless otherwise specified through the survey sheet.

If you have any questions prior, during or after the user study regarding the study, you may email me at paul@preissner-muc.de.

Page **1** of **1**

DOCUMENT: Orientation **revision 2**

User Study #2

for *ConText – A Text/Choice Adventure Game Framework*

This is an orientation sheet for this user study you are participating in. ConText is a framework intended for the creation of text/choice based adventure games for mobile devices. The framework is currently work in progress and as such, missing features and bugs are to be expected. This user study does not represent the final product.

Essentially, you will be using the framework to create such a text adventure game with a short story. For starters, you will be able to play test your game inside the framework, however export to an actual mobile device (Android) is be covered in the documentation if you wish to do it.

All files relevant to the study can be found here: https://www.dropbox.com/sh/vrao65c3ym3nkhc/AAC_WXLxHcPu04Gkaxne7wh9a?dl=0

It is advised you first complete the basic tutorial (Tutorial_UserStudy2_English.pdf). This will get you started in using the framework, getting to know its features, the user interface, etc.

After that, you should implement a short story of your choice and fantasy. Once done, fill in the survey (Survey_UserStudy2_English.pdf or online at http://goo.gl/forms/xpUYTaDGbpJWToIp2) and submit it (and if possible, your Unity project folder as well) to me by **Saturday July 23, 2016**. The story does not need to be complex or clever. If you have troubles using the framework, please first consult the documentation (Documentation_UserStudy2_English.pdf), contacting me should be a last resort.

Your story will not be used in the thesis paper unless you explicitly grant permission. This user study primarily serves the purpose of finding out whether the framework works as intended and how it resonates with various user types (as well as to judge the improvements made since the first study).

The results of this survey will not be shared with third parties, and only be used and evaluated by the author of the thesis. If any specific individual answers are used, they will be anonymous unless otherwise specified through the survey sheet.

If you have any questions prior, during or after the user study regarding the study, you may email me at paul@preissner-muc.de.

Page **1** of **1**

A.1.2. Survey form

DOCUMENT: Survey form **revision 1**

User Study #1 for *ConText – A Text/Choice Adventure Game Framework*

This survey is to be filled in by the participants of the first user study for the Bachelor's Thesis *ConText – A Text/Choice Adventure Game Framework*.

The process is as follows: the participant will download and install the framework as described in the additional tutorial, then spend until June 1, 2016 to implement a short story using the framework. They will answer this sheet as truthfully and thoroughly as possible. It is advised, the participant take notes during implementation in order to not miss any important details.

This sheet is comprised of the parts *Basic Information, General impression, Interface, Performance, Documentation/Tutorial.* Please make sure to fill in all parts.

Basic information

Name (optional, leave blank for anonymous participation)	Age (optional)	Time spent with the framework
If possible, list the computer system's specifications this framework was run on (e.g. either Device Model Number, or if possible detailed specs such as processor model, system memory size, graphics processor, drive type the framework was installed on, operating system, etc.)		
Briefly describe your existing knowledge/skills regarding programming, game development and story writing.		

Interface

On a scale from 1 (worst) to 7 (best), how intuitive did you find the framework to be? (e.g. immediate understanding of features, functions, processes)						
1	2	3	4	5	6	7
O	O	O	O	O	O	O
Why?						

Page **1** of **6**

DOCUMENT: Survey form **revision 1**

On a scale from 1 (worst) to 7 (best), rate the spatial arrangement of framework items (i.e. the placement of the different windows within the tool; e.g. Overview Window, Inspector/Settings Window, preview, Hierarchy window, Project/Asset folder view)						
1	2	3	4	5	6	7
O	O	O	O	O	O	O
Why?						

On a scale from 1 (worst) to 7 (best), rate the naming of framework items (Overview Window, Inspector/Settings Window, preview, Hierarchy window, Project/Asset folder view)						
1	2	3	4	5	6	7
O	O	O	O	O	O	O
Why?						

On a scale from 1 (worst) to 7 (best), rate the complexity of the framework items (e.g. bad if windows were overloaded, oddly sorted, etc.)						
1	2	3	4	5	6	7
O	O	O	O	O	O	O
Why?						

Describe the nature of your workflow in the framework, e.g. any hindrances you might have experienced, or on the contrary aspects that helped your workflow.

Page **2** of **6**

DOCUMENT: Survey form **revision 1**

Additional notes

Performance

On a scale from 1 (far too long) to 7 (fine), rate the saving and loading times of the framework itself (e.g. perceived hickups/stuttering when creating modules, changing settings, etc.)						
1	2	3	4	5	6	7
O	O	O	O	O	O	O
Why?						

On a scale from 1 (very stuttery) to 7 (fully smooth), rate the framerate of the game (in preview mode) (i.e. how visually smooth did the game run)						
1	2	3	4	5	6	7
O	O	O	O	O	O	O
Why?						

Page **3** of **6**

DOCUMENT: Survey form **revision 1**

On a scale from 1 (very slow) to 7 (very quick), rate the responsiveness of the interface (e.g. how quickly did the framework react to your input)						
1	2	3	4	5	6	7
O	O	O	O	O	O	O
Why?						

Additional notes

Documentation/Tutorial

On a scale from 1 (very vague) to 7 (very detailed), rate the detail of the provided documentation						
1	2	3	4	5	6	7
O	O	O	O	O	O	O
Why?						

Page **4** of **6**

DOCUMENT: Survey form **revision 1**

Was the tutorial useful in its sequential nature? (e.g. in providing step by step descriptions of the game creation process) Please explain your answer briefly.

What type of tutorial do you/would you prefer? (e.g. purely textual, video instructions, in-framework walkthrough, mixed, etc.)

Additional notes

DOCUMENT: Survey form **revision 1**

General impression

Describe your overall experience with the framework

Additional suggestions

Add any additional suggestions here

Page **6** of **6**

DOCUMENT: Survey form **revision 2**

User Study #2 for *ConText – A Text/Choice Adventure Game Framework*

This survey is to be filled in by the participants of the second user study for the Bachelor's Thesis *ConText – A Text/Choice Adventure Game Framework*.

The process is as follows: the participant will download and install the framework as described in the additional tutorial, then spend until July 23, 2016 to implement a short story using the framework. They will answer this sheet as truthfully and thoroughly as possible. It is advised, the participant take notes during implementation in order to not miss any important details.

This sheet is comprised of the parts *Basic Information, General impression, Interface, Performance, Documentation/Tutorial.* Please make sure to fill in all parts.

Basic information

Name (optional, leave blank for anonymous participation)	Age (optional)	Time spent with the framework
If possible, list the computer system's specifications this framework was run on (e.g. either Device Model Number, or if possible detailed specs such as processor model, system memory size, graphics processor, drive type the framework was installed on, operating system, etc.)		
Briefly describe your existing knowledge/skills regarding programming, game development and story writing.		

Interface

On a scale from 1 (worst) to 7 (best), how intuitive did you find the framework to be? (e.g. immediate understanding of features, functions, processes)						
1	2	3	4	5	6	7
O	O	O	O	O	O	O
Why?						

Page **1** of **6**

DOCUMENT: Survey form **revision 2**

On a scale from 1 (worst) to 7 (best), rate the spatial arrangement of framework items (i.e. the placement of the different windows within the tool; e.g. Overview Window, Inspector/Settings Window, preview, Hierarchy window, Project/Asset folder view)						
1	2	3	4	5	6	7
O	O	O	O	O	O	O
Why?						

On a scale from 1 (worst) to 7 (best), rate the naming of framework items (Overview Window, Inspector/Settings Window, preview, Hierarchy window, Project/Asset folder view)						
1	2	3	4	5	6	7
O	O	O	O	O	O	O
Why?						

On a scale from 1 (worst) to 7 (best), rate the complexity of the framework items (e.g. bad if windows were overloaded, oddly sorted, etc.)						
1	2	3	4	5	6	7
O	O	O	O	O	O	O
Why?						

Describe the nature of your workflow in the framework, e.g. any hindrances you might have experienced, or on the contrary aspects that helped your workflow.

Page **2** of **6**

DOCUMENT: Survey form **revision 2**

Additional notes

Performance

On a scale from 1 (far too long) to 7 (fine), rate the saving and loading times of the framework itself (e.g. perceived hickups/stuttering when creating modules, changing settings, etc.)						
1	2	3	4	5	6	7
O	O	O	O	O	O	O
Why?						

On a scale from 1 (very stuttery) to 7 (fully smooth), rate the framerate of the game (in preview mode) (i.e. how visually smooth did the game run)						
1	2	3	4	5	6	7
O	O	O	O	O	O	O
Why?						

Page **3** of **6**

DOCUMENT: Survey form **revision 2**

On a scale from 1 (very slow) to 7 (very quick), rate the responsiveness of the interface (e.g. how quickly did the framework react to your input)						
1	2	3	4	5	6	7
O	O	O	O	O	O	O
Why?						

Additional notes

Documentation/Tutorial

On a scale from 1 (very vague) to 7 (very detailed), rate the detail of the provided documentation						
1	2	3	4	5	6	7
O	O	O	O	O	O	O
Why?						

Page **4** of **6**

DOCUMENT: Survey form **revision 2**

Was the tutorial useful in its sequential nature? (e.g. in providing step by step descriptions of the game creation process) Please explain your answer briefly.

What type of tutorial do you/would you prefer? (e.g. purely textual, video instructions, in-framework walkthrough, mixed, etc.)

Additional notes

Page **5** of **6**

DOCUMENT: Survey form **revision 2**

General impression

Describe your overall experience with the framework

Additional suggestions

Add any additional suggestions here

Page **6** of **6**

A.1.3. Tutorial

DOCUMENT: Tutorial **revision 1**

Basic Tutorial for *ConText – A Text/Choice Adventure Game Framework*

This tutorial will guide you through the interface of the framework and in the process you will create a basic text adventure game.

Part 1 – Setup

1. Before anything else, you will need to download and install Unity Personal Edition from here: https://unity3d.com/get-unity/download?ref=personal
   Unity is the game engine this framework was developed in and is to be used in. The first time you start Unity, you will need to create/login with an account. This account is necessary to verify your license information. The account can be created and used free of charge.
2. Once you have installed Unity, you will need to either use the ConText package (ConText_UserStudy1.unitypackage) provided in your initial invitation email or download it from here: https://www.dropbox.com/sh/ad48e0u7hz3583b/AABUAKveQAnkVYoRA5sBqlAqa?dl=0. You may download the entire folder which contains the UnityPackage, a *'ConText Project.wlt'* file as well as all PDFs also attached to the email.
3. The *'ConText Project.wlt'* file needs to be copied to the Unity Editor Layout folder. In Windows, that is **%APPDATA%\Unity\Editor-5.x\Preferences\Layouts**. In Mac OS, it is **~/Library/Preferences/Unity/Editor-5.x/Layouts**. (Simply paste these directory paths to Explorer/Finder to access the respective folder)
4. Start Unity, log in with your account and select **NEW**. On the following screen, enter a name for your project and select **2D**.

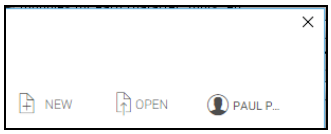


Figure 1: NEW

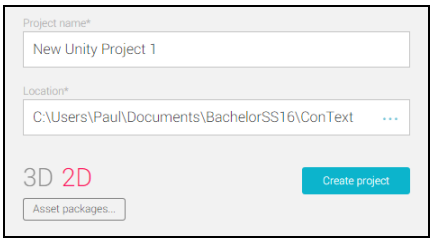


Figure 2: Create

5. Import the UnityPackage into Unity through **Assets – Import Package – Custom Package…** . A window will pop up listing all items that can be imported from that package. At the bottom of that window, select **All** and then click **Import**.

Page **1** of **6**

DOCUMENT: Tutorial **revision 1**

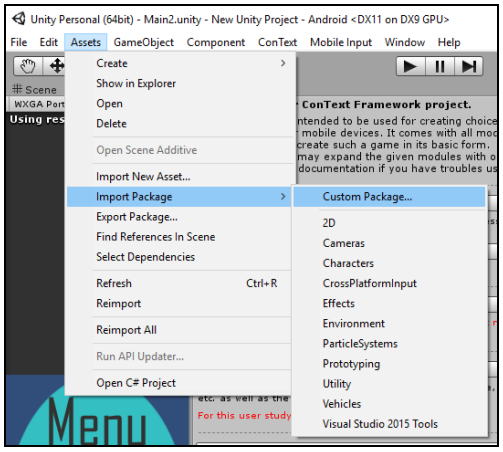


Figure 3: Import package

6. Almost done setting up the framework. In the upper right, select **ConText Project** as the Layout.

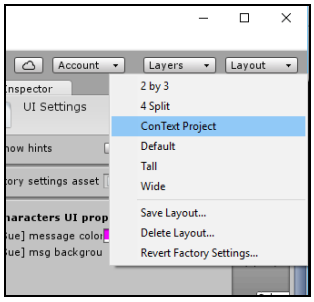


Figure 4: Layout

7. In the Project/Asset folder window, select the top most folder **Asset** and double click **Main** to open the main Unity scene.

Figure 5: Open Main

Now your Unity editor should look like this:

DOCUMENT: Tutorial **revision 1**

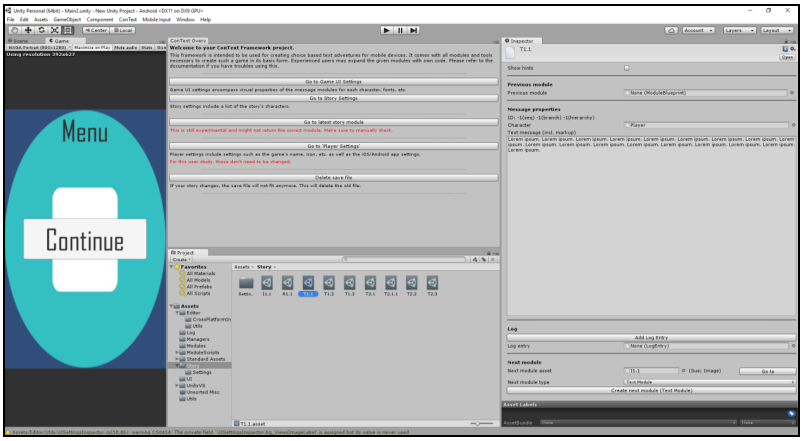


Figure 6: Overview

If so, the last thing to do is deselect **Maximize on Play** at the top of the left Game preview window, then you are done setting up the framework and can continue with Part 2 of the tutorial. If not, make sure you did all steps as described. If you are sure you did not miss a step and the overview is still wrong, feel free to contact me at paul@preissner-muc.de.

Part 2 – Sample game

In this part you will use the framework to create a simple text adventure game with a few different modules with the framework.

1. Given the layout as in Figure 5, the left part shows your *game preview window*. When you test/play your creation, it will be displayed in that window.
   The center part has the *ConText Overview* window in the top half, intended as a sort of hub from which you can go straight to the most important settings and windows. The bottom half is the *Project window*, which displays the directory structure of your asset folders, through which you may directly access story modules, settings files or ultimately script files.
   The right part is the *Inspector window*, which will always display the properties/details of the currently selected asset.
2. To get started, you will create the characters of your story. For this, click **Go to Story Settings** in the *ConText Overview* window. This will open the Story Settings asset in the Inspector window. There, click **Add a character**, which will create a new Character asset and automatically display its details in the Inspector window. You may then give it a name and a unique character ID. Add as many characters as you need.

DOCUMENT: Tutorial **revision 1**

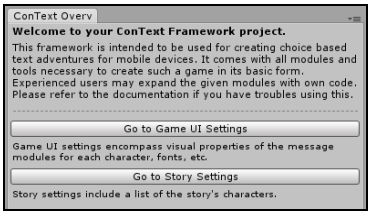


Figure 7: Go to story settings

Figure 8: Add character

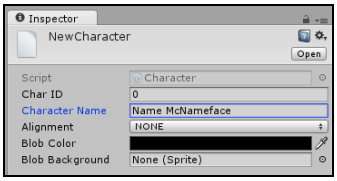


Figure 9: Customize character

3. Next, you will create the start of your story. For this, click **Create your story** in the ConText Overview. This will create the first node of your story, which (as of now) will be a Text module. The now created module will be displayed in the Inspector window. There you will need to choose the Character this message should be sent by by clicking on the small circle next to the Character field. Right below that is a text field which will hold the text this message consists of.

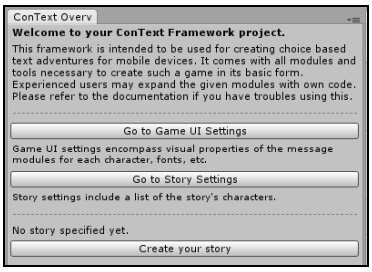


Figure 10: Create story

DOCUMENT: Tutorial **revision 1**

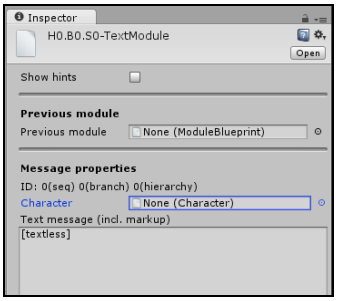


Figure 11: Message

4. Now your story consists of one message. Within each module, you can create the next module to follow after it (sometimes given a certain condition) through the *Next module* segment in its Inspector (or Replies for the Reply Module). For this tutorial, select the **module type as "Reply Module"** and click **Create next module**. This Create button will always reflect what type will be created.

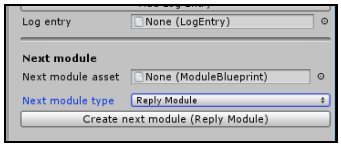


Figure 12: Next

5. This will have put that Reply module into focus in the Inspector. A specialty about the Reply module is that after displaying its text, it will not automatically send the next message, but leave the player with the respective number of replies to choose from. Each reply may lead to a different branch in the story. For this specific module in the tutorial, set up Character and Text as in step 3., then scroll down to the Replies segment. It works similarly to the Next module segment of the previous module, with the difference being you can add multiple entries. **Add a reply**.

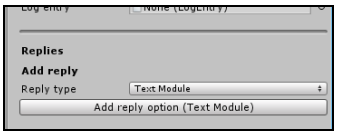


Figure 13: Replies

6. The newly created module will be in focus then. For the sake of this tutorial, look at the *Previous module* segment at the top of the Inspector. If the previous module is specified, it will have a **Go to** button next to it. Clicking it will naturally take you to that previous module, in this case the Reply module from step 5. There you may change the default description text of each reply, which is so far "new reply x". You may have seen by now that each previous or next module has that **Go to** button, which as of now is a simple way to traverse your story modules.

DOCUMENT: Tutorial **revision 1**

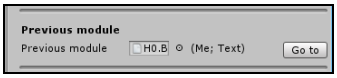


Figure 14: Go to

7. With this, you should know the basic process of creating modules. If you want to manually reconnect nodes in a different way, you can always click the little circle next to one and choose a different existing module. Be aware though, this can easily screw up the structure of the story and might lead to flawed playback.
8. At the bottom of each module's Inspector is a *Delete module* button, which will delete that module and attempt to properly connect the surrounding modules. Be aware, however, that when dealing with modules that have multiple next modules (like Reply modules), this may lose the connection to other branches.
9. As the last step of this tutorial, click on **Go to Game UI Settings** in the ConText Overview. This will take you to the visual properties of the messages and modules, where you might change color, font, font size and more.

Now that you have set up the framework and created your first basic game with it, click on the **play button** at the top center of the tool and check out what it looks like in the preview window.

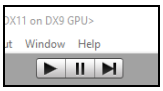


Figure 15: Play!

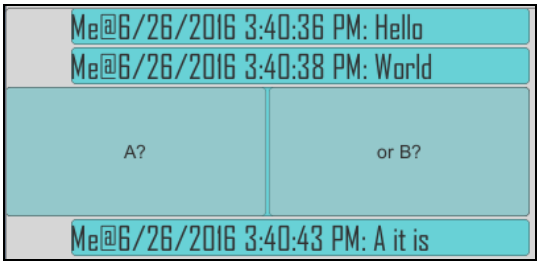


Figure 16: Small beginnings

Additional info about importing files into Unity:

It may likely happen that you want to import custom image files to use as background or in an image module or perhaps different fonts.
Files can generally imported into Unity by simply dragging and dropping them into the Project/Asset folder window. So you can just right click into any folder in that Project window, select Create – Folder to add a separate folder for your files, then drag and drop them in there.

In the case of image files, these will by default be imported as Sprites given you project was set to 2D upon creation. If that is not the case and the image is not available for selection as such an image, select the asset and in its inspector, set Texture Type to "Sprite".

Page **6** of **6**

DOCUMENT: Tutorial **revision 2**

Basic Tutorial

for *ConText – A Text/Choice Adventure Game Framework*

This tutorial will guide you through the interface of the framework and in the process you will create a basic text adventure game.

Part 1 covers the setup of the game engine the framework is using and the importing of the framework files.

Part 2 covers the steps to creating a simple small game with the framework, guiding you through the creation of characters, modules, linking of modules and changing settings regarding the visual appearance of your game.

Parts 3 and 4 contain troubleshooting suggestions and some information about importing (media) files into Unity.

Any further or more detailed topics are covered in the Documentation (Documentation_UserStudy2_English.pdf).

Part 1 – Setup (7 steps)

1. Before anything else, you will need to download and install Unity Personal Edition from here: https://unity3d.com/get-unity/download?ref=personal

   Info: Unity is the game engine this framework was developed in and is to be used in. The first time you start Unity, you will need to create/login with an account. This account is necessary to verify your license information. The account can be created and used free of charge.

   *Note: during installation you will be asked which components are to be installed. If you don't intend to do any custom programming, you can deselect **Visual Studio 2015 (Community)** and you may also deselect any **compilers** for platforms you don't intend to put your game on (e.g. Android/iOS/Windows Phone/WebGL compiler etc.), likely you will only need the Android compiler for initial testing. Especially deselecting Visual Studio will increase installation speed a lot due to reduced download size.*
2. Once you have installed Unity, you will need to either use the ConText package (ConText_UserStudy2.unitypackage) provided in your initial invitation email or download it from here:
   https://www.dropbox.com/sh/vrao65c3ym3nkhc/AAC_WXLxHcPu04Gkaxne7wh9a?dl=0.
   You may download the entire folder which contains the UnityPackage, a *'ConText Project.wlt'* file as well as all PDFs also attached to the email.
   A Unity Package is a collection of files and objects that can be integrated into Unity. Think of them as a sort of special zip/container file for Unity.

Page **1** of **12**

DOCUMENT: Tutorial **revision 2**

3. Start Unity, log in with your account and select **NEW**. On the following screen, enter a name for your project and select **2D**.

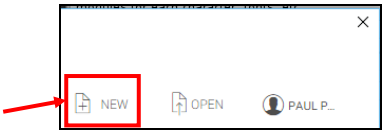


Figure 1: NEW

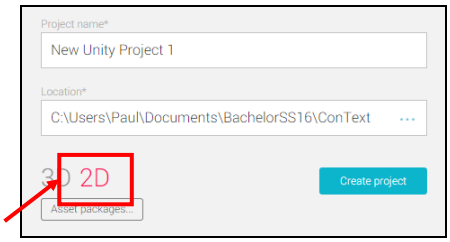


Figure 2: Create

4. Import the UnityPackage into Unity through **Assets – Import Package – Custom Package…** . A prompt will ask you to select the file location on your computer. Do so and confirm the choice. A Unity window will pop up listing all items that can be imported from that package. At the bottom of that window, select **All** and then click **Import**.

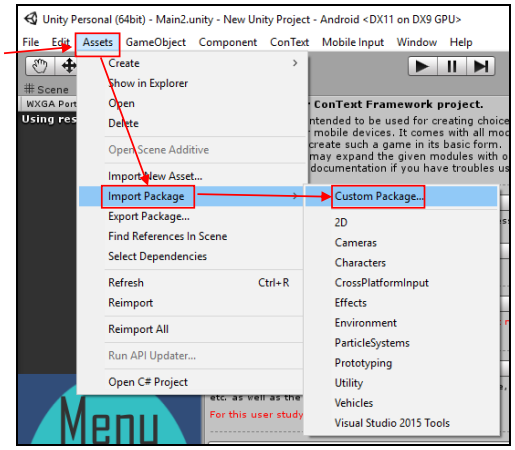


Figure 3: Import package

5. Next, you will need *to close Unity again* to add in the last part, the default framework layout. For this, the *'ConText Project.wlt'* file from the original package needs to be copied to the Unity Editor Layout folder. In Windows, that is **%APPDATA%\Unity\Editor-5.x\Preferences\Layouts**, in Mac OS, it is **~/Library/Preferences/Unity/Editor-5.x/Layouts**

DOCUMENT: Tutorial **revision 2**

(Copy & paste these directory paths to the top address bar in Windows Explorer/Mac Finder and press Enter to access the respective folder)

Info: This step only needs to be done once per Unity installation (i.e. usually once per computer). Technically you may also manually create the above listed folder and put the .wlt file in it.

6. Almost done setting up the framework. Start Unity again, open your now existing project in the initial screen. Once it is open, in the upper right, select **ConText Project** as the Layout. Ignore the red warning you will now see in the center window.

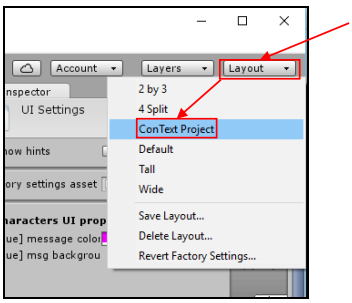


Figure 4: Layout

7. In the bottom Project/Asset folder window, select the top most folder **Assets** and double click **Main** to open the main ConText/Unity scene.

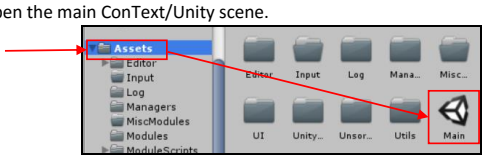


Figure 5: Open Main

DOCUMENT: Tutorial **revision 2**

Now your Unity editor should look like this:

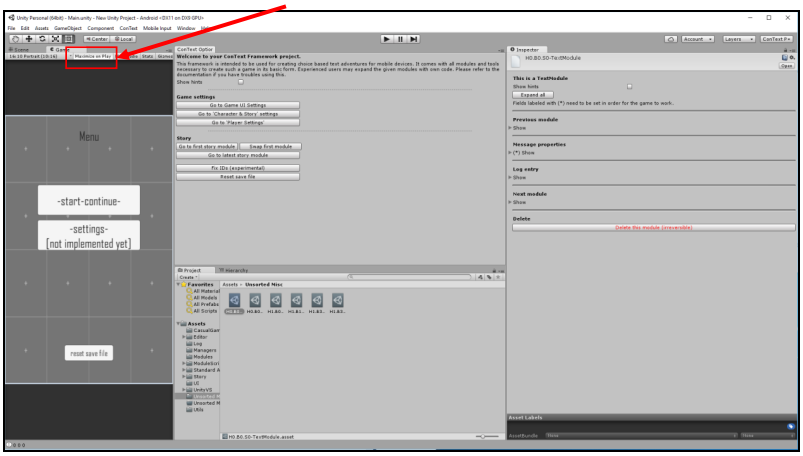


Figure 6: Overview

If so, the last thing to do is deselect **Maximize on Play** at the top of the left Game preview window, then you are done setting up the framework and can continue with Part 2 of the tutorial. If not, make sure you did all steps as described. If you are sure you did not miss a step and the overview is still wrong, feel free to contact me at paul@preissner-muc.de.

DOCUMENT: Tutorial **revision 2**

Part 2 – Sample game (10 steps)

In this part you will use the framework to create a simple text adventure game with a few different modules with the framework.

Note: the next steps mention the term "asset" multiple times. Assets in Unity are essentially all actual individual files representing objects. These objects can be or contain images, sounds, program code, GameObjects (i.e. objects that can be put into the actual game world/level), Prefabs (i.e. multiple GameObjects grouped together into one), Fonts, etc. Essentially everything that is permanently saved in files for the game or editor. "Asset" is thus an overarching term for all such files.

1. Given the layout as in Figure 6, the left part shows your *game preview window*. When you test/play your creation, it will be displayed in that window. The center part has the *ConText Overview* window in the top half, intended as a sort of hub from which you can go straight to the most important settings and windows. The bottom half is the *Project window*, which displays the directory structure of your asset folders, through which you may directly access story modules, settings files or ultimately script files. The right part is the *Inspector window*, which will always display the properties/details of the currently selected asset.

DOCUMENT: Tutorial **revision 2**

2. To get started, you will create the characters of your story. For this, click **Go to 'Character & Story' settings** in the *ConText Overview* window. This will open the Story Settings asset in the Inspector window. There, click **Add a character**, which will create a new Character asset and automatically display its details in the Inspector window. You may then give it a name and a unique character ID. Add as many characters as you need.

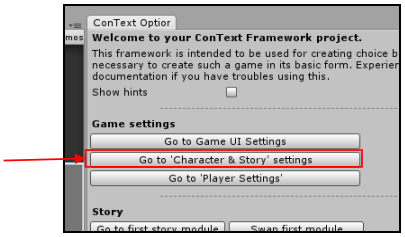


Figure 7: Go to story settings

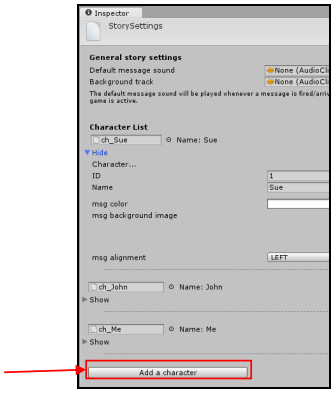


Figure 8: Add character

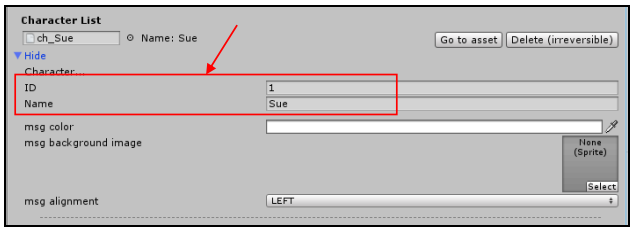


Figure 9: Customize character

DOCUMENT: Tutorial **revision 2**

3. Next, you will create the start of your story. For this, click **Create your story** in the ConText Overview. This will create the first node of your story, which (as of now) will be a Text module. The now created module will be displayed in the Inspector window. There you will need to choose the Character this message should be sent by clicking on the small circle next to the Character field and then double clicking an entry in the selection window that will pop up.

   Right below that is a text field which will hold the text this message consists of.

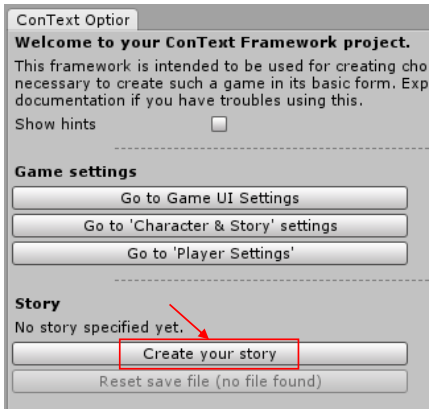


Figure 10: Create story

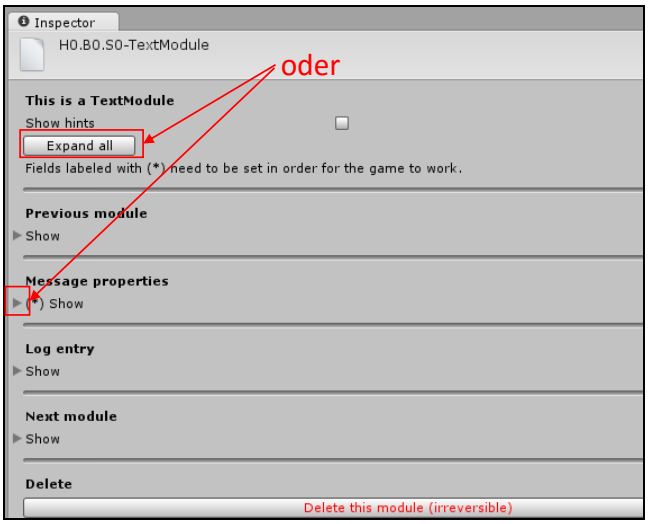


Figure 11: Message

Page **7** of **12**

DOCUMENT: Tutorial **revision 2**

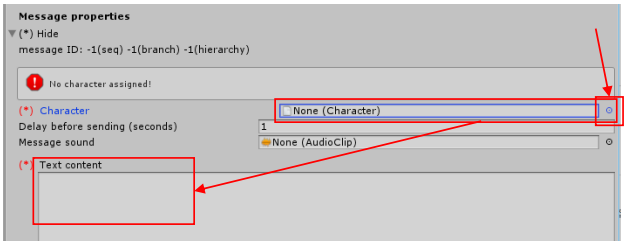


Figure 11b: Message

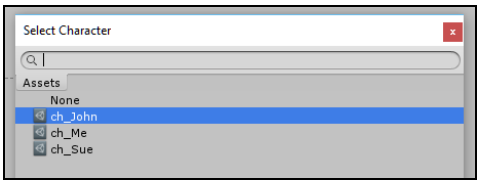


Figure 12: Select char.

You may also customize the Delay before sending (i.e. how long the game will wait before it virtually "sends" the message) or the specific sound the message should play upon sending *(Note: there is a default sound every message will play upon sending, so this one here should only be used for ones specific to that message)*.

4. Now your story consists of one message. Within each module, you can create the next module to follow after it (sometimes given a certain condition) through the *Next module* segment in its Inspector (or Replies for the Reply Module). For this tutorial, select the **module type as "Reply Module"** and click **Create next module**. This Create button will always reflect what type will be created. To navigate to the created module, click **Go To**.

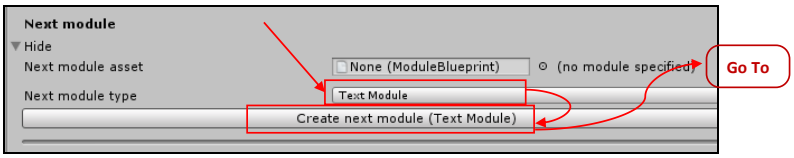


Figure 133: Next

DOCUMENT: Tutorial **revision 2**

5. This will have put that Reply module into focus in the Inspector.

   A specialty about the Reply module is that after displaying its text, it will not automatically send the next message, but leave the player with the respective number of replies to choose from. Each reply may lead to a different branch in the story.

   For this step in the tutorial, set up Character and Text as in step 3, then scroll down to the Replies segment. It works similarly to the Next module segment, with the difference being you can add multiple entries. **Add** at least one **reply option**. If you want to add multiple replies, simply repeat this action. For each reply, you can define the text its button should provide to the user (which is so far "new reply x"; *Note: this is not the text content of that reply, merely the text of the button that will trigger it*). Once you are done, click Go to next to the respective entry you want to configure next.

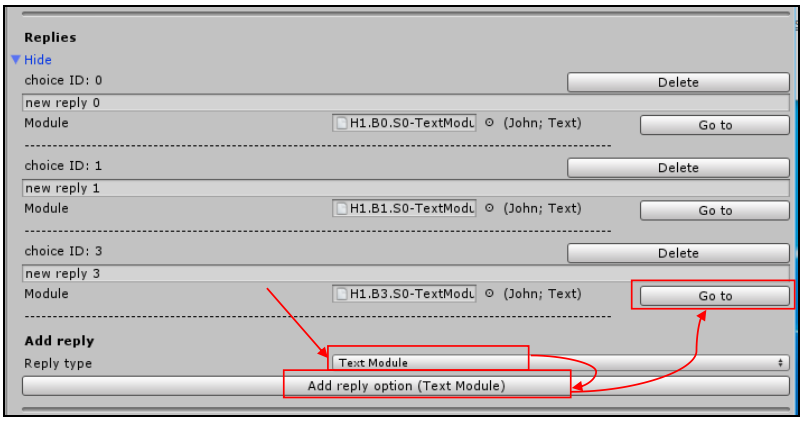


Figure 144: Replies

6. The selected module should be in focus now. For the sake of this tutorial, look at the *Previous module* segment at the top of the Inspector. If the previous module is specified, it will have a **Go to** button next to it. Clicking it will naturally take you to that previous module, in this case the Reply module from step 5. You may have seen by now that each previous or next module has that **Go to** button, which as of now is a simple way to traverse your story modules.

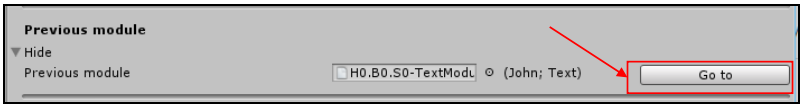


Figure 15: Go to

7. With this, you should know the basic process of creating modules. If you want to manually reconnect nodes in a different way, you can always click the little circle next to one and choose a different existing module. Be aware though, this can easily screw up the structure of the story and might lead to flawed playback if not done carefully.

DOCUMENT: Tutorial **revision 2**

8. At the bottom of each module's Inspector is a *Delete module* button, which will delete that module and attempt to properly connect the surrounding modules. Be aware, however, that when dealing with modules that have multiple next modules (like Reply modules), this may lose the connection to other branches.

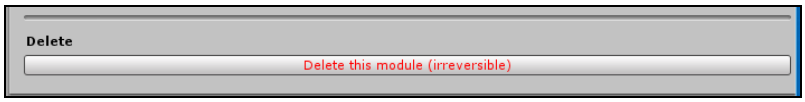


Figure 16: Delete

9. As the last step of this tutorial, click on **Go to Game UI Settings** in the ConText Overview. This will take you to the visual properties of the messages and modules, where you might change colors and background images for each character's message bubbles, the font and font size for messages as well as the font, font size, color and background images for each of the three main screens (Main menu, Text view, Log view).

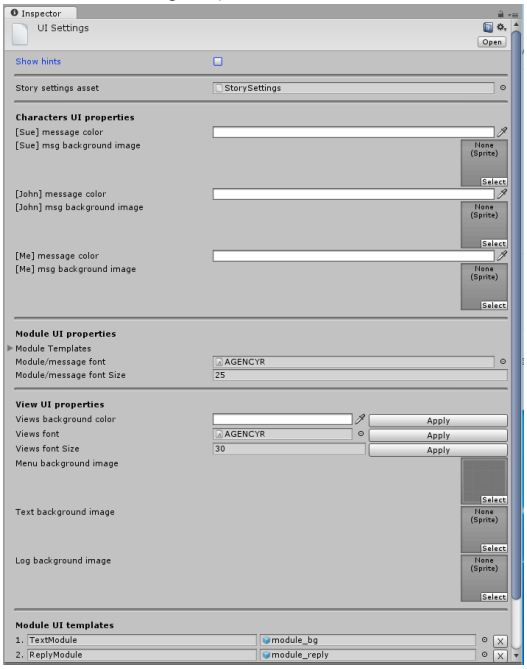


Figure 17: UI Settings

10. Go time! Now that you have set up the framework and created your first basic game with it, it is time to test it and play it. For this, Unity provides the editor play preview. At the top center of the program, you will see **three buttons, looking like typical Play, Pause and Forward** buttons.

DOCUMENT: Tutorial **revision 2**

They work as follows: Clicking the play button will start the game in editor preview, essentially showing you a close approximation of what the game currently is. Active Play mode is visually recognizable in the Play button being blue, as well as the whole interface becoming a darker gray. Pressing the Pause button will pause/un-pause the preview, again indicated by the button being blue when paused. Pressing the forward button will advance the game by individual frames (which would normally only take a few milliseconds per frame), but has likely no use for your game at this moment.

Important to keep in mind is that all changes made to assets while in Play mode are not saved, they will be reverted once play mode is disabled again.

Now, click on the **Play button** at the top center of the tool and check out what your story looks like in the preview window.

To stop game preview, simply press the (now blue) Play button again.

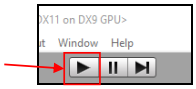


Figure 18: Play!

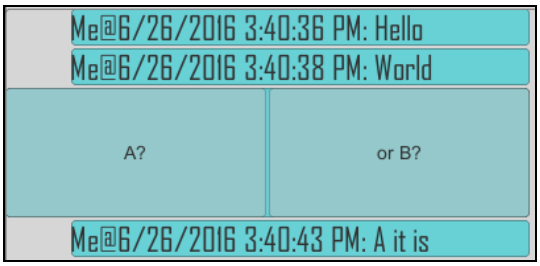


Figure 19: Small beginnings

DOCUMENT: Tutorial **revision 2**

Part 3 – Troubleshooting

Q: My new story modules are not showing up in the game preview!

A: There are multiple possible causes for this. The most likely are:

1) Play mode is still engaged. Make sure the Play button is not blue.

2) Previously existing story connections changed, as such the existing save file does not fit the storyline anymore. To fix this, click **Delete Save File** in the ConText Overview window.

3) Somehow the IDs of one or more modules in the storyline were corrupted. To fix this, click **Fix IDs** in the ConText Overview window (*Note: Fix IDs is still experimental and might not resolve the issue*).

Part 4 – Additional info about importing files into Unity

It may likely happen that you want to import custom image files to use as background or in an image module or perhaps different fonts.

Files can generally be imported into Unity by simply dragging and dropping them into the Project/Asset folder window. So you can just right click into any folder in that Project window, select Create – Folder to add a separate folder for your files, then drag and drop them in there.

In the case of image files, these will by default be imported as Sprites given you project was set to 2D upon creation. If that is not the case and the image is not available for selection as such an image, select the asset and in its inspector, set Texture Type to “Sprite”.

A.1.4. Documentation

DOCUMENT: Documentation **revision 1**

Basic Documentation for *ConText – A Text/Choice Adventure Game Framework*

This brief documentation attempts to explain the core features and structure of the framework.

It is by no means final or as detailed as possible. Feel free to suggest additions you'd find helpful.

Module system

The fundamental idea behind the framework is to provide a flexible and modular system for creating text/choice adventure games. In order to achieve this, messages are based on modules. There are four types of modules in the framework: Text, Image, TicTacToe and Reply Module. Their parent class is ModuleBlueprint. The basic implementation of a module contains a module ID comprised of four integers (hierarchy ID, branch ID, sequential ID, subpart ID) which uniquely designates each module, the sending character of the message, the content of the message, a previous as well as up to several next modules and a log entry.

Additionally, each module type is mapped to a specific UI template in the UI settings.

Depending on the individual implementation, certain details may change, such as the type of the message's content or the number of next modules.

The way the modules work in playback is as follows: The module manager triggers the first module of the story and as long as no user input is required will keep firing the respective next module until some form of user input is expected (or until a module for some reason breaks the cycle and does not tell the module manager which module is next). At that point, the automatic stream is stopped and has to be restarted by whichever user input is expected. This for example happens with the Reply Module, where pressing any reply will start the automatic stream again with whichever module was coupled with that reply. Whenever a module is fired and has an attached log entry, the respective entry is added to the log list or updated if it already is an element of the list.

Managers

The entire eventual game and parts of the framework use several manager classes to control actions, the UI, etc.

The Module Manager handles most manner of actions regarding the modules, such as the automatic stream, initial loading of existing and saving of new story progress on a superficial level and tracking of all currently instanced modules.

The Log Manager does likewise for the log entries.

The State Manager provides lower level functions for handling save files and tracks the current game state.

The UI manager handles instancing of modules and their representation in the game as well as switching of the three main screens.

Unify is not a manager class per se, but unifies access to all managers into one class.

Page **1** of **2**

DOCUMENT: Documentation **revision 1**

Story Settings and Characters

Characters and story settings are simple classes and mostly used for data storage. Characters each have their name, color of the messages, background image for messages (i.e. for more detailed message bubbles), a character ID and message bubble alignment.

The Story Settings contain a list of characters and a corresponding string list of just their names (for internal use).

UI Settings

The UI Settings store a range of data like the font and font size for modules, font, font size and color for the main screens as well as the list of pairings between module UI templates and module scripts. (Note here: due to internal constraints, these pairings consist of the template as a GameObject and the script represented by its class name only. If you want to add a custom pairing of your own, make sure the string value is exactly the class name of the intended class.)

Custom inspectors

The framework contains a custom Unity Inspector for each type of module as well as for UI Settings, Story Settings, Characters and log entries. These inspectors actually implement a lot of logic as they apply data and settings considering existing conditions and constraints.

The inspectors for modules have a parent class called ModuleInspectorAncestor which is intended to be a centralized place for handling creation of modules depending on a selected type. When you create a custom module class of your own, it is advised you expand the two provided functions in that class as well as the ModuleTypes and ModuleTypeEnumDescriptions enums in the Module Manager. That way your module will be available for selection in each module inspector.

Custom module classes

When creating a custom module class of your own, you should make it inherit from the ModuleBlueprint class and make sure to implement and thus possibly override all functions provided in that class. See the comments/descriptions for these functions in the class itself to learn what they do.

DOCUMENT: Documentation **revision 2**

Basic Documentation for *ConText – A Text/Choice Adventure Game Framework*

This brief documentation attempts to explain the core features and structure of the framework.

It is by no means final or as detailed as possible. Feel free to suggest additions you'd find helpful.

This documentation does only cover topics specific to this framework, features it adds on top of what Unity can do in itself. Thus, if one wants to know more about basic scripting and concepts of Unity, it's advised one look at the official Unity documentation and tutorials as well, to be found here:

https://unity3d.com/learn/tutorials

https://unity3d.com/learn/tutorials/topics/scripting

https://docs.unity3d.com/ScriptReference/

As well as perhaps the largest unofficial documentation: http://wiki.unity3d.com/index.php/Scripts

Topics of this documentation:

1. ConText Options Back to top

The ConText Options screen is intended to provide quick access to the most important core features of the framework right from the main screen. That includes

shortcuts to the **Game settings** comprised of

Game UI Settings,

'Character & Story' settings and

'Player Settings', as well as

shortcuts to **Story** related screens like

the **first story module**,

the **latest story module** (as far as possible, with branching storylines this will only return the single highest branch),

an option to attempt to **fix IDs** of the active storyline (which is experimental and will only work correctly given the function behind it is implemented correctly in each module's code) and

DOCUMENT: Documentation **revision 2**

to **reset the save file** (which should be used whenever elements in the existing storyline are changed).

2. Module system Back to top

The fundamental idea behind the framework is to provide a flexible and modular system for creating text/choice adventure games. In order to achieve this, messages are based on modules. There are four types of modules in the framework: **Text, Image, TicTacToe and Reply Module**. Their parent class is **ModuleBlueprint**.

The basic implementation of a module contains

a **module ID comprised of four integers (hierarchy ID, branch ID, sequential ID, subpart ID**, however the subpart ID is not set manually**)** which uniquely designates each module,

the **sending character** of the message,

the **content** of the message,

a **previous module**

as well as **up to several next modules** and

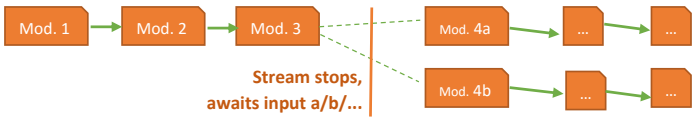
a **log entry**.

Depending on the individual implementation, certain details may change, such as the type of the message's content or the number of next modules.

(Additionally, each module type is mapped to a specific UI template in the UI settings (4.).)

The way the modules work in playback is as follows:

The module manager triggers the first module of the story and as long as no user input is required will keep firing the respective next module until a module does not tell the module manager which module is next (e.g. to await user input). At that point, the automatic stream is stopped and has to be restarted by that latest module (e.g. with the Reply Module, where pressing any reply will start the automatic stream again with whichever module was coupled to that reply). Whenever a module is fired and has an attached log entry, the respective entry is added to the log list or updated if it already is an element of the list.



DOCUMENT: Documentation **revision 2**

3. Managers Back to top

The entire eventual game and parts of the framework use several manager classes to control actions, the UI, module stream, logs, game state.

The **Module Manager** handles most manner of actions regarding the modules, such as the upkeep of the automatic stream, initial loading of existing and saving of new story progress on a superficial level, tracking of all currently instanced modules as well as resetting previously fired modules when resetting the save file within the game.

The **Log Manager** does likewise for the log entries.

The **State Manager** provides lower level functions for creating, loading and deleting save files and tracks the current game state.

The **UI manager** handles instancing of modules and their representation in the game as well as switching of the three main screens.

Unify is not a manager class per se, but unifies access to all managers into one class.

4. Characters & Story settings Back to top

Character is mostly used for data storage. Characters each contain their **name**, **messages' color, background image** (i.e. for more detailed message bubbles) **and alignment**, as well as their **character ID** (which until now is not specifically used).

The **Story Settings** (or 'Character & Story' settings) contain a **list of Characters** and a corresponding string list of just their names (for internal use), as well as both a variable for the **default message sound** to be played whenever a message is triggered and for a **background track**, which will be played on loop while the game is running.

5. UI Settings Back to top

The UI Settings store a range of data including

the **font and font size for modules**,

the **font and font size and color for the three main screens**

as well as the **list of pairings between module UI templates and module scripts**. (Note here: due to internal constraints, these pairings consist of the template as a GameObject and the script represented by its class name only. If you want to add a custom pairing of your own, make sure the string value is exactly the class name of the intended class.)

When viewed through the Inspector, the **Module UI properties** segment of the UI Settings includes a list of **Module Templates**. This should contain all prefabs that represent a module's UI template.

The **Module UI templates** segment at the bottom shows list of pairings between module UI templates and module scripts. If you want to add a custom pairing of your own, make sure the string value is exactly the class name of the intended class.

DOCUMENT: Documentation **revision 2**

6. Custom inspectors Back to top

The framework contains a custom Unity Inspector for each type of module as well as for UI Settings, Story Settings, Characters and log entries. These inspectors actually implement a lot of logistic logic as they apply data and settings considering existing conditions and constraints.

The inspectors for modules have a parent class called **ModuleInspectorAncestor** which is intended to be a centralized place for common functionality among the inspectors.

The class specifies the **OnEnable** function in its basic form, defining the labels for each section of the inspector as well as initializing the foldout status variables. This should ideally only be extended, not overridden for custom inspectors.

The class specifies the basic structure of the **OnInspectorGUI** function, consisting of six functions describing the six parts of the inspector layout. These parts are **PartInfo, PartPrevious, PartMessage, PartLog, PartNext, PartDelete**. Each of them is hidden in a foldout to give the inspector a cleaner look.

PartInfo includes basic utility functions such as Expand/Collapse all foldouts, toggle hints, and should be overridden and extended should one wish to display additional description or utilities at the top of the inspector.

PartPrevious includes any functionality related to the previous module, which by default is display of the object field specifying the module and a button to navigate directly to that module.

PartMessage by default holds only functionality to display an object field for the message sound and is the most likely to be overridden in another custom module inspector. It should contain all functionality regarding the message content such as the character, the message/module ID and of course the content itself in whichever form (e.g. text or image).

PartLog provides functionality for linking the respective log entry to a module, similar to how PartPrevious does for the previous module.

PartNext works analogously to PartPrevious but for the next module, with the addition that one can choose the type of module to create. This should be overridden when a custom module should offer multiple next modules, as is for example the case for ReplyModule and TicTacToe.

PartDelete offers functionality for deleting the currently focused module. By default it assumes one previous and one next module, upon pressing the Delete button will ask to confirm the choice, then connects the two surrounding modules to close the gap. For modules containing multiple next modules, this function should be overridden in order to make sure the surrounding modules can be connected correctly.

When you create a custom module class of your own, it is advised you expand and or override the Part[…] functions as necessary.

In order to make custom modules available for selection in the module inspectors (e.g. as next modules), first the **ModuleTypes** and **ModuleTypeEnumDescriptions** enums in the Module Manager need to be expanded with values (enum ID, name string). Secondly, in the ModuleInspectorAncestor class contains another two functions related to the handling of

DOCUMENT: Documentation **revision 2**

custom modules. For one, the **getShortDesc** function aims to provide a short description string for any custom module. It is used to display the little "(name; type)" info in the module inspectors. The other function is **createNextModule**, which is used to create and add actual asset instances of modules as e.g. next modules. It uses a switch statement over the ModuleTypes enum in Module Manager to distinguish which module type to use for creation. It also initializes the handful of necessary variable values such as the sending character and the module IDs.

7. Custom module classes Back to top

When creating a custom module class of your own, you should make it inherit from the **ModuleBlueprint** class and make sure to implement and thus possibly override all functions provided in that class. See the comments/descriptions for these functions in the class itself to learn what they do.

It is likely if not certain that the functions one will need to override are

setContent, which is dependent on the actual structure of the UI template, and is supposed to put whatever content is specified for the module into the actual UI instance,

getNextPart, which when called returns whatever is the next module for the automatic stream to fire. Note here, if you want to stop the automatic stream (e.g. to await user input), simply return null.

getModForChoice, which is given the ID of a choice and an **IDChoiceCapsule**. This function is called when loading existing story progress and expects in return the module that corresponds to the given choice ID. For example for a reply module with three replies, getModForChoice(2, […]) should return the second reply's module. The IDChoiceCapsule does not necessarily have to be considered, but can be used to check whether the modules match up (loaded data vs asset)

getHighestModule, which is a quasi-recursive function supposed to return the module furthest in the story line. Essentially, this needs to return the result of the function being called on the next module, or if the next module is not set, return the current module. If multiple next modules are set, query each next module and return whichever result has the largest/highest ID.

fixNextIDs, which is intended to repair the IDs of the story line. It should check for whether there is a next module, if so, set its ID according to the current module's ID (i.e. seq +1, branch and hierarchy remain), then call the function on the next module. Similar when multiple next modules are present (i.e. seq remains, branch increasing with each next, hierarchy +1 across all next modules)

resetModule, which should only be necessary when the module may be split into multiple subparts or contains data that is changed during runtime and needs to be reset for repeated firing (e.g. as in TextModule).

pushChoice may only possibly need to be changed. It is supposed to forward the IDChoiceCapsule corresponding to this module and being created at runtime to the choices list in the module manager. It is unlikely that a user needs to change it. It is called whenever a user's reply/next module choice (which may also be -1 if the module has only a single next

DOCUMENT: Documentation **revision 2**

module) is made. In that case, whichever class or instance calls the function so far generates the IDChoiceCapsule depending on that choice on its own and calls the function with it.

8. Export to Android Back to top

If you want to export your game to Android for testing, you should read into the Build process for Unity at

https://docs.unity3d.com/Manual/android-GettingStarted.html

and specifically follow the Android SDK setup instructions at

https://docs.unity3d.com/Manual/android-sdksetup.html

Once you have completed those steps, in Unity click on **File** -> **Build Settings…**.

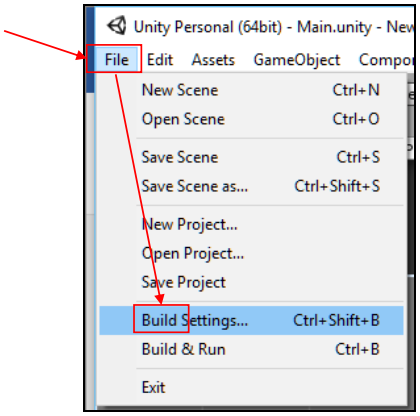


Figure 1: To Build

DOCUMENT: Documentation **revision 2**

This will open Unity's build settings window, where you can choose the platform to build for (**Android** since these instructions are about Android). Once **Android** is selected, you may set Texture Compression to ETC which may improve performance a bit. Additionally, while the game scene/level is open in the editor, press Add Open Scenes. This will add the scene to the actual standalone game build.

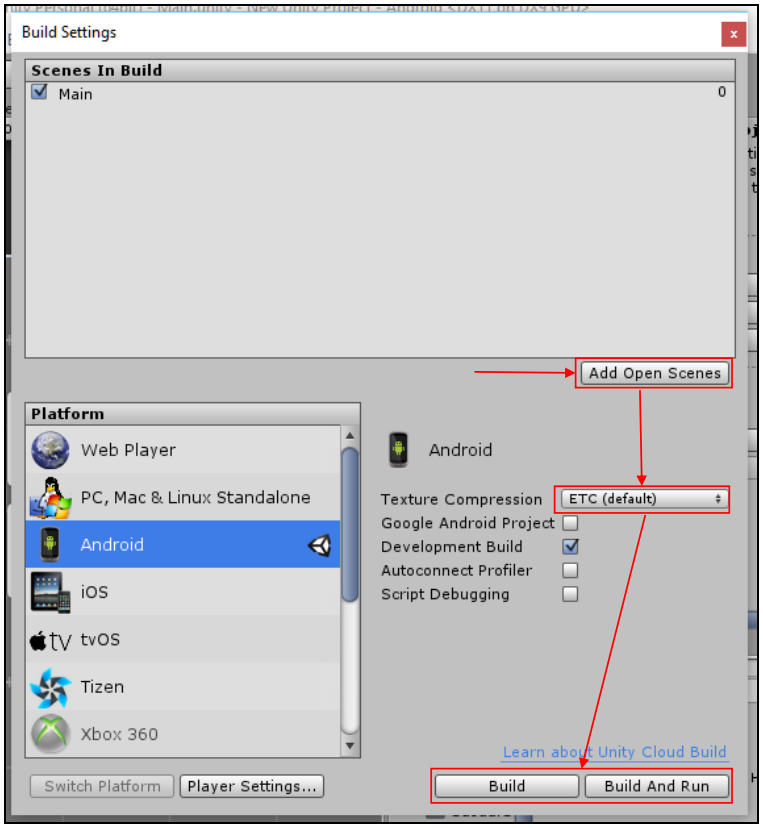


Figure 2: Build Settings

Once this is done, you may click **Build** or **Build And Run**. The difference here is

Build: this will prompt you to specify the name and location of the .apk file that will contain the standalone build of the game. After the process is done, you can move the .apk file manually over to your Android device, install it and then play.

Build And Run: this will prompt the same as Build, but after the process is done will attempt to run the game as well right away.

DOCUMENT: Documentation **revision 2**

9. FAQ Back to top

Q: What do I do if I want to create a new story?

A: So far there is no way of just creating a new story in the framework except for altering the existing message modules. If you want to create a new story without altering existing assets, you will need to create a new Unity project and import the plugin (and layout file) again, as described in ConText Tutorial Part 1.

Page **8** of **8**

A.2. Other (?)

TODO.

List of Figures

List of Tables

Bibliography

[Ale14] Alex Mitchell et al., ed. *Interactive Storytelling - 7th International Conference on Interactive Digital Storytelling, ICIDS 2014, Singapore, Singapore, November 3-6, 2014*. Vol. 7. Springer-Verlag, 2014.

[Dan12] Danu Pranantha et al. "Puzzle-it: An HTML5 Serious Games Platform for Education." In: *E-Learning and Games for Training, Education, Health and Sports*. Ed. by S. G. et al. Vol. 7. Springer-Verlag, 2012.

[Har14] Hartmut Koenitz. "Five Theses for Interactive Digital Narrative." In: *Interactive Storytelling*. Ed. by A. M. et al. Vol. 7. Springer-Verlag, 2014.

[run16] runevision. *Why doesn't Layout Element have Max values?* English. Unity Technologies. 2014-2016. URL: `http://forum.unity3d.com/threads/why-doesnt-layout-element-have-max-values.274221/#post-1816127` (visited on 08/01/2016).

[Ste12] Stefan Göbel et al., ed. *E-Learning and Games for Training, Education, Health and Sports - 7th International Conference, Edutainment 2012 and 3rd International Conference, GameDays 2012, Darmstadt, Germany, September 18-20, 2012*. Vol. 7. Springer-Verlag, 2012.

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