[Draw your reader in with an engaging abstract. It is typically a short summary of the document. When you’re ready to add your content, just click here and start typing.]

CE301 Final Report

Hypertext Game Project: HECC-IT

Project: **Hypertext Game Project**

Student Name: **Rachel Lowe**

Registration Number: **1804170**

Supervisor: **Dr. Richard Bartle**

Second Assessor: **Dr. John O'Hara**

Degree Course: **Computer Games BSc**

# Acknowledgements

I would like to thank Dr Richard Bartle for the support he was able to provide for this project as my supervisor, and for dealing with my general incompetence over the past year or so. Without his guidance, this project would have ended up in a much worse state than it would have been in otherwise.

Thanks to my pastoral and study skills mentors, Michelle Carpenter and Michelle Paul, for putting up with me, and helping me to stay relatively sane and on track with my studies over the past three years.

I would also like to thank David (my brother) for the invaluable support he was able to provide after several components of my desktop PC died; without his help, I would not have been able to recover that computer, which would have made it much harder for me to get this coursework done. Thanks to my parents as well for putting up with me and helping me to stay alive whilst I was at home, and thanks to Betty for the moral support (even if she did eat my ethernet cable in February).

Special thanks to the contributors of *showdown.js* (*A bidirectional Markdown to HTML to Markdown converter written in JavaScript*) [1], as this has been incorporated into the outputs produced by my tool.

Thanks to the academic and technical staff at the university for helping me to gain the necessary background knowledge to work on this project, and for providing the software and other technical support which allowed me to complete this project relatively smoothly. I would also like to thank Dr Dimitri Ognibene, for teaching the CE218 module last year, as the content for that module was particularly useful when creating the tool.

# Abstract

HECC-IT (Hypertext Editing and Creation Code Integrated Toolkit) is a toolkit for authoring hypertext games, designed for indecisive people. Users may write/edit a raw .hecc file themselves, or use the 'OH-HECC' GUI provided by the tool to assist them in the editing process, before converting their .hecc file into a playable hypertext game. Unlike most existing hypertext game authoring tools, which require authors to exclusively write raw source code or exclusively use a GUI, HECC-IT has been designed to allow authors to edit their games however they want (with or without the GUI), without having to go through a convoluted process of converting their drafts into different formats before using the other editing method.

This tool has been used to produce several demonstration games (playable at <https://11belowstudio.itch.io/the-hecc-it-demo>), as well as a more fully-featured game: *Backblast*; a murder mystery where **you** are the victim (playable at <https://11belowstudio.itch.io/backblast>).

HECC-IT itself is available for free on my itch.io page, here: <https://11belowstudio.itch.io/hecc-it>.

(note: Depending on when you are reading this, the latter two links may not yet be publicly viewable. If this is the case, the password to view those pages is ‘301’)

Contents

[Acknowledgements 1](#_Toc70253564)

[Abstract ii](#_Toc70253565)

[List of Symbols 3](#_Toc70253566)

[1. Literature Survey 4](#_Toc70253567)

[1.1: The tool-based research. 4](#_Toc70253568)

[1.2: The literature-based research 6](#_Toc70253569)

[1.3: Reviewing the research into actual hypertext games 7](#_Toc70253570)

[2. Project aims and objectives 9](#_Toc70253571)

[2.1: The specification for the tool 9](#_Toc70253572)

[2.2: The aims for the game I eventually built with the tool 10](#_Toc70253573)

[3. Technical Documentation 11](#_Toc70253574)

[3.1: Intro 11](#_Toc70253575)

[3.2: The full technical documentation discussing the design, development, and testing of every iteration of HECC-IT and the games produced with HECC-IT 11](#_Toc70253576)

[3.3: The summarized version of the technical documentation 11](#_Toc70253577)

[3.3.1: Summary of the pre-challenge week development 11](#_Toc70253578)

[3.3.2: Summary of the challenge week MVP development 12](#_Toc70253579)

[3.3.3: Summary of the term 1 MVP development 12](#_Toc70253580)

[3.3.4: Summary of the term 2 final product development 13](#_Toc70253581)

[3.4: Licensing 14](#_Toc70253582)

[4. Project Planning 15](#_Toc70253583)

[4.1: Intro 15](#_Toc70253584)

[4.2: Usage of Jira 15](#_Toc70253585)

[4.3: Usage of Gitlab 16](#_Toc70253586)

[4.4: Reflection on the project planning 16](#_Toc70253587)

[5. Conclusions 17](#_Toc70253588)

[5.1: Conclusions for HECC-IT (the ‘tool’ component of the project). 17](#_Toc70253589)

[5.2: Conclusions for the ‘game’ component of this project. 18](#_Toc70253590)

[Bibliography 19](#_Toc70253591)

# List of Symbols

* Hypertext Game
  + At the simplest level, a hypertext game is a game where the player is offered choices, which may lead to different sequences of events happening (or not happening). These can be presented as electronic html documents, but they don’t need to be.
* HECC-IT
  + Hypertext Editing and Creation Code Integrated Toolkit
  + This is the hypertext game authoring tool which I created. It will be explained in full throughout the rest of this document.
* HECC
  + Hypertext Editing and Creation Code
  + The intermediate scripting language used by HECC-IT, to store work-in-progress hypertext games (stored in .hecc files).
* OH-HECC
  + Optional Help for HECC
  + A GUI for editing .hecc files, included as part of HECC-IT.
* HECC-UP
  + HECC Ultra Parser
  + The part of HECC-IT responsible for turning .hecc files into playable hypertext games.
* HECCIN’ Game
  + HECC-Infused Nice Game
  + This is the ‘formal’ term for hypertext games produced with HECC-IT (after the .hecc code is parsed). ‘Nice’ in this context is not an indicator of the game’s quality, it’s an indicator of ‘oh nice you produced this hypertext game with HECC-IT’.
* HECCER/heccer.js
  + HECC Engine for Runtime
  + The (generic, pre-written) component of the HECCIN’ Game that works as the ‘engine’ for the game, responsible for all the backend logic.
* HECCED/hecced.js
  + HECC Exported Data
  + The component of the HECCIN’ Game that contains the exported game data (constructed from the .hecc file) produced by HECC-UP.
* HECC-SPECC
  + HECC Super Precise Explanation for Creating Code
  + The specification for the HECC language.
* JRE
  + Java Runtime Environment

# Literature Survey

The creation of HECC-IT was heavily influenced by the research I performed on some existing hypertext game authoring tools, and on the topic of hypertexts in general. In this section of this report, I shall summarize my findings and explain how these findings influenced HECC-IT itself.

## 1.1: The tool-based research.

I started by researching some existing hypertext game authoring tools. I did this because I wanted to gain an overview of the current state-of-the-art for the tools, to find a gap in the market that I could exploit. A full rundown of the tools (and most of the academic literature) I reviewed can be seen in the report on the background reading [2], so, to avoid reiterating those points again, I shall discuss the conclusions drawn from this reading.

The existing tools could be divided into several categories; some of them had a GUI, whilst others were all effectively just scripting languages (some with an IDE, some without an IDE), and most of these tools would require the author to exclusively use a GUI or a raw scripting language throughout the entire development process. In hindsight, this does make sense; it means that development time is not spent split between two ways of doing the same thing, only being concerned with one way of doing a task. There were two (pairs of) tools which did offer the user a choice between using a raw scripting language and a GUI, meaning that an author could, in theory, freely swap between whatever editing method they would currently deem more convenient, but, both of these did this with a caveat.

The *Inklewriter* [3] and *ink* [4] tools allowed some level of flexibility. *Inklewriter* is a server-side, GUI-based, authoring tool, but two of the obvious problems with it are how it’s server-side (meaning that if the company hosting it stops hosting it, this tool will cease to exist), and how authors need to make an account on the website to save/load their work (deterring authors who don’t want to deal with that). The *ink* tool is a client-side, scripting language-based authoring tool, offering the same functionality as *Inklewriter*, minus the GUI. Those two tools are somewhat interoperable, but in a rather inconvenient way. Whilst *Inklewriter* does have an option to export a game as an *.ink* file, this requires the author to manually copy and pastes the exported code into an .*ink* file. Then, to open an *.ink* file in the *Inklewriter* GUI, an author must first export that *.ink* file to JSON within the *.ink* tool, log in to *Inklewriter*, and then manually copy and paste the JSON into an ‘import from JSON’ option. This inconvenient process realistically means that no author would want to bother doing it.

*Twine* [5] and *Twee2* [6] were a bit less inconvenient in this regard. *Twine* is usable either via a web browser, or as a standalone executable, operates entirely client-side, and is a fully-featured GUI-based tool for producing hypertext games, offering plenty of flexibility for authors, even allowing authors to use different ‘formats’ (offering differing syntaxes/levels of functionality) for the games produced with that tool. Additionally, it presents a very helpful overview of games produced with it as networks of connected passages, making it very accessible for casual users. It saves the games in .html files, which can be opened in a web browser to be played, or opened within *Twine* for the actual game to be edited. *Twee2* is advertised as ‘Twine for power users’, effectively working as a pure code-based version of *Twine*, offering all of the options that *Twine* does (and then some), besides the GUI. The *Twee2* utility itself is a command-line program, which reads *.tw2* files, and exports them as .html files, as if those files had been made in *Twine*. Whilst this is more convenient than the *Inklewriter*/*ink* conversion, there are a couple of problems. If you are using Windows, you can’t use *Twee2* to convert from *Twine* format to *Twee2* (although, on macOS and Linux, it is possible). Additionally, the writer still needs to go out of their way to perform this conversion, so it still isn’t entirely convenient.

This exposed a clear gap in the market; a gap for a hypertext game authoring system that allowed users to freely choose if they wanted to edit their games using a GUI or by writing raw source code, so, if a writer wanted to make smaller edits to their game, they don’t need to use the GUI, and if they wanted to make larger changes to the overall structure, they can use the GUI.

Another key finding from this tool-related research was how many of the tools (*Twine* [5], *Inklewriter* [3], *Squiffy* [7], *Undum* [8], *eHyperTool* [9], *ChoiceScript* [10], *Inform* [11], *Quest* [12], *TADS3* [13], and *Ren’Py* [14]) are all capable of exporting games that can be played in .html format (even if, for some of these tools, some server-side legwork may be required by the author). This design choice makes these tools rather appealing from the perspective of a player of the games produced by the tools, as they won’t need to go out of their way to download/install anything beforehand, and can even play these games on their phones. When compared to *Storyspace* [15], which does not have this option, not including this option would very clearly discourage an author from using my tool. Even then, the games exported with some of these tools involve some server-side components, which can potentially discourage some authors from trying to distribute their games if they do not have a server which they can deploy their games on; therefore, I chose to make the games produced with HECC-IT consist of entirely client-side HTML and JavaScript code, for the sake of everyone’s convenience.

In terms of the tools themselves, the only ones which were usable via a web browser themselves were *Twine* [5], *Inklewriter* [3], *eHyperTool* [9], *Squiffy* [7], and *Quest* [12]; of these, only *Inklewriter* and *eHyperTool* could not be used as standalone applications, due to their inherent server-side nature. Therefore, I saw making the tool itself browser-based as not being a necessity, as the existing ‘standard’ for these tools did not extend to making the tools browser-based; in fact, it appeared more like the expectation was for the tools to be downloadable as standalone executable applications. Therefore, I chose to make HECC-IT a standalone application. This then begged the question of ‘what language should I write HECC-IT in?’. Ultimately, I chose to write in in Java. After noticing the operating system-related limitations of *Storyspace* [15] and *Twee2* [6], I knew that I wanted HECC-IT to not be plagued with such arbitrary platform dependence. I could have tried to find a language that has a cross compiler (allowing it to be compiled to multiple operating systems), but I chose Java, because the Java Virtual Machine already is available for multiple operating systems, so, I would only need to build HECC-IT once, and it would inherently run on any operating system. Java is also the language which I feel most confident in using, therefore, it was the obvious choice.

One other discovery I made during this initial research was about the existence of *The* *Treaty of Babel standard for Interactive Fiction Bibliography* [16]. The document itself specified some requirements for interactive fiction development tools that are signatories of this ‘treaty’, such that any works of interactive fiction produced with these tools can all be archived and identified in an appropriate way. The bare minimum requirements are that a tool should allow a title and an author name to be specified for the work produced, assign a unique ‘Interactive Fiction Identifier’ (or IFID) to the work, and for a C routine for the ‘babel’ utility to be contributed which can produce an ‘ifiction’ record for the work in question. The requirements for a tool which falls outside of the scope of the agreement are slightly different (the IFID must be the MD5 hash of the game file), however, during the development of HECC-IT, I chose to assign an IFID for my work as specified for a party which is a signatory to the agreement, and also produce an ifiction file for games produced with HECC-IT during the ‘parsing’ process that reads the input .hecc files and outputs the games. I justified this at the time because, at the time of the initial research, *Twine* [5], which was not currently a signatory to the treaty, assigned IFIDs to games produced with it as if it was a signatory, so I thought that there would be no problems if I were to do the same thing for HECC-IT. However, in January 2021, the tenth revision of *The Treaty of Babel* [17] was published, and *Twine* was now a signatory. Due to this, I am now not sure if the IFID-related components of HECC-IT would be considered ‘allowed’ or not. However, being realistic, I doubt that many people will find out about and use HECC-IT in the first place, so, for the time being, I suppose that I may not need to worry about this until HECC-IT somehow becomes widely used enough to justify asking if HECC-IT can become a signatory to the treaty.

In terms of functionality, the bare minimum supported by all tools was some method of linking between ‘passages’, some method of keeping track of what actions the player had previously taken, and some form of ‘guard conditions’/conditional statements. Some tools (such as *Quest* [12], *Inklewriter* [3], *eHyperTool* [9], and *ChoiceScript* [10]) implemented ‘links’ in the form of ‘pick one of these options’ at the end of each ‘passage’ in the text, always after the main passage content. In these tools, ‘guard conditions’ were implemented such that, depending on certain criteria, each of these ‘links’ would be selectively hidden/shown (with this condition being explicitly for the link), and in turn, could allow greater automatic validation of the game, at the cost of flexibility for the author. Conversely, *Twine* [5] (and the various story formats available for it), *Squiffy* [7], *Storyspace* [15], and *Undum* [8] all have the links defined ‘within’ the content itself. *Storyspace* is the odd one out here, as its links are considered ‘objects’ within the games it produces, and can each be individually configured. However, for the others, ‘guard conditions’ may be implemented in the form of conditional statements, which can be used to conditionally show/hide parts of the ‘content’ of the passages, and, in turn, conditionally show/hide links. This does mean that there is less scope for automatic checking of these ‘guard conditions’, however, due to the additional flexibility it offers the writer in terms of formatting, I chose to use this approach.

## 1.2: The literature-based research

I studied a range of literature for this project as well, both on the topics of hypertext itself, hypertext games, and some on the topic of producing hypertext games.

The article which had the biggest impact on HECC-IT was S. Kitromili, J. Jordan and D. E. Millard’s paper on *What Authors Think about Hypertext Authoring* [18], which, at the time of first reading it, was a very recently-published paper (published in July 2020), and was the ‘Best Student Paper’ at the conference it was presented at, giving it some credibility. It highlighted several key points about the process of authoring hypertexts, from the initial idea to the finished products, and several complaints that some authors had about existing tools. When making HECC-IT, I decided to address the points raised about unclear documentation (by aiming to make it completely clear what HECC-IT could/couldn’t do), debugging tools (by making HECC-UP refuse to produce a game with an obvious error, giving details about it, and also indicating the states of the passages within OH-HECC), some ‘separation of content and behaviour’ (through explicitly-defined ‘comment’ areas, separate from passage contents), and several of the post-lifecycle complaints (making ‘distribution’ easy via the internet, simplifying ‘maintenance’ because only the ‘hecced.js’ file would need updating if the game is updated, offering some avenue for ‘profitability’ due to the control an author can have over the ‘hecced.js’ file with the data, and some level of ‘curatability’ via the *Treaty of Babel*).

From the approach of a literature review, however, I probably should have looked at some other literature on the topic of hypertext game authoring, as that paper was the only one I looked at on this topic. I’m not entirely sure if these findings which I applied to HECC-IT actually will turn out to have been properly useful or not, and, if I had looked at more research on this topic, I may have been able to make even better-informed decisions regarding HECC-IT’s overall design from the perspective of an author. But, I didn’t, so this part of the research is a bit questionable in hindsight.

The rest of the research had fewer obvious problems. On the topic of hypertext itself, I started with E. J. Aarseth’s *Cybertext: Perspectives on Ergodic Literature* (or, at very least, the first chapter of it, as I was unable to legally obtain a copy of the full book). Despite not having access to the full text, it was rather enlightening, explaining that a ‘cybertext’ is a text which, effectively, can be seen as some sort of ‘machine’, where there is a ‘textual feedback loop’, such that ‘the cybertext reader is a player’. It also covered ‘ergodic literature’, where ‘non-trivial effort is required to allow the reader traverse the text’ [19]; this made it very clear that the game I would need to produce with HECC-IT would need to contain these things to qualify as more than just ‘text’.

M. Bernstein’s article *On Hypertext Narrative* was also rather informative, discussing things such as how hypertexts must ‘offer links, but the selection of links must be significant and consequential’, and that ‘the cycle, not the branch, goto, or jump, is the central hypertext structure’ [20]. The former appeared rather obvious, but the latter was rather unexpected, and did start to give me a few ideas about the game I would ultimately produce, even if the game produced didn’t quite include the form of ‘cycles’ discussed by Bernstein in this article. However, the discussion of how ‘using links to vary the *story* is less promising than using links to change the *plot*’ [20] was genuinely rather confusing, and I wasn’t sure how to actually go about doing such a thing. H. K. Rustad’s article on *A Four-sided model for reading hypertext fiction* provided more insight into specific reading methods of hypertexts, from ‘semantization’ (effectively a ‘search for meaning’), ‘exploration’, ‘self-reflection’ (where players ‘play a role’), and ‘absorption’ (where the player is ‘in a condition of confusion’) [21]. These, along with the information about how to invoke these modes of reading, gave me a few ideas about how I could potentially structure the hypertext game I was going to produce, and was re-consulted during the development process, in an atempt to induce the desired effects on the player.

The other pieces of academic literature I read during this research didn’t leave much of an impression that translated into an actual impact on the development of the tool itself, therefore, this aspect of the research was lacking. However, some of these papers did identify a few examples of hypertext games, which I did look into a bit further.

## 1.3: Reviewing the research into actual hypertext games

It would be foolish to attempt making a hypertext game without having first seen some existing hypertext games. The literature I looked at mentioned several existing hypertext games, so, I chose to try looking at them. Unfortunately, I was unable to play several of those existing games, due to them either costing money, being Java Applets (inoperable since 2014), or simply not being publicly available.

Gibbin’s talk on *Telling Tales: Hypertext Writing* gave a rather informative overview of the history of hypertexts, from the start of the genre to nowadays. It discussed hypertext fictions (simply involving the player making choices as the game progresses) as well as ‘ludic hypertexts’ (where some form of external state is involved, which may have some impact on what choices a player may have later on) [22]. The discussion of the latter provided the context for why the tools I had looked at included some forms of conditional logic, and also gave me an explanation for how conditional logic should be used: it turns the ‘reader’ into even more of a ‘player’, delivering on the ‘textual feedback loop’ discussed by Aarseth, and delivers various modes of ‘play’ (all of which map onto the ‘modes’ of reading Hypertext Fiction as proposed by Rustad). It also discussed some of the common tropes for these hypertexts, such as a second-person narrative, generally short ‘lexia’ (sections between choices), but varying levels of ‘constrained’ choices. It also pointed out that academia tends to concentrate on ‘literary hypertext’ (not ‘hypertext fiction’), having ‘non-linear narratives, not non-linear stories’ [22], which did explain why several of the other examples cited by the papers (which ultimately did not have much of an influence on the game I produced) were all varying levels of incomprehensible, compared to these examples. Ultimately, I opted to produce something that more closely resembled a ‘hypertext fiction’ than a ‘literary hypertext’, following the general tropes of the ‘choose your own adventure’ variety of hypertext explained in this talk.

I was able to play one of the literary hypertexts mentioned in some of the previously discussed papers: Pavić’s *The Glass Snail* [23]. The hypertextuality of this manifested itself by the first two ‘sections’ being tellings of the same events, but the reader chooses which ‘perspective’ of the events they want to read first, in turn affecting their perception of the events. There is another ‘choice’ of endings after a ‘middle’ section, again, describing the same events from two perspectives, albeit with both of them ending completely differently. This worked as a practical example to help me understand what Bernstein meant by ‘changing the plot’ [20], however, as this doesn’t really involve the reader as a ‘player’, I opted to not use this approach for my game.

I also played a selection of hypertexts produced with Twine. Anthropy’s *Queers in Love at the End of the World* [24] offered a rather good example of a game which induces the ‘absorptive’ mode of reading (as proposed by Rustad), through the very short time limit which the game introduces via the usage of Twine’s timer functionality, and, despite it simply being text, manages to convey a lot of physicality to the player. This sense of physicality is compounded by the ‘links’ to each passage being naturally embedded into the text, which in turn helps the player to feel more like an active participant than a passive reader. I took a note of this, as it helped me to better understand what the benefits of embedding the ‘links’ into text within a hypertext game actually are.

I also chose to play *Depression Quest* [25] (intentionally not taking the controversy regarding the reception of this game into account), which gave a rather good example of how hypertexts can be used to convey a message to the player, how the intentional restriction of choice can be used in a game, and served as an example of how to go about adapting the experience based on earlier choices made by the player (via an internal state recorded by the game). It also got me thinking about making a ‘serious game’ of sorts with this tool (which I eventually did attempt, even if that particular idea ended up unfinished), but overall, this was rather helpful in terms of working out what I can do.

*the uncle who works at nintendo* [26] contained a rather nice example of some form of ‘dialogue’ system, a ‘timer’ which increments as the player performs certain actions, various endings (keeping a record of what endings have been reached), ambient audio, and effectively abuses the Twine engine at points to install a sense of genuine fear in the player. This did show some of the unexpected advantages to having a hypertext game engine which shows errors at runtime instead of compile time, and provided a further example of what could be done via keeping track of an internal state. The sheer complexity of this game was a bit too much for me to consider imitating, however, the way that options are naturally hidden/shown in each playthrough according to the player’s choices (such that any choices which the player would not have ‘learned’ about so far, or have already ‘chosen’, are hidden, without anything to indicate that choices could have been there) made a lot of sense, and I chose to use that approach in the final ‘game’ I produced with HECC-IT.

Finally, I informally replayed *Zero Time Dilemma* [27] during the winter break, not with the express intent of doing this as research, but just because I simply wanted to play it again, however, this did still have some implications on the final game I produced. This game isn’t a ‘hypertext’ in the literal sense (instead being an ‘escape-the-room’ puzzle game), yet it has a very much hypertextual, non-linear overall plot structure. I knew that I would not be able to make a tool which could make games exactly like this, due to the inherent complexity of this game and the fact that the presentation of this game only makes sense in its own context. However, the plot is presented from several perspectives, several of the ‘branches’ in the plot differ due to the actions of the other characters (not just the player), and acknowledges that the player may know things from other ‘branches’, offering the player several opportunities to apply this knowledge. The latter two points ultimately heavily influenced the game I produced with HECC-IT, albeit by this point in development, HECC-IT had been mostly developed.

In hindsight, I still could have done some more research on individual hypertext games, especially regarding any more recent (post-2016) examples of hypertext games. Additionally, I didn’t actually work out what the current state of the market in terms of hypertext games is, so I’m genuinely not sure what market I should have tried to target with my game, or what specific features the games which my tool will be producing should incorporate to fit into a gap in the market, if there actually is a market which still exists. However, with the amount of free hypertext games which are playable online on websites such as itch.io, the game I shall produce will be published online as freeware, as, if I were to charge money for it, this barrier to entry would discourage people from wanting to play it, which, if the market turns out to be oversaturated/not present, could end up being rather detrimental to the game’s chances of being played.

# Project aims and objectives

The initial proposal for this project was to produce a hypertext game creation tool, and to use said tool to produce a hypertext game. Therefore, as there were two ‘parts’ to this project, I had two ‘specifications’ in mind when working on this. The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this section of this document are to be interpreted as described in RFC 2119 [28].

## 2.1: The specification for the tool

The requirements I set myself for the HECC-IT tool can be seen below. The tool was effectively built with the intent of having the HECC language as the ‘core’ of it, with a focus on producing a .hecc file and producing games as specified in the .hecc file.

1. MUST create an ‘engine’/’template’ for hypertext games, to be output by the tool. (The HECCIN’ Game)
   1. These MUST be in html/JavaScript format, they must be playable via a web browser, and must be 100% client-side code.
   2. These MUST consist of ‘passages’, holding textual content, with clickable ‘links’ which, when clicked, MUST replace the currently visible ‘passage’ with the ‘passage’ which the ‘link’ is ‘linked’ to.
   3. These MUST store a ‘history’ of the player’s visited ‘passages’ within a stack (of ‘gamestates’), and have a ‘back’ button which, when clicked, MUST load the previously visited ‘passage’ from the stack, popping the passage which the player was on when they pressed the ‘back’ button off the stack. Additionally, upon navigating to a new passage, that passage must be added to the stack. The user MUST NOT be allowed to go ‘back’ if there are no prior passages.
   4. SHOULD contain support for conditional statements, which MUST be implemented in the form of showing/hiding content depending on certain conditions. It MUST be possible for visited/unvisited passages to be usable as conditions, and it MAY allow other forms of conditions, as well as boolean logic, to be usable in the conditions.
   5. SHOULD include support for variables, and the current state of these variables when accessing a passage MUST be stored within the stack of ‘gamestates’, and when the player presses the ‘back’ button to return to an earlier passage, the earlier state of the variables MUST be loaded, discarding the current state.
2. MUST specify a high-level scripting language, which can be used to ‘declare’ game content for a hypertext game using the previously mentioned template (the HECC language, stored in .hecc files)
   1. MUST be possible to declare passages, specify passage content, and declare links between passages within the passage content.
   2. MUST be possible to declare metadata to indicate which passage is the ‘start’ passage (in the sense of ‘start from here’).
   3. SHOULD allow metadata about the game to be declared, such as the title of the game, and the author’s name.
   4. SHOULD require this metadata to be defined in such a way that complies with the Treaty of Babel.
   5. SHOULD allow metadata for individual passages to be declared.
   6. SHOULD allow ‘comments’ to be declared, but these MUST be declared in a way which makes them distinct from passage content.
   7. MAY allow authors to use markdown formatting within the passage content.
3. MUST create a utility program which can read files containing content written in this scripting language and export a playable hypertext game containing the declared game data. (HECC-UP).
   1. MUST allow a user to pick what ‘input’ .hecc file they want to export, and an ‘output’ folder to export the game to.
   2. MUST read the .hecc file and export the games exactly as they are declared.
   3. If the file does not contain an explicit declaration for a field of metadata, it MUST use a default value for the undefined metadata field. In the case of the ‘start’ reference, this default value MUST be ‘Start’ unless a user explicitly defines it differently. Any other fields MAY be defined with default values as specified by the Treaty of Babel (if relevant), and the user MUST be given instructions for explicitly defining these other fields.
   4. MUST NOT export a game if the input file is invalid (no content, no passages, passage contains empty content, a link to a passage which doesn’t exist is detected, passages share names, the referenced start passage does not exist, or any other problems with the file itself are detected). In these cases, it MUST tell the user what the error is.
   5. MUST (will) be written in Java.
   6. SHOULD have a GUI (Graphical User Interface).
   7. MAY export ‘.ifiction’ metadata for the game, with a structure as specified by the Treaty of Babel.
4. SHOULD create a program which allows for the editing of .hecc files written in the scripting language via a GUI. (OH-HECC)
   1. A user MUST NOT be required to use this program in order to write their games.
   2. It MUST allow a user to create a new .hecc file, or to open an existing one for editing.
   3. If the given .hecc file is valid, it MUST read it as-is. If there are any problems with the .hecc file (missing metadata, references to undefined passages, duplicate passage names, etc.), it MAY ignore those errors, but it SHOULD attempt to automatically resolve those errors.
   4. It MUST graphically represent the structure of the game as a ‘network’/’directed graph’ of connected ‘passages’, all of which MUST be labelled with their name. It MUST be clear which ‘passages’ contain links to other passages; this SHOULD be in some form of arrow pointing from the passage with the link to passage the link points to.
   5. It MUST allow a user to edit the data held within the individual ‘passages’ and the ‘metadata’ for the game. It MUST NOT allow a user to enter any input which would render the .hecc file invalid (as specified in requirement 3.d), although an empty passage MAY be tolerated. If the change requires the structure of the .hecc file to change to retain validity (adding more passages, editing contents of other passages, etc.), it MUST perform those macro-level changes.
   6. It MUST allow ‘passages’ to be added/deleted from the ‘network’
   7. It MUST allow a user to save their work, as a .hecc file. The .hecc code it produces MUST be valid, and readable by HECC-UP and OH-HECC itself. This MAY overwrite the .hecc file they first opened. If there is an IO-related problem with saving the .hecc file itself, the user SHOULD be presented with their raw .hecc code, in such a way that allows them to manually copy and paste it into a new file.
   8. This program MUST (will) be written in Java.
   9. HECC-UP and OH-HECC MAY be packaged into a single executable, such that a user MUST be allowed to choose to open a .hecc file directly within HECC-UP or OH-HECC, or open HECC-UP from OH-HECC (after having saved it, and MAY be dependent on a validation check within OH-HECC).

## 2.2: The aims for the game I eventually built with the tool

I didn’t set myself any sort of formal requirements like those I gave myself for the HECC-IT tool. The eventual implications of this are discussed later, within the technical documentation section of this report, specifically, within section 3.2.4.3.3 (as part of the documentation on Gitlab) and summarized within section 3.3.4 (within in this document). However, I did have these informal requirements:

1. It MUST contain multiple connected passages, with a start, and MUST contain multiple endings.
2. It SHOULD demonstrate as much of the functionality of the HECCIN’ Game and the .hecc format as possible, and do this in a way which MUST make sense in context of itself as a game.
3. It MUST be free of any errors, whether they may be to do with the internal logic of the game as a piece of software, or in the sense of improper content displayed to the player.
4. It SHOULD offer meaningful choices to the player. These choices MAY be presented in a way that makes it obvious that they will be meaningful, but these meaningful choices MUST change the range of choices offered to the player later on/the ‘route’ that the player is taking through the game such that the potential range of experiences is different to what they could have been if the player had picked a different option.

# Technical Documentation

## 3.1: Intro

The technical documentation for this project is held on Gitlab, and can be seen, in full, [here](https://cseegit.essex.ac.uk/ce301_2020/ce301_lowe_richard_m), on the project’s Gitlab repository [29].

This documentation is in several parts.

Full JavaDoc-style documentation, explaining every single class, method, and class attribute, is held within the ‘/JavaDocs.zip’ folder on the repository. You will need to download it, unzip it, and open it in a web browser in order to peruse it. I tried to work out how to put it on ‘Gitlab Pages’, so it would be easier to browse, but it doesn’t look like the Gitlab instance provided by the university has this configured, meaning that this was not an option. I did consider putting it in a restricted itch.io page, but the JavaDoc-format documentation doesn’t lend itself to being embedded into another html page. Therefore, this approach for showing it was the best out of several equally undesirable choices.

Shorter package-level explanations of the general purpose of each class in the codebase are present in the form of ‘README.md’ files, which can be seen within the ‘/src’ folder of this project’s repository, and within the various subfolders of that folder (albeit the ‘src/assets’ folder only has a single ‘README’, which covers the contents of its subfolders).

Finally, the bulk of the technical documentation, covering the design, development, and testing of the overall deliverable can be seen in the ‘/Technical Documentation’ folder on this project’s gitlab repository, presented as a series of markdown documents. Yes, the numbering for the sections of those documents start anew from ‘1’, so, for the sake of not having ridiculously long prefixes in front of every single section name, consider them to have been prefaced with a ‘3.2.’ in front of the numbers that are there.

## 3.2: The full technical documentation discussing the design, development, and testing of every iteration of HECC-IT and the games produced with HECC-IT

Once again, is in the ‘/Technical Documentation’ folder of the repository, here: <https://cseegit.essex.ac.uk/ce301_2020/ce301_lowe_richard_m/-/tree/master/Technical%20Documentation> [29]

## 3.3: The summarized version of the technical documentation

### 3.3.1: Summary of the pre-challenge week development

Refer to section 3.2.1 (on [Gitlab](https://cseegit.essex.ac.uk/ce301_2020/ce301_lowe_richard_m/-/blob/master/Technical%20Documentation/Designing%20HECC-IT.md)) for the full details.

The first part of HECC-IT to be produced was a rather crude prototype of the ‘HECCIN’ Game’, consisting of a prewritten ‘heccer.js’,’hecced.js’, and ‘index.html’ file. The ‘game’ held within the ‘hecced.js’ file was more of an incomprehensible tech demo instead of a proper game, lovingly referred to as the ‘HeccSample’. This was quickly followed by a rudimentary draft for the ‘HECC-SPECC’, and a retrospectively produced ‘HeccSample.hecc’ file. I then created a crude Java program which was capable of reading the aforementioned ‘HeccSample.hecc’ file and could use that input to create and export a copy of the ‘HECCIN’ Game’, just like the example. Before the academic year started, I made a few improvements to this crude parser, by making it more object-oriented (with ‘Passage’ objects and a ‘Metadata’ object), and making it throw some exceptions if it was given invalid inputs (which it would then proceed to complain about in the console). This parser was when I first had the idea of storing the passage objects like an adjacency list graph within HECC-IT; the ‘passages’ were stored in a Map<String, Passage>, with the passage names being used as the key, and, within the parser, the ‘passages’ had a set of the named passages they are ‘linked’ to, as an indirect ‘pointer’ of sorts to the appropriate entry of this map.

### 3.3.2: Summary of the challenge week MVP development

Once again, refer to section 3.2.2 (on [Gitlab](https://cseegit.essex.ac.uk/ce301_2020/ce301_lowe_richard_m/-/blob/master/Technical%20Documentation/Development%20of%20the%20Challenge%20Week%20MVP.md)) for full details.

I started Challenge Week by making some improvements to this parser to make it less terrible (improving the overall architecture of it so it was less crudely held together), making the outputs look somewhat presentable (removing the garish colours which were added for the sake of making it easier to work out the sizes of each element when I was trying to set up the CSS), and, most importantly, adding a GUI for the parser. This GUI was the final piece in the puzzle which turned this ramshackle, nigh-useless single-purpose Java script into an actually useful utility: HECC-UP. I also considered trying to add in markdown formatting during Challenge Week, however, after a day of very little progress, and a realization about just how much work would actually be needed to make a markdown parser, this was put on the backburner. I also made another HECCIN’ Game whilst I was thinking about it, at the very end of Challenge Week. This game, ‘A Conversation’, was framed as a ‘conversation’ between the player and their internal monologue, and was also an intentionally poorly disguised advert for HECC-IT.

With HECC-UP working, and able to convert .hecc files into playable HECCIN’ Games, this, strictly speaking, meant that I had a Minimum Viable Product before the end of the 2nd week of the academic year. Yes, it might have had more emphasis on the ‘Minimum’ than the ‘Viable’, but the point was that it could be used to create simple hypertext games much easier than doing it manually. Theoretically, I could have stopped here. But I didn’t.

### 3.3.3: Summary of the term 1 MVP development

Refer to section 3.2.3 (on [Gitlab](https://cseegit.essex.ac.uk/ce301_2020/ce301_lowe_richard_m/-/blob/master/Technical%20Documentation/Development%20of%20the%20Term%201%20MVP.md)) for full details.

During the remainder of Term 1, I focused my efforts on creating the ‘OH-HECC’ editing GUI. The first step, however, was to perform a bit more refactoring of the HECC-UP data classes, to essentially clean them up, with the intent to re-use those same data classes within OH-HECC (this re-use eventually didn’t happen). I chose to start making OH-HECC by making ‘editable’ passage/metadata classes, as, if OH-HECC couldn’t edit the individual components of a game, it would be completely unfit for purpose. It was around this time that I chose to use UUIDs as the ‘keys’ for the OH-HECC passage map instead, as this could ensure that every single ‘passage’ had a constant immutable identifier, which, in turn, made it easier to implement the ‘renaming’ operation for passages. These editable metadata/passage objects also had some unit tests created for them, to ensure that the getters/setters worked correctly, refusing invalid inputs but accepting valid inputs. I also created some ‘component editor windows’ for the editable passages/metadata, to make it possible for these ‘editable’ data classes to be edited. Later, I chose to encapsulate all of these ‘data’ objects within a ‘GameDataObject’, however, I will admit that, in practice, this encapsulation was far from perfect.

I then proceeded to start writing the OH-HECC GUI. I chose to use a model-view-controller architecture, like the one I had used for my CE218 coursework and several other games I made as personal projects over the summer. This GUI would be similar in nature to *Twine*’s GUI, showing the game as a network of linked passages, but would not be a direct copy of it. I started by making the classes that represent the ‘objects’ that would be visible in this ‘model’, before then creating the ‘model’ itself. Initially, I was considering reusing the same update loop-based logic as well, but, upon realizing that an update loop would be pointless when there’s no need for anything to happen without user input, this internal logic was changed to only use the event handling functionality of Java Swing. I eventually created a parser which could read a .hecc file into the data structure used by OH-HECC (recycling parts of the HECC-UP parser), a save routine (to turn the OH-HECC data into a .hecc file), as well as a ‘main menu’ allowing a user to create a new .hecc file, or open an existing .hecc file in OH-HECC.

At the end of term 1, OH-HECC was usable. It could be used to save, load, and edit .hecc files, representing the game as a network of connected ‘passages’. Authors could edit individual passages by clicking them, edit the game metadata by clicking a button in the GUI (with the editing dialog being on a new window), drag the ‘passages’ around by holding the left mouse button, ‘scroll’ the viewport with their arrow keys, and could automatically handle the creation/deletion/renaming of passages (updating the overall data structure as appropriate). HECC-UP itself still worked and was compatible with the outputs produced by OH-HECC. At this stage of development, they still used different ‘main’ methods, so HECC-UP and OH-HECC were packaged into separate .jar files. However, they worked.

Now that I had a GUI for editing .hecc files (which users didn’t need to use if they didn’t want to), and a utility for converting .hecc files into playable games, I, once again, could have stopped here. But, again, I didn’t.

### 3.3.4: Summary of the term 2 final product development

Refer to section 3.2.4 (on [Gitlab](https://cseegit.essex.ac.uk/ce301_2020/ce301_lowe_richard_m/-/blob/master/Technical%20Documentation/Final%20version%20development.md)) for full details.

I started Term 2 by working on the HECCIN’ Game a bit further. I decided to use a premade Markdown implementation within the HECCIN’ Game, due to the aforementioned issues with making a homemade implementation. After looking into some implementations, I found ‘*Showdown*’ [1]. I ultimately chose to use it, due to it having a permissive license (MIT), and having an ‘extensions’ feature, allowing me to declare custom ‘rules’ for the parser; I opted to move the ‘link’ formatting code from HECC-UP into a custom ‘extension’ for the *Showdown* instance incorporated into the HECCIN’ Game, as a test run to see if it worked as expected: it did. I then had the idea of abusing these ‘extensions’ to implement conditional formatting into the game. I created and unit-tested some code which could be used to check if a given passage/passage with a given tag had been visited, as well as a class which would be usable as a dedicated conditional statement checker (which I named the ‘checcer’), via the magic of eval abuse. Then, I added a new ‘rule’ to that custom extension, as a dedicated handler for the conditional formatting logic (formulating a syntax for ‘if’/’else’ statements for the .hecc language in the meantime), and unit-tested that to see if it worked as intended. It did. I then cleaned up the CSS for the ‘index’ page for the games, and, later on, I added some functionality for the game’s metadata to be recorded on within this ‘index’ page and within the ‘hecced.js’ file.

I made some major improvements to the rest of HECC-IT. I refactored HECC-UP’s file-related code a bit to make it more compatible with OH-HECC, changed the main menu of OH-HECC to include an option to open HECC-UP after selecting a .hecc file (thereby making it a main menu for ‘HECC-IT’), and added an option in OH-HECC to open HECC-UP (after some newly-added automatic validation checking code ‘passes’, letting the author know about the obvious error otherwise), which meant that I could put HECC-UP and OH-HECC into a single ‘HECC-IT.jar’ executable. Some ‘idiotproofing’ was added to OH-HECC, making it nigh-impossible to input something which could break the entire structure of the output .hecc file, as well as some code which could automatically mitigate any obvious major problems within the .hecc file read by OH-HECC (such as renaming passages which had the same name, adding passages which were referenced but didn’t exist, and automatically giving a position to any passages that didn’t have a defined position). I also made it possible for an author to ‘drag’ the viewport within OH-HECC by holding the right mouse button and moving the mouse, so they aren’t restricted to the fixed increment scrolling provided by the keyboard, and I also added in some functionality to constrain the viewport relative to the positions of the passage objects (so the author can’t get completely lost).

There were several other backend-related improvements, such as more abstraction via additional interfaces leading to less overhead at runtime, writing some proper unit tests for the parsing/game data classes, unit-testing several of the newly introduced classes, deprecating classes which were ultimately redundant, declaring every variable and attribute which was never updated as ‘final’, and several other improvements, both documented and made on the spur of the moment. This cut down on some of the runtime code bloat within HECC-IT, and overall, made the overall codebase somewhat more coherent, even if there are a few minor code style problems that could use some fixing, although there isn’t enough time left before the deadline for me to fix those problems (and document those fixes) now.

With HECC-IT done, I started to write a hypertext game with it. I wrote *Countdown* in January, as a tech demo of sorts for the ‘conditional’ functionality of the HECCIN’ Game. It was still mostly incoherent, but it worked as a demonstration of these conditional statements, remembering prior choices made by the player. I then started work on *Backblast*. Like *Countdown*, it could ‘remember’ certain choices made by the player earlier on in the game, which, later on, could show/hide options for the player as appropriate (which, in turn, changes which ending branches will be accessible to the player after the main ‘loop’ area of the game concludes). However, I will admit that the overall design process for *Backblast* had many problems, it barely qualifies as a ‘game’, most of the choices presented to the player were utterly meaningless, and the finished product genuinely suffers as a result. If I had properly designed *Backblast* before writing it, it may not have ended up as badly as it did.

However, the final product for the whole HECC-IT suite is still functionally complete and fulfils the criteria I set out at the start. Authors can write hypertext games in a raw scripting language, they have the option (not obligation) to use a GUI to edit these files, and they can compile this scripting language into playable hypertext games. The games themselves are playable via a browser, using entirely client-side JavaScript code to run them, in turn meaning that minimal additional effort is needed to host these games on a server or to download and run them, and are even playable via a smartphone. It’s usable (at least, in theory) on any desktop computer which has the JRE for Java 8 installed. HECC-IT has also been demonstrably used to create several hypertext games of varying levels of complexity throughout its development cycle.

The final deliverable of HECC-IT consists of ‘HECC-IT.jar’, with a copy of the ‘HECC-SPECC’ (with instructions for the .hecc language and for using HECC-IT), a ‘README’ file (with more general information), and a couple of example .hecc files (those files being ‘A Conversation’ and an updated version of ‘HeccSample’, with some conditional statements included in it).

## 3.4: Licensing

HECC-IT (along with the copies of heccer.js produced by HECC-UP) is distributed under the terms of the Mozilla Public License v. 2.0 [30] (MPL). This license was chosen because it’s a copyleft, free (as in ‘freedom’) software license, which isn’t overly restrictive on other people who may want to contribute to HECC-IT/incorporate HECC-IT into their own works in the future. The main requirement for using MPL-licensed works is that any modifications made to MPL-licensed components must themselves be made publicly available, for free (again, as in ‘freedom’), licensed under the MPL. If someone wishes to use an MPL-licensed work (such as HECC-IT) within some proprietary work, they can, if the components based on the MPL-licensed work are, again, made freely available under the MPL. This means that HECC-IT will remain as free software, yet not forcing any works which merely use part of HECC-IT to be wholly bound by any restrictions related solely to the HECC-IT component.

The ‘index.html’ files output by HECC-UP are licensed under the terms of the MIT license [31] (the same license as the *Showdown* markdown parser is licensed under), giving the author freedom to do whatever they want with those files. Any .hecc files produced by OH-HECC, along with the ‘heccer.js’ file exported with HECC-UP, are considered to belong to the author of the HECCIN’ Game which those files contain the data of, and the author is free to distribute those files (which hold the game they wrote) under whatever license they want to distribute them under. This licensing choice (and the choice to make it explicit) was made to give any prospective author peace of mind that anything they produce with HECC-IT is officially theirs, so they can do whatever they want with it. If they want to make it freeware, they can. If they want people to pay to play their HECCIN’ Game, they can do that. However, any modifications they make to the HECC-IT tool, or the ‘heccer.js’ engine, modifications which the vast majority of writers may never make (as the source code for these components do not hold any data specific to a particular HECCIN’ Game), still need to be made publicly available, for the collective benefit of everyone who has used/may want to use HECC-IT.

# Project Planning

## 4.1: Intro

The development of this project was performed using a Kanban-style agile methodology. I also used Jira and Gitlab being throughout the development cycle, to log what work had been done, record what work I had planned to do, and to ensure that I had a backup of my progress. I will admit that the level of care I used with each of these tools did vary considerably over the course of development, and that I did neglect to plan certain aspects of this project in advance, with many instances of me just ‘jumping in’ to do something without having it properly recorded happening over the development cycle.

## 4.2: Usage of Jira

For this project, I had access to a Kanban board on the university’s instance of Jira, which I used to plan out most of my work and log all my work. I will admit that I didn’t utilize Jira to the fullest, and, especially during the second term, my usage of Jira was rather inadequate. I didn’t utilize ‘story points’ or the ‘estimated time’ on any issues, mostly because I wasn’t sure how long any of the issues I raised on Jira would actually take to complete, and I didn’t want to waste time agonizing over predictions of complexity at the expense of time I could have spent actually working on the aforementioned issues. I also neglected to use any ‘user stories’ on Jira (only using a grand total of one), with the vast majority of the items logged on Jira simply being ‘issues’, with the occasional epic/bug thrown in there every once in a while. Additionally, there were many occasions where I neglected to add subtasks to Jira, did not mark subtasks as ‘done’ when I should have been moving between subtasks, and logged work which was done on the spur of the moment as part of the work on the other task I was in the middle of working on at the time.

Despite this, there were still some parts of my Jira usage which have been rather good. I have been fastidiously logging all the work I have done on Jira, linking to the appropriate git commit(s) containing the progress I have made. When certain unexpected problems did show up during development (such as when my dog ate my ethernet cable, or when my desktop’s SSD died), I logged them on Jira as soon as I could, and documented how those problems were addressed as well. Additionally, I have been using epics and releases, to organize the development of HECC-IT into each stage, using a release and an epic for the ‘preparation’ work performed over the summer term, another one for the challenge week work, and another one for the MVP work performed during Term 1. As soon as this report is done (and this has been logged on Jira), I intend to create another release on Jira for the ‘final deliverable’, allowing me to file this out of the way with all of the other completed stages of development. Additionally, I have still been breaking down some tasks into subtasks, allowing progress made on them to be done in a somewhat more granular manner.

## 4.3: Usage of Gitlab

In comparison, my usage of Gitlab has been a bit less terrible. Since I started work on this project, I have been trying to commit relatively frequently, and having at least one commit on Gitlab for every instance of logged work on Jira (as of the time of writing this, my repository has 245 commits on it), with 4 branches as of the time of writing (albeit with all but the master branch being used as ‘archives’ of the repository as of the start/end of challenge week and as of the end of term 1). I have 3 releases on the Gitlab repository; one for the challenge week iteration of HECC-IT, another for the term 1 iteration of HECC-IT, and one final one for the ‘final’ version of HECC-IT. Additionally, all of the work that I have done is on my Gitlab repository. This was rather helpful, as it meant I still had a backup of my work available in spite of the various hardware failures I had to endure during the development process of HECC-IT, and, from the commit messages, it’s also possible to get an overview of what work I had done.

Could my usage of Gitlab have been improved? Yes. I didn’t commit my work quite as frequently as I should have (usually making a bulk commit after finishing working on a certain task, instead of committing more granular changes), which in turn lead to several commits being much larger than may be considered reasonable. I didn’t actively use branches for my project, with most of my work being committed directly to the ‘master’ branch. I could try to defend this with an excuse of ‘I’m the only person working on this project’, however, it still could have resulted in broken code being put onto the master branch. On that note, I also failed to utilize any of the continuous integration/delivery tools available via Gitlab, which I could have used to automate testing of my codebase, and could have potentially been used to present the Javadoc-based documentation in a somewhat more coherent manner. Finally, my commit messages could have been made a bit more descriptive, with some of them being too vague to be of any use to anyone.

## 4.4: Reflection on the project planning

From an organizational standpoint, this entire project was a complete mess from start to finish. I barely spent any time trying to properly plan things out before doing them, I neglected to engage in test-driven development (with the vast majority of the unit tests I did implement being performed in hindsight, as regression tests, instead of engaging in test-driven development), and I didn’t even consider getting any feedback about this tool from any potential end users. It’s honestly a miracle that HECC-IT even made it to the MVP stage, let alone making it to the state of being a half-decent fully usable program that it currently is in. I don’t know if I’m being overly critical of myself here, but I cannot, in good faith, label any of the planning-related aspects of this project as being ‘a success’.

Once again, these failures most blatantly manifested themselves within *Backblast*, where my lack of a proper vision beyond a general vague idea singlehandedly squandered any potential which it may have had, reducing it into an overly-railroaded, nigh-incomprehensible disappointment of a ‘game’ which ended up failing to deliver on any sort of satisfying conclusion, due to a lack of planning behind it. Contrast this to *A Conversation*, which, despite being thrown together from scratch within a span of 81 minutes, had a clear vision behind it (being a poorly-disguised advert for HECC-IT, presented as a dialogue between the player and their internal monologue), does not overstay its welcome, and doesn’t end up completely disappointing a player. Yes, that hypertext didn’t have much effort put into its design or its implementation either, but it delivered the experience which I intended for it to deliver.

Next time I do a project like this, I will need to put a lot more effort into the overall design of the software, and not just jump in blindly into the programming without much forethought. Additionally, I will need to put some effort into learning how to use the various tools which I had at my disposal but chose not to use, and to use those tools effectively. I got lucky this time, but I doubt that this success will be repeatable unless I make significant improvements to my planning processes in the future.

# Conclusions

## 5.1: Conclusions for HECC-IT (the ‘tool’ component of the project).

Simply looking at the specification of HECC-IT (section 2.1), this aspect of this project appears to have mostly been a success. The only element of the HECC-IT specification which was ultimately left unimplemented was point 1.e (variables), which was a ‘SHOULD’ requirement. Whilst HECC-IT is still very much functional even without that one element of the specification being implemented, it does mean that HECC-IT has a deficiency which, when compared to most of the existing tools, does make it a somewhat less appealing option. However, the vast majority of those other tools were produced as a collaborative effort from many people, over several years of development, whilst HECC-IT was a solo effort, written mostly from scratch (besides a few utility classes and a prewritten markdown implementation) over the span of 8 months, produced in tandem with a selection of hypertext games (each of which reflecting the capabilities of the HECC-IT tool at the time of writing).

I am under no false impressions regarding the chances (or, rather, the lack of) which HECC-IT has for gaining any mass appeal. If a writer wants to create a hypertext game which has all the bells and whistles, in most situations, a tool such as *Twine* or *Squiffy* will ultimately be a much better choice than HECC-IT. However, if a writer does not care for the additional functionality offered by every other tool, and wants to produce simpler hypertext games, whilst not be restricted to exclusively using a GUI/raw code for the entire development process, HECC-IT will get the job done for them.

Asides from those issues, the development of HECC-IT has still been, overall, successful. This does not change the fact that the development process behind it needed significant improvements, as discussed at length in section 4.4. I should have used test-driven development, I should have documented changes in a more formal manner, and I should have sought end user feedback. But, simply looking at the product, I would say that it is not entirely terrible.

I have produced, from scratch, the ‘HECCIN’ Game’ system, which can be used to play hypertext games in a web browser, which records an internal state, and this ‘state’ may be used, at runtime, to change the options available to a player. Whilst this engine does include an existing markdown implementation, as proven by the MVP iterations of this project, it is effectively just a cherry on top of the self-made internal mechanisms.

I have defined a standard ‘language’ (HECC) which can be used to declare the content of a hypertext game. It may be similar in syntax to existing ‘languages’ for this task, but this one comes with several features not present in other such ‘languages’, such as truly explicit metadata declarations, and properly separates game content from any ‘comments’ which the author does not intend to include in the game content.

I have produced, from the ground up, the HECC-UP utility, which converts valid .hecc files into playable HECCIN’ Games. If the given .hecc file is invalid, it will inform the user if there is a problem with the given .hecc file, and states what the problem is.

I have produced, from scratch, the OH-HECC utility, which allows users to create/edit existing .hecc files and save their work as .hecc code within these .hecc files, and keeps a backup of the most recent version of the .hecc file that contained ‘valid’ code. Whilst some elements of the architecture were re-used from some of my prior projects even those legacy components were built, from scratch, by me. It can read a .hecc file, fix any major problems detected in the aforementioned .hecc file, presents the hecc file as an interactive directed graph, and outputs a .hecc file. It can be used to make changes to the overall structure of the .hecc file, it can be used to make changes to individual ‘passages’ or the metadata of the .hecc file, and it will not permit a writer to (un)intentionally cause the output .hecc file to be rendered invalid.

## 5.2: Conclusions for the ‘game’ component of this project.

I will admit that the ‘game’ aspect of this project has been a bit less successful. Whilst the project proposal simply said to ‘produce a hypertext game’, one could argue that I have produced four times as many hypertexts as specified in the proposal, yet I do not feel as if I have made any which I can, with confidence, call a ‘game’.

*HeccSample* (see section 3.2.1.4.1) fulfilled its purpose of being an example output for the HECC-IT software suite, but it’s simply a tech demo, nothing else. *A Conversation* (see section 3.2.2.4) has a branching narrative, with a total of four endings, and incorporates the player as an active participant within it. However, I do not feel that it can truly be called a ‘game’, as the player cannot ‘lose’. One could consider *Countdown* (see section 3.2.4.1.3) to almost be a game, as there is a condition which may be considered as a ‘lose’ condition, yet the player will always reach that condition when going through it. Additionally, *Countdown* still has an overall very linear structure.

Finally, there’s *Backblast* (see section 3.2.4.3). It has a somewhat branching narrative, with five endings, however, the bulk of the content is still ultimately linear, with few meaningful choices. Additionally, just like *Countdown*, there is no real ‘lose’ condition, with the closest thing to a ‘lose’ condition being the ‘deaths’, which are ultimately treated as simply being parts of the rather linear narrative. Additionally, because most of these ‘deaths’ are all ultimately unavoidable (with only two deaths, from choosing the ‘doing nothing’ and ‘fighting’ options within the ‘subject A’ ending, being avoidable), they cannot be considered to be actual ‘lose’ conditions. The development of *Backblast* was fraught with bad decisions, the impacts of which have been discussed in section 3.2.4.3.2, and ultimately boil down to the lack of a clear vision at the start of development. This is something which I will need to address in any future projects I embark on, so that I can avoid producing any more similarly disappointing end products.in the future.

Whilst I will not deny that the quality of the HECCIN Games I produced with HECC-IT are somewhat questionable from an artistic standpoint, I would still like to point out that, from a technical perspective, these works are fully functional. They behave as intended, they have a textual feedback loop (in the form of the player making choices, which impact the content shown to the player), and require non-trivial effort on behalf of a reader to traverse (as they will need to continually make choices throughout the course of the game, with these choices, again, affecting the content the player has access to). Therefore, even though they are far from perfect, they do what they need to do, and they prove that HECC-IT does work.

## 5.3: Conclusions to the project as a whole

There is still plenty of scope for improvements which could be made to this product. For example, I could extend the engine to allow variables to be used within games, I could add some functionality to allow an author to define their own custom CSS for their games, I could improve the way that the HECCIN’ Game handles conditional statements such that it does not rely on ‘eval’, and so on. I could also attempt to write a much better game using HECC-IT than the ones which I have produced.

However, this doesn’t mean that the product I have delivered is terrible. HECC-IT can be used to produce hypertext games, via the medium of the HECC scripting language, and has been used to produce several hypertext games. It goes above and beyond the bare minimum requirements, with this being most evident through the existence of the OH-HECC GUI and including support for conditional statements. *Backblast* has enough content to keep a player engaged for at least 15 minutes or so, and, despite its flaws, it is still much more than a simple tech demo for the HECC-IT system. So, overall, I would say that this project has been a success.

# Bibliography

|  |  |
| --- | --- |
| [1] | E. Santos, P. Deschênes, C. Innis, R. Sharp, K. Käfer, R. Braun, D. Tarr, C. Chen, T. Stone, R. Sutherland, P. Lang, B. Combee, A. Backstrom, H. Wolfe, A. Courtiol, K. Balakrishnan, rheber, J. Gruber and J. Fraser, "showdownjs/showdown: A bidirectional Markdown to HTML to Markdown converter written in Javascript," 2 November 2019. [Online]. Available: https://github.com/showdownjs/showdown. [Accessed 19 Janaury 2021]. |
| [2] | R. Lowe, "The 'Summary of Background Reading' report I produced," 25 September 2020. [Online]. Available: https://cseegit.essex.ac.uk/ce301\_2020/ce301\_lowe\_richard\_m/-/blob/master/Reports%20n%20such/Summary%20of%20background%20reading.docx. [Accessed 25 September 2020]. |
| [3] | inkle Ltd., "inklewriter," inkle Ltd., [Online]. Available: https://www.inklestudios.com/inklewriter/. [Accessed 11 August 2020]. |
| [4] | inkle Ltd., "ink - inkle's narrative scripting language," inkle Ltd., [Online]. Available: https://www.inklestudios.com/ink/. [Accessed 11 August 2020]. |
| [5] | Interactive Fiction Technology Foundation, "Twine / An open-source tool for telling interactive, nonlinear stories," [Online]. Available: https://twinery.org/. [Accessed 5 August 2020]. |
| [6] | D. Q, "Twee2 | Interactive Fiction for Power Users," 2015. [Online]. Available: https://dan-q.github.io/twee2/. [Accessed 17 August 2020]. |
| [7] | textadventures.co.uk, "Squiffy - A simple way to write interactive fiction," 2020. [Online]. Available: https://textadventures.co.uk/squiffy. [Accessed 13 August 2020]. |
| [8] | I. Millington, A. Yakovlev, A. Plotkin, B. Dias, D. Fabulich, D. Eyk, D. Eliseev, I. Narozhny, J. Grams, J. Leinonen, M. N. Tenuis, Selene and Zonnah, "Undum – A client side framework for hypertext interactive fiction games," 2010-2018. [Online]. Available: https://idmillington.github.io/undum/. [Accessed 14 August 2020]. |
| [9] | Beck & Bartle Limited, *eHyperTool Design Specification Version 1.02 (unpublished),* 2011. |
| [10] | Choice of Games LLC, "Introduction to ChoiceScript - Choice of Games LLC," Choice of Games LLC, 2020. [Online]. Available: https://www.choiceofgames.com/make-your-own-games/choicescript-intro/. [Accessed 11 August 2020]. |
| [11] | G. Nelson, "Inform 7 | Inform is a natural-language-based programming language for writers of interactive fiction.," 2015. [Online]. Available: http://inform7.com/. [Accessed 10 August 2020]. |
| [12] | textadventures.co.uk, "Quest - Write text adventure games and interactive stories," 2020. [Online]. Available: https://textadventures.co.uk/quest/. [Accessed 11 August 2020]. |
| [13] | M. J. Roberts, "TADS - the Text Adventure Development System, an Interactive Fiction authoring tool," tads.org, 16 May 2013. [Online]. Available: https://www.tads.org/. [Accessed 4 August 2020]. |
| [14] | T. Rothamel, "The Ren'Py Visual Novel Engine," [Online]. Available: https://www.renpy.org/. [Accessed 6 August 2020]. |
| [15] | Eastgate Systems, Inc., "Storyspace: Storyspace," 2020. [Online]. Available: http://www.eastgate.com/storyspace/. [Accessed 3 August 2020]. |
| [16] | Interactive Fiction Technology Foundation, "The Treaty of Babel (Revision 9)," 24 October 2014. [Online]. Available: http://babel.ifarchive.org/. [Accessed 10 August 2020]. |
| [17] | Interactive Fiction Technology Foundation, "The Treaty of Babel (Revision 10)," 23 January 2021. [Online]. Available: https://babel.ifarchive.org/babel\_rev10.html. [Accessed 5 March 2021]. |
| [18] | S. Kitromili, J. Jordan and D. E. Millard, "What Authors Think about Hypertext Authoring," in *HT '20: Proceedings of the 31st ACM Conference on Hypertext and Social Media*, Virtual Event, 2020. |
| [19] | E. J. Aarseth, Cybertext: Perspectives on Ergodic Literature, Baltimore, MD: Johns Hopkins University Press, 1997. |
| [20] | M. Bernstein, "On hypertext narrative," in *HT '09: Proceedings of the 20th ACM conference on Hypertext and hypermedia*, Torino, 2009. |
| [21] | H. K. Rustad, "A Four-Sided Model for Reading Hypertext Fiction," *Hyperrhiz: New Media Cultures,* vol. 6, 2009. |
| [22] | N. Gibbins, "Telling Tales - EdShare Southampton," 6 November 2019. [Online]. Available: http://edshare.soton.ac.uk/20145/. [Accessed 21 September 2020]. |
| [23] | M. Pavić and S. Sofrenovic, "The Glass Snail," Word Circuits, August 2003. [Online]. Available: http://www.wordcircuits.com/gallery/glasssnail/. [Accessed 17 September 2020]. |
| [24] | A. A. Anthropy, "Queers in Love at the End of the World by anna anthropy," itch.io, 12 March 2016. [Online]. Available: https://w.itch.io/end-of-the-world. [Accessed 17 September 2020]. |
| [25] | Z. Quinn, P. Lindsey and I. Shankler, "Depression quest: An interactive (non)fiction about living with depression," 14 February 2013. [Online]. Available: http://www.depressionquest.com/. [Accessed 17 September 2020]. |
| [26] | M. Lutz, "the uncle who works for nintendo by ztul," itch.io, 6 December 2014. [Online]. Available: https://ztul.itch.io/the-uncle-who-works-for-nintendo. [Accessed 17 September 2020]. |
| [27] | Spike Chunsoft, "Zero Escape: Zero Time Dilemma," Aksys Games, Minato City (Tokyo), 2016. |
| [28] | Internet Engineering Task Force, "RFC 2119 - Key words for use in RFCs to Indicate Requirement Levels," March 1997. [Online]. Available: https://tools.ietf.org/html/rfc2119. [Accessed 24 April 2021]. |
| [29] | R. Lowe, "ce301 / ce301\_lowe\_rachel\_m · GitLab (The Gitlab Repository for this project)," 2021. [Online]. Available: https://cseegit.essex.ac.uk/ce301\_2020/ce301\_lowe\_richard\_m. [Accessed 2021]. |
| [30] | Mozilla Foundation, "Mozilla Public License, version 2.0," 3 January 2012. [Online]. Available: https://www.mozilla.org/en-US/MPL/2.0/. [Accessed 16 October 2020]. |
| [31] | Open Source Initiative, "The MIT License | Open Source Initiative," [Online]. Available: https://opensource.org/licenses/MIT. [Accessed 19 January 2021]. |