Evaluation of Narrative Tools and Systems for Game Writers

GDEV60001 GAMES DEVELOPMENT PROJECT

Emily Perry
SUPERVISOR: LUKE BARSBY
SECOND SUPERVISOR: SHAUN REEVES

Contents

Key Words 2

Abstract 2

Introduction 3

Aims and Objectives 4

Literature Review 5

- Games and Narrative 5
- Narrative Models in Games 7
- Narrative Delivery and Systems 9
- Narrative Development and Tools 11

Research Methodology 14

Results and Findings 18

- Section 1 - Prior Experience 18
  - Programming Experience 18
  - Game Engine Experience 18
- Section 2 - System Usability Scale 20
  - System Usability Scale 20
- Section 3 – General Feedback 22
  - Overall Improvement Feedback 22
  - Approachability Feedback 23
  - Improving Approachability 25
  - Bug and Error Reporting 26

Discussion and Analysis 28

Conclusion 33

Recommendations 35

References 36

Appendices 39

- Appendix 1 – Methodology Materials 39
  - 1.1 – Social Media Pitch (X Thread) 39
  - 1.2 – Manu-Scriptwriter Testing Manual 39
  - 1.3 – Testing Questionnaire 51
- Appendix 2 – Results 53
  - 2.1 – Research Data 53

Key Words

Narrative, Story Scripting, Cutscene, Tools Development, User Experience, Usability

Abstract

Despite modern video games using increasing amounts of narrative elements in games, writers rely on programmers to implement their stories for them, which leads to a divide between the writing and writer. Better narratives could be implemented into games by removing this divide and allowing writers to implement their stories directly, without the need for programming or development skills.

This paper considers the development of a narrative tool which is easy to use for writers without experience in programming or game development, and how narrative tools can be improved in future to further empower writers while keeping usability in mind. By researching the elements and requirements of narratives in games and considering existing approaches to narrative tools and dialogue engines within the wider game industry, a method of narrative scripting is proposed and developed which makes use of natural language commands in English.

A study was conducted with the hypothesis that natural language scripting is easier for writers to use and understand, using a plugin of this system for the Unity Engine within a simple framework which provides a movable player character that can interact with static objects, and a small set of dialogue commands. The usability and points of improvement for the system was determined using a questionnaire sent to participants after using the tool to create a short scene. It was determined that the system had an above average usability score, and that the documentation for the system combined with the natural language element was appealing to use. However, strict grammatical requirements were detrimental for some participants, and other approaches are worth researching for comparative purposes in future.

Introduction

One of the most common tools in use for writing dialogue are spreadsheets (Bateman, 2021), or separate text files and documentation that require a developer to implement separately from the writer (Kauhanen, 2009). This requires back and forth between writers and developers every time edits are needed, or if a scene wasn't implemented correctly. This also requires liaison between the two disciplines regarding what is achievable within a given engine, forcing compromises in the creative vision for the game due to a possible lack of communication or transparency between developers and writers. Ultimately, this leads to a slower workflow for writers implementing narratives within games.

This system of work also requires a writer to have experience within the games industry. Ince writes in the introduction to his book (Ince, 2009) that a studio once expressed to him that working with writers without experience within the industry is a struggle. He theorises that their issue was not related to the quality of writing, but their lack of development knowledge and the lack of resources within the studio to accommodate for the change in discipline. This attitude towards games within the industry greatly limits who can work within it, requiring writers have experience with the game development process that they might not be able to acquire due to a lack of interest in training, and makes the field more difficult to get into. Supporting writers regardless of experience would greatly improve the diversity of writers hired to work on games, which could allow for more creative or artistic works within the field.

While various tools programmers within the industry have reported different methods and tools that work well for writers within their development teams (Gregory, 2014; Kipnis, 2014; Birke, 2015; Armstrong and Ewing, 2017), and there has been some research into the development of story scripting (Mclaughlin and Katchabaw, 2006; Zhang, McLaughlin and Katchabaw, 2007) there is a gap in the area regarding research into the usability of tools and systems that are easiest for writers without experience within the industry to understand and use.

The lack of development in this area compared to other disciplines within game development is odd, especially when considering the role narrative plays within games. In Ip's analysis of a set of commercial games (Ip, 2011), he found that up to 28% of a game's total game time could be dedicated to forms of narrative delivery, and up to 70% of all narrative in the studied games is delivered through cutscenes, text, or prompts. Enezi and Verbrugge (Al Enezi and Verbrugge, 2023) found that players benefitted from and found greater enjoyment in playing a stealth game with randomly moving guards and contextual dialogue barks that communicated their actions, while Toh (Toh, 2023) found overall that players enjoyed exploring the story in narrative games more than playing the game itself. Overall, there is evidence to show that a game's narrative benefits player enjoyment, so filling this gap in research towards writer friendly tools would aid in the development of games with better narratives.

Aims and Objectives

The aim of this study is to propose and develop an extensible, writer friendly tool for directing narratives in games, and determine what can be done to make the field more accessible to writers without experience in programming or game development.

To do so, the following objectives should be met:

Objective 1: Research the presentation of narratives in games.

Carry out a study of how narrative is presented within interactive fiction and games, and the features required to produce these narratives within a game engine. This should consider both static and dynamic elements used to convey dialogue, stories, or scenes and convey information to the player.

Objective 2: Research existing writer friendly tools.

Consider how elements of these systems are supported within game development editors and tools to allow for complex behaviour and stories, determining the advantages and disadvantages of different tools in use within commercial and proprietary game engines, focusing on usability for writers and non-programmers and considering compromises regarding the technical implementation.

Objective 3: Implement common narrative features into a simple system.

Design and develop a simple, user-friendly narrative tool for non-programmers, which implements a baseline narrative engine, a framework for simple narratives to be written into, and a simple tool or method through which writers can interface with the engine and develop their own narrative scenes within the framework.

Objective 4: Evaluate the usability of the developed system for non-programmers.

Determine the usability and approachability of the developed system for writers with varying degrees of experience within the industry, using a reproducible testing method which can be applied to other types of tooling than the system developed.

Objective 5: Consider wider applications for this research.

Use the results of testing and user feedback to determine how developers can make better game engines and tools for creatives and non-programmers and allow for more creative works within video games by supporting these disciplines.

Literature Review

Games and Narrative

Whether video games have narrative is a long-standing topic of debate between ludologists and narratologists. In an article following on from his thesis, Juul (Juul, 2001) argues that many computer games contain narrative elements, but that you cannot have a continuously interactive story. His thesis (Juul, 1999) follows the definition that narratives are a linear, fixed chain of events, while games are interactive and non-linear – works can be created with alternating narrative and interactive elements, but these cannot be considered true interactive fiction.

This is corroborated by Aarseth in his book (Malloy and Aarseth, 1998), who makes the argument that a football match and a story both consist of a succession of events, but that the actions within a football match are not narrative actions. He uses the term ergodic to refer to a situation in which a chain of events has been produced by the nontrivial efforts of one or more individuals or mechanisms, going on to argue that if narratives consist of description and narration, then Pac-Man has description in the images used to represent visual elements of the game, and ergodics as the forced succession of events, but it does not have narration inbuilt into the game .

Arguing against the conclusion that games are not narratives, Jones writes (Jones, 2008) that Juul assumes that narrative must always be a static artifact with the intention that the narrative will always be experienced in a particular manner, preventing works from having emergence or creativity as its purpose. Following this assumption, the audience is ignored as an independent entity with their own knowledge and experience that may consume the text in an unintended manner. He then adds that narrative is built by both the author and the audience, between the setting, characters, and plot, and how the reader reconstructs and reinterprets the writing – a split which is also present in the consumption of games such as SOCOM 3, which presents the moral idea that killing “terrorist” non-player characters (NPCs) is justified and “civilian” NPCs are not, while allowing Jones the freedom to think outside of the logic of the game and interpret it through his own values. Therefore, the argument that games cannot have narrative elements due to their interactive, emergent elements is disproven, where emergence is necessary for the discursive level of any narrative.

Despite this, Juul argues that the relationship between the reader of a story and the player of a video game is completely different, as the reader can only experience the plot as an outsider, experiencing a story as a prior event, while the player of a game is also able to undertake a role within the game itself. His thesis argues that interactive fiction is a utopian idea and presents an alternative to the desire for games to have content close to a novel – a hypothetical game that emphasises flexibility and possibilities. He notes that this would have a large amount of complexity which could not be readily simulated but suggests that this could be circumvented by making use of a program similar to Eliza, an early therapist program which responds to user inputs with simple questions to prompt further details (Weizenbaum, 1966) – ideally, anything that can give the impression of intelligence, rather than fully modelling it.

Façade, an interactive drama, is a recent example of innovation in this field. The player is given the freedom to influence how events play out, parsing natural language inputs as dialogue and rewarding the player with information or progression within the story, reacting to the input text where possible. In their paper discussing the director system used to drive the plot, Mateas and Stern (Mateas and Stern, 2003) describe the system as generative in the sense that it mixes and sequences pre-determined behaviours, but it does not generate those behaviours itself. It also does not achieve general purpose natural language understanding, focusing on set phrases that fit within the environment and ignoring or sometimes misinterpreting anything which falls outside of this pre-determined framework. Therefore, it cannot be considered truly interactive fiction on its own, according to Juul's definition.

In a more recent paper, Aarseth reflects on his previous work and argues that reducing the argument down to the idea that games do or do not have narrative is "unnuanced, untenable, and unproductive" (Aarseth, 2012). He considers that games and stories share several elements based on a hierarchical theory of narratology, which he breaks down into the world and its agents, objects and events. He compares this to a model by Jenkins (Jenkins, 2003), who describes categories of narrative as spatial, enacting, emergent and embedded, in which emergent narratives are not pre-structured events, but take shape through the gameplay itself, such as in The Sims. Aarseth argues against the idea that emergent narrative classifies as any interesting experience in a game, as this definition has no limit to the point that it becomes hard to distinguish narratives from other worldly experiences.

However, Jenkins also introduces his paper (Jenkins, 2003) by echoing the claim that not all games tell stories, suggesting that simple graphic games such as Tetris do not lend themselves to narrative exposition. This contrasts the abstract interpretation of the same game by Murray in her book, Hamlet on the Holodeck (Murray, 2017), who compares the tension of clearing lines as comparable to the overtasked lives of Americans in the 1990s, suggesting that the spatial ideas represented by the game could be represented in other mediums such as dance, in which this kind of association is more easily formed due to the humans that enact them. However, Jenkins argues against this interpretation, noting that while some ballets tell stories, storytelling is not an intrinsic or defining feature of dance as Murray suggests.

In his article, Juul (Juul, 2001) also considers that Space Invaders has a prehistory suggested in the title that the player must fight back against, but that it is impossible to restore the initial state of a world without invaders. Therefore, while the player is fighting to realise an ideal sequence of events, the act of playing the game is not this sequence. Juul concludes that some games use narratives for some purposes and suggests that his original claim that games and narratives are completely unrelated is untenable. This interpretation returns to Aarseth's idea (Malloy and Aarseth, 1998) that games have elements of narration and an ergodic succession of events, but no inbuilt narration.

Simons suggests in his own article (Simons, 2007) that the difference between a narrative and a game is merely a matter of perspective, that game theory and narratology converge at the level of history or story. She suggests that narratologists use the concept of kernels and satellites to distinguish between necessary events and events that, if removed, would not remove coherence or prevent the story from being recognisable. Aarseth also states (Aarseth, 2012) that satellites define the discourse, but the kernel is a key point of the story that can't be removed. Simons continues that to identify kernels, a narratologist must look at a story retrospectively, while a player's perspective is prospective, as the outcome of the game is still hidden in the future. She argues that the logic of narrative is moving towards the conception of narrative as an assemble of characters, settings and actions - a similar breakdown to Aarseth's model.

In his book, Domsch (Domsch, 2013) echoes a claim by Ryan (Ryan, 2006) that there is an elective affinity between computer games and narrative that explains why some but not all games have narrative, to which he adds that this affinity is explained by a common element of both fiction and games: rules, or suggestions to assume that something is the case. In fiction, elements are referred to as if they exist, while games follow rules as if they are necessary – life has non-negotiable rules in the form of physics, but there are no restrictions on what a person can do in the same way that a player has restrictions on where they can move or what they can say.

Considering narratives and stories in games, Ryan later considers (Ryan, 2009) the idea of narrative games versus playable stories and how they reflect the definition of ludus and paidia, different types of games. Paidia games refer to building imaginary scenarios with toys, with spontaneous rules and no specific goal, which she compares to a playable story in which the purpose of the player is not to beat the game, but to observe the evolution of the storyworld, such as in The Sims. Ludus games are transformations of abstract playfields into concrete fictional worlds with recognisable objects and characters, which is comparable to narrative games in which the player plays to win, and the story is a lure into the game world, such as in Grand Theft Auto.

Narrative Models in Games

The book Hamlet on the Holodeck (Murray, 2017) uses the holodeck from the Star Trek series as a reference for thinking about the future of interactive narratives – a system which can project a holographic world and people that can be interacted with as if in real life, telling the user stories with typical literary genres as if they starred within it. Murray considers digital environments to be procedural and participatory, or interactive, as well as spatial and encyclopaedic, or immersive. Interactive environments exhibit rule-generated behaviour that responds to input and are immersive in their power to represent navigable space and their ability to store and retrieve vast amounts of data.

In her article, Ryan (Ryan, 2009) states that it would take an incredibly advanced artificial intelligence to process the user's inputs and integrate this into a creative, well-formed plot, discussing compromises between the science fiction holodeck and existing technology. She emphasises three features of the holodeck described in Murray's book: the natural interface, in which users interact with the computer generated world in the same way as the real world; integration of user action within the story, or the creation of narrative from a user's choices; and dynamic creation of the story, computing and responding to the effects of those choices in real time and updating a model of the fictional world accordingly.

Rules of Play, a book covering design fundamentals in games (Tekinbas and Zimmerman, 2003), uses the structure of an embedded or emergent narrative to represent ways that a game system produces narrative, as defined by Leblanc in his GDC talk (LeBlanc, 2000), in which he describes emergent narrative as short vignettes, considering gameplay to be largely emergent, and embedded, authored narrative as a frame for interaction, limited to short, discrete, non-interactive moments. The book adds to these definitions, building from Jenkin's definitions of embedded, emergent, spatial and enacting narratives to define embedded narrative as pre-generated content that exists prior to interaction with the game, usually providing motivation for the events and actions of the game. On the other hand, emergent narratives arise from the set of rules governing interaction with the game system to provide an experience unique to each player – a constrained form of user action affecting the fictional world.

Different genres of game emphasize specific types of narrative – Rules of Play uses The Sims as an example, having a setting that resembles suburban southern California as embedded narrative which contextualizes the emergent narrative events that occur during play. These events are considered by Ryan previously (Ryan, 2009) to fall under a playable story, in which the player receives pleasure from coaxing a good story out of the system. On the other hand, adventure games such as The Secret of Monkey Island consists primarily of an embedded narrative with pre-scripted descriptions, dialogue, interactions and actions, in which the player has a limited emergent experience based on the order in which they progress and figure out puzzles. Aarseth concludes in his book (Malloy and

Aarseth, 1998) that “the standard concepts of narratology are not sufficient to explain the literary phenomena of adventure games” or their differences from other literature.

This concept of embedded narrative in games is different to the narratological definition, as discussed in Wei’s paper (Wei, 2010), who highlights a theory by Nelles (Nelles, 2020) that shifts in narrator, narrative or diegetic level, and reality mark the border between embedded and embedding narrative. Wei divides these categories further into horizontal embedding, vertical embedding, and modal embedding. Horizontal embedding refers to a shift in narrator, but not narrative level, such as when the narration is handed to an in-game character, the player, an object such as a book with an uncertain narrator, or moved into a flashback, so we hear the main and embedded stories side by side. Vertical Embedding requires a shift in both narrator and narrative level to create depth in the storytelling, referring to stories in dialogue, narrative objects such as journals in-game, non-interactive sequences that do not take place in flashbacks, and voice-over narration. Finally, modal embedding refers to a shift of the reality or storyworld, such as in dreams, hallucinations, or alternate dimensions, which can include shifts between game levels with different thematic designs.

Aarseth’s paper (Aarseth, 2012) breaks down narrativity into world, objects, agents, and events, stating that every game configures these elements differently. Gameworlds are physical or virtual structures with clear limits and geometry that can be explored directly by an independent agent. The game world is measured in ludic space, meaning the arena of gameplay, and extra-ludic, referring to the surrounding, non-playable space. Any objects in a game are categorized in terms of malleability – static and non-interactable, static and usable, destructible, changeable such as upgradable weapons, creatable such as crafting attributes into items, or inventible, where the player can create new objects. Characters can be classified in terms of depth and malleability as well, which he categorises as bots with no individual identity, shallow characters without much personality, and deep characters who change as the story progresses. Finally, he categorises events using the concept of kernels and satellites, referring to pure story, dynamic satellites in a playable story, dynamic kernels with multiple paths, and no kernels for a pure game.

In his paper, Lindley (Lindley, 2005) discusses different ways in which games structure narratives. He considers that a branching narrative refers to the change in how something is narrated, branching plot structure refers to alternative pathways through the representation of an overall plot, and branching story refers to the interactive selection of a representation of a story based on predefined elements. He suggests that actions in a game form a version of the concept of fictive blocks, as proposed by Mackay (Mackay, 2017), which are basic fragments or units of fictional or narrative significance which can be strung together to form a higher-level narrative. In games, fictive blocks have a tangible, predefined form as the constrained set of valid game moves, which then have a bearing on the player’s play style.

Lindley also considers various story structures, such as the three-act restorative structure (Rush, Dancyger and Keyt, 2023), in which a conflict is established, the implications are played out, and the conflict is resolved. He suggests that key scenes during these acts are typically achieved using cutscenes and non-interactive sequences, while the story itself is mostly a structure imposed on top of gameplay. He suggests that some elements that would satisfy story preferences in a game would be outside the scope of user-selectable moves, and that current design conventions for these to support dramatic performance and immersion are of limited effectiveness and poorly developed, arguing that good game design achieves better integration of the gameplay and narrative structures, but not all players will accept the approach taken.

In an article on narrative structures in games, Ip (Ip, 2011) suggests that simple games do not require extensive backstories, which have variable effectiveness in games. However, common techniques for narrative delivery included backstories, linear and branching game structures, the portrayal of emotion and reactive environments, and narrative structures such as the monomyth (Vogler, 1985; Campbell, 2008), with technical deliveries in the form of on-screen text, audio cues and various combinations of the above with gameplay and cutscenes. Ip found that narrative never exceeded a quarter of the total experience of the studied games.

Regarding nonlinear approaches, Ip considers that the player is given the impression of a greater degree of control than is possible, and that interactive cutscenes is one way that this choice is provided to players. However, branching structures are limited by the amount of content needed for each decision, plot changes, and maintaining a coherent story, and more work is necessary to keep up with audience demands in this area as hardware becomes more powerful. Despite this, he also found that there was a low ratio of kernel to satellite events and considers that key story events may not be sufficiently amplified by supporting scenes. Of the games studied, he noticed that linear games were prevalent but sometimes used branching structures to provide a greater sense of freedom in confined sections or provided side-quests as minor additions to create the illusion of branching.

Eladhari discusses structures in interactive narratives in her thesis (Eladhari, 2002), which Lindley summarizes as trees, exploratorium games with linear structures that allow exploration of the surroundings, parallel plot structure, nodal or dead-end structures, a modulated dynamic labyrinth structure where new interactions become available after different parts of the story have been experienced, open structures where links between places are open, and open structures with no story arc. Lindley then suggests that these can be modelled using formal graph theory, made up of nodes, links between nodes, and constraints on links such as directed links, conditional links, and restricted nodes. Once represented as a graph, the overall shape of an interactive narrative can be seen from the high-level topology. He also considers that nodes could contain any kind of interactive structure, allowing for nested substructures.

Eladhari's thesis defines the concept of object-oriented story construction as allowing all (relevant) objects in the world to have integrity and contain their own stories, functions, conditions, possible developments, and counter reactions. Having integrity means that the object's information is only available through the object itself and its conditions – an NPC cannot spill information that ought to arise later in the intrigue because the player cannot ask for it. Story driven games consist of a code level in the engine, framework, and game-specific code, the story level for narrative content, which she models using a flow chart, and the discourse level, which characterises the told order of the story as the player experiences the game, and private story discourses regarding individual objects.

Narrative Delivery and Systems

In his book for narrative skills in games, Bateman (Bateman, 2021) discusses various methods of narrative delivery through which a writer can get a story across to the player. He considers that different game types support different techniques for advancing the narrative, and the techniques used take shape from and reinforce the games they are part of to support player immersion.

Delivery methods discussed include:

- On-screen text, used for non-subtitled dialogue, tutorials, and on-screen artefacts such as notes, scrolls or books.

- Recorded dialogue (and subtitles), which can play at any time during gameplay or during cutscenes and scripted events. Some dialogue may have multiple variants.
- Static images, drawings, paintings, or computer generated (CG) stills. These can be used during loading or as part of a cutscene.
- Camera cases, or scenes created through camera movements within the game world, such as flybys.
- In-engine cutscenes which give a writer total control over events, allowing for visual polish and actions that cannot occur in-game.
- Scripted events, or brief moments where the game takes control of the camera or action to force certain events, while the player retains control of their avatar.
- FMV cutscenes and pre-recorded visuals, referring to computer graphics imagery (CGI) or live-action video, most useful for displaying events that the game engine can't handle.

Cheng discusses interactive cutscenes in his paper (Cheng, 2007), in which players must quickly press a button to affect the outcome of an action. He criticises this solution as a regression to interactive movies, where the player loses a sense of control in comparison to the standard mechanics of the game, rather than gaining it. However, he also suggests that these interactive cutscenes provide representational agency, where the player experiences agency in terms of a fictional figure, and suggests that having these rendered in-engine and keeping a uniform aesthetic and relative smoothness to transitions sustains the illusion of a coherent game world and improves this sense of agency.

He also considers a form of longform scripted events in which the player is presented with narrative information while retaining control of the player. He suggests that these require designers to balance the delivery of information with player agency, as the player can move away from a conversation and miss important information, since attention is not forced as in non-interactive sequences. However, he also suggests that this can improve immersion by subverting earlier instances of interactive scripted events, such as by removing elements of control if the player character has been tied up.

Bateman's book also proposes that non-interactive methods run the risk of disrupting pacing, forcing failures to accomplish a narrative goal which may cause frustration if not handled carefully, as it removes player choice or denies players the central role. This has the benefit of ensuring that events in the narrative can occur without interruption, allowing for cinematic use of narrative techniques commonly found in film and television. In his study, Ip (Ip, 2011) found that up to 70% of a game's narrative are communicated through cutscenes, which are frequently used in complex story-based games. He suggests that cutscenes are becoming a standard method of narration, despite being criticised as a passive mode of narrative that disrupts the interactive experience.

The book lists different types of dialogue engines that are available in games. In particular:

- Event-driven engines, in which different events trigger different responses. Events can be triggered by inputs and game state. This can also be used to drive commentary engines, which produce a stream of chatter based on current game events.
- Topic-driven engines, in which dialogue is triggered based on choices in a conversation. This can take the form of character scripts, which organise dialogue choices with a single NPC by a set of conditions, or token-based, in which a player has access to a series of tokens that they can present to any NPC, such as inventory items.

- Dialogue trees, which are converging and diverging chains of conversation, divided into segments. These usually create linear conversations as branching trees can cause a "combinatorial explosion", becoming costly and inefficient to develop.

One example of an event-driven engine is a dynamic or contextual dialogue system, as discussed by Elan Ruskin in his 2012 GDC talk (Ruskin, 2012). He discusses existing systems in which enemies react to the player's actions as they move across a level or perform certain actions by performing barks that convey their current state. These barks are usually tied directly to their decision-making and are used to inform the player, even while enemies are out of sight.

He then describes the system used in the development of Left 4 Dead, a game in which a player and 3 NPC characters driven by artificial intelligence (AI) shoot zombies to survive and complete missions. Throughout a level, objects are described using tags within the engine which trigger specific dialogue responses when the AI performs a given action or meets certain conditions. This concept can also be used to trigger conversations between multiple agents.

The engine used by The Last of Us (Gregory, 2014) builds on this system, adding random probabilities to prevent repetition in repeated lines, and prioritising events to ensure that only appropriate reactions occur, allowing traditional barks to interrupt idle conversation when combat starts. They suggest that the system could be improved with better Boolean logic and branching and considers that rules could be utilised for non-dialog mechanics.

Narrative Development and Tools

According to Bateman's book (Bateman, 2021), a writer must work within the capabilities of the game and the engine, producing their work in such a way as to make the content easy for the programmers to deal with. This means considering how text will fit onscreen, localisation into other languages, and file naming conventions for organisation. Localisation issues arise from translated phrases being longer in other languages or having a completely different meaning, which can be clarified by annotating lines with writing decisions, highlighting the intention, meaning, or context of a line, and indicating critical phrases. Annotation also benefits voice actors, as context, emotion, and emphasis can influence the performance of the line. In his GDC talk discussing dialog tools for Firewatch, Ewing (Armstrong and Ewing, 2017) mentions that they used pseudo-languages to test Unicode support and languages with longer phrasing without the need to translate the text, allowing them to sidestep this issue during development.

Firewatch's dialogue engine is also based on the system discussed in Ruskin and Gregory's talks (Ruskin, 2012; Gregory, 2014). Their dialog tools use a collection of variables stored in Blackboards and uses a visual scripting graph to listen to and trigger events. Events are viewed from a separate window to individually edit responses, targets, and other properties, including the type of event such as dialogue or changing variables. The dialogue itself is written in plaintext using very lightweight syntax to denote the conversation name, dialogue speakers and content, and choices, which is then imported into a separate database. The database allows them to track lines, translations, voice overs, and other metadata during development, and provides functionality to export lines into excel spreadsheets to produce a full script for voice actors. Bateman suggests that spreadsheets are easiest for game engines to understand, and useful for organising large amounts of data, but argues that some writers are intimidated by the format.

Bateman states that interactive scripting doesn't have one standardized format due to the use of different game engines, storytelling styles, and differences in linearity – a sentiment agreed by Despain (Despain, 2020). The use of standard text documentation and prose does have its uses as information resources for developers, but this style of writing is difficult to import into the engine. He suggests that screenplay formats are common in mediums with visual and spoken elements, inspired by stage play formats, which are intended to make information easy to understand within a short period of time. Games sometimes use a modified screenplay format as a compromise between human and computer readability, additionally allowing for dynamic elements, conditions and variables communicated through human readable pseudocode. This style of formatting is best for on rail narratives, but the script could be modified to accommodate for decisions. Despain suggests (Despain, 2020) that cutscenes, cinematics, and scripted events are best communicated in this format due to their linear, non-interactive format.

One solution investigating scripting for cutscenes (Zhang, McLaughlin and Katchabaw, 2007) makes use of an XML-based specification as a basis from which to add additional elements for video game cut-scenes, however they suggest that XML isn't a natural or convenient method to write with without the use of additional conversion software. Their system also doesn't cover cutscenes in 3D spaces, with additional functionality, testing, and platforms falling under future work.

In his master's thesis, Kauhanen evaluates a similar XML-based system (Kauhanen, 2009), in which a company developed tools to convert between word and spreadsheet documents and the XML format used by the game. However, he determined that the system was not intuitive or easy to learn and understand, and that defining XML rules was more difficult and required additional support. Kauhanen concludes that a domain-specific language and support with automatic mappings to game code would be more suitable. He investigates support for narrative scripting and determines a set of design patterns for game scripting tools to follow:

- Authoring tools should support non-programmers.
- Natural language and familiar, domain specific terminology is more accessible for creative writers.
- Support reuse of assets across multiple projects.
- Provide a means to quickly playtest and edit a story.

Supporting workflows where automation isn't possible is highlighted by Birke in his GDC talk (Birke, 2015), in which he briefly discusses the tools used for The Adventures of Bertram Fiddle, who suggests that he dislikes having users input text directly into a tool as it is error prone, particularly for non-technical users who need to learn to understand the format. Their solution was to create a tool that allows users to create the story and interaction within a scene using a sequence of blocks with instructions that can be edited in the inspector. Additional instruction blocks were created when requested by developers on the team. He suggested that controlling the types of instructions that developers could use within the tool reduced errors for the team and saved time. He also created a timeline based cutscene editor for cinematics to time dialogue for a given conversation and tweak the duration of different events.

The tools used for The Last of Us (Gregory, 2014), builds on Ruskin's talk, making use of its own scripting language with heavy code-style syntax which makes it easier to import into the engine during runtime for rapid iteration. Dialog files are split up by character and by act where possible, to allow multiple writers to work on the dialogue at any one time and prevent conflicting work within a team.

Ink is an open-source narrative scripting language developed for games with a text and choice-based interface in mind, styled after 'choose your own adventure' (CYOA) books and originally developed for the games "80 Days" and "Sorcery!". Humfrey discusses how flowcharts and changing the flow on a word-by-word basis would cause too many connections and too much branching, becoming unmanageable for the writers. The language uses a set of syntax to identify choices, variables, and different passages to move to (Humfrey, 2016), later adding support for 3D games to add emotions that change sprites, directions, animations, and interaction choices (Humfrey, 2017). He makes the argument that not all writers are this technical, but that there's little in the games industry that doesn't have that requirement.

The Ink editor, Inkle, has been compared against Twine (Interactive Fiction Technology Foundation, no date) and the paid Celtx scriptwriting software (Clarke and Zioga, 2022), both of which use a tree and node-based interface to structure narrative, for use during the development of an interactive film. The study determined that Inkle had better functionality for testing and debugging, able to detect loose ends and preview the final format. However, they note that the coding required can be challenging for writers without prior experience, and exporting to HTML was less accessible than exporting to PDF, as with Celtx, which they concluded was best for their purposes.

The Versu engine was developed by Richard Evans to create interactive stories, focusing on character interaction and choice, while its scripting language was developed by Graham Nelson, previously known for his work on Inform, another interactive fiction creation tool (Nelson, 2014). Prompter was designed with natural language in mind, aiming to achieve faster development and human readability by making use of scriptwriting structures. Defining characters and scenes for the engine uses very little syntax based on structured sentences. He estimates that test stories written in the engine using its original, syntax heavy language (Praxis) took a month to write, while Prompter allowed the same to be written in a day.

The Novella model (Green *et al.*, 2018) was developed based on work by various ludologists, including Aarseth's earlier model (Aarseth, 2012), breaking down elements into world, objects, characters and events. The model is based on a flow chart, like Twine, starting from an overall Story that stores an initial and current scenario based around a Context object. Contexts consist of a set of gameplay rules, which can include flow nodes and graphs, dialogues, and cutscenes. However, the proposed system was never developed into an authoring environment for game development and fails to fully define cutscenes, both of which are concluded as future work.

Patel discusses the tools that have been used by Obsidian Entertainment for 8 years, as of her GDC talk in 2019 (Patel and Szymczyk, 2019), which consists of a dialogue tree, made up of individual nodes and with the ability to fine-tune the behaviour of each node or branch. She highlights various quality of life features of the tool, such as automatic spacing, hotkeys, and writing text directly into nodes, which keep the tool organised and makes workflows faster for quick iteration, while mirroring the player's experience. She also discusses the tool's ability to interface with conditional scripts and trigger game functionality, which she argues allows conversations to have an impact on the rest of the game.

In the development of The Witcher 3 (Tomsinski, 2016), quests are structured using a flow chart, which contain dialogue nodes with nested dialogue flow charts. These link together different sections and choices, written using a screenplay format. Designers then determine which models are used within the scene. Dialogue nodes can include quest logic and update variables to affect the world itself. Separately, a conversation is edited in a timeline tool to specify animation, dialogue and voiceover timings, camera transitions, and other basic properties necessary for cutscenes, some of which can be automated from the screenplay to generate a simple scene that can be tweaked later.

Research Methodology

This research poses the hypothesis that writers without prior experience with programming or game development may find a system which uses a custom narrative scripting language using natural language sentences to be easier to understand and write with when developing scenes within a game or game engine.

Due to the gap in research into this area, it would be most beneficial to gather primary research directly from subjects to contribute to the current understanding of writer friendly narrative tools, which may inspire further research into the field.

The population for this research includes people above the age of 18, regardless of gender, with an interest or experience in writing for games or the development of narrative games. Allowing participants with a range of experience with both writing and games engines or development would confirm the system's viability for use in narrative development for games and could provide an insight into how much their experience, if any, factors in on the system's usability.

Participants are to be recruited a week before testing, using a mixture of voluntary response sampling, purposive sampling, and snowball sampling. A short pitch of the concept and what would occur during testing would be posted on various social media websites, requesting that anyone interested reaches out for more information to sign up – see Appendix 1.1. The same pitch will be sent to specific students or alumni of computer science or game development courses, including programmers, developers, designers and writers, as well as students with an interest in writing, regardless of formal education in the subject. Participants are allowed to forward the pitch to other possible interested parties. The expected sample size is 10 participants.

Any volunteers are required to be able to run a copy of the Unity 6 editor and text editor of their choice to take part. This will be communicated using an information sheet covering the testing process in detail and what will be expected of these volunteers, as well as ensuring they understand that they are aware of their right to withdraw and understand how their data will be processed under the General Data Protection Regulation. To confirm participation, an informed consent form is required from each subject.

A small artefact is to be developed for testing over the course of 6 weeks, following the design principles proposed by Kauhanen (Kauhanen, 2009), while keeping the scope of this research in mind. As discussed previously, these principles focus on support for non-programmers, writer friendly terminology, ease of asset re-use, and quick debugging tools. In particular, the implementation should focus on supporting non-programmers and familiar terminology, while relying on Unity 6 to provide debugging support and use of prefabs to support level creation and asset reuse, including object placement, which would not be supported by the tool itself.

This is to be developed in C#, using Unity 6, version 6000.0.19f1, and Rider 2022.3.2 as the Integrated Development Environment (IDE). Unity 6 was chosen over other commercial engines, such as Unreal Engine 5 or Godot 4, due to having lower system requirements which would make testing more accessible to low end computers, reliability as a commercial engine, and its use of C#, which can handle memory using garbage collection and therefore reduce the possibility that a writer could cause a memory leak unknowingly, an issue which would be difficult to debug without technical knowledge. Rider was chosen for development due to the researcher's preference, as it is a fast, powerful IDE with Unity integration for debugging, making the artefact easier to develop and test.

Based on various information provided by the literature review, the basis of the artefact for the purposes of this research is a way to parse and interpret commands in a custom language, while allowing functions to be defined and implemented separately within C#. Planning for additional flexibility within the tool would allow it to be used for a variety of different games, without tying the functionality to any one genre or implementation. The natural language element comes from how functions are defined, using human readable sentences in which different words, such as nouns or adjectives, act as input parameters. To increase the tool's viability for game development, compatibility with an event driven engine could also be implemented, ideally allowing for use with a contextual dialogue system, should time constraints allow.

To make use of this implementation, testers would need access to a simple framework, with basic functionality such as a movable player and the ability to interact with objects using an event-based system which could theoretically allow for additional events in future. This framework should also define a set of standard commands within the custom language, decided based on Bateman's methods of narrative delivery (Bateman, 2021). To keep scope small, this should focus on on-screen text and camera cases, as these cover simple use cases without requiring heavy asset usage, as in the case of pre-recorded visuals, dialogue, or images, which would reduce the amount of additional assets that might be needed for the artefact.

At the start of the testing period, subjects are to be sent a folder to download using GitHub, which will include a Unity project containing the scripting language, framework, and some demo scenes (Figure 1, 2, and 3) to demonstrate what the project can do. This will be provided alongside a manual for testing, which details how to install Unity, how to locate different elements of the project and how to access them, links to resources on how to use the Unity editor, and short documentation on what commands are available to use – see Appendix 1.2. Participants are to be given little other instructions or guidance, other than to play around with the project and fill out a questionnaire, provided alongside the link to the project, once satisfied with testing. This testing can occur at home or using a publicly available computer such as those available at the University of Staffordshire if possible. The testing period is expected to last one week, though the testing itself should take an hour or less, including answering the questionnaire. Subjects are expected to test the project whenever they have time during the testing period, to allow for flexibility around other responsibilities.

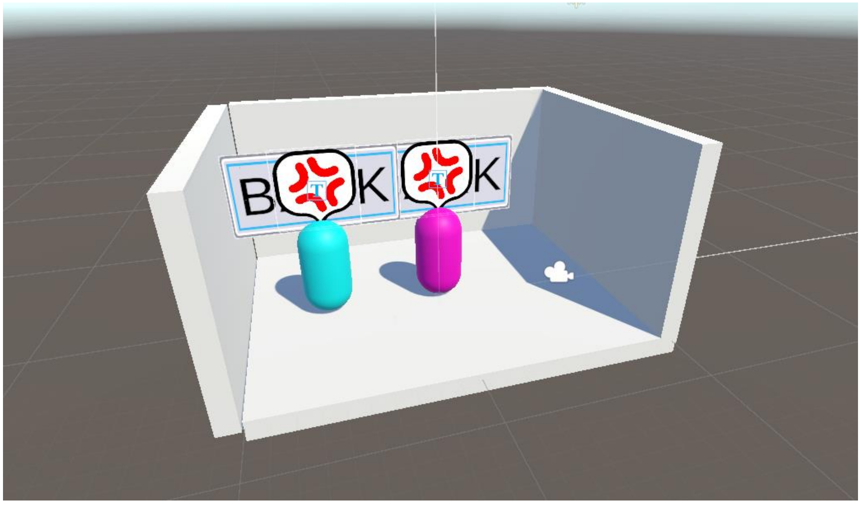


Figure 1 - The Template Scene provided to testers.

This is based on the testing process used by Jon Manning during the development of YarnSpinner (Manning and North, 2021), an open-source narrative scripting language, in which he provided a writer, Ryan North, with an incredibly simple Unity Project and a short script demonstrating what the system could do. North was asked to develop a short demo for YarnSpinner, and to let him know what he came up with, and he developed a prototype with a playtime of multiple hours as a result. While testers will not be expected to produce the same amount of content, and North had some experience with programming and game development as a writer before testing, this approach appears to be an effective way of judging the usability and approachability of a given tool for non-programmers.

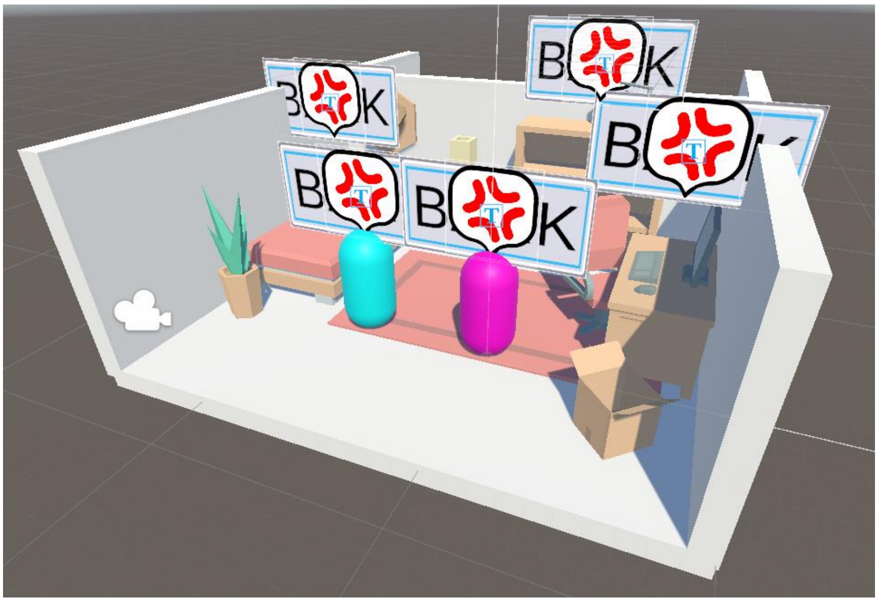
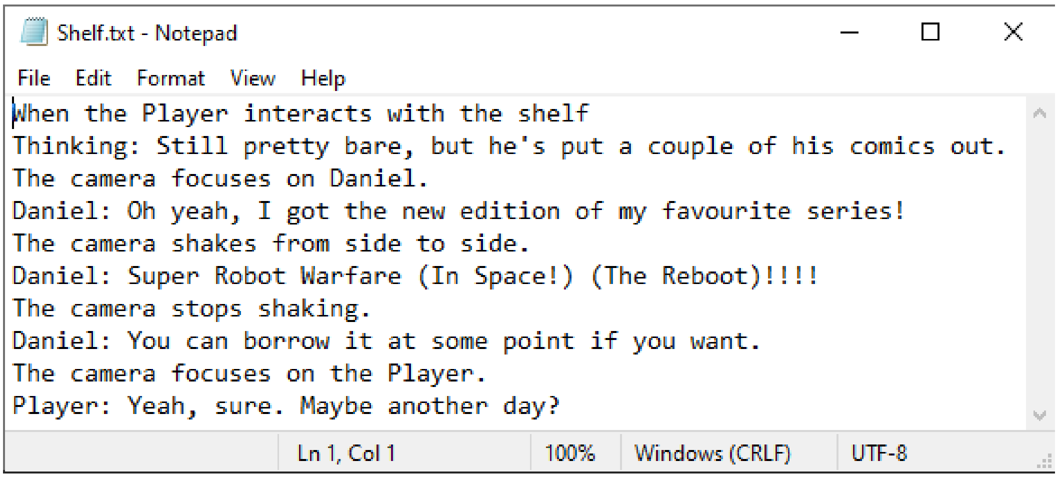


Figure 2 - A sample Scene, Bedroom, provided to testers.

Manning's method was intended for use during a development cycle, in which YarnSpinner would be improved based on feedback from North. However, iteration would require more time than is available for this research. Kauhanen's research is more applicable to the topic, previously performed on existing narrative scripting tools and an investigation of a proprietary system. He interviewed a professional team before and after the implementation of improved dialogue tools in their internal toolchain (Kauhanen, 2009). However, this occurred a year apart, which is not possible either. Alternatively, McLaughlin and Katchabaw tested the usability of their narrative scripting engine by recreating scenes from film, TV, and existing games (Mclaughlin and Katchabaw, 2006; Zhang, McLaughlin and Katchabaw, 2007), which proved the functionality of the system, but did not necessarily prove its usability for non-programmers. Therefore, testing with writers directly using Manning's method would be a more accurate way of proving this hypothesis in the time allotted, despite being unable to iterate on the project in this time.



```
When the Player interacts with the shelf
Thinking: Still pretty bare, but he's put a couple of his comics out.
The camera focuses on Daniel.
Daniel: Oh yeah, I got the new edition of my favourite series!
The camera shakes from side to side.
Daniel: Super Robot Warfare (In Space!) (The Reboot)!!!!
The camera stops shaking.
Daniel: You can borrow it at some point if you want.
The camera focuses on the Player.
Player: Yeah, sure. Maybe another day?
```

Figure 3 - A sample script from Bedroom, demonstrating dialogue and camera functions.

Once subjects have concluded testing, they will be asked to answer a questionnaire, using a mixture of quantitative and qualitative questions – see Appendix 1.3. These questions were screened during the ethics review performed by the University of Staffordshire in advance of the development and testing period.

For quantitative data, the questionnaire asks after existing experience with programming and game development, as well as ask respondents to rank their experience with the tool using the System Usability Scale (Brooke, 1996), which asks a set of questions using a 5-point scale to determine how much a given respondent agrees with a statement, from “strongly disagree” to “strongly agree”. This scale is commonly used to broadly judge the usability of software and calculate an overall usability score, allowing possible avenues of improvement in future development or research to be determined. Comparing usability with a subject’s experience with game development or programming tools should provide a way to put each subject’s feedback into context.

For qualitative data, the questionnaire will ask after the respondent’s opinions of the tool, including what made the tool feel approachable to them, and how they felt it could be improved. Qualitative data is necessary to determine actionable feedback for improvement, where the System Usability Scale doesn’t provide guidance, outside of general categories. The results of these will be sorted and analysed using thematic analysis, as originally proposed by Braun and Clarke (Braun and Clarke, 2006), to allow the data to be analysed in a quantitative way.

In theory, this research methodology should be repeatable for alternative narrative tool implementations, which could determine the usability of narrative tools in comparison with other tools researched using similar methods. However, it is limited by various factors, such as the lack of a single controlled environment for testing, which could allow for uncontrolled variables to impact the testing process. Future research may wish to standardise the testing conditions to reduce this. Additionally, the varied sampling methods could allow for sampling bias, particularly as the chosen sources are poorly defined or too wide, which could be solved by focusing on a more specific sampling frame or a selection of sampling frames to allow for variance without losing control over the variables. Finally, while the estimated number of participants was based on the timeframe and scope available for this research, 10 participants is not enough to allow for variance within the sample while also allowing for statistical significance, which could be improved with a larger research scope and more time allotted for sampling.

Results and Findings

The final sample included 15 participants, each of which filled out the provided questionnaire, which can be viewed in Appendix 1.3. The full set of data can be viewed in Appendix 2.1.

Section 1 - Prior Experience

The first section of the questionnaire evaluates the respondents experience with programming and game engine use.

Programming Experience

How much experience do you have with writing in a programming language on a scale of 1-5, where 1 is no experience, and 5 is professional experience?

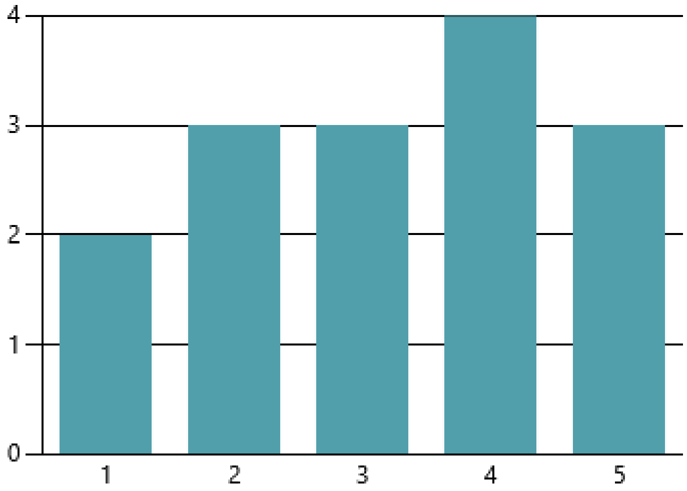


Figure 4 - The programming experience reported by each respondent, on a scale of no experience (1) to professional experience (5).

Question 1 (see Figure 4) focuses on experience with programming languages in general, asking each participant to rank their experience with programming on a scale of 1 to 5, from zero experience to professional experience. The average rating across each respondent was 3.2. Most respondents, four in total, ranked themselves with an experience of 4, while the least respondents, two in total, ranked themselves with an experience of 1.

Game Engine Experience

Did you have any experience with any of the following game engines before taking part in this study?

Question 2 focuses on each respondent's experience with various game engines. The available engines were picked based on popular engines within the games industry for indie development, including Unreal Engine and Unity, as well as Scratch, which is an educational introductory engine which may have been taught to participants during early education. Ren'Py and Twine were included due to their status as engines for developing narrative games, each using a different type of tool. An option for participants to suggest other engines with which they have experience was also included.

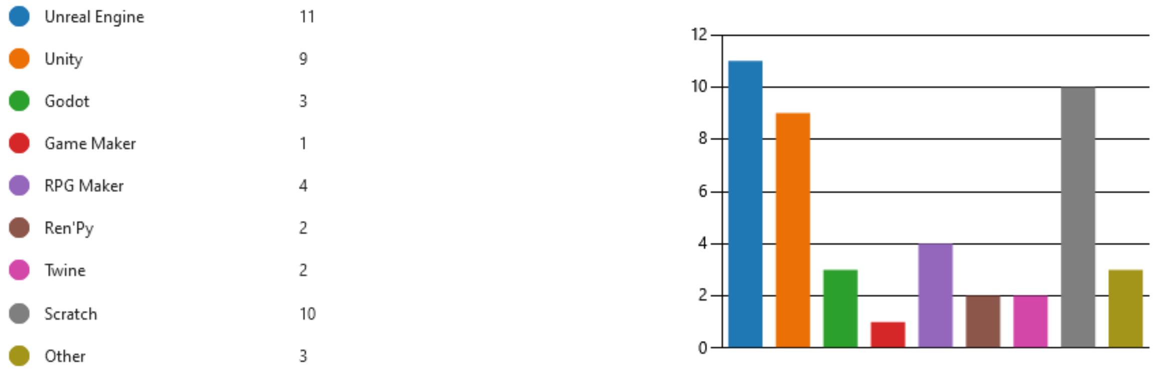


Figure 5 - The game engine experience reported by each respondent.

13 respondents answered this question (see Figure 5), leaving 2 respondents reporting a lack of experience in any game engine. The most popular engine recorded was Unreal Engine with 11 respondents, closely followed by Scratch with 10 respondents, and Unity with 9 respondents. Game Maker only had 1 user from the sample, while Ren'Py and Twine, which are both tools for narrative games, only had two users. Overall, each engine had at minimum 1 respondent with experience in that engine, or 11 at most, with a range of 10 and a mean of 3.46 excluding the two without experience, or 3 including those without experience.

The 3 engines reported under the 'Other' category, includes 'proprietary engines', Roblox Studio and Inkle. While further inferences cannot be made based on the listed proprietary engine, Roblox Studio uses Luau for scripting, which is a text-based multi-paradigm language, and Inkle is an engine agnostic narrative scripting language, as discussed during the literature review.

Section 2 - System Usability Scale

The second section covers questions from the system usability scale, aiming to rank the overall usability of the developed artefact, according to the respondents. Each question within this section asks the respondent to rank how much they agree with each statement, from 1 to 5, where 1 represents strongly disagreeing, while 5 represents strongly agreeing. 3 represents neither agreeing nor disagreeing with the statement.

System Usability Scale

The combined data for the System Usability Scale questions can be seen in Figure 6. Exact distributions and data can be seen in Appendix 2.1.

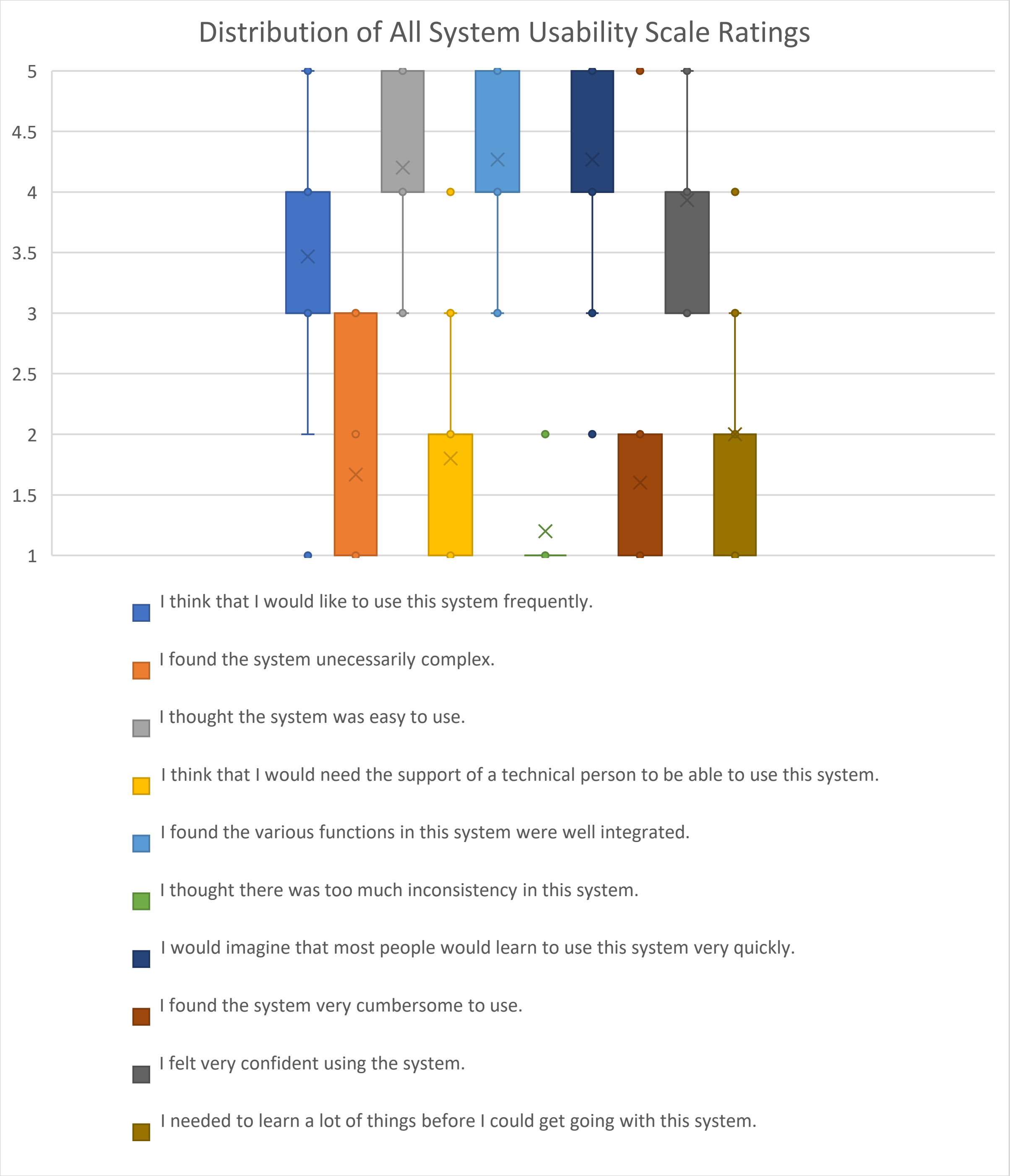


Figure 6 – The distribution of all answers to the SUS questions.

Questions in the SUS are designed to alternate between positive and negative sentiments. The individual data for questions for each sentiment are grouped and displayed in Figure 7 and 8.

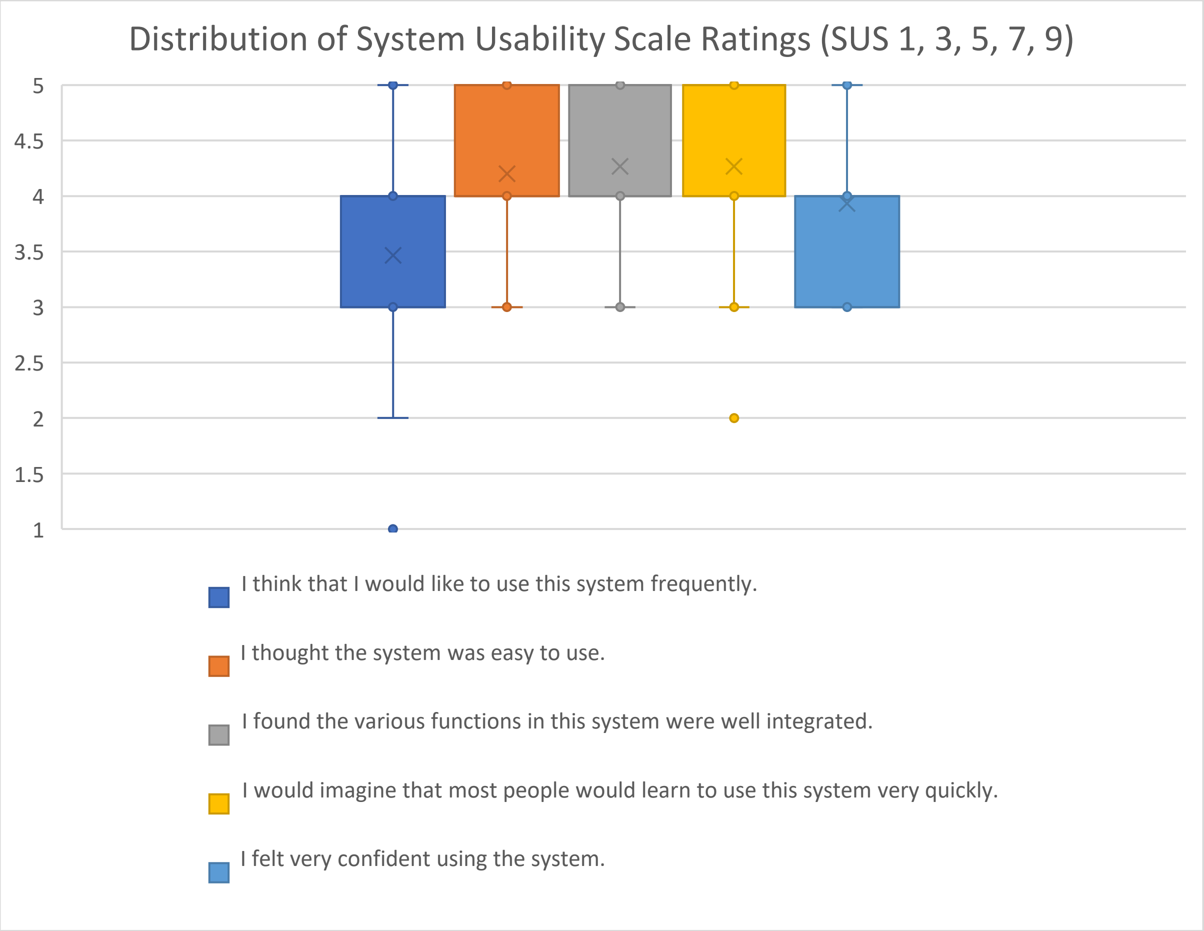


Figure 7 – The distribution of answers to positive tone questions on the SUS.

Ratings for these questions tend positively, with averages at 3.47, 4.2, 4.27, 4.27, and 3.93 respectively. The mode for each is 4, 4, 5, 5, and 4, showing that most respondents agreed or strongly agreed with each statement.

In SUS01, asking whether respondents would use the system frequently, only 2 respondents voted in disagreement, with one vote for 1 and 2.

In SUS07, asking whether respondents thought the system would be quick to learn by most, only one respondent voted in disagreement with a 2.

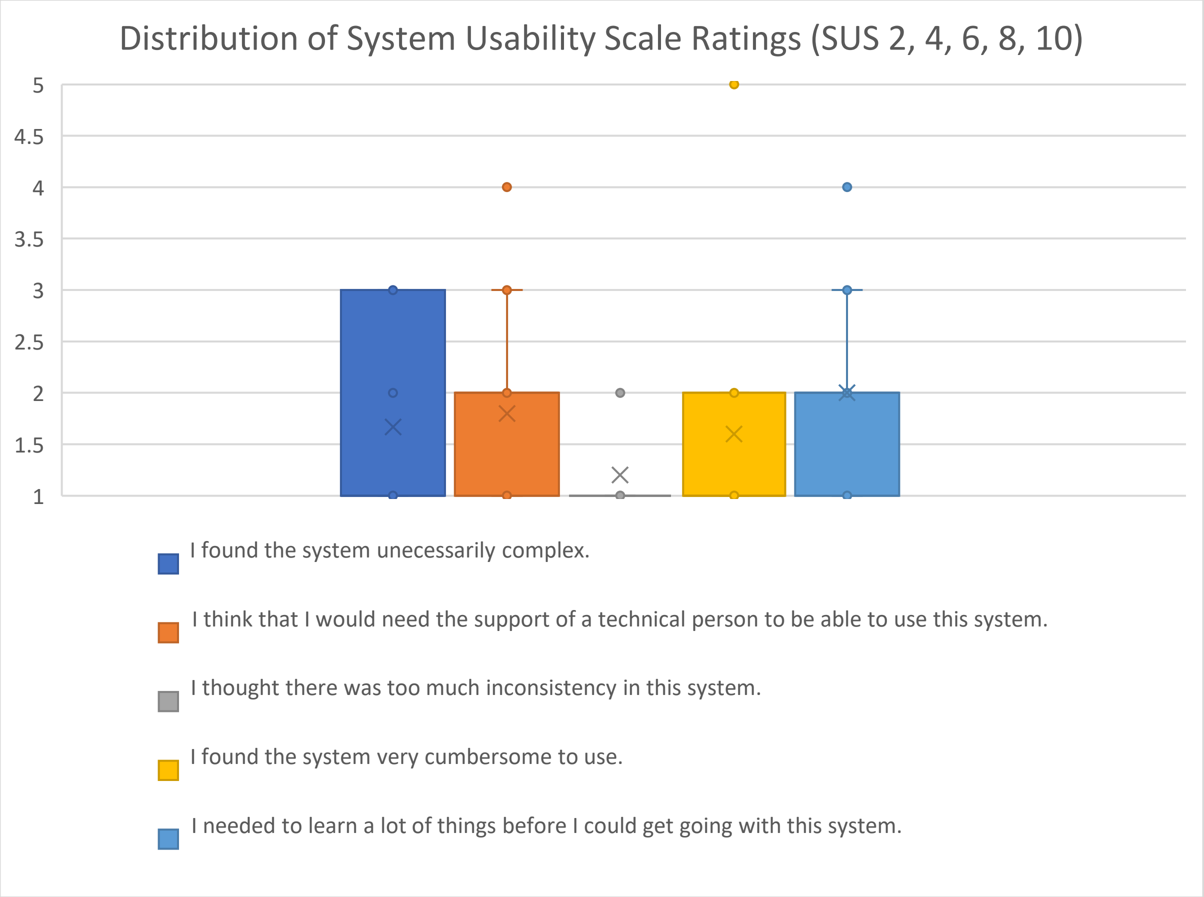


Figure 8 – The distribution of answers to negative tone questions on the SUS.

Ratings for these questions tend negatively, with averages at 1.67, 1.8, 1.2, 1.6, and 2 respectively. The mode for each is 1, 1, 1, 1, and 2, showing that most respondents strongly disagreed with the statements.

In SUS04, 1 participant voted 4, agreeing that they would need the support of a technical person.

In SUS08, 1 participant voted 5, strongly agreeing that they found the system cumbersome to use.

In SUS10, 1 participant voted 4, agreeing that they felt they would need to learn a lot of things before they could get going with the system.

Section 3 – General Feedback

The final section of the questionnaire focuses on qualitative data, relying on user feedback to determine specific advantages, disadvantages, or improvements that could be made to the tool.

Full responses to each question can be viewed in Appendix 2.1.

Overall Improvement Feedback

How could the narrative tool be improved?

Elements of each response were categorised into a set of common themes, using an inductive approach, as shown in Figure 9.

- **Plugin improvements** cover any custom tooling within Unity.

- **Framework improvements** cover any features which rely on an engine for implementation, which could be included within the framework.
- **Documentation improvements** cover any changes or additions to external documentation or demo scenes.
- **Technical improvements** cover engine agnostic features which should not rely on either Unity or the framework.
- **Feature improvements** cover improvements to existing features that were implemented into the framework.
- **Debugging improvements** covers the improvement of debugging tools or error messaging.

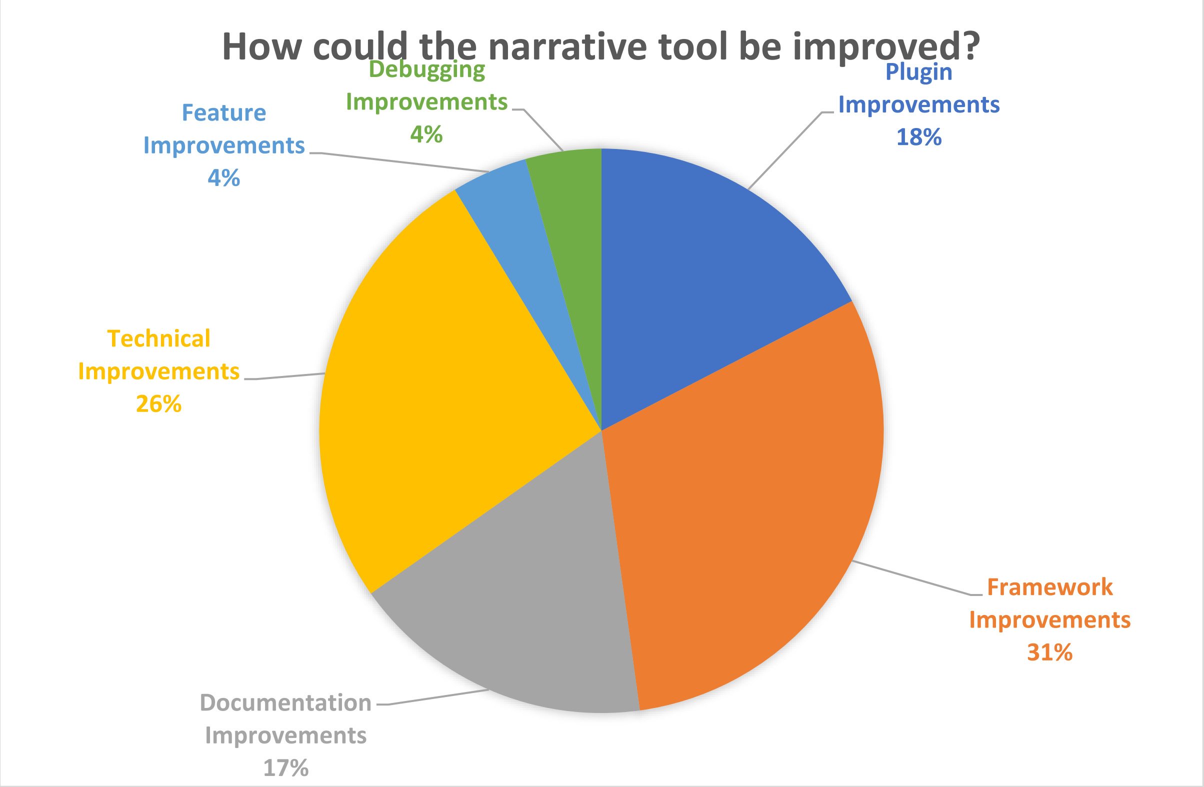


Figure 9 – A chart showing the proportion of responses containing common themes within feedback.

Two responses did not report any improvements. For the remaining responses, 31% of responses covered a desire for additional features, such as the ability to play sound effects or animations. 26% of responses requested technical improvements, such as branching dialogue or narrative variable support. 18% cited improvements to the plugin itself, including porting the plugin to Unreal Engine, editing text files within the editor, and improved tooling for adding new emotions. Documentation improvements was also requested by multiple respondents, taking up 17% of the responses in total, asking after video tutorials, more specific information on how the event system worked, and suggesting that it highlight the fact that sentences are case and punctuation sensitive.

Approachability Feedback

In what ways was the tool approachable or easy to understand?

Elements of each response were categorised into common themes using a deductive approach, based on Kauhanen's design principles for narrative tools, as well as two additional categories for provided documentation and features in the framework. The data is visualised in Figure 10.

- **Supporting non-programmers** covers elements of the tool which reduced the amount of technical knowledge necessary for use.
- **Natural language and terminology** covers the structure of the language or terminology used for features of the tool.
- **Asset reuse** is related to supporting non-programmers, specifically covering the use of existing assets to aid development.
- **Playtesting and debugging** covers functionality used for bug fixing and, error communication, and editing.
- **Features** covers framework functionality which is not engine agnostic.
- **Documentation** covers the external documentation and demo scenes.

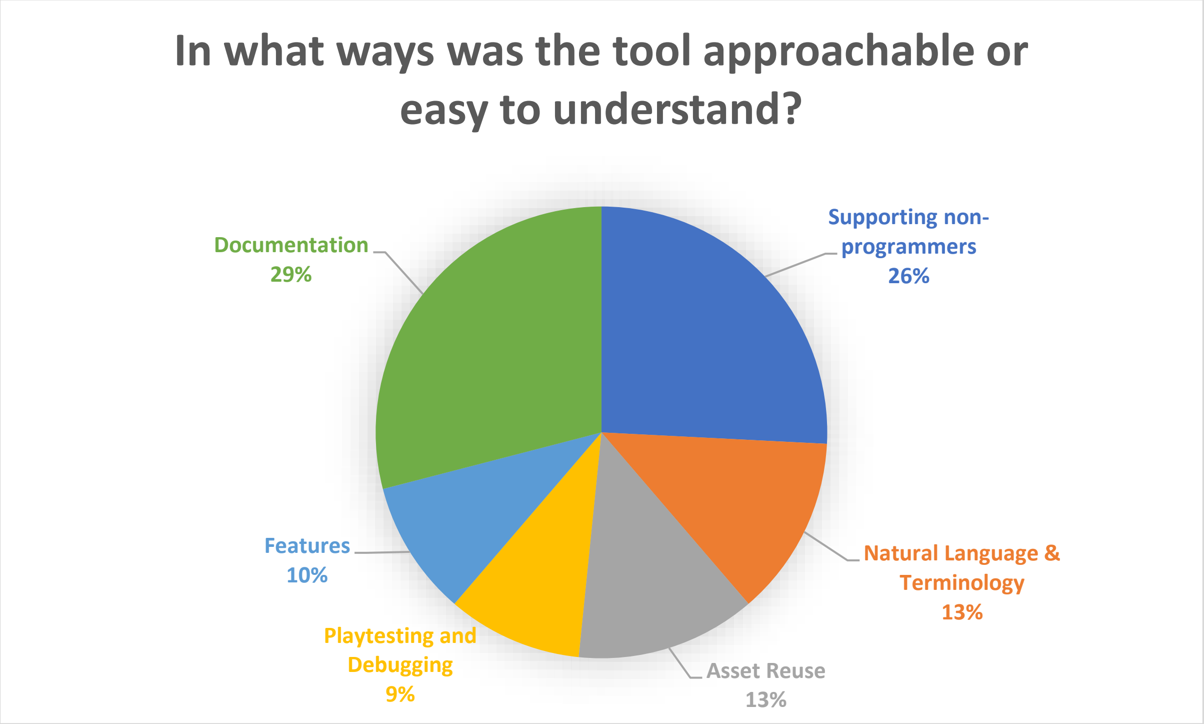


Figure 10 - A chart showing the proportion of responses containing common themes regarding what made the tool approachable.

29% of responses cited the testing manual as a useful reference or mentioned that the demo scenes helped them to understand how the tool worked. 26% of respondents highlighted elements of automation, such as the ability to make any object into an 'interactable' object that can be used within a narrative script using a right-click menu command, or the use of text files as a format.

13% of responses cited the tool's use of natural language as approachable, due to the understandable formatting and ease with which it could be written, similarly to a script. Another 13% cited the reuse of assets, ranging from use of prefabs to the general organisation of the framework reducing the amount of time spent getting lost within navigation.

10% of responses cited specific features within the framework as powerful tools within the simple framework that allow a writer to do a lot with a little, barks and camera options specifically. In terms of playtesting and debugging, 9% of users cited using a mixture of documentation or the existing demo scenes to figure out what went wrong.

Improving Approachability

In what ways could the tool be improved to be more approachable or easier to understand?

Elements of each response were categorised into the same themes as in question 14, following Kauhanen's principles once more. The data is visualised in Figure 11.

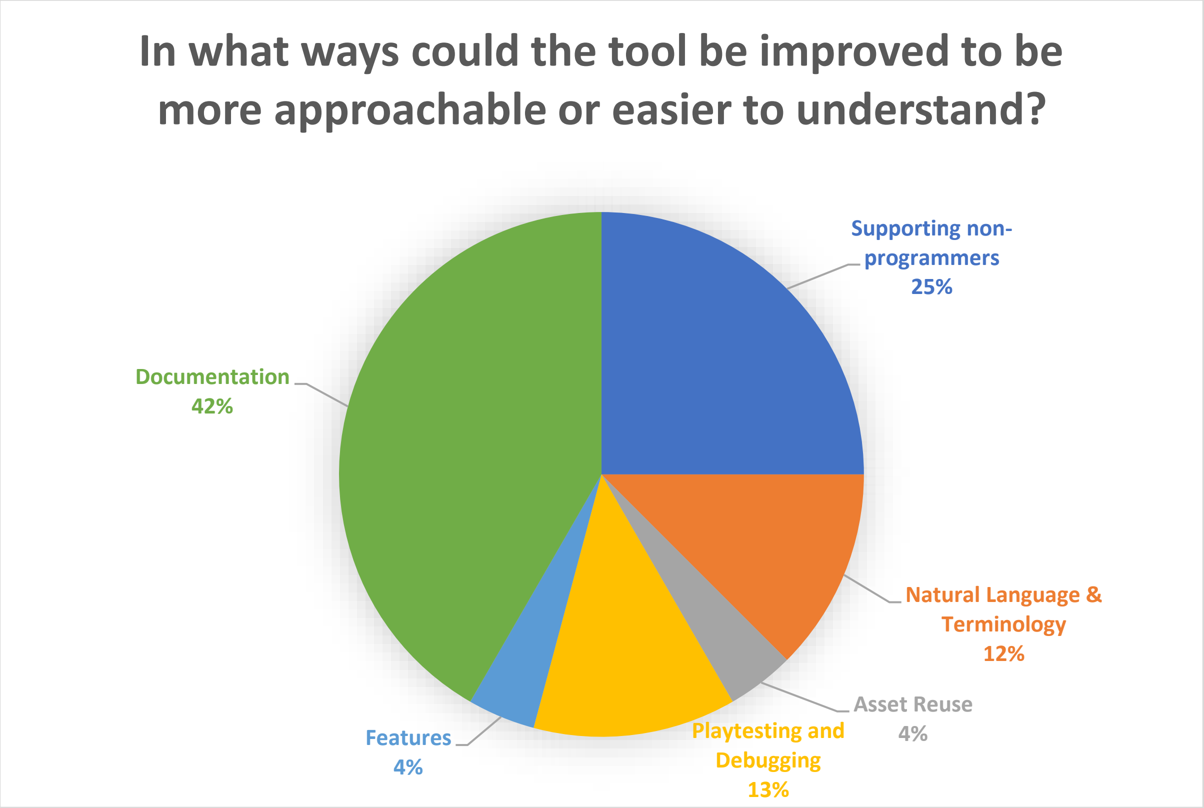


Figure 11 - A chart showing the proportion of responses containing common themes regarding how the tool could be more approachable.

2 responses did not report any improvements. Two other responses copied or referred to answers in question 13, which were then categorised under this question as a result.

42% of responses suggested that improvements to the documentation provided would make the tool more approachable, including in-depth API documentation for developers, an in-depth tutorial scene and documentation within the Unity Editor, better specification regarding grammar and smaller features such as text colour. 25% of responses cited a need for additional support for non-programmers, which includes editing text files within Unity to reduce the amount of time spent swapping between windows, or the use of alternative front-facing software made specifically for editing files for the system.

13% of responses discussed improvement of error logging to aid capture and debugging in the case of incorrect syntax or other errors, and that the technically of existing error logging made debugging harder to approach. Another 12% referred to the natural language element of the tool in some form, citing strict punctuation and grammar requirements, or the possibility of a visual, block-based tool.

Bug and Error Reporting

Did you encounter any bugs or unexpected behaviour while working with the tool? If so, what happened, and how was it caused?

Elements of each response, including responses with no comments, were categorised inductively into a set of categories, shown in Figure 12.

- **None** covers responses that fall under no other categories and did not encounter unexpected behaviour, as perceived by the respondent.
- **In-Game UI** covers issues with the framework's UI implementation within the gameplay.
- **Grammatical** covers issues with user input into the natural language.
- **Empty Lines** covers a specific bug in which empty lines and comments would prevent compilation.
- **Plugin UI** covers issues with the framework's UI implementation within the Unity Engine.
- **Gameplay** covers issues with the framework's gameplay implementation within the Unity Engine.

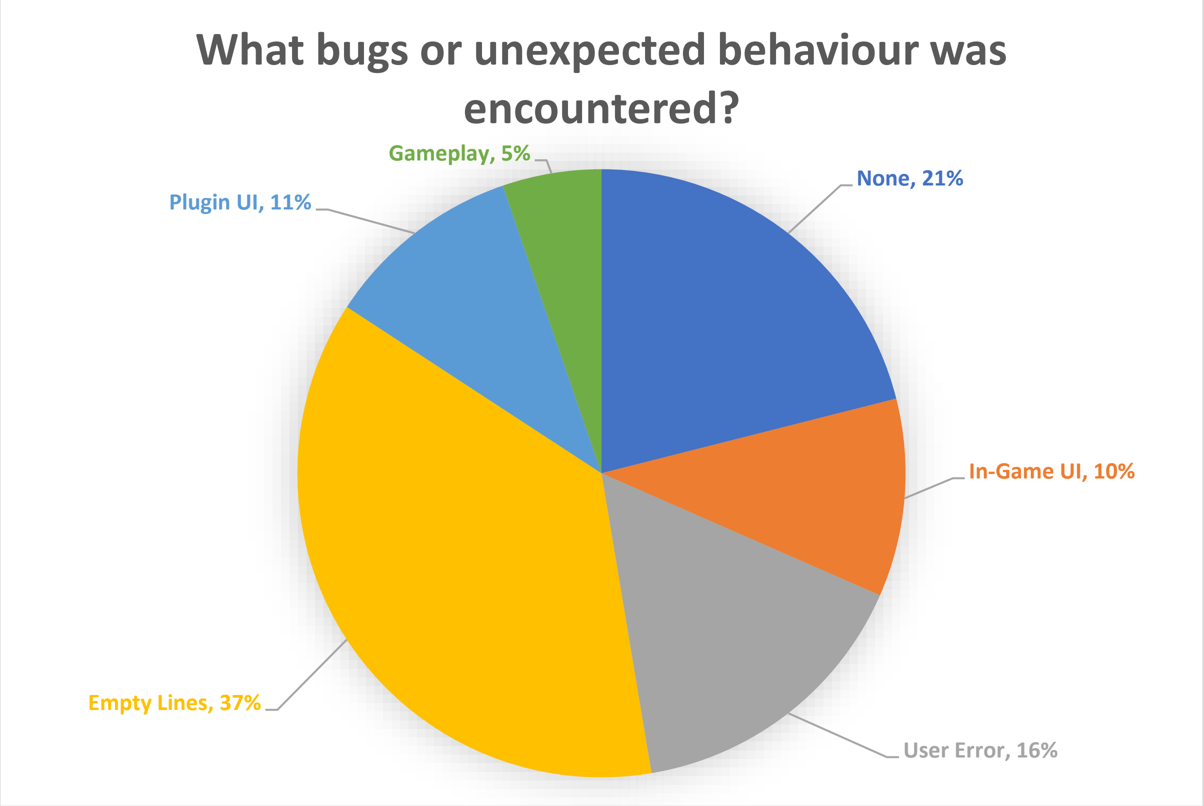


Figure 12 - A chart showing the proportion of responses containing common bugs and unexpected behaviour.

Overall, 4 respondents did not experience any issues that weren't expected by that respondent, making up 21% of content within responses overall.

7 respondents, or around 37% of response contents, mentioned the 'empty lines' bug, an out-of-range exception within the framework that was not caught during the development period that occurred during compilation of narrative script files for any empty or whitespace lines, including comments. Most respondents encountered this issue within one of the demo scenes, but did not report encountering it in their own scenes.

Another 16% mentioned having issues due to user error, missing out punctuation at times and encountering issues as a result. Of the 3 respondents with this issue, does not specify a grammatical issue, but mentions that errors disappeared once they figured out how things worked.

11% of contents, or 2 respondents, mentioned issues with the plugin UI, both encountering different issues. One had issues with making objects interactable using a right-click menu, while the other had issues assigning an action script using the Unity selector.

Less pressingly, 2 respondents mentioned issues with the UI within gameplay, suggesting improvements for features including the barks and emotions, and 1 other respondent mentioned falling off the stage often, which has been attributed to Gameplay.

Discussion and Analysis

The results from the SUS questions can be combined into a set of overall usability scores for the system. This requires use of the following equation, as composed by Lewis in his review of the system's use (Lewis, 2018) between 1996 and 2018.

$$SUS = 2.5\,(\,20 + \sum(SUS01, SUS03, SUS05, SUS07, SUS09) - \sum(SUS02, SUS04, SUS06, SUS08, SUS10))$$

Using the score formula, each respondent's answers were calculated into a single System Usability Scale score between 0 and 100, as shown in Figure 13.

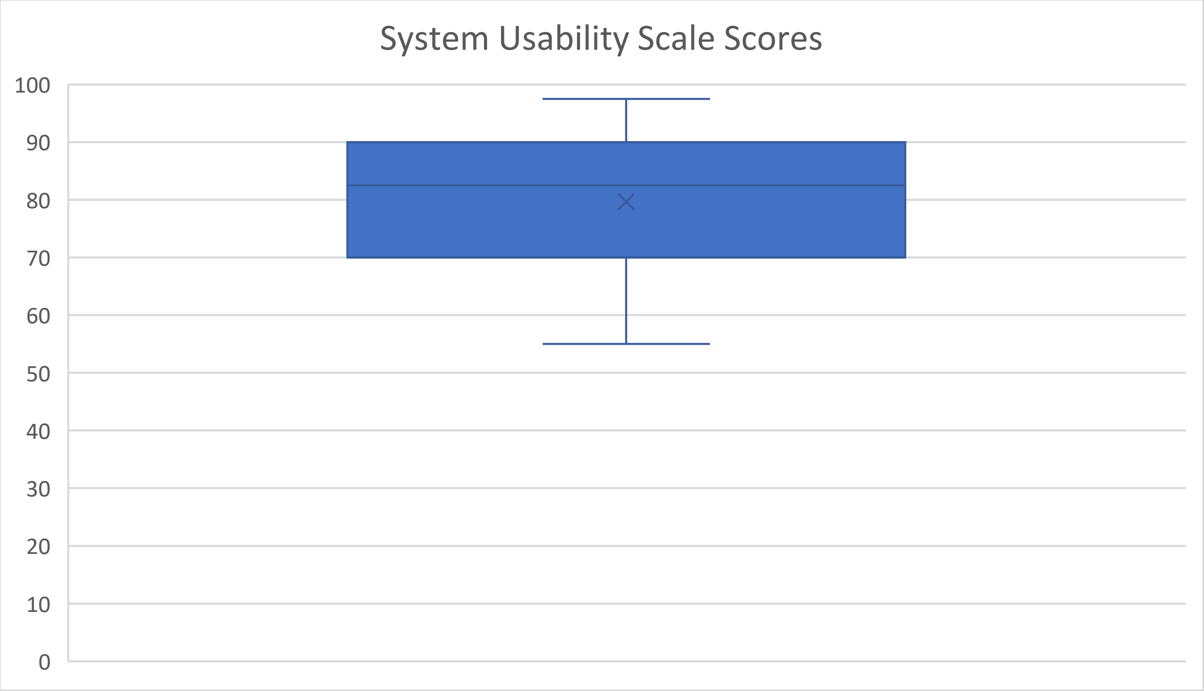


Figure 13 – The spread of SUS scores calculated from the responses of each respondent.

Scores ranged from between 55 to 97.5, with an average score of 79.67. According to Lewis, the average overall score for the SUS is 68, while above average ranks around 80. As such, the data suggests that most participants felt that the system served well in terms of usability for their purposes.

The participant who gave the system an overall ranking of 55 ranked the SUS questions at either 2, disagreeing with multiple statements, or 3, having no opinion either way, ranking themselves as having a lot of experience with programming and game engines. Their answers to the general feedback questions suggest that they struggled with learning how to get started in a scene, mentioning that "it was a little difficult to understand with what needed putting in the text file", and suggesting that the documentation could have been improved by being more specific on how to set up a text file for dialogue.

On the other hand, the participant who gave a ranking of 97.5 had more engine experience, but less programming experience, primarily responding that they strongly agreed or disagreed with all statements, except for agreeing that they would use the system frequently and disagreeing that they needed to learn a lot. They suggested that the tool could be improved with more features such as branching dialogue, mentioning that the tool was “immediately clear and simple to understand, given basic unity experience”, implying that the tool still required a level of technical experience. Their only suggestion for approachability was to add a tutorial within Unity on project startup.

While the extents of the data were both ranked by somewhat experienced participants, the study does appear to have gotten data from a small range of individuals with differing amounts of experience with coding, with an average experience of 3.20 and at least two participants per experience ranking. However, there were few participants without any experience within a game engine, with two respondents having no answer for question 2, and one respondent with only experience in Scratch. Most respondents had some form of experience using Unreal Engine, Scratch, or Unity, which could skew the data somewhat regardless of development discipline.

Participant 6 reported no engine experience, and programming experience at 2, ranking the overall system at 92.5, far above average. They mention finding the functionality of the tool and explanation given in the testing manual to be “clear and concise, and more approachable for it”, suggesting that the testing manual was “invaluable”, and “made working in an engine [they] have no experience with much less intimidating”. Although they experienced a bug with a file selector used by the plugin, they otherwise had no suggestions for improvement.

Participant 11, with no programming experience with anything other than Scratch, gave the system a ranking of 85, also above average, and did not rank any of the SUS questions at a 3, agreeing or disagreeing with each statement to some degree with no notable outliers. They mention that everything was “laid out in a clear and easy to navigate system”, and their only suggestion for improvement regarded the addition of a user guide on startup.

On the other hand, participant 14, while having no experience within game engines, does have professional programming experience, and ranked the system at 62.5. Strongly disagreeing that they would like to use the system frequently, they have few other outlier rankings, with suggestions for improvement surrounding framework features, choices, and quests, which would need implementation in the framework and improved variable support for the system. They state that their “favourite approachability features revolve around anything that keeps the user in the unity application more”, which lines up with their suggestions for improving the tool, which includes a more in-depth tutorial scene.

Notably, this participant suggests the use of a system that does not rely on natural language, as they found it “more confusing during the writing scenes process”, providing an example describing a block-based system like the system described by Birke (Birke, 2015) or the block-based implementation of the Scratch language (Burd *et al.*, 2004). While another participant shared the sentiment of keeping work within Unity, suggesting a text editor for the custom language within the engine, this participant’s answer suggests the need to test other variants of narrative tools in future research for comparison with these results, as well as the possibility that different writers are likely to understand different types of tools better than others.

Overall, when comparing the SUS scores against each participant’s programming experience as in Figure 14, there is near zero correlation between the two values, with a slight negative trend. While this may be due to both values being particularly subjective to an extent, or due to the small sample size, this suggests that programming experience had little impact on the resulting usability of the system.

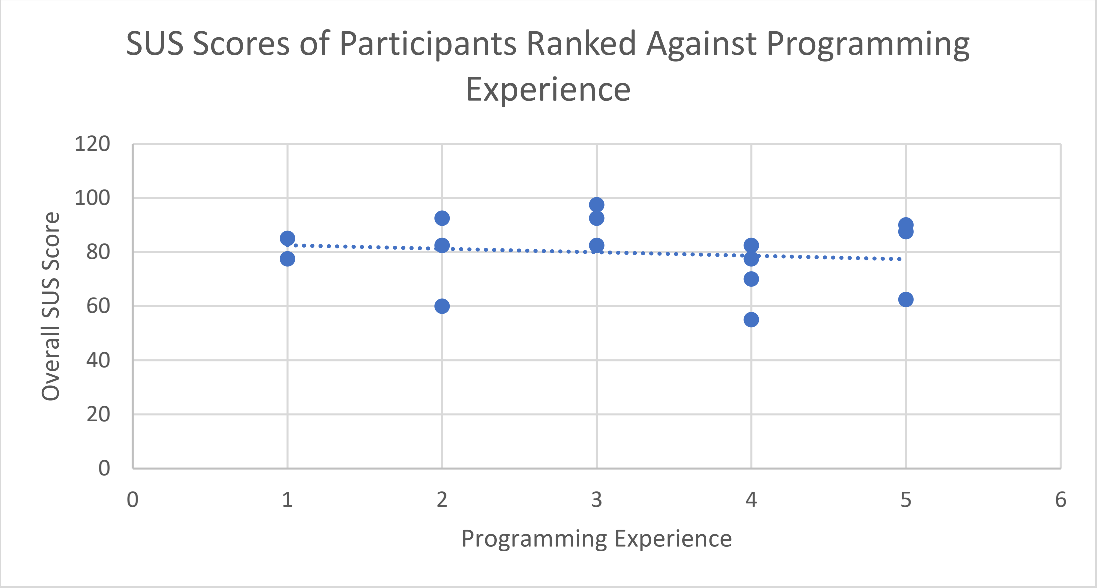


Figure 14 - SUS Scores as ranked against the programming experience of each participant.

Similarly, when comparing SUS scores to the number of engines a participant has used as in Figure 15, there is near zero correlation between these two values, with a slight positive trend. This could provide further evidence to show that the development experience of participants had little impact on the usability of the narrative system.

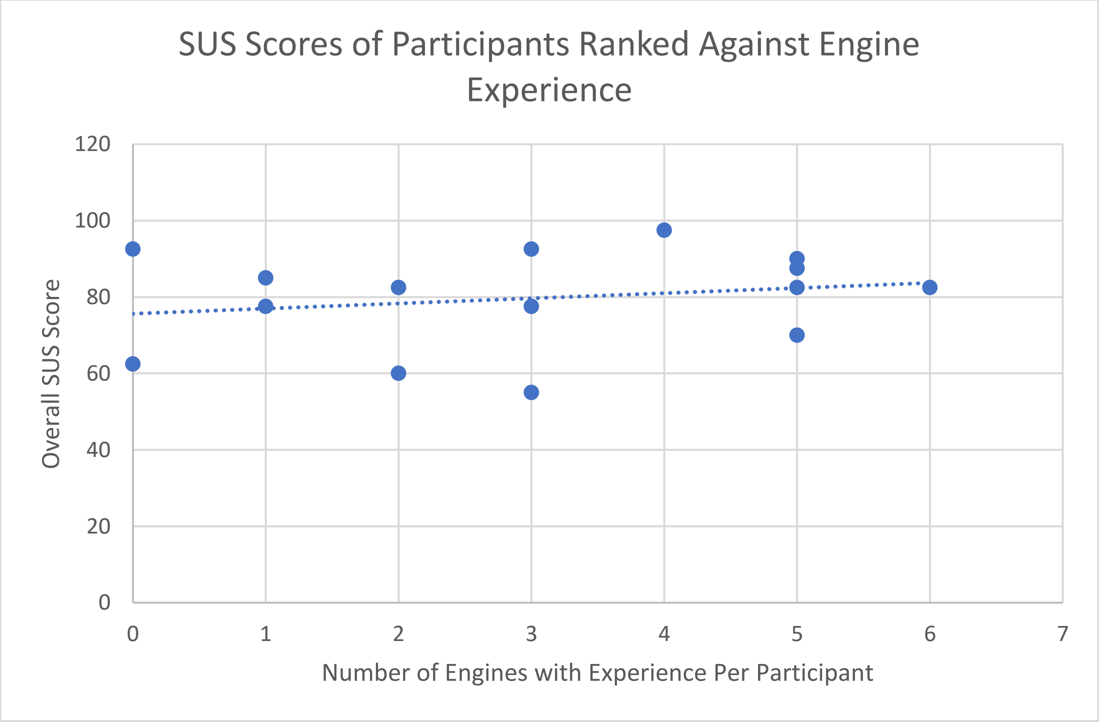


Figure 15 - SUS Scores as ranked against the game engine experience of each participant.

While this could support Kauhanen's proposed design principles (Kauhanen, 2009), as participants without experience within game engines did just as well with the tool as others, it could be argued that this data fails to prove that the success of the tool was related to the implementation of those principles within the artefact.

Regarding how the system could be improved, a significant number of participants suggested improvements to the framework and general features, many suggesting new emotions, or expansions to the existing text or camera systems. While these aren't the focus of this study, providing writers with more tools to create scenes are likely to make them find the tool easier to work with as more options are available to them, allowing them to create a more satisfactory scene and more complicated narratives.

On the other hand, technical improvements also made up a significant portion of suggestions, with participants 10, 12 and 14 all suggesting the addition of branching dialogue, a feature which was planned but could not be implemented due to scope and time. Similarly, improving these could also add to the usability of the final system simply by providing a larger narrative toolset, and would allow for greater player agency as a result.

Participants generally felt that the documentation made the tool particularly approachable, with many citing the demo scenes as a useful resource, such as participant 4, who suggested that "the example scene was more than enough to figure out how to construct a scene". Participant 6 suggested that the manual made working within Unity without experience "much less intimidating". However, most suggestions to improve approachability also covered documentation, with many respondents suggesting that a guide on startup within Unity would reduce the amount of time spent swapping between windows. Another common criticism was a request for better communication of the commands available within the manual, and more detail regarding the setup of a new scene.

While documentation was not one of Kauhanen's specific design principles, the emphasis on documentation from the respondents seems significant. It could be argued that this falls under 'supporting non-programmers', as it provides them with a frame of reference without the need for a technical person to be present. However, this could impact the results of the SUS, as more documentation may require users to learn more before they can get going with the system. It may be worth compromising here, as better startup information would make the tool more approachable as a result and may instead improve scores for this question.

Of Kauhanen's principles, 'supporting non-programmers' was the most common theme regarding usability. 3 participants mentioned that the use of text files made the system more approachable, as did handling various features under the hood. However, supporting non-programmers also featured heavily regarding improvements, with 3 participants suggesting that there could be more front-facing tooling, plugins for text editors such as notepad++, or a separate, custom program for editing scripts, with participant 15 claiming that people with little programming experience "are much happier to use a program with a UI than a tool".

There was also a lot of praise for the natural language element, with participant 2 mentioning that it was "practically the same as just reading a script", and participant 4 noting that the natural typing allowed them to preplan beforehand and implement their idea directly. However, 2 people brought up the rigidity regarding punctuation, new lines, and case sensitivity as something to improve approachability, as this was prone to cause errors, and debugging information in the project was not sufficient to explain where the problem might lie. Participant 14 mentions that the natural language was more confusing during the writing process, despite reading nicer., and made it "harder […] to jump into the middle of a file", suggesting the block-based editor as a result. This supports a need to re-evaluate the core design of the tool, regarding how sentences should be structured, the differentiation of key words and identifiers for readability, and grammar flexibility. Alternatively, better tooling could reduce the impact of these issues by providing a spell-check or autocomplete for different sentences.

Some of Kauhanen's design principles were neglected in the responses. Four positive responses mention aspects of asset reuse inbuilt within Unity, referring to use of prefabs and the ease with which users can create new interactable objects. Only one participant mentioned this negatively, in terms of a desire to add more to a prefab, such as an additional camera, which ties into the limited functionality of the framework itself. Playtesting and debugging also received little feedback, with 3 participants mentioning it positively, suggesting that the documentation or existing demo scenes helped with debugging. However, there were also multiple mentions of ways in which debugging could be improved within the tool, suggesting more user friendly and helpful error logging, as well as support within text editors to prevent errors before they occur.

Overall, the research supported Kauhanen's design principles, as participants found the implemented elements of these principles to be useful for approachability while using the tool. These have also highlighted some key avenues for improvement with future development which could improve approachability for non-programmers, particularly better debugging and a nicer front-end. However, it also shows that there is room for improvement where different approaches to a narrative tool may benefit some users over others.

However, this research methodology has key room for improvement. As mentioned, the sample included 15 subjects, which is not enough for statistical significance. In addition, due to the sampling method and timescale, most participants were friends of the researcher, or friends of friends due to the snowball sampling method, which could have introduced a significant amount of bias towards the research. Future reproductions of this method should look to get a larger sample size of unrelated participants to reduce the amount of bias within the results, as well as getting a larger range of participants with differing backgrounds, especially more writers without game development experience. There is also a gap in the research area, meaning there is little to compare the results to, so further research into other possible tools or improvements to this style of tool would greatly benefit the field.

Conclusion

At the start of this paper, various objectives were set out to reach the aim of proposing, developing, and learning from a writer friendly narrative tool. Each objective must be discussed to determine if this aim was achieved, and to determine how to move forward with this research.

What constitutes narrative in games continues to be the subject of debate amongst narratologists and ludologists alike, due to conflicting views on whether narrative should be interacted with, as in games. There has been research into the development of true interactive narratives, from cybertext and text adventures as discussed by Aarseth (Malloy and Aarseth, 1998), to interactive dramas like Façade (Mateas and Stern, 2003), which would allow players to interact with the world as they would in real life and have an impact in the narrative based on their choices, as discussed in Juul's thesis (Juul, 1999) and in Hamlet on the Holodeck (Murray, 2017).

However, this is not as common within games at large, which make use of a mixture of emergent and embedded narrative techniques, according to Rules of Play (Tekinbas and Zimmerman, 2003), which communicate narrative through the game world and the ergodic sequence of events that the player experiences within gameplay, alongside static methods of narrative delivery such as textual artefacts, audio, and cutscenes which take techniques from traditional forms of narrative.

Within the industry, various companies, including Obsidian Entertainment (Patel and Szymczyk, 2019), Valve (Ruskin, 2012), and Naughty Dog (Gregory, 2014), report different toolsets for use with narrative development that allows writers to directly interface with the narrative of different games. While spreadsheets are commonly used in some form (Bateman, 2021), particularly for organisation purposes (Armstrong and Ewing, 2017), toolsets can range from natural language or a code-like structure, such as YarnSpinner (Manning and North, 2021) or Prompter (Nelson, 2014), to visual tools such as block-based or node-based scripting, such as Scratch (Burd *et al.*, 2004) or Twine (Interactive Fiction Technology Foundation, no date).While cutscenes are the most used form of narrative delivery within any one game (Ip, 2011), various dialogue engines still find use to allow a degree of player agency within the story by dictating how players can communicate with NPCs and how these affect the game itself (Bateman, 2021), most commonly event-based and choice-based dialogue engines.

There appears to be a gap within the field regarding guidance towards the development of narrative tools for writers that has been backed up with evidence, however the design patterns proposed by Kauhanen (Kauhanen, 2009) are a useful resource for developing potential narrative tools. The artefact used as the subject of this paper's research used these design patterns to develop a plugin and framework for the Unity Engine, using a natural language scripting approach.

From testing this artefact with a small sample of 15 participants, it was determined that the system has a score on the System Usability Scale of 79.67 (Brooke, 1996), based on the answers to the questionnaire. This is around above average for systems scored using the SUS (Lewis, 2018), at a score of 80. However, due to the lack of statistical significance, and the possibility of biases due to the sampling method and lack of controlled environment, it would be worth investigating this further for more accurate data.

The narrative language system could be improved by developing a way to reduce the need for technical support while improving the ease at which the system can be learned and used. From qualitative feedback, this could require the development of an editor or text software plugin for the narrative scripting language, which could be used to correct spelling issues and use formatting to help writers identify different sections of the script, without changing the structure of the language.

Notably, the documentation was heavily focused on by subjects, as many within the sample referenced the documentation as points of both approachability and improvement. This suggests that proper documentation of the system and what is possible for writers to use improves the overall user experience, and as such should not be neglected during the development of narrative tools.

Additionally, many were happy with the natural language scripting, suggesting that use of text files was “seamless and easy to understand” and that it was “practically the same as just reading a script”. This also made the system easier to pick up, using the information provided within existing scripts as a frame of reference. This also adds to the argument that pre-existing, detailed but simple materials, examples and documentation are effective for onboarding writers for a narrative tool.

However, this came with its own issues, as some testers had issues with the strict spelling and punctuation requirements for commands, with one noting that they struggled to jump into the middle of a script since “the break points are less clear”. While this solution could theoretically be translated, this does also suggest the possibility that this approach may be significantly less usable for writers with less proficiency in English. Additional research determining the effectiveness of a natural language approach in other languages could also be worth investigating going forward, as this could reduce the impact of a language barrier to narrative game development for non-native English speakers.

One participant’s feedback suggests that alternative tooling may work better for some writers compared to others. They proposed a block-based system within the Unity editor, which suggests that research into other tools would be beneficial to the field and could be worth comparing to determine which is best. However, it could also suggest that which narrative tool is the most usable for writers is subjective. Therefore, it could be more beneficial for research purposes to determine which features of these tools are usable and why, and how to apply these principles to other tools to get the best experience possible. It could also point towards a need to have multiple methods to visualise narrative during development.

Overall, the developed system received a lot of positive feedback from a sample with a range of technical backgrounds, with strong feedback and usability scores from writers without experience within a game engine. While the tool and the framework are not currently complex enough for use within a development environment, it is worth developing further to include more of the requested features and make changes to some of the presentation to align more with Kauhanen’s design principles. It would also be worth testing in a development environment at this stage to determine if the tool is viable for narrative development at a greater scale than was asked of testers for research.

Responses to the research indicate that concise, accessible documentation and examples are key to improving approachability for a tool for writers without experience in game development, which also reduces dependency on other developers. In addition, the natural language element was generally considered easy to use and understand, though could be improved with better debugging tools or guards against user mistakes such as spell-checking. However, additional research is necessary to determine how a natural language approach compares in usability to other narrative tooling for use in game development.

Recommendations

Future research in this area should attempt to reduce or remove biases in the data, particularly those caused by relation to the researcher, by testing with a larger range of participants with a wider variance in backgrounds. A study which achieves statistical significance would fill a large gap within this area of research and may provide more accurate information regarding the usability of narrative scripting tools by non-programmers and writers.

The methodology could be improved by introducing more methods through which to control external variables, such as the testing conditions, which were not consistent due to participants being sourced in a remote manner and using their own hardware to run both Unity and the artefact. Limiting access to unrelated materials during the testing period and removing the step to install Unity and the package itself could reduce the amount of unnecessary feedback regarding engine installation.

In addition, the data gathered over the course of this study would be best compared against similar studies with alternative implementations of narrative tools, as well as existing tooling and systems. Doing so would allow these studies and tools to be compared against each other, allowing future developments within this field to be compared against a base scale using existing data. It could be beneficial for a project with a larger scope to build and test multiple different types of narrative tools using different narrative engines and commercial editors to refine this data with a range of consistent options.

Regarding the artefact developed for this research, much of the feedback regarded bug reports and support for additional functionality, including the addition of branching paths and variables. It would be beneficial to develop this artefact further to include some of these requests, such that more complex stories can be created without the need to interface directly with the code of a given engine. In addition, fixing internal bugs and reducing the severity of different user-side issues, such as being more lenient with spelling and grammar, would be beneficial to usability overall as writers can feel more empowered while using the tool, and this would reduce confusion when something goes wrong. Additional debugging information which is more useful for non-technical writers would also help in this regard.

While additional feature support may not add to the usability of the tool on its own, including more detail within the documentation in a concise and writer-friendly manner and improving the UI of the tool further to be more approachable could also improve a writer’s experience with the tool, and therefore improve the usability of this approach. In addition, it is possible that integration of the artefact into the Unity engine affected the tool’s usability overall. It could be worth either creating a custom engine or editor for writers or creating versions of the tool which are compatible with other commercial engines, such as Unreal Engine. Doing so may determine if the engine itself has an impact on the usability of narrative tools.

References

Aarseth, E. (2012) “A narrative theory of games,” in *Foundations of Digital Games 2012, FDG 2012 - Conference Program*. Available at: https://doi.org/10.1145/2282338.2282365.

Armstrong, W. and Ewing, P. (2017) *Do You Copy? Dialog System and Tools in “Firewatch.”* Available at: https://gdcvault.com/play/1024000/Do-You-Copy-Dialog-System (Accessed: November 2, 2024).

Bateman, C. (2021) *Game Writing: Narrative Skills for Videogames*. Second edition. New York: Bloomsbury Academic.

Birke, A. (2015) *Automate, Streamline, Win! Content Creation for Small Teams*. Available at: https://gdcvault.com/play/1022777/Automate-Streamline-Win-Content-Creation (Accessed: November 3, 2024).

Braun, V. and Clarke, V. (2006) “Using thematic analysis in psychology,” *Qualitative Research in Psychology*, 3(2). Available at: https://doi.org/10.1191/1478088706qp063oa.

Brooke, J. (1996) “SUS -A quick and dirty usability scale Usability and context,” *Usability evaluation in industry*, 189(194).

Burd, L. *et al.* (2004) “Scratch: a sneak preview [education],” *Proceedings of Second International Conference on Creating, Connecting and Collaborating through Computing* [Preprint].

Campbell, J. (2008) *The hero with a thousand faces, 3rd ed.*, *The hero with a thousand faces, 3rd ed.*

Cheng, P. (2007) “Waiting for something to happen: Narratives, interactivity and agency and the video game cut-scene,” in *3rd Digital Games Research Association International Conference: “Situated Play”, DiGRA 2007*.

Clarke, A. and Zioga, P. (2022) “Scriptwriting for Interactive Crime Films,” *Interactive Film and Media Journal*, 2(1). Available at: https://doi.org/10.32920/ifmj.v2i1.1524.

Despain, W. (2020) *Professional Techniques for Video Game Writing*, *Professional Techniques for Video Game Writing*. Available at: https://doi.org/10.1201/9780429196539.

Domsch, S. (2013) *Storyplaying: Agency and narrative in video games*, *Storyplaying: Agency and Narrative in Video Games*. Available at: https://doi.org/10.1515/9783110272451.

Eladhari, M. (2002) *Object oriented story construction in story driven computer games*. Translated by S. Olsson. Masters. Stockholm University.

Al Enezi, W. and Verbrugge, C. (2023) “Investigating the Influence of Behaviors and Dialogs on Player Enjoyment in Stealth Games,” in *Proceedings - AAAI Artificial Intelligence and Interactive Digital Entertainment Conference, AIIDE*. Available at: https://doi.org/10.1609/aiide.v19i1.27512.

Green, D. *et al.* (2018) “Novella: A Proposition for Game-Based Storytelling,” in *Proceedings of the 7th International Workshop of Narrative and Hypertext hosted at ACM*. Baltimore. Available at: https://doi.org/10.475/123_4.

Gregory, J. (2014) *A Context-Aware Character Dialog System*. Available at: https://gdcvault.com/play/1020386/A-Context-Aware-Character-Dialog (Accessed: November 2, 2024).

Humfrey, J. (2016) *Ink: The Narrative Scripting Language Behind “80 Days” and “Sorcery!”* Available at: https://gdcvault.com/play/1023221/Ink-The-Narrative-Scripting-Language (Accessed: November 3, 2024).

Humfrey, J. (2017) *Creating Interactive Film Scripts for 3D Adventures with Ink*. Available at: https://gdcvault.com/play/1023990/Creating-Interactive-Film-Scripts-for (Accessed: November 3, 2024).

Ince, S. (2009) *Writing for Video Games*. 1st edn. A & C BLACK.

Interactive Fiction Technology Foundation (no date) *Twine / An open-source tool for telling interactive, non-linear stories*. Available at: https://twinery.org/ (Accessed: February 24, 2025).

Ip, B. (2011) “Narrative structures in computer and video games: Part 1: Context, definitions, and initial findings,” *Games and Culture*, 6(2). Available at: https://doi.org/10.1177/1555412010364982.

Jenkins, H. (2003) “Game Design as Narrative Architecture,” *Response*, 44(3).

Jones, D. (2008) “Narrative reformulated: Storytelling in videogames,” *CEA Critic*, 70(3).

Juul, J. (1999) *A Clash between game and narrative: A thesis on computer games and interactive fiction*, *Institute of Nordic Language and Literature*.

Juul, J. (2001) “Games telling stories,” *Game Studies*, 1(1). Available at: https://www.gamestudies.org/0101/juul-gts/ (Accessed: February 24, 2025).

Kauhanen, M. (2009) *Examining Support of Narrative Scripting for Serious Games*. Carleton University.

Kipnis, A. (2014) *Dialog Systems in Double Fine Games*. Available at: https://gdcvault.com/play/1020864/Dialog-Systems-in-Double-Fine (Accessed: November 3, 2024).

LeBlanc, M. (2000) “Formal Design Tools: Emergent Complexity, Emergent Narrative,” in *Game Developers Conference*.

Lewis, J.R. (2018) “The System Usability Scale: Past, Present, and Future,” *International Journal of Human-Computer Interaction*, 34(7). Available at: https://doi.org/10.1080/10447318.2018.1455307.

Lindley, C.A. (2005) “Story and Narrative Structures in Computer Games,” in *Developing Interactive Narrative Content*.

Mackay, D. (2017) *The Fantasy Role-Playing Game: A New Performing Art*. McFarland, Incorporated, Publishers. Available at: https://books.google.co.uk/books?id=s8YRVbDknyUC.

Malloy, J. and Aarseth, E.J. (1998) “Cybertext, Perspectives on Ergodic Literature,” *Leonardo Music Journal*, 8. Available at: https://doi.org/10.2307/1513408.

Manning, J. and North, R. (2021) *Game Narrative Summit: Turn your Writers Into Programmers: Greyboxing Narrative with Story Languages*. Available at: https://gdcvault.com/play/1027215/Game-Narrative-Summit-Turn-Your (Accessed: November 2, 2024).

Mateas, M. and Stern, A. (2003) *Façade: An Experiment in Building a Fully-Realized Interactive Drama*. Available at: www.interactivestory.net.

Mclaughlin, M. and Katchabaw, M. (2006) “A Reusable Scripting Engine for Automating Cinematics and Cut-Scenes in Video Games,” *Game Studies* [Preprint].

Murray, J.H. (2017) “Hamlet on the Holodeck, Updated Edition,” *Free Press* [Preprint].

Nelles, W. (2020) *Frameworks: Narrative Levels and Embedded Narrative*. reprint. Wipf and Stock Publishers.

Nelson, G. (2014) *Prompter: A Domain-Specific Language for Versu*.

Patel, C. and Szymczyk, D. (2019) *Technical Tools for Authoring Branching Dialogue*. Available at: https://gdcvault.com/play/1025962/Technical-Tools-for-Authoring-Branching (Accessed: November 2, 2024).

Rush, J., Dancyger, K. and Keyt, J. (2023) *Alternative Scriptwriting*, *Alternative Scriptwriting*. Available at: https://doi.org/10.4324/9781003242307.

Ruskin, E. (2012) *AI-driven Dynamic Dialog through Fuzzy Pattern Matching. Empower Your Writers!* Available at: https://gdcvault.com/play/1015528/AI-driven-Dynamic-Dialog-through (Accessed: November 2, 2024).

Ryan, M.-L. (2006) *Avatars of Story*. University of Minnesota Press.

Ryan, M.-L. (2009) “From Narrative Games to Playable Stories: Toward a Poetics of Interactive Narrative,” *StoryWorlds: A Journal of Narrative Studies*, 1(1). Available at: https://doi.org/10.1353/stw.0.0003.

Simons, J. (2007) “Game Studies - Narrative, Games, and Theory,” *Game Studies*, 7(1).

Tekinbas, K.S. and Zimmerman, E. (2003) *Rules of Play: Game Design Fundamentals (The MIT Press)*, *MIT PRESS*.

Toh, W. (2023) “The Player Experience and Design Implications of Narrative Games,” *International Journal of Human-Computer Interaction*, 39(13). Available at: https://doi.org/10.1080/10447318.2022.2085404.

Tomsinski, P. (2016) *Behind the Scenes of Cinematic Dialogues in “The Witcher 3: Wild Hunt.”* Available at: https://gdcvault.com/play/1023285/Behind-the-Scenes-of-Cinematic (Accessed: November 3, 2024).

Vogler, C. (1985) “The Writer’s Journey,” *Physica Status Solidi (a)*, 89(1).

Wei, H. (2010) “Embedded narrative in game design,” in *Future Play 2010: Research, Play, Share - International Academic Conference on the Future of Game Design and Technology*. Available at: https://doi.org/10.1145/1920778.1920818.

Weizenbaum, J. (1966) “Eliza.”

Zhang, W., McLaughlin, M. and Katchabaw, M. (2007) “Story scripting for automating cinematics and cut-scenes in video games,” in *Proceedings of the 2007 Conference on Future Play, Future Play ’07*. Available at: https://doi.org/10.1145/1328202.1328229.

Appendices

Appendix 1 – Methodology Materials

1.1 – Social Media Pitch (X Thread)

Are you interested in narrative games, or writing dialogue and/or cutscenes for games?
Would you like to volunteer to help out with my dissertation research?

If so, please reach out to me via Twitter or LinkedIn for more details on signing up!

🧵For more info...

I'm working on a human readable scripting language for Unity 6, looking for people with interest and time to test the system and answer a form on usability between the 20th and 26th of January. Game dev experience isn't needed, as long as you can access Unity 6 and a text editor.

Testing should take about an hour at most, followed by a short questionnaire, asking after your background with programming (if any) and usability feedback for the system.

All data collected is covered by the GDPR - full info will be sent to anyone interested before sign-up.

1.2 – Manu-Scriptwriter Testing Manual

Thank you for volunteering to take part. This document should cover any setup necessary for testing, as well as the available functionality.

Setup

If you have any issues during setup, or with using the Unity Engine itself, feel free to request help.

What do you need?

- A copy of Unity 6, version 6000.0.19f1 or later.
- A text editor, such as Notepad, TextEdit, Notepad++, or otherwise. Most computers come with a text editor pre-installed.
- A copy of the project itself, which should have been provided alongside this manual.

Installing Unity 6

If you already have a copy of Unity 6 installed on your device, you can skip this heading.

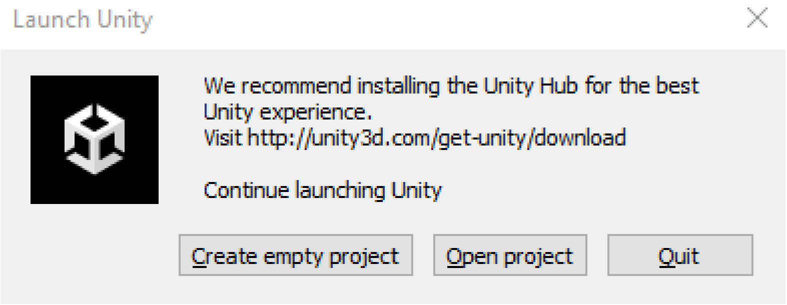
An installation for version 6000.0.19f1 for Windows, MacOS, or Linux, can be found on the Unity website. https://unity.com/releases/editor/whats-new/6000.0.19

To install, download the installer for your machine, run the executable file, and follow the setup wizard. Wait for the install to finish before continuing.

Opening the Project

Download and unzip the provided copy of the Manu-Scriptwriter project.

On opening Unity 6, select Open Project.



Launch Unity

We recommend installing the Unity Hub for the best Unity experience.
Visit http://unity3d.com/get-unity/download

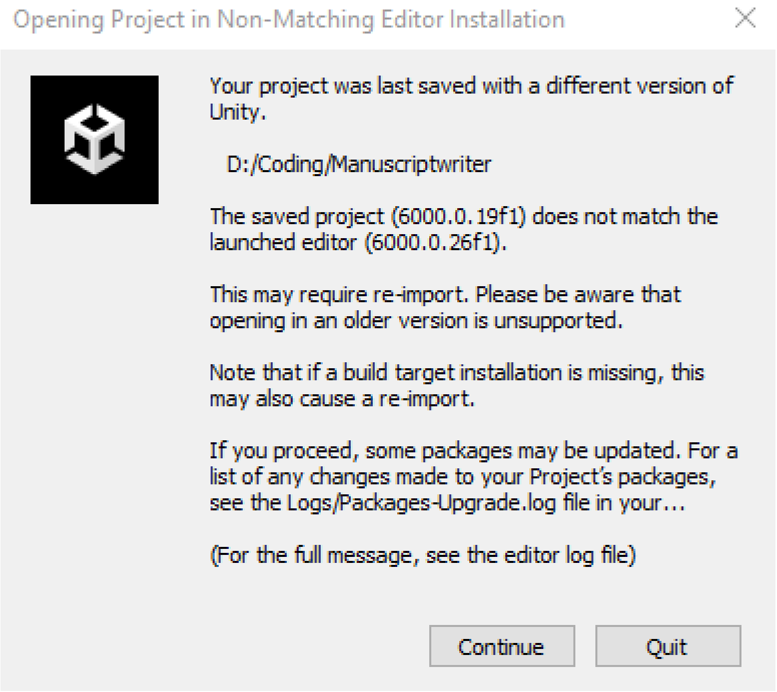
Continue launching Unity

Then, navigate to the unzipped Manu-Scriptwriter folder. Click on the folder or open it, then click select folder. Wait for Unity to load the project.

Optional: Updating the Project to a Later Version

If, on opening the project, you are met with the following window, it is safe to click continue, then wait for it to load once more.

The project does not depend on version 6000.0.19f1, and the update process should not require any other input. If you experience any issues, feel free to request help, or delete and re-download the project and the correct version of the editor.



Opening Project in Non-Matching Editor Installation

Your project was last saved with a different version of Unity.

D:/Coding/Manuscriptwriter

The saved project (6000.0.19f1) does not match the launched editor (6000.0.26f1).

This may require re-import. Please be aware that opening in an older version is unsupported.

Note that if a build target installation is missing, this may also cause a re-import.

If you proceed, some packages may be updated. For a list of any changes made to your Project's packages, see the Logs/Packages-Upgrade.log file in your...

(For the full message, see the editor log file)

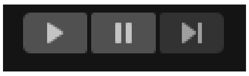
Testing Instructions and Hints

From here on, what you do with the project is up to you. You should not need to touch any of the code itself, outside of anything within a text file.

Play around with the provided functionality using the Manu-Scriptwriter language or add new things to scenes based on existing objects until you are satisfied that you've made a good scene.

Controls

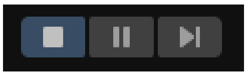
To start a scene, enter Play mode by clicking the Play button in the top middle of the editor.



While in Play mode:

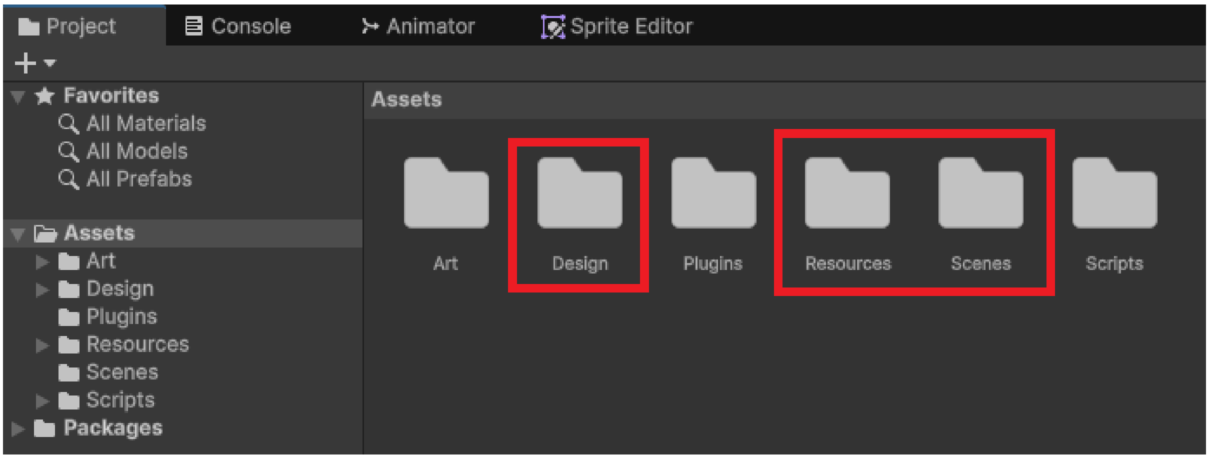
- Use WASD or the arrow keys to move the player character.
- Press Enter to interact with something.
- Press Enter to advance text during an interaction.

To stop the scene at any time, leave Play mode by clicking the stop button. You should do this whenever you want to edit a script, or if you encounter an issue, such as falling off the world, or encountering bugs.



Organisation

There are a few folders which you'll either work in or take assets from, which can be viewed in the project window.

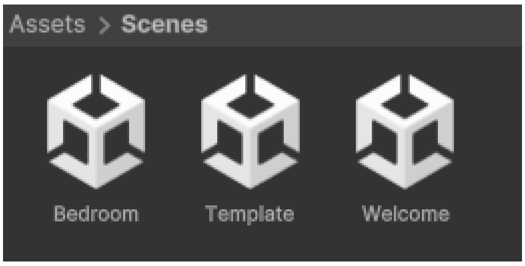


Scenes Folder

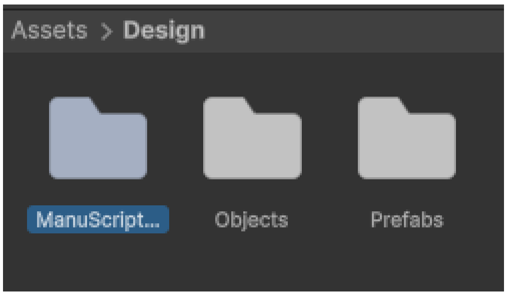
Within this folder are two sample scenes with simple interactions, and one template with everything but the interaction set up for use.

To open a scene, double click the icon for the desired scene. For a simple example, I recommend the 'Welcome' scene!

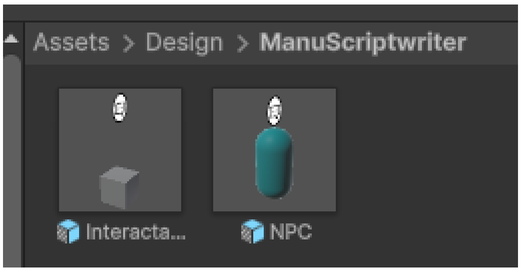
Feel free to edit an existing scene and scripts to your own liking, or create something entirely new within 'Template'. Whichever is easiest for you.



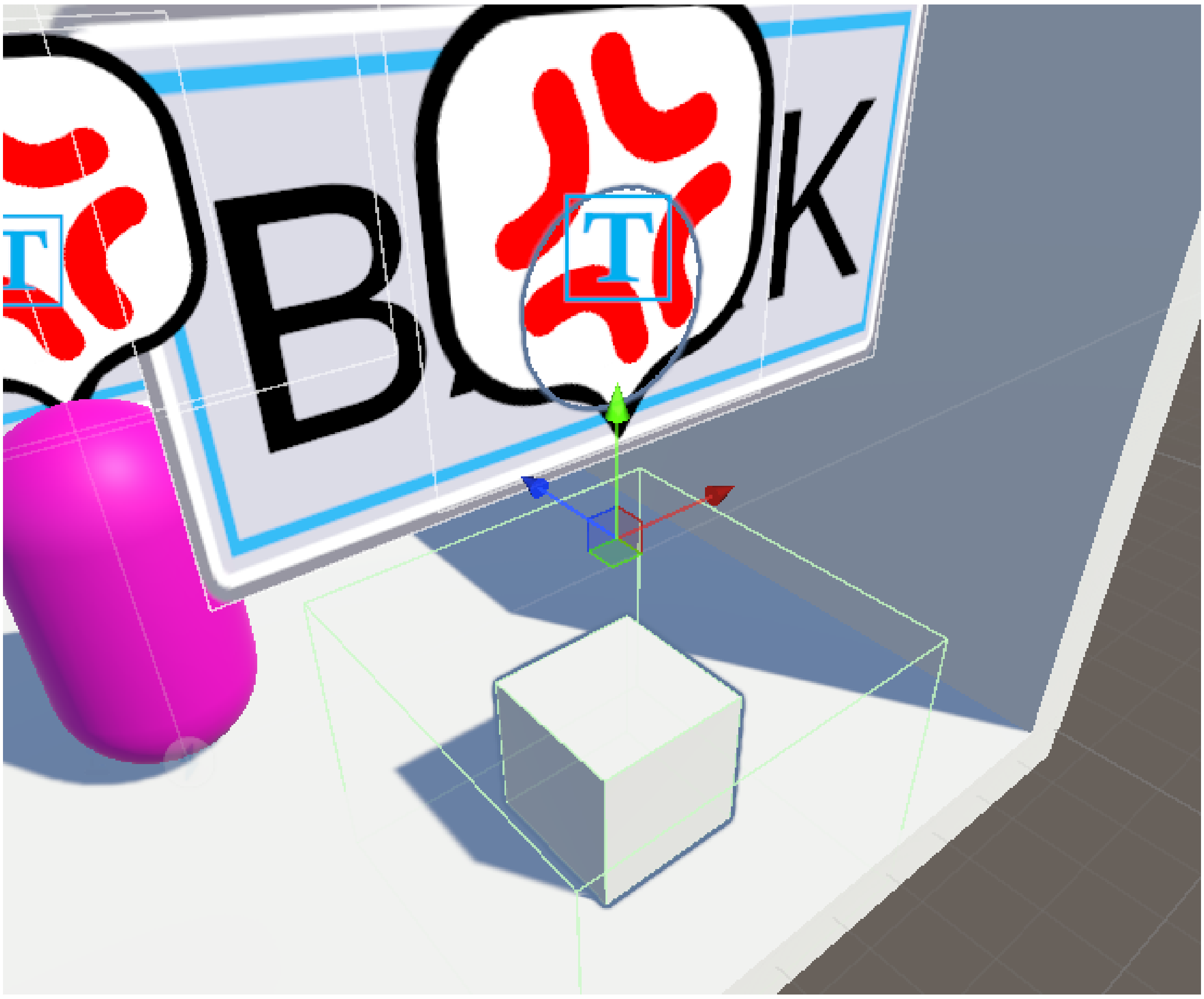
Design/ManuScriptwriter Folder



Within this folder are common, reusable objects for a scene/interaction, known as prefabs. Importantly, this includes an NPC and an "Interactable Object" prefab, which has been setup with everything necessary to work with the text scripts in advance.



To use a prefab, simply click and drag from the project window and into the scene or the hierarchy windows. You can move that object within the world either using the transform tool or the Transform section of the inspector window.



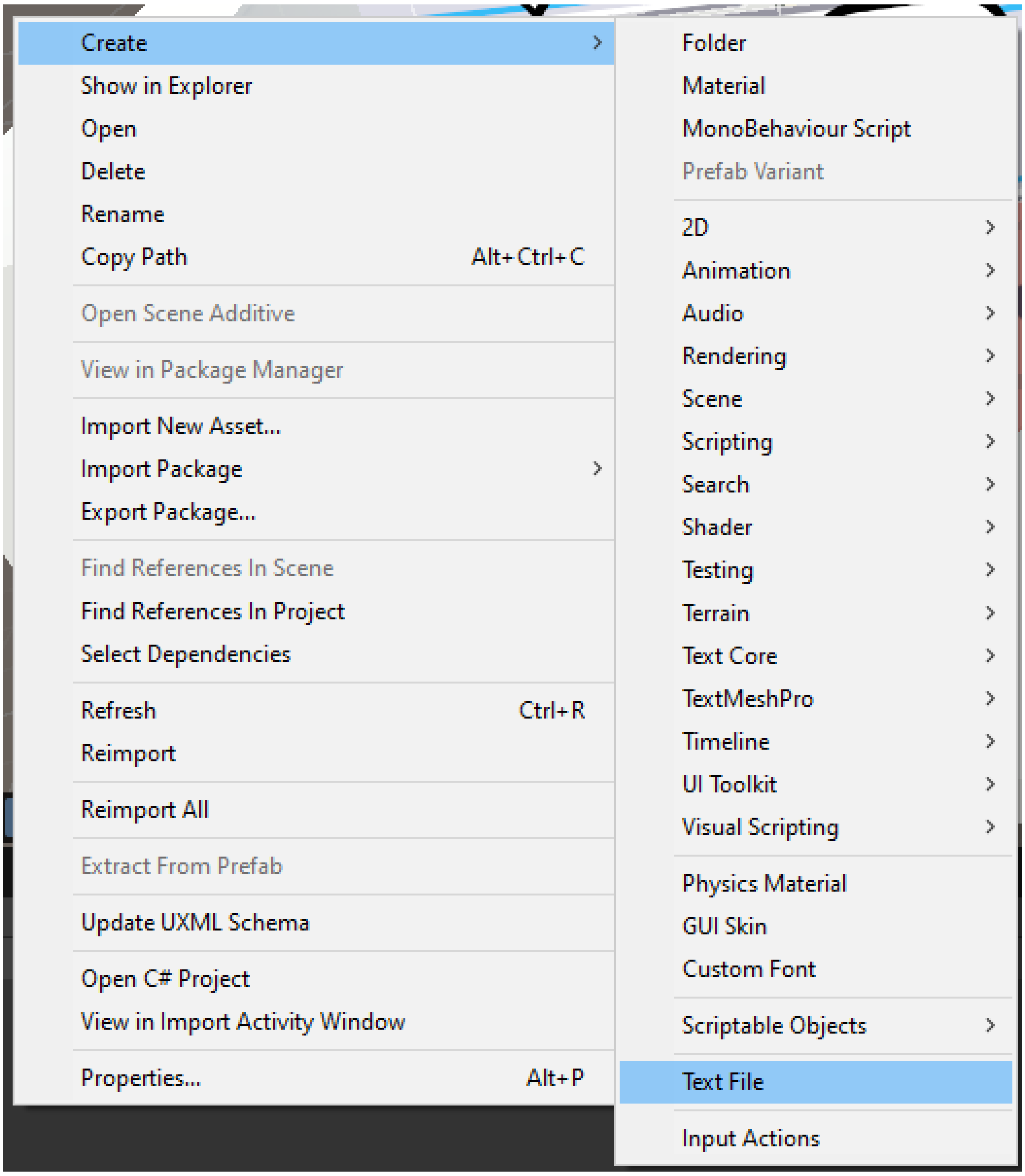
If you place an object in the scene that doesn't use one of these prefabs, and then decide to make it interactable later, you can right click that object in the hierarchy, then go to ManuScriptwriter -> Make Selection Interactable. This should add everything necessary.

Resources Folder

Within this folder are any text files used within scenes in the project. All scripts you create should be placed somewhere within this folder, regardless of how you organise them.

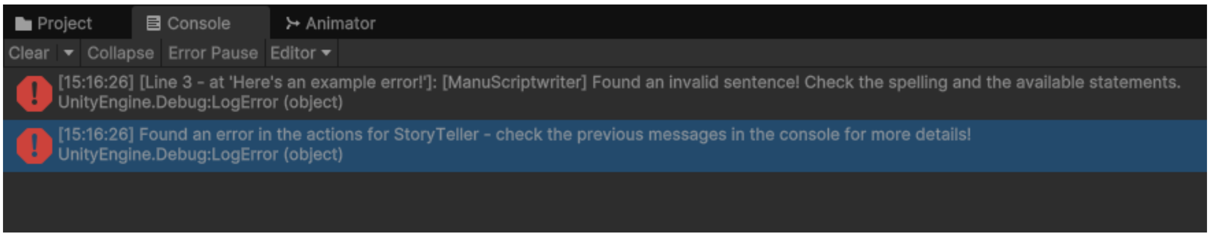
You should be able to open a text file in this folder by double clicking the file.

To create a new text file, right click within the project window, go to Create -> Text File.



Errors, Problems, and Debugging

Chances are, one of your custom scripts will break if they aren't understood by the game. In the case that an error occurs, check the Console window.



```
[15:16:26] [Line 3 - at 'Here's an example error!']: [ManuScriptwriter] Found an invalid sentence! Check the spelling and the available statements.
UnityEngine.Debug:LogError (object)
[15:16:26] Found an error in the actions for StoryTeller - check the previous messages in the console for more details!
UnityEngine.Debug:LogError (object)
```

If the error message mentions "[ManuScriptwriter]", you'll want to follow the instructions to correct something in one of your own text files. See if you can figure out how to fix the issue yourself!

If not, then you may have encountered a bug with my code that is not your fault. While this project has been provided to you with the expectation that it will not break for the purposes of this research, there is always the chance that something like this will occur.

In general, make note of the problem, but see if you can work around it.

Additional Guidance

If you have trouble with some of the core functionality within the Unity editor, unrelated to the provided language, you may want to look towards the official Unity documentation or tutorials.

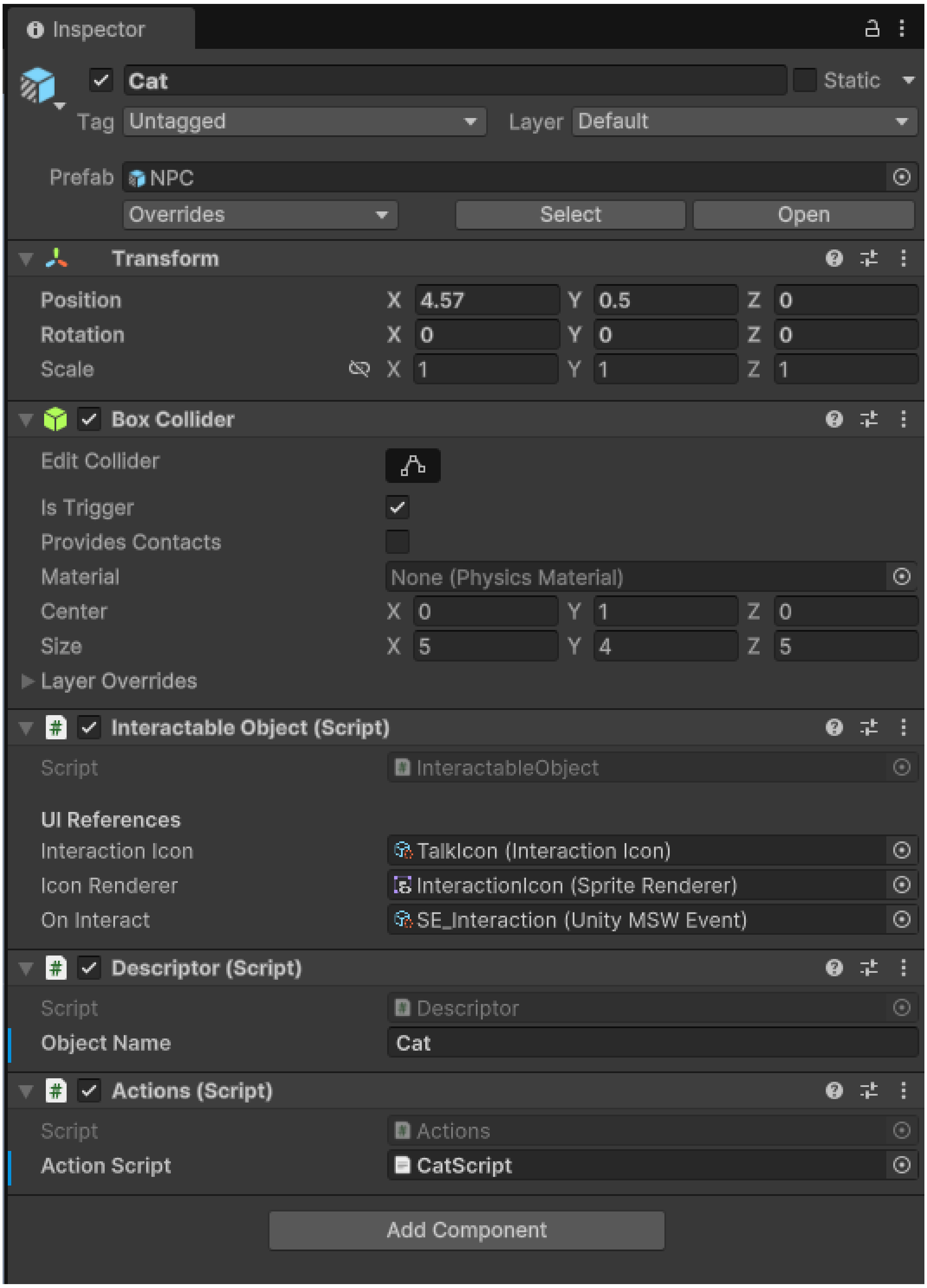
This tutorial serves as a good introduction to the Unity Editor, though not all the content covered may be relevant or necessary.

https://learn.unity.com/tutorial/explore-the-unity-editor-1

Using Manu-Scriptwriter

Within Unity Editor

Some use of the Unity Editor is required for a scene to work, as a script will not run otherwise. This logic is displayed in the inspector panel, usually to the right of the screen.



☑ Cat □ Static

Tag Untagged Layer Default

Prefab NPC

Transform

Position	X	4.57	Y	0.5	Z	0
Rotation	X	0	Y	0	Z	0
Scale	X	1	Y	1	Z	1

☑ Box Collider

Edit Collider

Is Trigger ☑

Provides Contacts □

Material None (Physics Material)

Center	X	0	Y	1	Z	0
Size	X	5	Y	4	Z	5

Layer Overrides

☑ Interactable Object (Script)

Script InteractableObject

UI References

Interaction Icon TalkIcon (Interaction Icon)

Icon Renderer InteractionIcon (Sprite Renderer)

On Interact SE_Interaction (Unity MSW Event)

☑ Descriptor (Script)

Script Descriptor

Object Name Cat

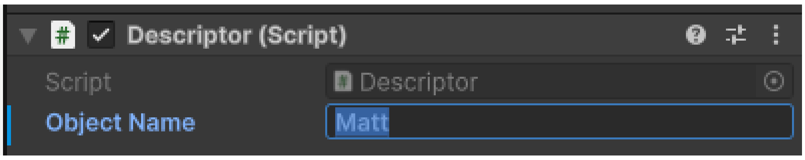
☑ Actions (Script)

Script Actions

Action Script CatScript

Objects and NPCs can be given a name using the “Descriptor” component in the inspector. This can be changed by editing the “Object Name” property.

Multiple objects in one scene should not share the same name.

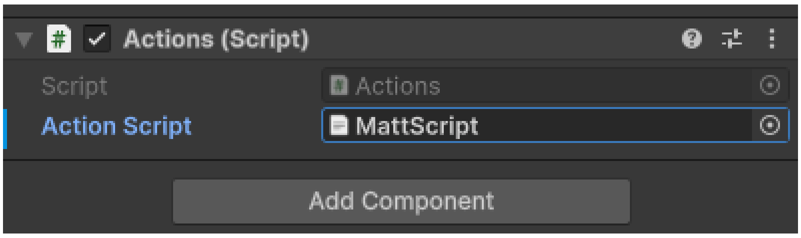


Script: Descriptor

Object Name: Matt

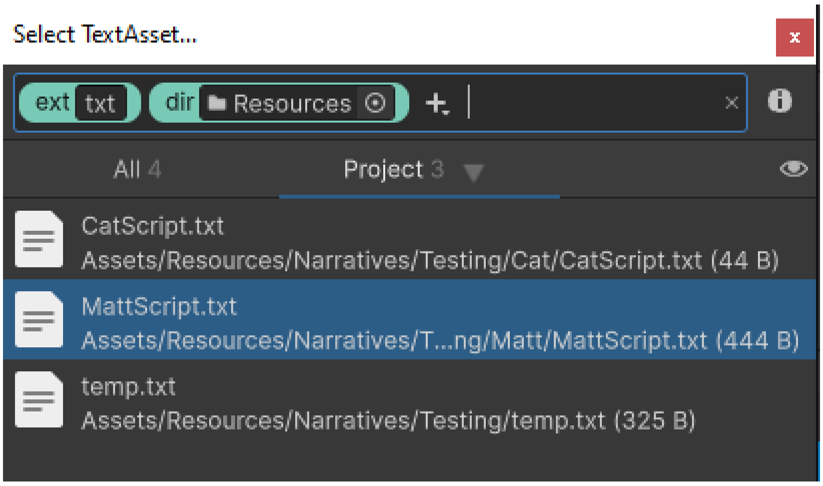
Each object and NPC holds its own script within the “Actions” component in the inspector. This can be changed by editing the “Action Script” property.

Either click and drag the script from the resources folder in the project window to this property or click the target button and search for the file in the search window.



Script: Actions

Action Script: MattScript



CatScript.txt
Assets/Resources/Narratives/Testing/Cat/CatScript.txt (44 B)

MattScript.txt
Assets/Resources/Narratives/T...ng/Matt/MattScript.txt (444 B)

temp.txt
Assets/Resources/Narratives/Testing/temp.txt (325 B)

Within Manu-Scriptwriter Scripts

Parts of any sentences that can be changed are highlighted in blue.

Events

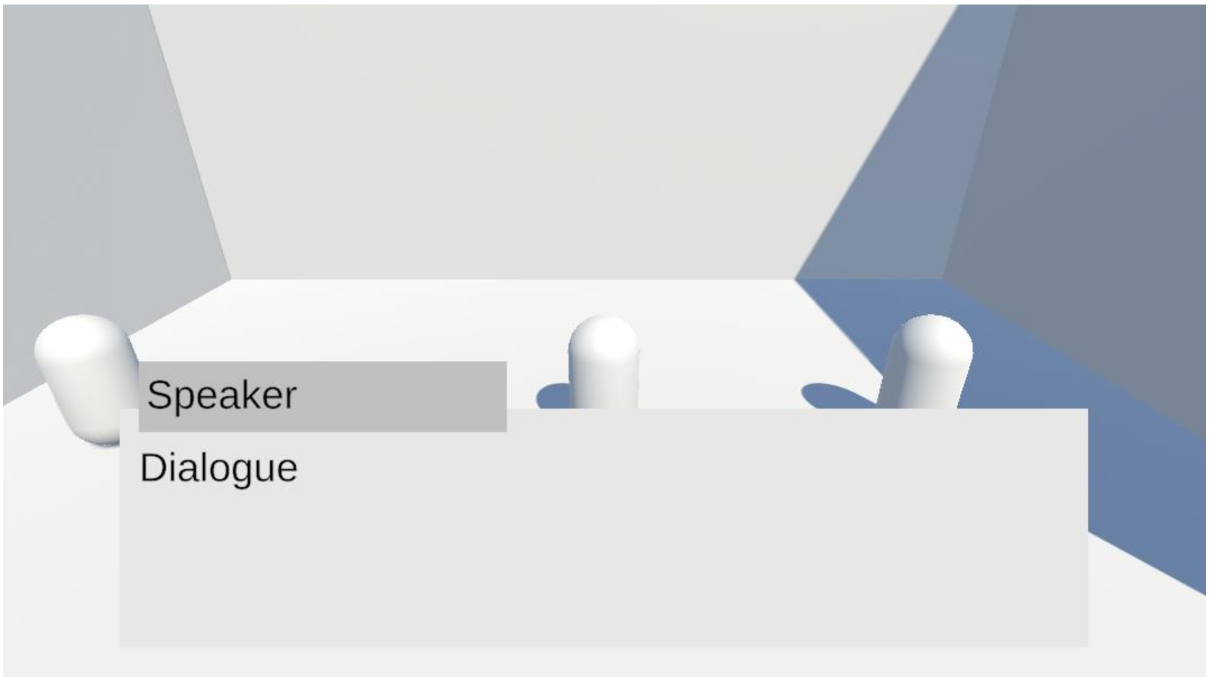
When the Player speaks with Target
When the Player talks to Target
When the Player talks to the Target
When the Player interacts with the Target

The only currently available event within Manu-Scriptwriter. This indicates that dialogue should occur after interaction with the specified target. “Target” must share a name with an object or NPC.

Dialogue

Speaker: Dialogue

The standard method of writing a line of dialogue. The Speaker does not need to share a name with an object or an NPC.

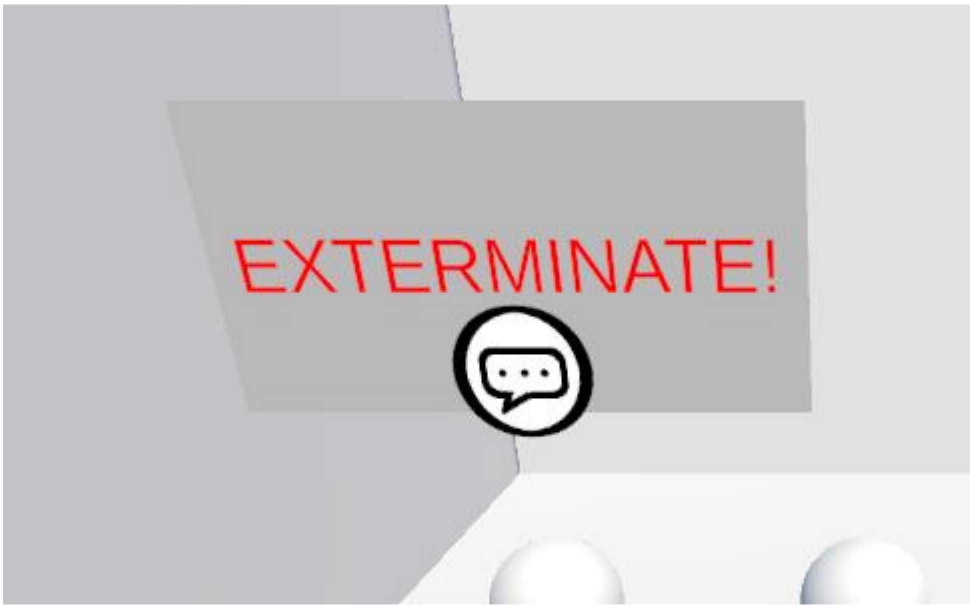


Speaker

Dialogue

Speaker barks: Dialogue
The Speaker barks: Dialogue

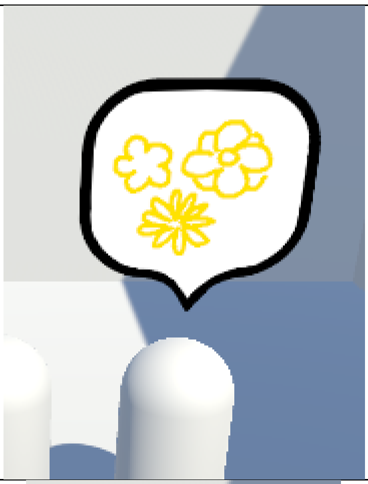
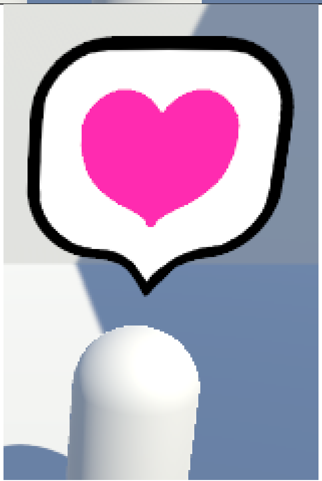
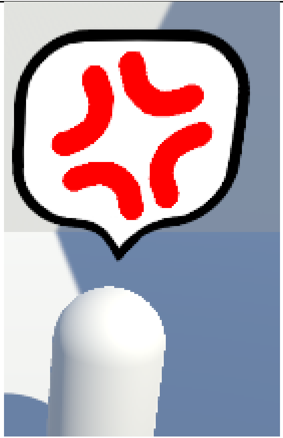
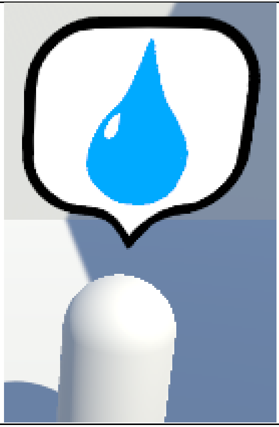
An alternative method of communicating dialogue. This is intended for short sentences. “Speaker” must share a name with an object or an NPC. After a short period of time, the bark will disappear automatically.

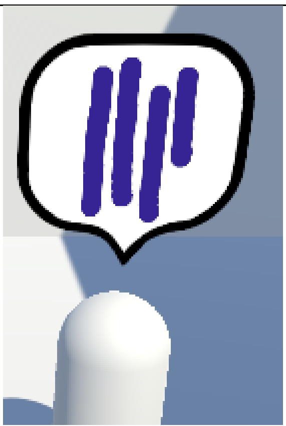
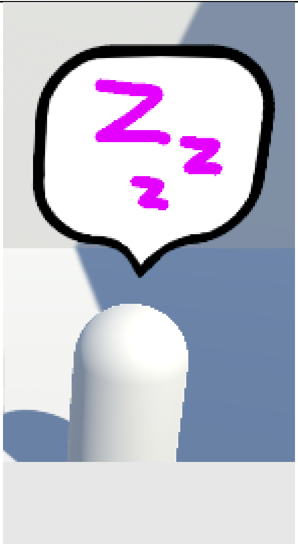


EXTERMINATE!

Target feels emotion.
The Target feels emotion.

A simple method of conveying little emotions using iconography. “Target” must share a name with an object or an NPC. The available emotions are as follows:

“happy” or “sunny”	
“love”	
“mad”, “angry”, or “annoyed”	
“nervous” or “awkward”	

"sad" or "depressed"	
"tired"	

Camera

The camera freezes.

The camera will stop moving entirely. This continues after the end of an interaction.

The camera focuses on Target.
The camera focuses on the Target.

The camera will follow the specified target. "Target" must share a name with an object or an NPC. This continues after the end of an interaction. Return focus to the player by specifying "**The camera focuses on the player.**"

The camera shakes from side to side.

This causes the camera to shake erratically from left to right. This effect is removed at the end of an interaction.

The camera stops shaking.

This stops the camera from shaking if it was previously shaking.

1.3 – Testing Questionnaire

Investigating Narrative Systems and Tools for Games - Testing Questionnaire

Once you have spent some time working within the provided project, please fill out this questionnaire.

Please see the information sheet provided before your participation in the study for full details regarding the study, and regarding the storage and use of your data, including the full data protection statement as covered by the General Data Protection Regulation 2016 (GDPR).

Section 1 – Participant Background

Question 1. How much experience do you have with writing in a programming language on a scale of 1-5, where 1 is no experience, and 5 is professional experience.

Question 2. Did you have any experience with any of the following game engines before taking part in this study?

- Unreal Engine
- Unity
- Godot
- Game Maker
- RPG Maker
- Ren'Py
- Twine
- Scratch
- Other [Allow custom input]

Section 2 –System and Tool User Experience

For this section of the form, please rank your experience with the provided narrative scripting language based on the following scales from 1-5.

Question 1. Rank your experience of the tool based on the following scales from 1-5:

a) I think that I would like to use this system frequently.
b) I found the system unnecessarily complex.
c) I thought the system was easy to use.
d) I think that I would need the support of a technical person to be able to use this system.
e) I found the various functions in this system were well integrated.
f) I thought there was too much inconsistency in this system.
g) I would imagine that most people would learn to use this system very quickly.
h) I found the system very cumbersome to use.
i) I felt very confident using the system.
j) I needed to learn a lot of things before I could get going with this system.

Section 3 – General Feedback

Question 1. How could the narrative tool be improved?

Question 2. In what ways was the tool approachable or easy to understand?

Question 3. In what ways could the tool be improved to be more approachable or easier to understand?

Question 4. Did you encounter any bugs or unexpected behaviour while working with the tool? If so, what happened, and how was it caused?

Appendix 2 – Results

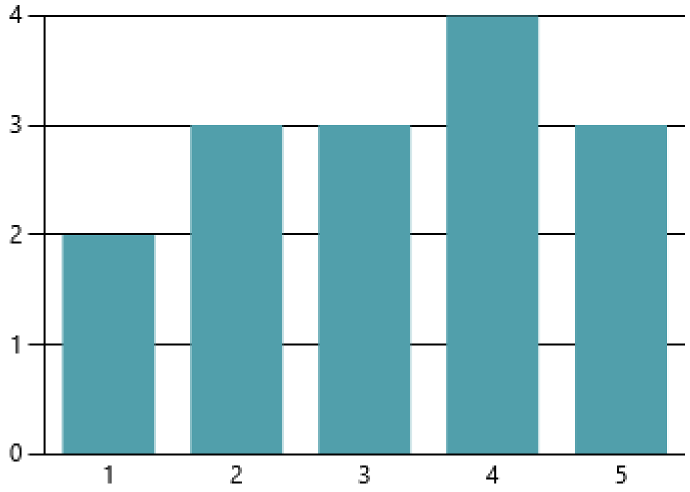
2.1 – Research Data

2.1.1 – Section 1, Prior Experience

Id	How much experience do you have with writing in a programming language on a scale of 1-5, where 1 is no experience, and 5 is professional experience?	Did you have any experience with any of the following game engines before taking part in this study?
1	4	Unreal Engine;Unity;Godot;RPG Maker;Scratch
2	5	Unreal Engine;Unity;Godot;Scratch;Proprietary Engines
3	3	Unreal Engine;Unity;Scratch
4	5	Unreal Engine;Unity;Godot;RPG Maker;Scratch
5	1	Unreal Engine
6	2	
7	3	Unity;Ren'Py;Twine;Scratch;Roblox Studio
8	4	Unreal Engine;Unity;Scratch
9	2	Unreal Engine;Scratch
10	3	Unreal Engine;Unity;RPG Maker;Inkle
11	1	Scratch
12	4	Unity;Unreal Engine;Game Maker
13	2	Scratch;Unreal Engine
14	5	
15	4	Unreal Engine;Unity;RPG Maker;Ren'Py;Twine;Scratch

Question 1

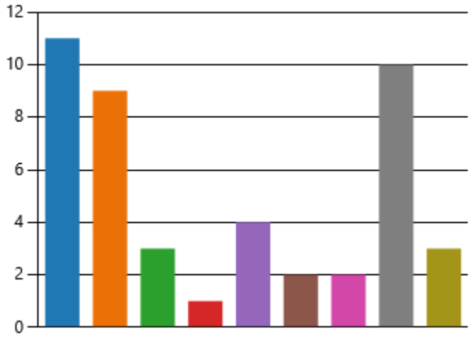
How much experience do you have with writing in a programming language on a scale of 1-5, where 1 is no experience, and 5 is professional experience?



Question 2

Did you have any experience with any of the following game engines before taking part in this study?

●	Unreal Engine	11
●	Unity	9
●	Godot	3
●	Game Maker	1
●	RPG Maker	4
●	Ren'Py	2
●	Twine	2
●	Scratch	10
●	Other	3

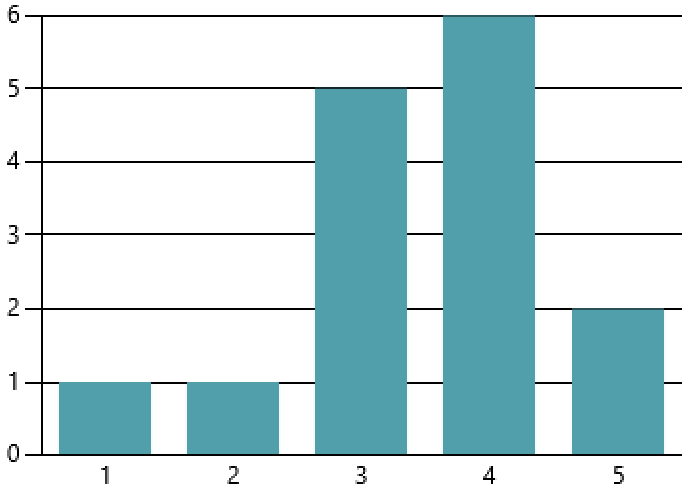


2.1.2 – Section 2, System Usability Scale

Id	SUS01	SUS02	SUS03	SUS04	SUS05	SUS06	SUS07	SUS08	SUS09	SUS10	SUS Score
1	3	3	4	1	3	2	3	2	4	1	70
2	4	2	4	1	4	1	5	1	5	1	90
3	5	1	5	2	5	1	4	1	5	2	92.5
4	4	1	4	1	4	1	5	1	4	2	87.5
5	4	3	4	1	5	2	5	2	4	3	77.5
6	4	1	5	1	4	1	5	1	4	1	92.5
7	3	1	4	2	5	1	4	1	3	1	82.5
8	3	3	3	3	3	2	2	2	3	2	55
9	3	1	4	2	5	1	5	1	3	2	82.5
10	5	1	5	1	5	1	5	1	5	2	97.5
11	4	1	5	4	5	1	5	1	4	2	85
12	3	2	3	2	4	1	5	1	4	2	77.5
13	2	1	4	3	5	1	3	5	4	4	60
14	1	3	4	2	3	1	4	2	4	3	62.5
15	4	1	5	1	4	1	4	2	3	2	82.5

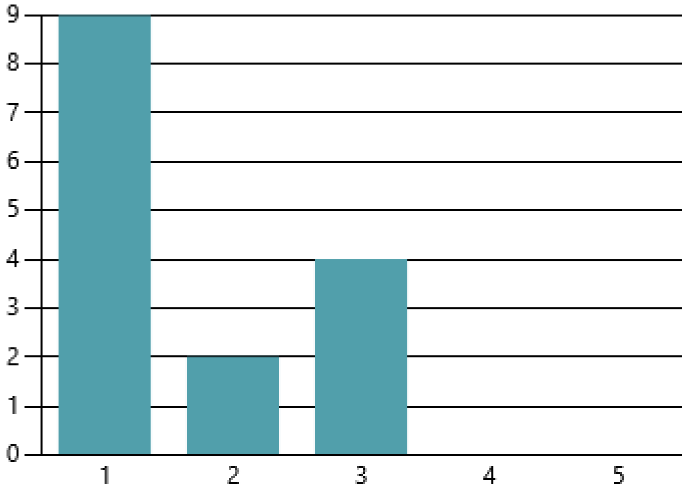
Question 3 – SUS01

I think that I would like to use this system frequently.



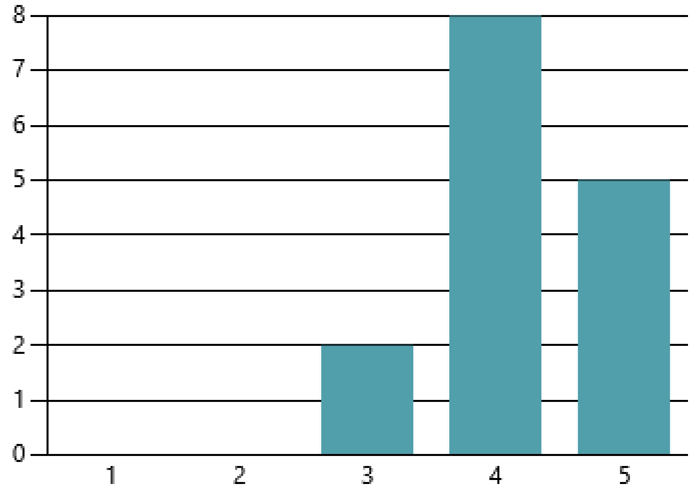
Question 4 – SUS02

I found the system unnecessarily complex.



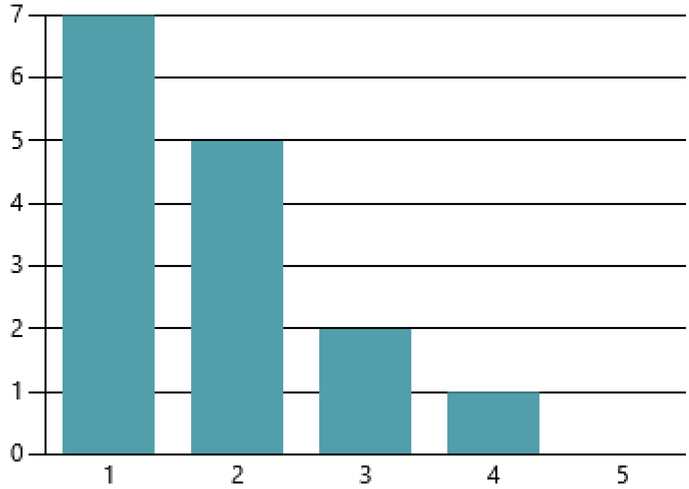
Question 5 – SUS03

I thought the system was easy to use.



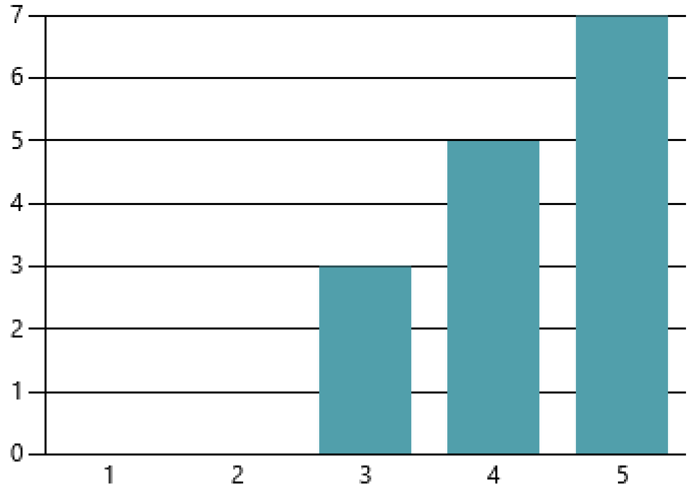
Question 6 – SUS04

I think that I would need the support of a technical person to be able to use this system.



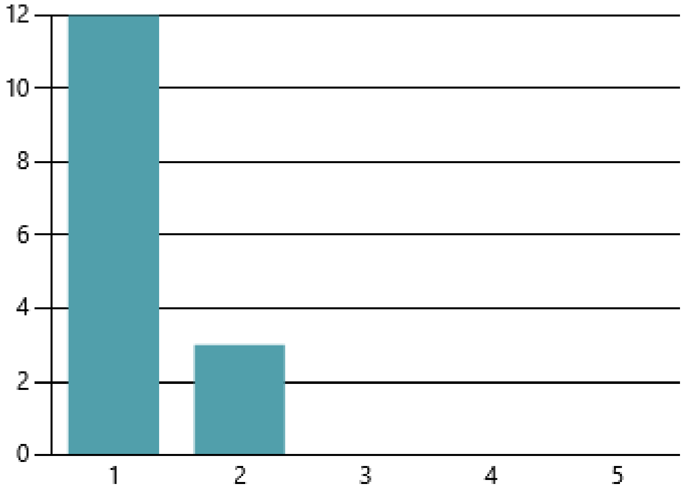
Question 7 – SUS05

I found the various functions in this system were well integrated.



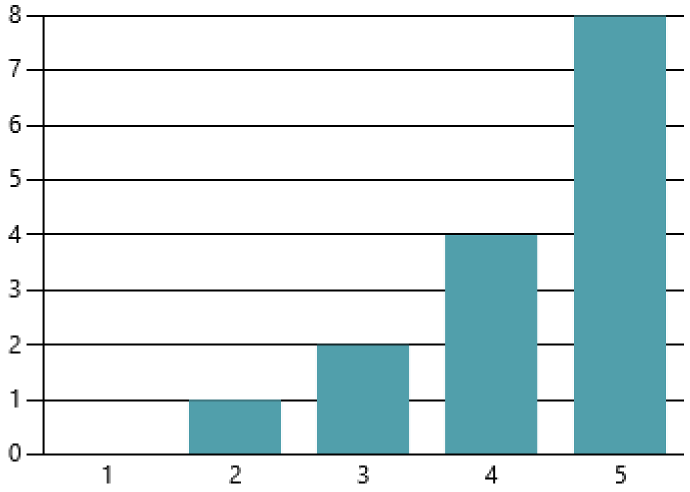
Question 8 – SUS06

I thought there was too much inconsistency in this system.



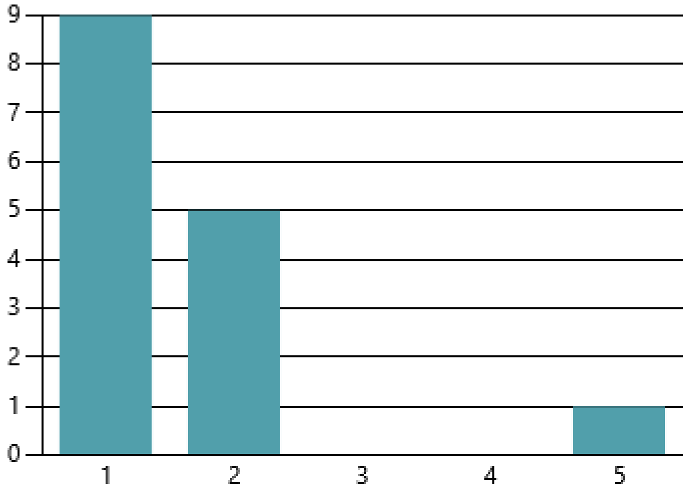
Question 9 – SUS07

I would imagine that most people would learn to use this system very quickly.



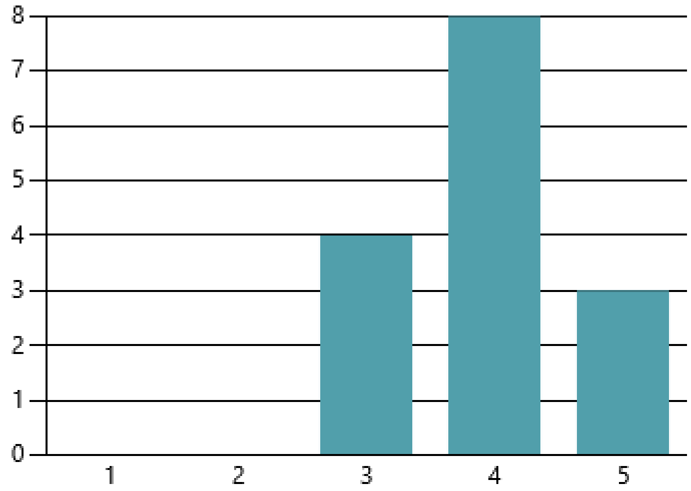
Question 10 – SUS08

I found the system very cumbersome to use.



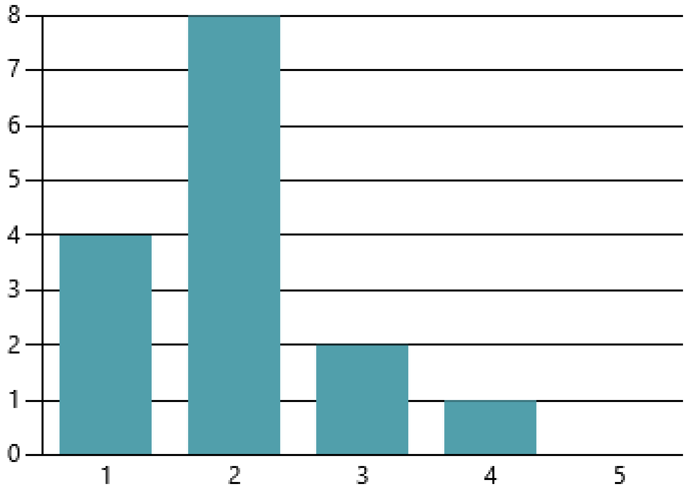
Question 11 – SUS09

I felt very confident using the system.



Question 12 – SUS10

I needed to learn a lot of things before I could get going with this system.



2.1.3 – Section 3, General Feedback

Id	How could the narrative tool be improved?	In what ways was the tool approachable or easy to understand?	In what ways could the tool be improved to be more approachable or easier to understand?	Did you encounter any bugs or unexpected behaviour while working with the tool? If so, what happened, and how was it caused?
1	If the text files could be changed in Unity itself, the workflow would be much smoother. More ways to trigger dialog sequences and being able to associate character portraits may also be good features to increase the number of use	The natural language used to write dialog is very approachable. The barks are a particularly useful feature for approachability. They enhance dialog greatly while being very simple to use!	Having to use a separate editor for the text files adds an extra layer of complexity, so being able to edit directly in Unity would make the tool more approachable as a user.	I encountered an issue where the name box of the speaking character would be in the wrong position and as a result end up behind the center of the main dialog box. The name box would update correctly,

	cases for the tool. While more of a fringe option, being able to pause between dialog lines within the same conversation would be nice as well (e.g. two characters have an awkward moment in a conversation, leading to a pause)			but not be visible during play.
2	I think the tool is great and very useful, the improvement that I can think of is just expanding upon it to have more features, but creating custom features is easy enough, so not much else.	It is easy due to the fact that its practically the same as just reading a script. In terms of use it is easy as most things are delt with automatically, just need to attach a the action script.	Either slightly less rigidity with punctuation/new lines, or some debug error checking to help find the issue/warn users. If an action script accidentally was edited and ended with an empty line, could result in game crashes.	Small issues with punctuation or new lines. Example having a text action script with the last line being empty reports an issue with index being out of bounds.
3	Perhaps attaching instructions in the form of a video tutorial that demonstrates using it so people who are less confident with the engine can mimic the creator's actions first.	The use of the txt files was very seamless and easy to understand. The files within the project are organized and it is easy to find what you need. Since an NPC prefab is already created for you, it is easy to add new characters to the scene. Having a welcome scene also helps introduce the user to the tool more gently.	I believe having a demonstration/tutorial video creating a scene would be more than sufficient in making the tool more approachable as the player can practice using the tool by copying the creator's tutorial. If there is any confusion they are able to rewatch the video as many times as necessary. This would remove needing an in-person demonstration to each writer and save a significant amount of time.	Adding a space at the end of the txt file blocks the user from witnessing the interaction between an NPC.
4	From a technical perspective, allowing users to input their own schema to parse key phrases. That would be fantastic since then this tool can be customised for specific needs. An engineer would need to provide support to allow this though.	It was easy as I was handling only a couple of text files. I could naturally type out the scene the scene I wanted so I could preplan beforehand what the scene should look like and then directly implement it. The example scene was more than enough to figure out how to construct a scene	Maybe providing these commands in a debug log would be nice or some way to output these commands/schema? That way I don’t need to look back and forth at a manual I can just look at Unity directly?	N/A
5	Restructuring the guide to put a emphasis on case/punctuation sensitive to avoid common user bugs if this little caveat cannot be	Most functions handled under the hood by the plugin such as emote visibility and camera lerps.	Restructuring the guide to put a emphasis on case/punctuation sensitive to avoid common user bugs if this little caveat cannot be	Punctuation sensitive with missing full stops breaking the script

	changed. Object names could be required to be in ALL CAPS so that there is a better differentiation between dialogue lines and variable changes Emotions can be shown for a certain amount of seconds set by the designer before being turned off, or have them be controlled by the designer to disappear upon the next line of dialogue, as emotions can overlap and overstay if the player skips through lines of dialogue on repeat play through. More emotions can be added overtime such as question marks for more intricate cutscenes Maybe some syntax in the game world when the script wont start due to a grammatical issue. Though this is a minor change out of the list. Within the NPC prefabs, being able to spawn and place a camera in that prefab and then giving it a object name such as camera 2 could maybe allow designers to allow for more cinematic camera angles when using the line the camera focuses on Kacy (Camera 2)		changed. Object names could be required to be in ALL CAPS so that there is a better differentiation between dialogue lines and variable changes Emotions can be shown for a certain amount of seconds set by the designer before being turned off, or have them be controlled by the designer to disappear upon the next line of dialogue, as emotions can overlap and overstay if the player skips through lines of dialogue on repeat play through. More emotions can be added overtime such as question marks for more intricate cutscenes Maybe some syntax in the game world when the script wont start due to a grammatical issue. Though this is a minor change out of the list. Within the NPC prefabs, being able to spawn and place a camera in that prefab and then giving it a object name such as camera 2 could maybe allow designers to allow for more cinematic camera angles when using the line the camera focuses on Kacy (Camera 2)	
6	I can't think of anything that'd make it easier to use than it is. Its functionality and the explanation are clear and concise, and more approachable for it,	The Manu-Scriptscriptwriter Testing Manual was absolutely invaluable. The fact that most of the work with the tool was confined to editing .txt files made it	Little comes to mind.	When searching for a present Action Script though the target button beside the "Action Script" property, the search function could not locate

	especially with the aid of the Testing Manual.	particularly approachable, and knowing where exactly I'd be working within Unity (the Design, Resources, and Scenes folders within the Assets folder) and what exactly I'd be doing in them made working in an engine I have no experience with much less intimidating.		the file. When attempted via dragging and dropping the Action Script file into the property instead, this worked fine.
7	It feels limited in that you can only interact with objects/NPCs and feels like a Visual Novel where you are stuck in one scene. If there was a way to switch scene with the tool that would make it (in my opinion) better. For example: interacting with a door and changing scene to a different room. This isn't an issue for me since I have knowledge of Unity and C# but the plugin feels like a way to circumvent scripting in C#	It was easy to understand in terms of it having the Manual with it which explained how to use it. Once I had read that I found it easy to use. Having the templates provided also helped with understanding how it worked as I could reference what I was doing to something already made to check if I was using it right (e.g. I forgot to put "When the Player talks to Target" with the first NPC I made, looked at the script for Daniel and realised my mistake)	N/A - I can't think of a way to make it more approachable/easier to understand. I think it explains itself well in the manual, even just to downloading the Game Engine (Unity). This could be because of my experience with Twine and RenPy though.	No
8	Making my own text boxes was fun, but it was a little difficult to understand with what needed putting in the text file E.G it took me a little bit to figure out I needed to add "When X interacts with Y" at the start, thanks to my prior knowledge i was able to figure that out though. So more documentation on how things work maybe	Once I understood everything It was pretty simple to use, It was just a case of learning how it worked that proved slightly hard	Mostly documentation, Explaining what is needed when creating a new txt file for the dialogue and on setting up new scenes. I tried setting up a new scene and ran into some difficulty setting everything up, eventually I did get everything working but this was again thanks to my prior knowledge with Unity and Game Engines	I did encounter some errors but they dissapeared once I understood what i was doing and how things worked
9	Can't think of any improvements on the current code other than to just develop it further / add more features. I barely know anything about coding so take everything I say with a heaping spoonful of salt.	Manual given was very easy to read and any issues were completely user error (not knowing you can't have an empty line of code, not knowing to not use quotation marks, etc). After a bit of trial and error I completely got the hang of	Can't think of any, the only way it could be more approachable is through being more familiar with the software myself, which is no fault of yours.	When trying to interact with NPCs in the "Welcome" room, no text would appear and the error message "Index was outside the bounds of the array" would be at the bottom of the screen. This however was

	It would be nice for the system to be able to change text colour, speed, position and even add noises if those are possible? Like how you can move the camera with it. My main thought is that if these things aren't in the system, it may be difficult to connect it to said system later after coding them separately? If this doesn't make sense or wouldn't be an issue then please disregard it.	it and was able to utilise every provided feature.		not at all a problem with any other rooms or entities.
10	Obviously it could be improved by expanding the range of features and abilities, but even in this simple state I think it would be extremely useful to game designers looking to include a simple to medium complexity narrative line to their games without any experience. With more experienced coders, I think the tool could be implemented and made to produce complex narrative formats quite easily even with the current feature base. I think the next most useful step would be to add simple choice based dialogue as a base feature of the code (ie, question and answer) (As an Unreal Engine user it would be extremely improved by becoming an unreal plugin, I want it)	The tool was immediately clear and simple to understand, given basic unity experience. The documentation was clear and precise and I was able to install it, get it running and start editing scenes within less than 20 minutes (including unity install time). I think even newcomers to unity and code would be able to approach a simple narrative very easily with this tool.	I genuinely can't think of anything - even before I opened the documentation, it was immediately clear how it worked and applied itself to the provided test scenes. Short of it having a step-by-step tutorial popup on install, I don't think it could be any clearer.	Visual studio did briefly refuse to open the script - but then I closed and re-opened it and it suddenly worked, so I don't think that's a problem with the tool so much as my computer. Other than that I didn't encounter any errors!
11	I feel that the narrative tool worked perfectly for the purpose I personally set out to use it for, so I am unsure how to answer this one.	I have next to no knowledge of coding or using an engine that would allow me to create a functioning scene, but through picking through the available items and using the tools laid out in a clear and easy to navigate system, I was able to	A user guide could potentially be useful, that appears on initial startup to explain the uses of each tool- although this would only be necessary if the user had never used anything related to coding before.	No.

12		produce a functional end product with no prior knowledge.		
12	The base behaviours provided are good, there's some areas I think there is for improvement or that might have a lot of room for user error I'll point out. - this is likely beyond the scope of the project but it would be good if there was an easy way to add new behaviours, e.g. theres a base class thats inherited + maybe an interface that allows you to assign the text and an action to happen so that people can quickly add their own behaviours if they have programming knowledge. - setting new emotions - I know there is a menu for creating new emoticon scriptable objects, but when i create one it doesnt automatically get added somwhere which allows me to use it immeditely, and im not entirely sure where that can be done. If all of the emotions, aliases and images were stored in one scriptable object as a struct it might be easier to use possibly, then again that could make it more complicated to use when theres a lot of emotions. Either way, when i make a new emoticon i dont know how to link it into the system which might be complicated from a user perspective. - This is beyond your scope, but having an ability to branch dialogue, define	I found it easy to understand how to change dialogue, set new things up, etc. As a system it's very friendly for designers to get into immediately and start changing things.	While you said i shouldnt go into the code for me at least knowing how some things work helps me to work with them, i think having an API reference or explainations of what each component does could be helpful for users (or commenting the code or doing summaries for functions users might touch), along with step by step explaintions of how to set stuff up. While the PDF is good i think theres a way to include docs like that within unity which could make it more user friendly.	Not any that you don't know about

	options for inputs and assigning variables from inside the text itself (thinking yarn spinner with games like disco elysium where you can pick dialogue options and different dialogue options can call functions which's return effect the next line of text, add EXP, etc.) would make it so designers have a lot of power without needing to go into code.			
13	Add a Unity Tutorial stage/ PDF Detailing the locations and specifics of the tool. Perhaps finding a way to make the code text files more visually clear for future editing or longer projects.	Loved that everything was clearly labeled and sectioned within the inspector panel. It made renaming or adding in different .TXT files very intuitive. Along with making a new object interactable very simple.	Same as previously stated.	Had errors trying to load the Manuscript into Unity 6 despite following the tutorial the PDF provided had. I found that some of the objects really did not want to become interactable until I tried it a few times. Along with i kept running off the stage due to some floaty controls.
14	Some options for interactivity beyond camera pans would be a major improvement, for example adding a teleport to chosen location option would be great to let a player move past a doorway. Similarly, adding the option to make choices in the dialogue and having the NPC say the corresponding response would help make scenes feel more like a chose your own adventure book and less like a stage play. Also, adding “quests” would add a lot, e.g. a character requests you to go collect a book, you go click on the book, when you return the character has new dialogue.	My favourite approachability features revolve around anything that keeps the user in the unity application more. The ability to right click and select “make object interactable” is a great example, and really alleviates a new users issues with the Unity UI (I tried to make a text box without it, and struggled to get the scaling correct), whilst also being logically where you’d expect it (I want to do a thing with this object - > right click the object - > select the do thing option - > it now does the thing!). Additionally, the integrated camera options really help avoid a new player needing to fuss with the Unity UI, and help scenes feel much more alive.	As someone used to coding, the lack of highlighting for text in the code editor makes it harder to distinguish what is the target, and what is the action. Maybe a simple VSCode/notepad++ plugin could help? However, given the aim of simplicity, having to leave the unity interface at all can be intimidating. 1) I think a tutorial in Unity would help, especially for people unfamiliar with Unity’s UI. Something along the lines of a. NPC: Hey there, I’m currently talking from script 1, swap to script 2 to hear a joke! To swap, click on me in the hierarchy tab (with the game preview stopped), then click on the script box in the bottom right. There	Barks seem to scale how long they appear with sentence length. This is good for readability, but makes convos desync, and also the initial message appears at the same time rather than separate like a normal convo. The interactable icon for NPC’s appeared on top of the Barks, whereas in my opinion they could be hidden whilst the dialogue is progressing (maybe add an option so that pressing enter hurries the barks along, with an enter indicator on the bark implying that). Lastly I tried to run the welcome scene, but had the following error on load, and no dialogue would happen.

			should be an option called NPCScript2. Select that, then preview the application again! 2) Maybe try and avoid the need for text files all together. Whilst the natural language might read nicer, I personally find it more confusing during the writing scenes process. Also, it makes it harder for me to jump into the middle of a file I've written, as the break points are less clear. I don't have any experience with Unity, but I played around with it for a little bit and managed to create a custom script (using C#) that had the following UI. The Dialogue Box and Text was just how I linked to the text box, so ignore that, but the dialogue lines array is the list of lines that an NPC responds with in sequence. Maybe a solution similar, where you add "steps" with a drop down for what event type it is, what is doing it, and a field to the right for the details would be. (Example mockup, with each section being a dropdown, except dialogue being free text)	
15	I think the ability to use the tool to play animations or for it to remember what interactions you've used as variables for things like puzzles and progression would be a really cool next step!	Your manual was well written and to the point! Any errors I encountered were my own fault for skipping ahead and refusing to read.	I feel like the best way to go about this would be having a proper front end program for the system, though that would be a lot of work lol. I find most people who don't have a lot of programming experience are much happier to use a program with a ui than a tool, yknow? Perfectly accessible and approachable for someone like me. Maybe include details like being	Not while working with it, but for some reason the 'welcome' test room just did NOT work. Like I was spamming enter and every key I could think of and none of them were letting me talk to Matt or the cat. I looked at it and I could not for the life of me figure out why.

			able to choose text colour in the main manual also.	

Question 13

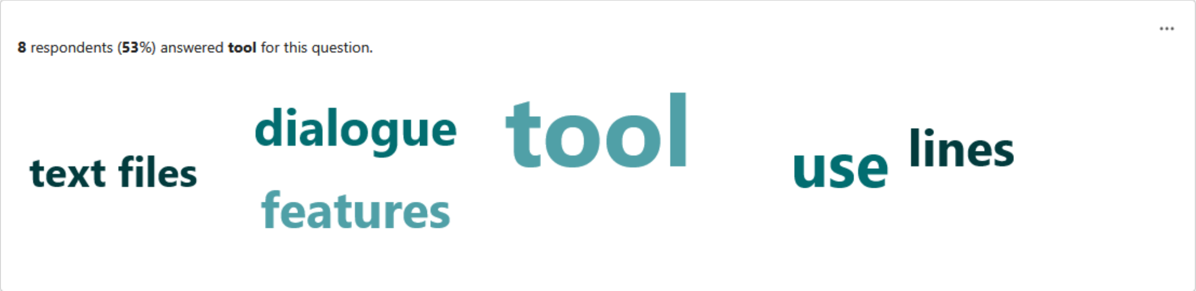
How could the narrative tool be improved?

Id	How could the narrative tool be improved?	Plugin	Features Requests	Documentation	Technical	Feature Changes	Debugging
1	If the text files could be changed in Unity itself, the workflow would be much smoother. More ways to trigger dialog sequences and being able to associate character portraits may also be good features to increase the number of use cases for the tool. While more of a fringe option, being able to pause between dialog lines within the same conversation would be nice as well (e.g. two characters have an awkward moment in a conversation, leading to a pause)	1	1				
2	I think the tool is great and very useful, the improvement that I can think of is just expanding upon it to have more features, but creating custom features is easy enough, so not much else.		1				
3	Perhaps attaching instructions in the form of a video tutorial that demonstrates using it so people who are less confident with the engine can mimic the creator's actions first.			1			
4	From a technical perspective, allowing users to input their own schema to parse key phrases. That would be fantastic since then this tool can be customised for specific needs. An engineer would need to provide support to allow this though.				1		
5	Restructuring the guide to put a emphasis on case/punctuation sensitive to avoid common user bugs if this little caveat cannot be changed. Object names could be required to be in ALL CAPS so that there is a better differentiation between dialogue lines and variable changes Emotions can be shown for a certain amount of seconds set by the designer before being turned off, or have them be controlled by the designer to disappear upon the next line of dialogue, as emotions can overlap and overstay if the player skips through lines of dialogue on repeat play through.		1	1	1	1	1

	More emotions can be added overtime such as question marks for more intricate cutscenes Maybe some syntax in the game world when the script wont start due to a grammatical issue. Though this is a minor change out of the list. Within the NPC prefabs, being able to spawn and place a camera in that prefab and then giving it a object name such as camera 2 could maybe allow designers to allow for more cinematic camera angles when using the line the camera focuses on Kacy (Camera 2)							
6	I can't think of anything that'd make it easier to use than it is. Its functionality and the explanation are clear and concise, and more approachable for it, especially with the aid of the Testing Manual.							
7	It feels limited in that you can only interact with objects/NPCs and feels like a Visual Novel where you are stuck in one scene. If there was a way to switch scene with the tool that would make it (in my opinion) better. For example: interacting with a door and changing scene to a different room. This isn't an issue for me since I have knowledge of Unity and C# but the plugin feels like a way to circumvent scripting in C#		1					
8	Making my own text boxes was fun, but it was a little difficult to understand with what needed putting in the text file E.G it took me a little bit to figure out I needed to add "When X interacts with Y" at the start, thanks to my prior knowledge i was able to figure that out though. So more documentation on how things work maybe			1				
9	Can't think of any improvements on the current code other than to just develop it further / add more features. I barely know anything about coding so take everything I say with a heaping spoonful of salt. It would be nice for the system to be able to change text colour, speed, position and even add noises if those are possible? Like how		1					

	you can move the camera with it. My main thought is that if these things aren't in the system, it may be difficult to connect it to said system later after coding them separately? If this doesn't make sense or wouldn't be an issue then please disregard it.						
10	Obviously it could be improved by expanding the range of features and abilities, but even in this simple state I think it would be extremely useful to game designers looking to include a simple to medium complexity narrative line to their games without any experience. With more experienced coders, I think the tool could be implemented and made to produce complex narrative formats quite easily even with the current feature base. I think the next most useful step would be to add simple choice based dialogue as a base feature of the code (ie, question and answer) (As an Unreal Engine user it would be extremely improved by becoming an unreal plugin, I want it)	1			1		
11	I feel that the narrative tool worked perfectly for the purpose I personally set out to use it for, so I am unsure how to answer this one.						
12	The base behaviours provided are good, there's some areas I think there is for improvement or that might have a lot of room for user error I'll point out. - this is likely beyond the scope of the project but it would be good if there was an easy way to add new behaviours, e.g. theres a base class thats inherited + maybe an interface that allows you to assign the text and an action to happen so that people can quickly add their own behaviours if they have programming knowledge. - setting new emotions - I know there is a menu for creating new emoticon scriptable objects, but when i create one it doesnt automatically get added somwhere which allows me to use it immeditely, and im not entirely sure where that can be done. If all of the emotions, aliases and images were stored in one scriptable object as a struct it might be easier to use possibly, then again that could make it more complicated to use when theres a lot of emotions. Either way, when i make a new emoticon i dont know	1			1		

	how to link it into the system which might be complicated from a user perspective. - This is beyond your scope, but having an ability to branch dialogue, define options for inputs and assigning variables from inside the text itself (thinking yarn spinner with games like disco elysium where you can pick dialogue options and different dialogue options can call functions which's return effect the next line of text, add EXP, etc.) would make it so designers have a lot of power without needing to go into code.						
13	Add a Unity Tutorial stage/ PDF Detailing the locations and specifics of the tool. Perhaps finding a way to make the code text files more visually clear for future editing or longer projects.	1		1			
14	Some options for interactivity beyond camera pans would be a major improvement, for example adding a teleport to chosen location option would be great to let a player move past a doorway. Similarly, adding the option to make choices in the dialogue and having the NPC say the corresponding response would help make scenes feel more like a chose your own adventure book and less like a stage play. Also, adding "quests" would add a lot, e.g. a character requests you to go collect a book, you go click on the book, when you return the character has new dialogue.		1		1		
15	I think the ability to use the tool to play animations or for it to remember what interactions you've used as variables for things like puzzles and progression would be a really cool next step!		1		1		
		4	7	4	6	1	1


8 respondents (53%) answered tool for this question.

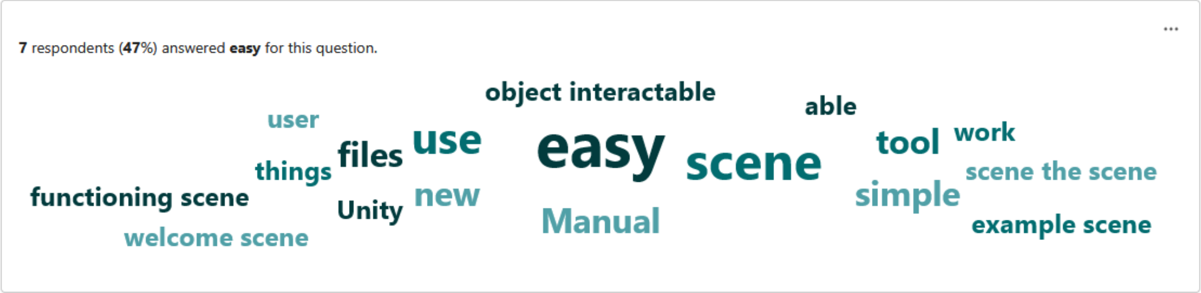
Question 14

In what ways was the tool approachable or easy to understand?

Id	In what ways was the tool approachable or easy to understand?	Supporting non-programmers	Natural Language & Terminology	Asset Reuse	Playtesting and Debugging	Features	Documentation
1	The natural language used to write dialog is very approachable. The barks are a particularly useful feature for approachability. They enhance dialog greatly while being very simple to use!		1			1	
2	It is easy due to the fact that its practically the same as just reading a script. In terms of use it is easy as most things are delt with automatically, just need to attach a the action script.	1	1				
3	The use of the txt files was very seamless and easy to understand. The files within the project are organized and it is easy to find what you need. Since an NPC prefab is already created for you, it is easy to add new characters to the scene. Having a welcome scene also helps introduce the user to the tool more gently.	1		1			1
4	It was easy as I was handling only a couple of text files. I could naturally type out the scene the scene I wanted so I could preplan beforehand what the scene should look like and then directly implement it. The example scene was more than enough to figure out how to construct a scene	1	1		1		1
5	Most functions handled under the hood by the plugin such as emote visibility and camera lerps.	1				1	
6	The Manu-Scriptscriptwriter Testing Manual was absolutely invaluable. The fact that most of the work with the tool was confined to editing .txt files made it particularly approachable, and knowing where exactly I'd be working within Unity (the Design,	1		1			1

	Resources, and Scenes folders within the Assets folder) and what exactly I'd be doing in them made working in an engine I have no experience with much less intimidating.						
7	It was easy to understand in terms of it having the Manual with it which explained how to use it. Once I had read that I found it easy to use. Having the templates provided also helped with understanding how it worked as I could reference what I was doing to something already made to check if I was using it right (e.g. I forgot to put "When the Player talks to Target" with the first NPC I made, looked at the script for Daniel and realised my mistake)				1		1
8	Once I understood everything It was pretty simple to use, It was just a case of learning how it worked that proved slightly hard						1
9	Manual given was very easy to read and any issues were completely user error (not knowing you can't have an empty line of code, not knowing to not use quotation marks, etc). After a bit of trial and error I completely got the hang of it and was able to utilise every provided feature.				1		1
10	The tool was immediately clear and simple to understand, given basic unity experience. The documentation was clear and precise and I was able to install it, get it running and start editing scenes within less than 20 minutes (including unity install time). I think even newcomers to unity and code would be able to approach a simple narrative very easily with this tool.						1
11	I have next to no knowledge of coding or using an engine that						1

	would allow me to create a functioning scene, but through picking through the available items and using the tools laid out in a clear and easy to navigate system, I was able to produce a functional end product with no prior knowledge.						
12	I found it easy to understand how to change dialogue, set new things up, etc. As a system it's very friendly for designers to get into immediately and start changing things.	1	1				
13	Loved that everything was clearly labeled and sectioned within the inspector panel. It made renaming or adding in different .TXT files very intuitive. Along with making a new object interactable very simple.	1		1			
14	My favourite approachability features revolve around anything that keeps the user in the unity application more. The ability to right click and select “make object interactable” is a great example, and really alleviates a new users issues with the Unity UI (I tried to make a text box without it, and struggled to get the scaling correct), whilst also being logically where you’d expect it (I want to do a thing with this object - > right click the object - > select the do thing option - > it now does the thing!). Additionally, the integrated camera options really help avoid a new player needing to fuss with the Unity UI, and help scenes feel much more alive.	1		1		1	
15	Your manual was well written and to the point! Any errors I encountered were my own fault for skipping ahead and refusing to read.						1
		8	4	4	3	3	9



7 respondents (47%) answered easy for this question.

Question 15

In what ways could the tool be improved to be more approachable or easier to understand?

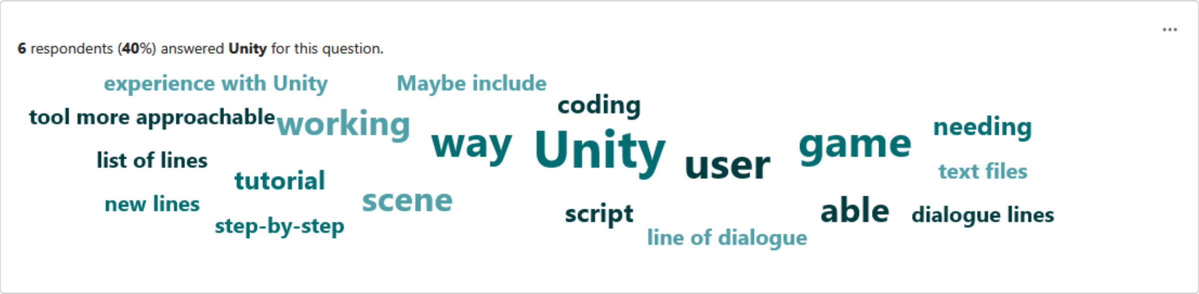
Id	In what ways could the tool be improved to be more approachable or easier to understand?	Supporting non-programmers	Natural Language & Terminology	Asset Reuse	Playtesting and Debugging	Features	Documentation
1	Having to use a separate editor for the text files adds an extra layer of complexity, so being able to edit directly in Unity would make the tool more approachable as a user.	1					
2	Either slightly less rigidity with punctuation/new lines, or some debug error checking to help find the issue/warn users. If an action script accidentally was edited and ended with an empty line, could result in game crashes.		1		1		
3	I believe having a demonstration/tutorial video creating a scene would be more than sufficient in making the tool more approachable as the player can practice using the tool by copying the creator's tutorial. If there is any confusion they are able to rewatch the video as many times as necessary. This would remove needing an in-person demonstration to each writer and save a significant amount of time.						1
4	Maybe providing these commands in a debug log would be nice or some way to output these commands/schema? That way I don’t need to look back						1

	and forth at a manual I can just look at Unity directly?						
5	Restructuring the guide to put a emphasis on case/punctuation sensitive to avoid common user bugs if this little caveat cannot be changed. Object names could be required to be in ALL CAPS so that there is a better differentiation between dialogue lines and variable changes Emotions can be shown for a certain amount of seconds set by the designer before being turned off, or have them be controlled by the designer to disappear upon the next line of dialogue, as emotions can overlap and overstay if the player skips through lines of dialogue on repeat play through. More emotions can be added overtime such as question marks for more intricate cutscenes Maybe some syntax in the game world when the script wont start due to a grammatical issue. Though this is a minor change out of the list. Within the NPC prefabs, being able to spawn and place a camera in that prefab and then giving it a object name such as camera 2 could maybe allow designers to allow for more cinematic camera angles when using the line the camera focuses on Kacy (Camera 2)		1	1	1	1	1
6	Little comes to mind.						
7	N/A - I can't think of a way to make it more approachable/easier to understand. I think it explains itself well in the manual, even						

	just to downloading the Game Engine (Unity). This could be because of my experience with Twine and RenPy though.						
8	Mostly documentation, Explaining what is needed when creating a new txt file for the dialogue and on setting up new scenes. I tried setting up a new scene and ran into some difficulty setting everything up, eventually I did get everything working but this was again thanks to my prior knowledge with Unity and Game Engines						1
9	Can't think of any, the only way it could be more approachable is through being more familiar with the software myself, which is no fault of yours.	1					
10	I genuinely can't think of anything - even before I opened the documentation, it was immediately clear how it worked and applied itself to the provided test scenes. Short of it having a step-by-step tutorial popup on install, I don't think it could be any clearer.						1
11	A user guide could potentially be useful, that appears on initial startup to explain the uses of each tool- although this would only be necessary if the user had never used anything related to coding before.	1					1
12	While you said i shouldnt go into the code for me at least knowing how some things work helps me to work with them, i think having an API reference or explainations of what each component does could be helpful for users (or commenting the code or doing summaries for functions users might touch), along with step by step explaintions of how to set stuff up. While the PDF is good i think						1

	theres a way to include docs like that within unity which could make it more user friendly.						
13	Same as previously stated.	1					1
14	As someone used to coding, the lack of highlighting for text in the code editor makes it harder to distinguish what is the target, and what is the action. Maybe a simple VSCode/notepad++ plugin could help? However, given the aim of simplicity, having to leave the unity interface at all can be intimidating. 1) I think a tutorial in Unity would help, especially for people unfamiliar with Unity's UI. Something along the lines of a. NPC: Hey there, I'm currently talking from script 1, swap to script 2 to hear a joke! To swap, click on me in the hierarchy tab (with the game preview stopped), then click on the script box in the bottom right. There should be an option called NPCScript2. Select that, then preview the application again! 2) Maybe try and avoid the need for text files all together. Whilst the natural language might read nicer, I personally find it more confusing during the writing scenes process. Also, it makes it harder for me to jump into the middle of a file I've written, as the break points are less clear. I don't have any experience with Unity, but I played around with it for a little bit and managed to create a custom script (using C#) that had the following UI. The Dialogue Box and Text was just how I linked to the text box, so ignore that, but the dialogue lines array is the list of lines that an NPC responds with in sequence. Maybe a solution similar, where you add "steps" with a drop down for what	1	1		1		1

	event type it is, what is doing it, and a field to the right for the details would be. (Example mockup, with each section being a dropdown, except dialogue being free text)						
15	I feel like the best way to go about this would be having a proper front end program for the system, though that would be a lot of work lol. I find most people who don't have a lot of programming experience are much happier to use a program with a ui than a tool, yknow? Perfectly accessible and approachable for someone like me. Maybe include details like being able to choose text colour in the main manual also.	1					1
		6	3	1	3	1	10



6 respondents (40%) answered Unity for this question.

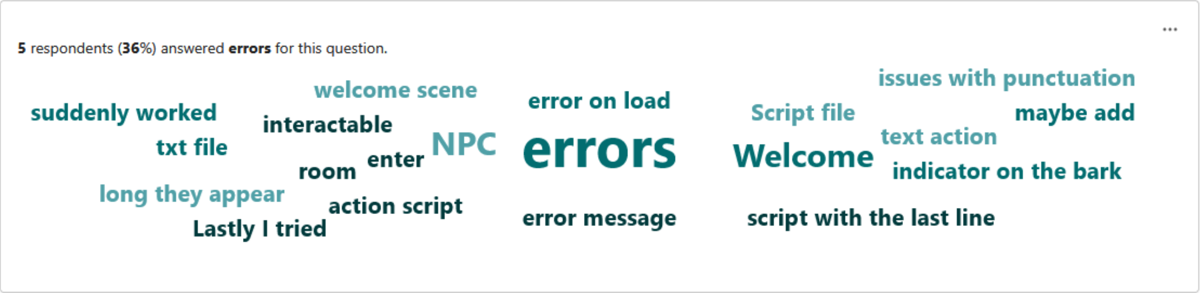
Question 16

Did you encounter any bugs or unexpected behaviour while working with the tool? If so, what happened, and how was it caused?

Id	Did you encounter any bugs or unexpected behaviour while working with the tool? If so, what happened, and how was it caused?	None	In-Game UI	User Error	Empty Lines	Plugin UI	Gameplay
1	I encountered an issue where the name box of the speaking character would be in the wrong position and as a result end up behind the center of the main dialog box. The name box would update correctly, but not be visible during play.		1				
2	Small issues with punctuation or new lines. Example having a text action script with the last line			1	1		

	being empty reports an issue with index being out of bounds.						
3	Adding a space at the end of the txt file blocks the user from witnessing the interaction between an NPC.				1		
4	N/A	1					
5	Punctuation sensitive with missing full stops breaking the script			1			
6	When searching for a present Action Script though the target button beside the "Action Script" property, the search function could not locate the file. When attempted via dragging and dropping the Action Script file into the property instead, this worked fine.					1	
7	No	1					
8	I did encounter some errors but they dissapeared once I understood what i was doing and how things worked			1	1		
9	When trying to interact with NPCs in the "Welcome" room, no text would appear and the error message "Index was outside the bounds of the array" would be at the bottom of the screen. This however was not at all a problem with any other rooms or entities.				1		
10	Visual studio did briefly refuse to open the script - but then I closed and re-opened it and it suddenly worked, so I don't think that's a problem with the tool so much as my computer. Other than that I didn't encounter any errors!	1					
11	No.	1					
12	Not any that you don't know about				1		
13	Had errors trying to load the Manuscript into Unity 6 despite following the tutorial the PDF provided had. I found that some of the objects really did not want to become interactable					1	1

	until I tried it a few times. Along with i kept running off the stage due to some floaty controls.						
14	Barks seem to scale how long they appear with sentence length. This is good for readability, but makes convos desync, and also the initial message appears at the same time rather than separate like a normal convo. The interactable icon for NPC's appeared on top of the Barks, whereas in my opinion they could be hidden whilst the dialogue is progressing (maybe add an option so that pressing enter hurries the barks along, with an enter indicator on the bark implying that). Lastly I tried to run the welcome scene, but had the following error on load, and no dialogue would happen.		1		1		
15	Not while working with it, but for some reason the 'welcome' test room just did NOT work. Like I was spamming enter and every key I could think of and none of them were letting me talk to Matt or the cat. I looked at it and I could not for the life of me figure out why.				1		
		4	2	3	7	2	1



5 respondents (36%) answered errors for this question.