PROCEDURAL STORYLINE IN GAMES USING LANGUAGE PROCESSING

TABLE OF CONTENTS

TERRIFIC DISPOSITION BY JUSTIN ALEXANDER SHANKS

ABSTRACT

Through the combination of both natural language, processing and fuzzy logic engines an approach to procedural generation for a game's storyline should be possible. The implementation of these elements allows the game to better adapt to the player's input, by processing their input and processing it further through the usage of a fuzzy inference engine.

INTRODUCTION

Terrific Disposition is designed to be a text adventure game that somewhat adapts to the player's input by tokenising their input and processing this through a fuzzy inference engine. The majority of this processing comes in the form of tokenising the player's interaction attempts and then handing it over to the interactable's fuzzy inference engine.

The project is primarily implemented with C++, utilising; JsonCpp library for json file manipulation, SFML for graphical user interface, FuzzyLite for the fuzzy logic functionality, and Pybind11 to embed python. Python modules are utilised for the natural language processing and the definition of words, the former provided by the Natural Language Toolkit, and the latter by PyDictionary.

The player is placed in a dark and plain room with a console awaiting them and victim to a slight amnesia, they have to equip themselves with what they think they will need and leave the room.

DEFINITION OF TERRIFIC

The use of terrific in this project has a double meaning, of both great and terrifying, as it applies to the themes of cyberpunk, nuclear winter, and passive invasion present in the developed game. Although the definition pertaining to the attribute of being terrifying is rarely used, it is still applicable.

	DICTIONARY.COM	MERRIAM-WEBSTER	OXFORD
FIRST	Extraordinarily great or intense	Unusually fine	Of great size, amount, or intensity
SECOND	Extremely good; wonderful	Extraordinary	Extremely good; excellent.
THIRD	Causing terror; terrifying	a) Exciting or fit to excite fear or awe b) Very bad	Causing terror.
SOURCE	(Dictionary.com LLC, 2018)	(Merriam-Webster Incorporated, 2018)	(Oxford University Press, 2018)

REQUIREMENT CONSIDERATION

TEXT ADVENTURE GENRE

Terrific Disposition is a text adventure game and as such was designed to conform to some of the conventional elements of said games. The game centres on the provision of commands to the environment and to the non-player characters that the player would come into contact; this is a staple of the genre and as such is indispensable.

Text adventure games also typically boast a variety of commands that can be used in order to better navigate the game that, although optional, are highly recommended in order to facilitate play and to get the most content from the game itself. As a result, Terrific Disposition had to allow the user to play without using the core commands while highly incentivising the practice.

PROCEDURAL GENERATION

FUZZY LOGIC USAGE

The usage of fuzzy logic is a core component of this final year project, as outlined by Dr Jethro Shell, and is therefore responsible for the majority of the procedural content in the game. The idea is to provide the player with a more tailored experience based on how they interact with the entities in the game, the usage of fuzzy inference systems provides the ability to use player behaviour to alter the behaviour of those characters giving the player information.

NATURAL LOGIC

Natural logic is the use of regular processing techniques in conjunction with natural language processing techniques, for the purposes of this project this is taken to be the

NATURAL LANGUAGE PROCESSING

The application was deemed to require the ability to identify the nature of the words that the players inputs, primarily whether they are hostile or pleasant. To further narrow down what words are processed, three main components of sentences were chosen for processing, these being adjectives, nouns, and verbs. By focusing on these three sentence components, an accurate preliminary profile of the player inputs can be constructed.

These words are then compared to a dictionary or word list of hostile or pleasant terms in order to create a profile on the player that will be taken into account when creating or determining the interactions the player will have when playing.

Further development could include other sentence components, such as adverbs, in order to better construct a profile on the player's interactions.

TEMPLATE USAGE

As outlined in the literature review, the usage of templates allows for a more natural presentation of information in contrast to the often clunky and inelegant construction of wholly computer-generated text. However, the templates have to be created in such a way that terms are available throughout that are identified and then changed by the system in order to tailor to the game's theme.

TECHNICAL CONSIDERATIONS

PROCESSING SPEED AND CORE FUNCTIONALITY

C++ AND SFML

Due to years of experience both inside and outside of academia, a solid knowledge of C++ ensures the development of the project. Aside from the familiarity with the language, C++ is a language with excellent memory management features, as well as having a high performance, meaning that it would pair very well with a slower language such as Python (discussed later on) for the task.

FUZZYLITE

Fuzzylite facilitates in the design of fuzzy inference systems, while also being efficient to run without taking up much of processor speed and without requiring additional libraries. (FuzzyLite Limited, 2018) Therefore, fuzzylite was a good candidate for the implementation of the fuzzy inference system as it is indeed easy to use and in fact uses much of the same language as Matlab within a C++ environment.

NATURAL LANGUAGE PROCESSING

PYTHON, SPACY.IO, AND NLTK

Although there exist some C++ libraries that handle natural language processing, they are not reliable in an application. This is because many require the use of a cloud-based api that would therefore require an internet connection, lending a highly variable processing time depending on the player's internet connection. Although this may be an acceptable risk to undertake the benefit to utilise a python language module was far greater, as the language has several modules or libraries well designed for natural language processing.

Initially, the python module chosen to process the player entries was spacy.io, due to its lightweight and rapid start-up/processing speed. However, when it came to embedding the python code into the C++ application some issues arose, and the standard Natural Language Toolkit was utilised in its stead.

Although Python has an api allowing for its methods to be directly called by C++ code, the decision to utilise a third party library to handle the embedding of the python code was made to allow for an automatic handling of the python interpreter. PyBind11 is the library that ensures this functionality as it is sufficiently lightweight while also being easy to handle.

IMPLEMENTATION

ESSENTIAL GAME OBJECTS

INTERACTABLES

Initially, the interactables were going to be hardcoded as three distinct versions called type 80a, 80b, and 20. It was decided that these three would be the types of entities the player could interact with, and that they would appear 80% and 20% of the time respectively, with 80a and 80b alternating. These entities would all have a fuzzy inference system for their interaction engine, but their behaviour profile would have been set from the beginning.

In order to make the game more reactive to player experience however, it was decided to change this into utilising a fuzzy inference system at the moment of construction.

FUZZYBEHAVIOUR

Two fuzzy inference systems were created for the interactable entities, the first is the World Context Engine, the second being the Fuzzy Interaction Engine, which are both modelled below.

The World Context Engine determines what type of behaviour profile the interactable will have and takes into account the honesty index of the world, the user's play-style, and the player's overall hostility. This ensures that the world's environment and the player's actions are taken into account.

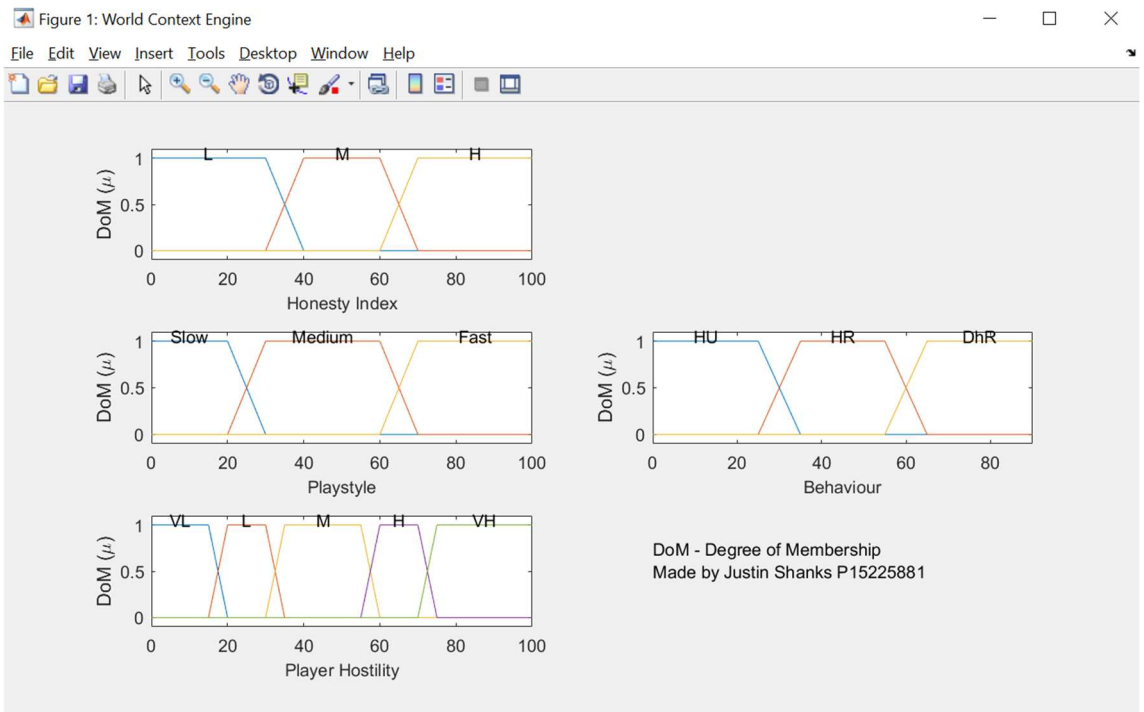


Figure 1 – World Context Engine

(TYPE) VALUE	DESCRIPTION	JUSTIFICATION
(INPUT) HONESTY INDEX	The overall ratio of honest behaviour profiles to dishonest behaviour profiles.	This allows the game to keep behaviour presence somewhat balanced. Designed to avoid overwhelming the player with a single behaviour.
(INPUT) PLAYSTYLE	How correct the player was at answering the calibration questions in the room escape sequence.	The faster the player or the more accurate they were during the calibration sequence the better they understand the writing. This allows presenting them with less helpful entities to up the difficulty.
(INPUT) PLAYER HOSTILITY	The player's hostility throughout the play-through.	The idea is that in a story as a character continues being hostile; their personality starts to match that hostility.
(OUTPUT) BEHAVIOUR	The entity's behaviour profile.	As the output of the system, this determines the profile the entity has and shapes all future interaction.

The Fuzzy Interaction Engine determines the magnitude of the behaviour profile, this is to mean that the behaviour profile has a standard level that is increased based on player interaction. Helpful behaviours become even more helpful, unreliable becomes more unreliable, and dishonesty becomes more dishonest.

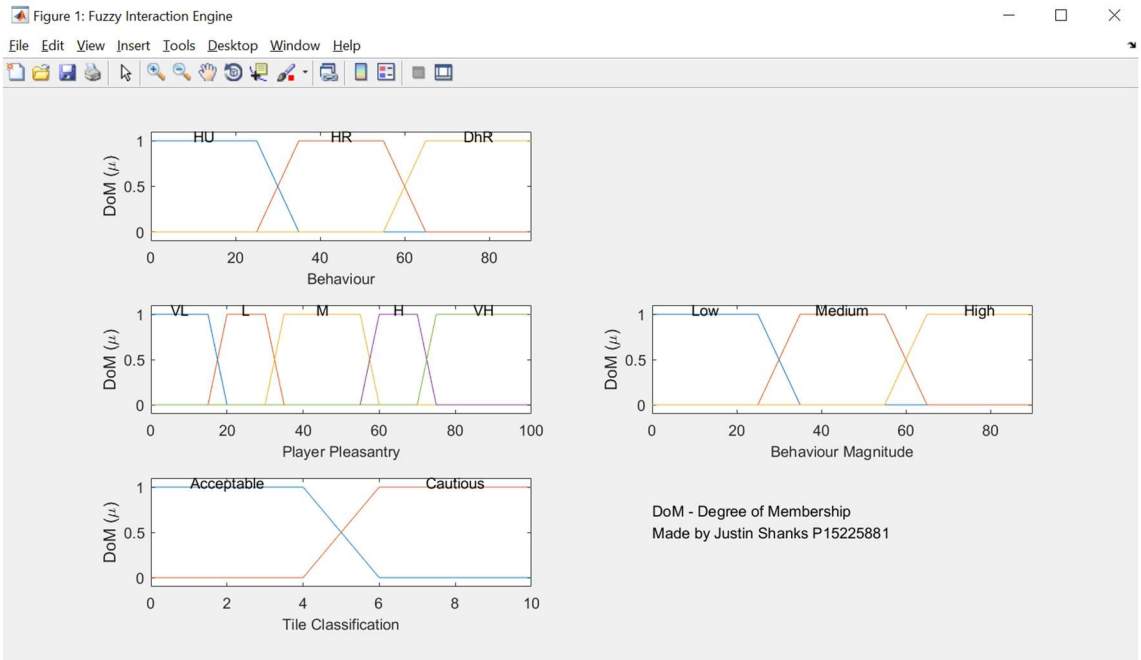


Figure 2 – Fuzzy Interaction Engine

(TYPE) VALUE	DESCRIPTION	JUSTIFICATION
(INPUT) BEHAVIOUR	The entity's behaviour profile.	Rather than having three different types of engines or systems in place to handle behaviour this engine handles the behaviour profile.
(INPUT) PLAYER PLEASANTRY	The player's use of pleasant words when interacting with the entity.	The player has the option to be straight to the point, courteous, or extremely courteous. This decision will alter whether someone would want to help them, or take advantage of them.
(INPUT) TILE CLASSIFICATION	What type of tile the entity is standing on.	This allows determining how a behaviour would react to being in a position of heightened danger. Honest entities might not be as helpful, and dishonest ones might be bolder.
(OUTPUT) BEHAVIOUR MAGNITUDE	Viewing the overall behaviour as a vector, the direction is its behaviour profile, whereas its magnitude is the behaviour magnitude.	The behaviour magnitude determines how helpful honest entities might prove, how tricky dishonest ones might be. Without this aspect the player might receive the same interaction every time they interact with an entity.

Figures one and two were generated by a Matlab script designed to mimic the behaviour of the fuzzylite fuzzy inference system present in the game. The decision to copy the fuzzylite system over to matlab was to ensure development speed, due to the limitations of the development system and the time required to build the fuzzylite files.

LANGUAGE PROCESSING

The game processes the player's inputs if they follow an interact command, identifies if the words being said are adjectives, nouns, or verbs and then checks to see the word lists or dictionaries to see if they are vulgarities, hostile, or pleasant.

This is done to build the player's pleasantry index and their hostility index, which are relevant with each interaction and through the entire play-through respectively. The idea behind this decision being that in a hostile environment the player would find themselves becoming more and more hostile, with pleasantries being employed during each interaction in order to get the information that is needed at that moment. As a result, the player's overall pleasantry is not being tracked, as it is deemed more self-serving than their hostility, which is reasoned as being a product of their environment.

It was deemed that no other command would elicit the need for natural language processing, as they would constitute an internal monologue. It could be argued that the player character could move menacingly north or look around with suspicion, but this was deemed as minimal and small enough to not take into account.

The decision to only keep track of adjectives, nouns, and verbs is due them being core components of speech that hold a great deal of information. It could be argued that it always benefits to process and keep track of the entirety of the given interaction command, but the "big three" allows the game to process sufficient information to get an accurate enough profile of the player's speech or interaction with other characters.

A small note on the differentiation of hostile part of speech terms and vulgarities. Seeing as many vulgarities can be used fairly loosely as any part of speech it was deemed to process them separately in order to ensure that hostility is still taken into account, without having to place the same words in all three word lists or dictionaries.

MANAGEMENT

RESOURCEMANAGER

The majority of the resources are saved in readable json files that are loaded when the game is loaded. This allows for the addition or alteration of the resources that will be used in the game, and although it is currently in a readily readable format, it would be reasonable to assume that a real game of this type would have encrypted resources to avoid spoiling the plot.

POTENTIAL CONFLICT

In the folder containing the projects and solution for this final year project is a folder called *Potential Conflict*, which contains a ResourceManager developed for the first assignment of the Games Programming Module IMAT3606.

However, upon reviewing the two versions of the ResourceManager it should be apparent that sufficient differences exist between the two. Although the ResourceManager used in the final year project was based upon the one for IMAT3606, it has been worked upon and sufficiently developed to the point that it is a different class.

NATURALLOGICMANAGER

The NaturalLogicManager was originally implemented with the name PythonManager, as it initially dealt exclusively with running python scripts while gathering data from the ResourceManager. However, as implementation carried on it was decided to add to the PythonManager the functionality to gather the loaded templates and processing them for user viewing.

As a result, this manager was renamed as it became less python focus and more generally focused on the general natural logic processing as outlined by the literature review.

USER CALIBRATION

ROOMESCAPE

When the player begins the game, they are placed into an escape the room scenario under the pretence of slight amnesia and hiding out from an unknown entity. This allows the player to pick what theme to play the game, how some of the commands work, and to figure out how perceptive the player is at answering given answers.

The player is told that they need to equip themselves to leave the room, through this the theme selection is carried out; this prompts them to look around the room and find some items on the floor, which correspond to the three themes available to be played. Although nine items are present, only six are able to be equipped and the player needs only to pick up and in order to leave the sequence. The player is not allowed to pick up items of different themes, but is allowed to drop picked up items, the items also have descriptions which serve as a hint as to what themes are available.

When answering the questions required to leave two things occur that affect subsequent gameplay; both the player's success rate when answering and the profile of their answers are logged. The former information allows to identify how easily they understand the questions, as this will dictate how well they understand the theme and story they are about to interact with, whereas the latter allows to build up a record of their hostility.

The questions provided to the player have some keywords associated, which identify whether or not the answer is acceptable, and are most always hostile in nature. This hostility helps build up the player's hostility record which is crucial to the game as the universe that all three themes are set in are inherently hostile and filled with struggle, meaning that from the get go the player is slowly seen more and more hostile by the in-game interactables.

PROCEDURAL GENERATION

PLOT STRUCTURE

Initially the game was going to be completely free form and be determined as the player progresses in the game. However, it was later decided to add a lose plot structure in order to give the game a sort of framework to build upon and in order to help give the game a focus.

Due to the plot points being divisible by four, and the theme of the game being somewhat pessimistic, it was decided to give the game a four-part structure of sorts. A three-act structure is typically constructed of a setup, confrontation, and resolution (Moura, 2014) and the five act structure consists of prologue, conflict, rising action, falling action, and resolution (Ray, 2018). The game's four act structure consists of proving worth, personal benefit, local benefit, and systemic benefit, ending with the peak of conflict and resolution as the player invariably dies.

The parts of this structure denote the player proving their worth to a group or gang in the game, defeating an enemy that primarily targets the player, defeating an enemy that affects the local community and then defeating a larger enemy that poses the main threat in the story. Seeing as the game's themes are fairly pessimistic and bleak it was deemed fitting that the player character dies at the end of the story, gaining victory only at the expense of their life.

WORLD GENERATION

WORLD SIZE AND DIFFICULTY

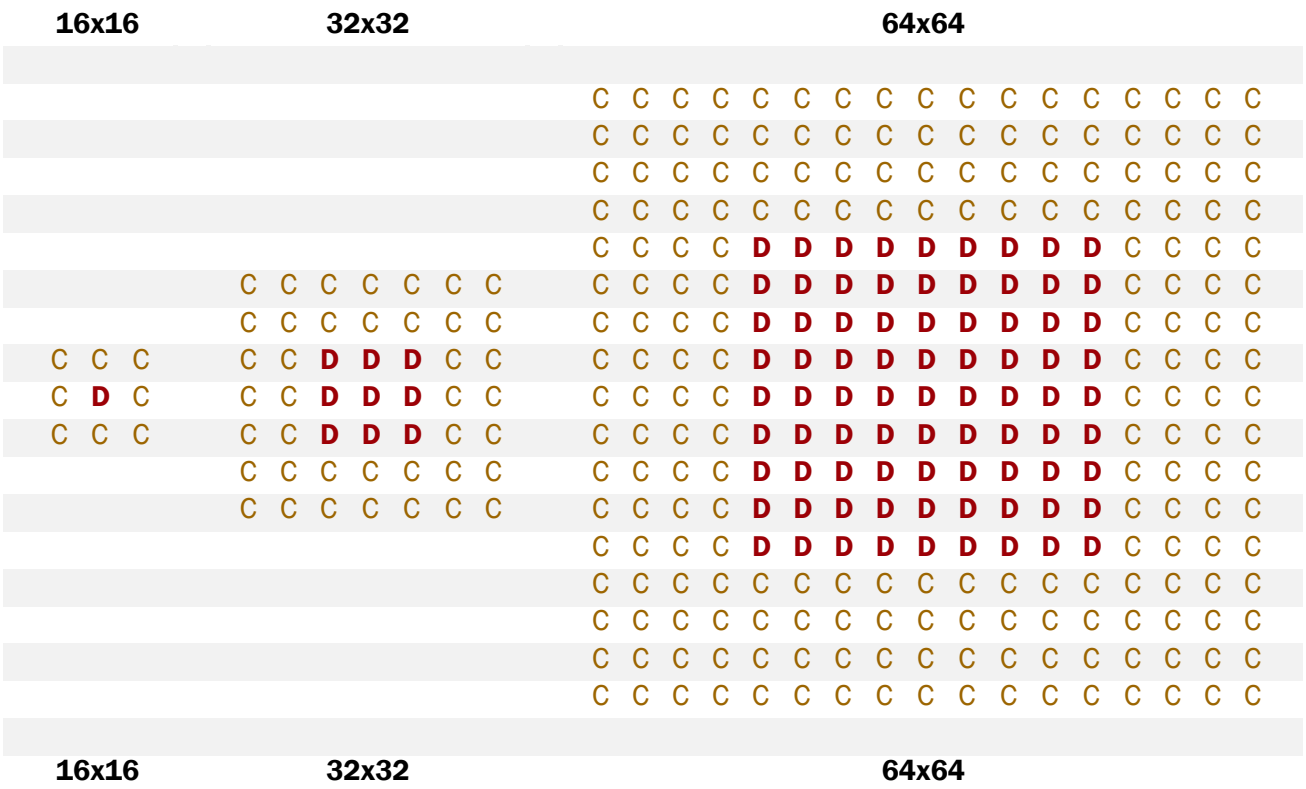
The game allows the player to select different world sizes in order to have different difficulty levels. The different sizes are not scaled up versions of the smallest map; they have a moderate increase in the number of both cautious and deadly tiles as can be seen in the following table.

	Total Tiles	***Plot Positions***	***Dangerous***		***Deadly***	
			Total Tiles	**Percentage**	**Total Tiles**	**Percentage**
16x16	256	12	36	**14.06%**	4	**1.56%**
32x32	1024	24	196	**19.14%**	36	**3.52%**
64x64	4096	48	1156	**28.22%**	324	**7.91%**

Ramping up the number of dangerous and deadly tiles with the world size allows for the definition of harder play-styles without merely labelling them differently. The larger amount of dangerous tiles also helps increase the difficulty in ways other than merely prolonging gameplay with more plot points.

The danger zones allow setting a reason for caution, as well as one for death. Without this the player would have little reason to actually be wary of their surroundings as well as reducing the save facility of the game as a mere time-management tool.

The danger zones are delineated below as a centre of deadly tiles surrounded by cautious tiles.



DESIGN DECISIONS

INCLUSION OF AUDIO

Text adventure games do not typically have audio playing throughout gameplay; however, it was decided to add audio in order to add another layer of immersion. This immersion comes in the way of having different audio files play based on the type of tile the player finds themselves in as well as what theme they have decided to play.

The inclusion of audio files based on what behaviour profile the interactable the player is interacting with was considered, but was deemed as making it too easy for the player to know what behaviour profile they are talking to. The idea behind these behaviour profiles was to require the player to read into what is being said and figuring out whether they are honest, reliable, both, or neither.

USER INTERFACE

Initially the user interface was similar to a console command window with three main sections, notes, display and commands. The notes were to allow the player to take notes of anything they wanted, such as their inventory or their character name or important locations. The display is where the player gets the relevant information from the world they are interacting with, whereas the commands window merely shows the text the player is and has typed.

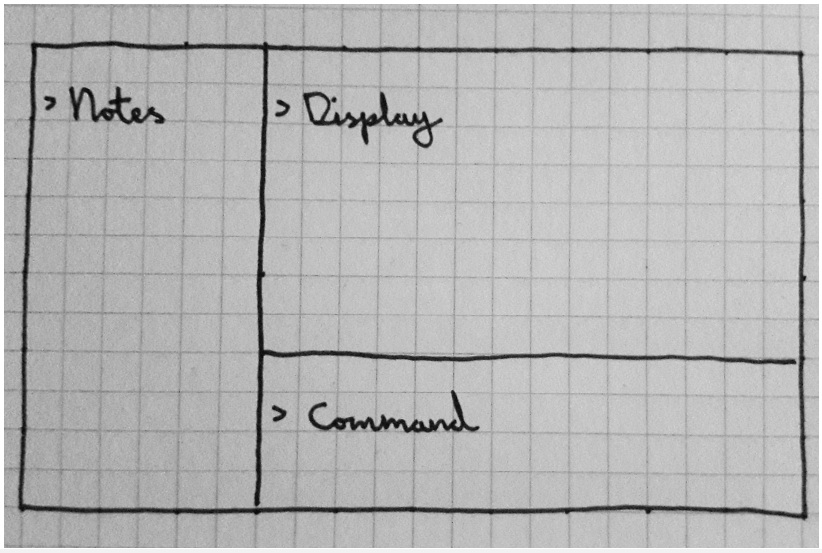


Figure 3 – Old UI Design

It was later decided to add another section called journal, with the other three sections being called very much the same with display being renamed output. The decision behind the addition of the journal section was to keep track of the base information that the player may benefit from, such as their character name, their position in the map, and their equipped items.

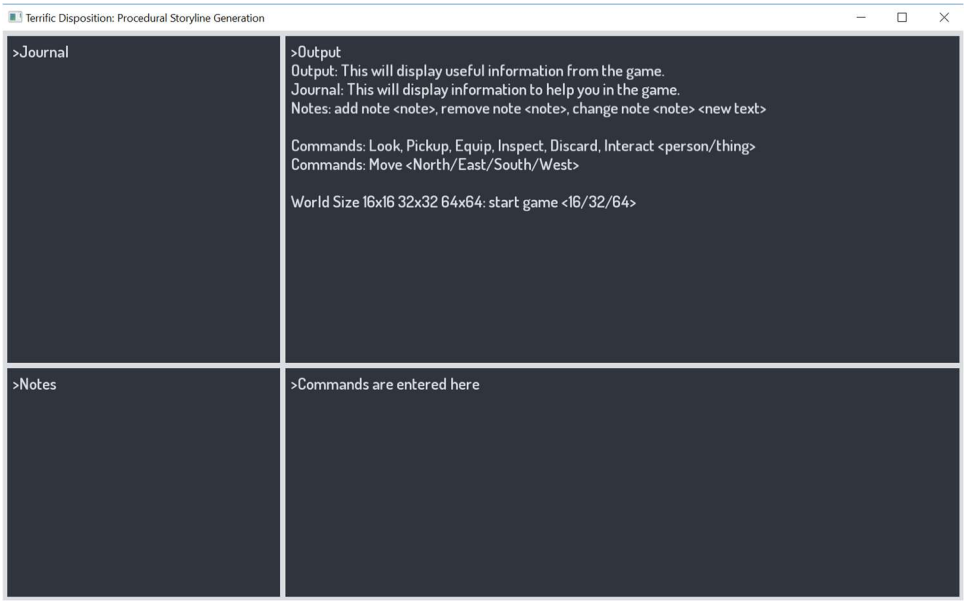


Figure 4 – Terrific Disposition Main Menu

ROOM ESCAPE

The room was designed to be a gloomy and cramped area with items strewn across the floor that correspond to the different themes. The player cannot move in the room as it is too small, the only thing that seems to be occurring in the room are a flashing light, and the only items are the ones in the floor, the moss, and the console-door exit.

Some of the elements in the room direct the player to the door and the console, whereas the others suggest to the player that the room is somewhat dangerous but that it is now safe to exit. The light, for instance, suggests to the player that this room protects them from the outside world while also conveying that it is now safe to leave. The moss in the corners suggests to the player that the room is not regularly inhabited, and allows for the suggestion that the fungal/biotic invasion is quite pervasive.

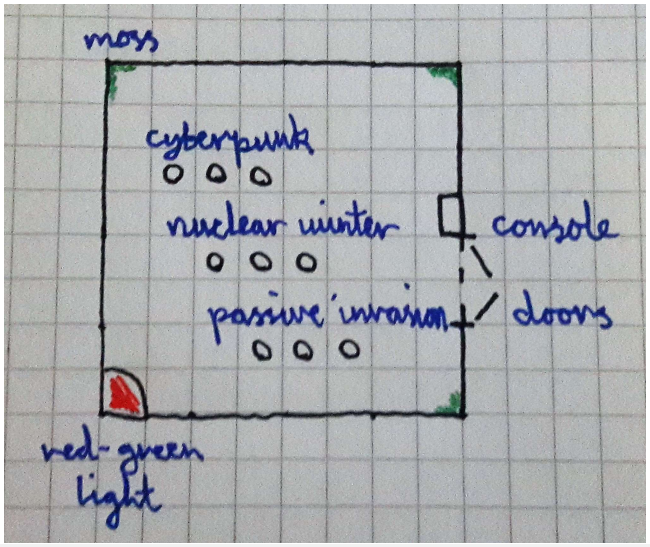
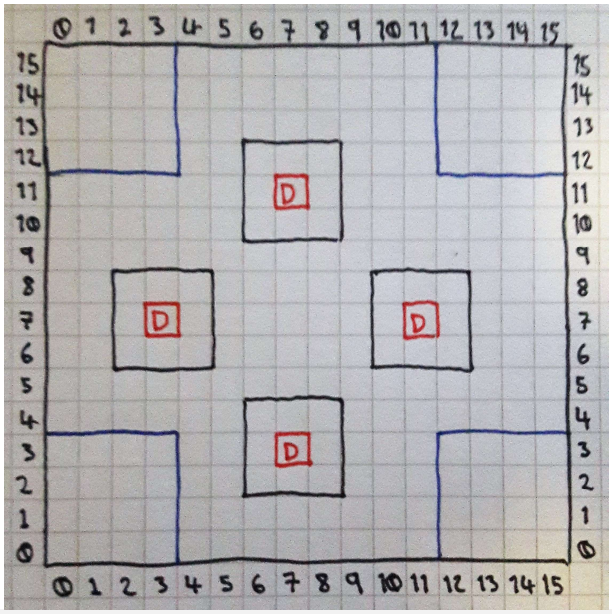


Figure 5 – Room Escape Design

WORLD

The decision to make the world maps be square was, in part, for simplicity as well as giving the player enough room in all direction without favouring any one axis.



Red Tiles: Deadly Tiles

Black Squares: Dangerous Zones

Blue Squares: Player Spawn Zones

Figure 6 – 16x16 World Map

The above figure shows the danger zones present in the 16x16 map, as well as the four zones in the corners which show the potential player start position. The actual player start position undergoes randomisation during world construction within one of these four zones in order to help change gameplay start conditions between each play-through.

The world is static, in the sense that the map layout is static, the plot changes between each play-through, with the interactable behaviours changing as, and when the player interacts with them. The idea behind this that the geography of the world remains the same but the population is what actually changes and reacts to the player as they navigate and interact.

This decision was primarily made in order to create both a sense of familiarity for the player, while also allowing for enough change to occur between each play-through.

EVALUATION

TESTING

MATLAB TESTING

As previously mentioned, the FuzzyLite mamdani engines that were to be used in the project proper were also recreated in a MatLab environment. The primary reason was to allow for a greater amount of alteration and testing without significantly slowing down the implementation process, as building the project anytime the fuzzy engines are significantly altered often took five to ten minutes due to the system specifications of the computer being used for development.

A benefit of the system's low technical specification is that more documentation and testing logs are available to ensure proper functionality. Although, anytime a change is made to the fuzzy engine requires

altering not only the C++ unit tests but the MatLab script as well which does require some more development time.

UNIT TESTING

The majority of unit testing was focused around the behaviour and management classes, primarily those for FuzzyBehaviour and NaturalLogicManager. This is due to the fact that these classes are the most important when it comes to testing the required functionality for the final year project's goals.

The FuzzyBehaviour unit tests essentially re-create the aforementioned MatLab testing regiment, in order to ensure that the two systems are indeed mirror images of one another. This involved testing that the different combinations of inputs lead to the expected outputs for the mamdani fuzzy inference engine, and as a result many of the values being tested for were taken directly from the MatLab testing regiment.

The NaturalLogicManager is in charge of the python modules and of processing the templates that have been prepared for the project. Therefore this class had to be tested for the proper tokenisation of input, as well as ensuring that the template blanks have been removed and replaced with the provided plot or theme keywords.

NOTE ON PYTHON TESTING

Pybind11 provides the user with a scoped interpreter to allow for a better automatic management of the python interpreter. However, as with most automated processes, this is at the lack of control over said interpreter.

Seeing as the game always keeps the same NaturalLogicManager alive throughout the game's runtime this poses little to no problem, but when running the unit tests this causes subsequent python tests to fail. The interpreter is still technically running between unit tests, making it impossible to create a new scoped interpreter. Although this can be worked around by running the failed tests again until they run, it does slow down testing somewhat.

PLAY TESTING

During development, the game was tested informally by the student behins this final year project, however, many times this was very brief or merely testing the visual component of the application.

What follows are playtests that were carried out by others in order to ensure that the application was reasonably playable by a person not involved with development. The testing carried out by the developer will not be greatly discussed as it suffers from a certain bias as the developer had an in depth knowledge of how the system worked, and may have avoided certain bugs due to said knowledge.

Flat mates, and their guests, known to the developer carried out tests. An outline of the provided information, the carried out actions, the successful application functionality, and the failures or bugs in the application are delineated below. The tests are given a numeric identifier as well as a pseudonym for the tester in question for the purposes of reference while also maintaining anonymity, the same pseudonym means that the same person carried out the different tests.

TEST	0	1	2
INFORMATION	The game is text-based; you start in a room and have to leave somehow. There will be items on the floor; you select the theme by picking up and equipping the items of a theme. Please pick the cyberpunk ones for these tests.		
TESTER	Alpha	Bravo	Charlie
TESTING	Room Escape or Calibration		
BRIEF DESCRIPTION	Player attempts to pick up and equips the items corresponding to the cyberpunk theme. Player approaches the console and attempts to answer the questions.	Player attempts to pick up and equips the items corresponding to the cyberpunk theme. Player approaches the console and attempts to answer the questions.	Player picks up and equips the items corresponding to the nuclear winter theme, attempts to equip the cyberpunk items. Player attempts to drop the nuclear winter items and then attempts to pick up and equip the cyberpunk theme items. Player approaches the console and attempts to answer the questions.
SUCCESS	• Picking up items • Equipping items • Approaching the console • Answering console questions	• Picking up items • Equipping items • Approaching the console • Answering console questions	• Picking up items • Equipping items • Denying the mixing of thematic items • Dropping items • Approaching the console • Answering console questions
PROBLEMS	• Answer key words hard coded, difficult to alter. • Answer keys too specific	• Answer key words hard coded, difficult to alter. • Answer keys too specific	• Answer key words hard coded, difficult to alter. • Answer keys too specific • Picking up incorrect item displays empty message, neither confirming or denying command

The first round of testing dealt with ensuring that the base room escape sequence works as required, as a result this testing checked if the player could pick up and equip the required items. Most of the required functionality worked as expected, but certain problems did arise, primarily that the required key items were too specific for the testers. When considering that the key words were hard coded and would require significant change of the code to alter, it was decided to add these keywords to the resource file required for the room escape sequence.

TEST	3	4	5	6
INFORMATION	The game is text-based; you start in a room and have to leave somehow. There will be items on the floor; you select the theme by picking up and equipping the items of a theme. Please pick the cyberpunk ones for these tests.			
TESTER	Alpha	Charlie	Delta	Echo
TESTING	Room Escape or Calibration			
BRIEF DESCRIPTION	Player attempts to pick up and equips the items corresponding to the cyberpunk theme. Player approaches the console and attempts to answer the questions.	Player picks up and equips the items corresponding to the nuclear winter theme, attempts to equip the cyberpunk items. Player attempts to drop the nuclear winter items and then attempts to pick up and equip the cyberpunk theme items. Player approaches the console and attempts to answer the questions.	Player attempts to pick up and equips the items corresponding to the cyberpunk theme. Player approaches the console and attempts to answer the questions.	Player attempts to pick up and equips the items corresponding to the cyberpunk theme. Player looks at the various elements around the room. Player approaches the console and attempts to answer the questions.
SUCCESS	• Picking up items • Equipping items • Approaching the console • Answering console questions	• Picking up items • Equipping items • Denying the mixing of thematic items • Dropping items • Approaching the console • Answering console questions	• Picking up items • Equipping items • Approaching the console • Answering console questions	• Picking up items • Equipping items • Approaching the console • Answering console questions
PROBLEMS	• Keeping track of answered questions	• Keeping track of answered questions	• Slight confusion as to how to leave the room for some time. • Keeping track of answered questions	• Keeping track of answered questions

The change from hard coded keywords to json file-based keywords allowed to easily alter the required keywords as well as adding new ones if necessary. The main difficulty that arose were that the answered questions were not being recorded correctly, and that a tester found it difficult to find the exit and had to

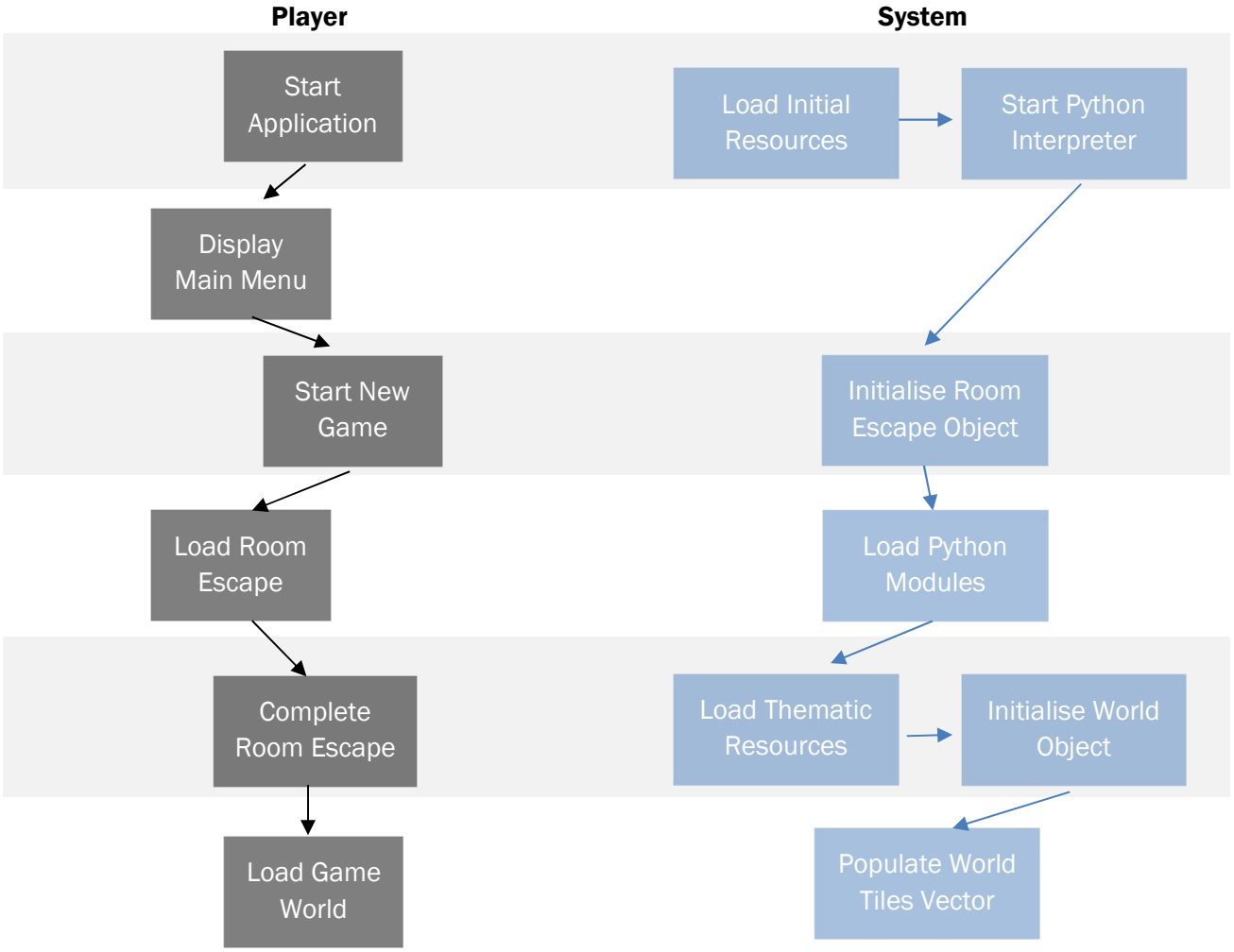
be instructed how to do so. The former issue was caused due to a design change from using a timer to determining the ratio of correct-incorrect answers and was easily solved, whereas the latter was solved by adding flavour text in the sequence designed to let the player know that a door exists and that it might be the way out.

TEST	7	8	9	10
INFORMATION	The game is text-based; you start in a room and have to leave somehow. There will be items on the floor; you select the theme by picking up and equipping the items of a theme. Please pick the cyberpunk ones for these tests. Interact with people/things on each tile and try to go to the next plot point, the interactables may or may not be helpful. They may also lie to you, so be careful.			
TESTER	Alpha	Bravo	Charlie	Echo
TESTING	Generation of plot points around the game map.			
BRIEF DESCRIPTION	Room escape scenario carried out similar to tests 0, 1, 3, and 5. Player would move and interact with the interactables as and when chosen. Attempts to follow the plot points procedure, stopping at four. Decided to stop play testing.	Room escape scenario carried out similar to tests 0, 1, 3, and 5. Player would move and interact with the interactables as and when chosen. Attempts to follow the plot points procedure, stopping at two. Decided to stop play testing.	Room escape scenario carried out similar to tests 0, 1, 3, and 5. Player would move and interact with the interactables as and when chosen. Attempts to follow the plot points procedure, stopping at four. Unable to continue play testing.	Room escape scenario carried out similar to tests 0, 1, 3, and 5. Player would move and interact with the interactables as and when chosen. Attempts to follow the plot points procedure, finishing the plot points.
SUCCESS	• Moving around the map • Interacting with the tile interactables • Presenting templates	• Moving around the map • Interacting with the tile interactables • Presenting templates	• Moving around the map • Interacting with the tile interactables • Presenting templates • Death process due to walking onto a deadly tile	• Moving around the map • Interacting with the tile interactables • Presenting templates • Death process due to winning the game
PROBLEMS	• Some templates are improperly processed	• Some templates are improperly processed	• Some templates are improperly processed • Fifth plot point placed outside the bounds of the map	• Some templates are improperly processed

This final play testing uncovered some issues when loading and presenting the templates to the player, as well as the improper allocation of plot points to the game board.

PLAYER-SYSTEM OPERATION

STARTING A NEW GAME

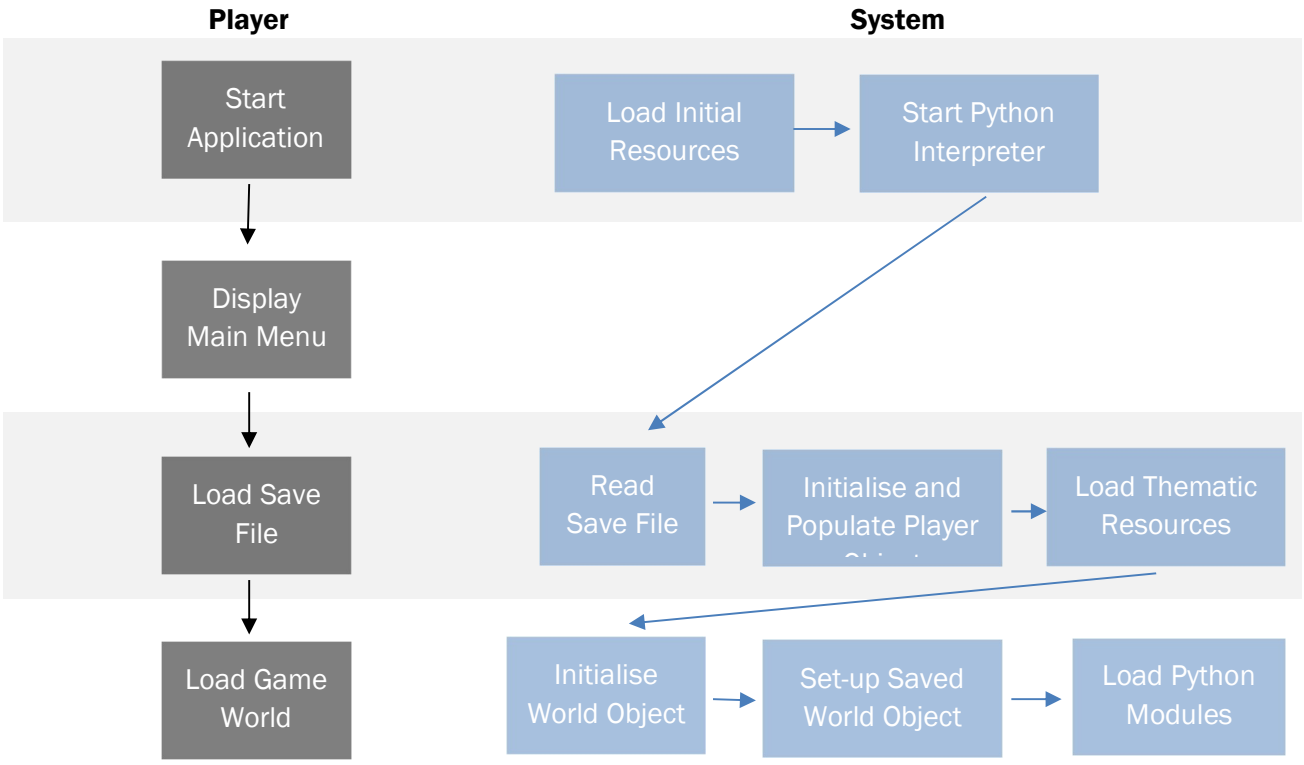


The first interaction a player has with the application is to start a new game, which requires them to play and complete the initial calibration sequence called and styled as a room escape scenario. As the player start up the application, they are greeted with the main menu that lists some relevant information in order to help them start a new game. They select the size/difficulty of the game world and then load the room escape scenario, once the sequence is completed the game world is loaded and the player is allowed to play out the plot sequence.

The key system operations involve the loading of all the relevant resources, located in various json files. The majority of the resources are loaded in the application's start up sequence, but certain resources are loaded once the game world is loaded as they require the player to select one of the available themes.

The python modules are not necessarily loaded in the room escape specifically, but rather are loaded once the player inputs an interact command.

LOADING A SAVEFILE

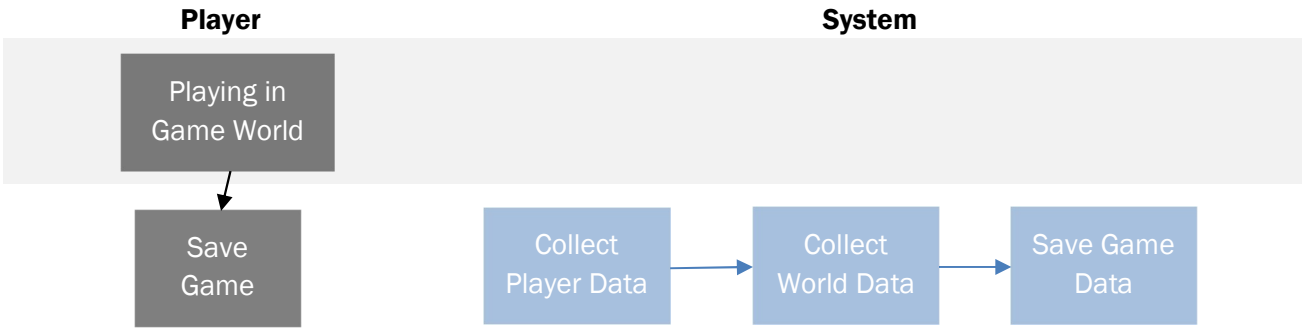


The initial sequence is very much the same, with the start-up sequence being carried out as normal. Once the player has reached the main menu they have the opportunity to try and load a previously saved game, the application then checks to see if the save file exists and reads the relevant json file. Once the save file has been read, the room escape sequence is skipped over with the game world being loaded with the collected information.

The collected information populates the initialised player object and then the game world object is initialised. This initialisation sequence is similar to a new game being loaded, with the exception that the player's start position and their current position are both known and set based on the read-in data. The world's plot positions are also loaded in, as well as the player's current position within the plot's progression.

The python modules are not loaded automatically, but are loaded once the player provides with an interact command.

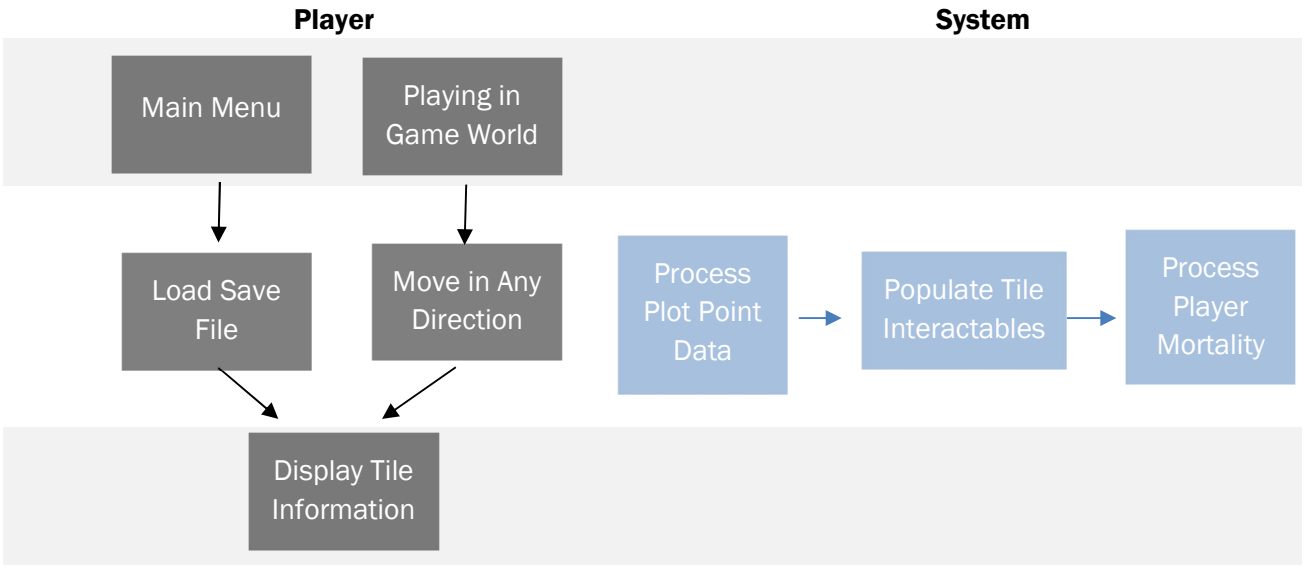
SAVING CURRENT GAME



When saving the current game, certain aspects were able to be saved, whereas others were not. For example, the interactables could not be reliably saved without requiring a lengthy save sequence as the fuzzy behaviour profile would have to be saved as well as their locations in the game world.

Important information was able to be collected for the saving sequence, this being the complete player object's variables as well as some rudimentary world information. The saved world information being the map size, the player's start location, and the generated plot points. This allows the game to be saved to a satisfactory degree, with the lack of interactable data being explained by having the interactables moving on to a new area not covered by the world map. Although this is a purely lore or in-game explanation, it serves well enough for the time being, and this limitation ends up providing new player interactions.

PROCESS TILE SEQUENCE



The process tile sequence is called every time the player moves to a new tile, or when they successfully load an existing save file. The sequence's main operations involve the processing of whether plot points exist on the tile, if so a new one must be allocated, populating the tile with interactables, and whether the tile is deadly to the player or not.

The player mortality processing requires checking if the tile is a deadly tile or if the plot points have been exhausted, as the player invariably dies at the end of the game.

FINAL CONSIDERATIONS

CONCLUSION

Although not ideal, the developed system demonstrates that through a combination of both natural language processing, and fuzzy logic systems an approach to procedural storyline generation is possible. The current system still heavily depends on the outlined four-part structure and might be able to be retooled with the calibration sequence being used to determine what type of story structure the player prefers and adapting to said preference.

Further development could centre on extending the number of sentence components are taken into consideration when processing player input, or even processing the relationship between sentence components. Another option for future development would be creating a library of pleasant and hostile words based partially on natural language processing. As it currently stands, the dictionaries or word lists have to be manually compiled, although the creation of this may be manual at first, but should then be aided with natural language processing tools.

The project helped with a better understanding on how to merge multiple programming languages and how to best take advantage of the characteristics of the utilised programming languages. Although the chosen languages are not typically used in the manner present in the project, as python is usually extended with C++ rather than being embedded in C++, it was still far better to go through the effort.

RUNTIME AND PYTHON

It should be noted that although most of the include and lib files were included with the submitted project, in specific Include and Library folders, those for python were not due to the requirement of keeping the python directory unchanged. Therefore, these paths were not relative, as python had to be installed into the computer being used.

WORKS CITED

Dictionary.com LLC, 2018. *Dictionary.com Definition of Terrific.* [Online]
Available at: http://www.dictionary.com/browse/terrific?s=t
[Accessed 15 March 2018].

FuzzyLite Limited, 2018. *Fuzzylite Github.* [Online]
Available at: https://github.com/fuzzylite/fuzzylite/
[Accessed 5 April 2018].

Merriam-Webster Incorporated, 2018. *Merriam-Webster Definition of Terrific.* [Online]
Available at: https://www.merriam-webster.com/dictionary/terrific
[Accessed 15 March 2018].

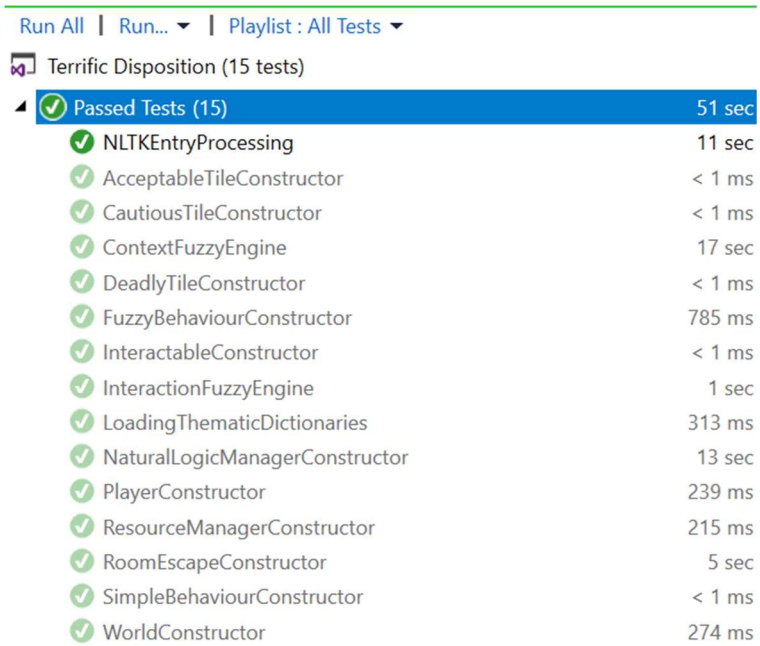
Moura, G., 2014. *Three Act Structure.* [Online]
Available at: http://www.elementsofcinema.com/screenwriting/three-act-structure/
[Accessed 02 March 2018].

Oxford University Press, 2018. *Oxford Definition of Terrific.* [Online]
Available at: https://en.oxforddictionaries.com/definition/terrific
[Accessed 15 March 2018].

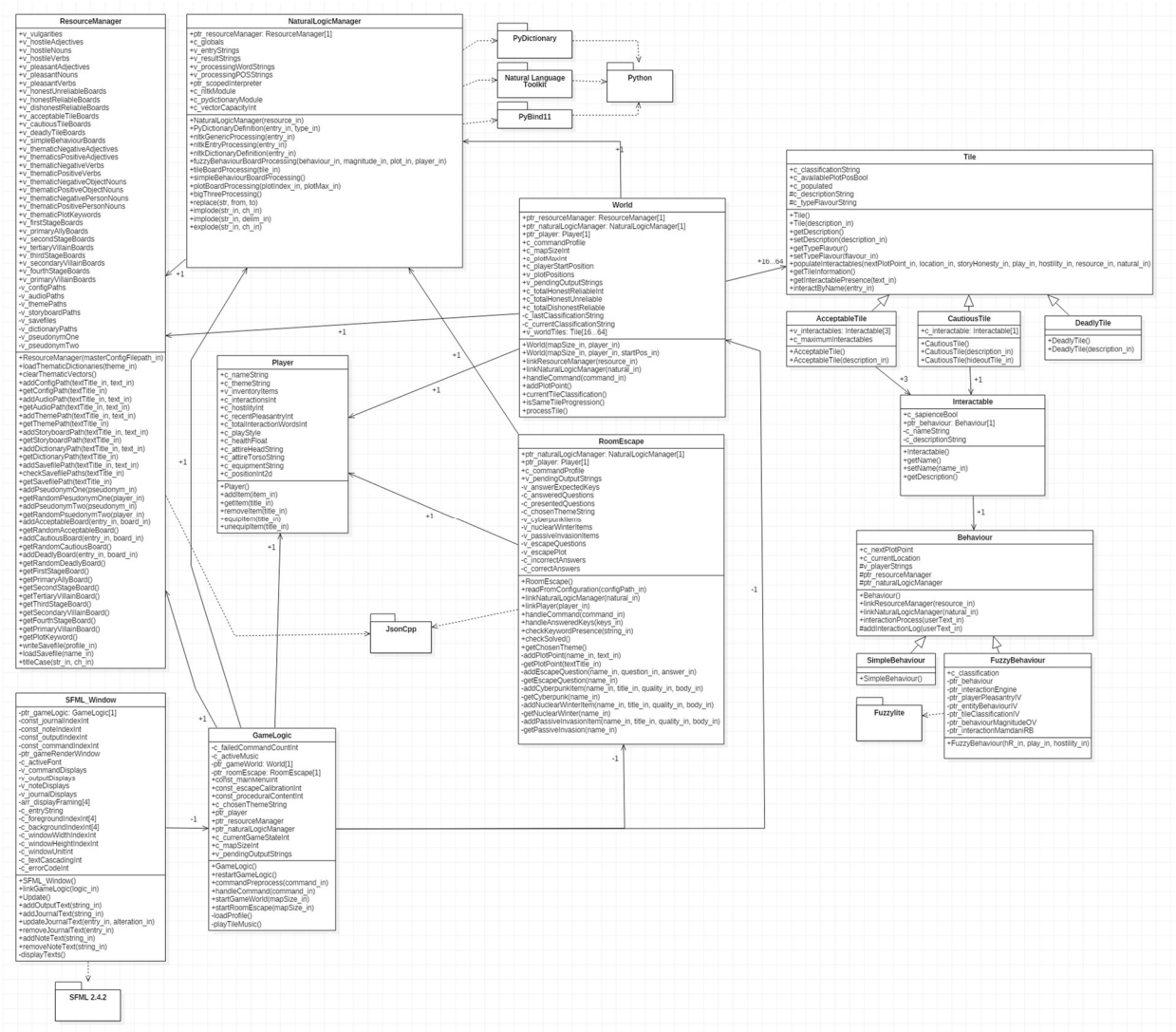
Ray, R., 2018. *The Five Act Play (Dramatic Structure).* [Online]
Available at: https://www.storyboardthat.com/articles/e/five-act-structure
[Accessed 2 March 2018].

APPENDIX

UNIT TEST



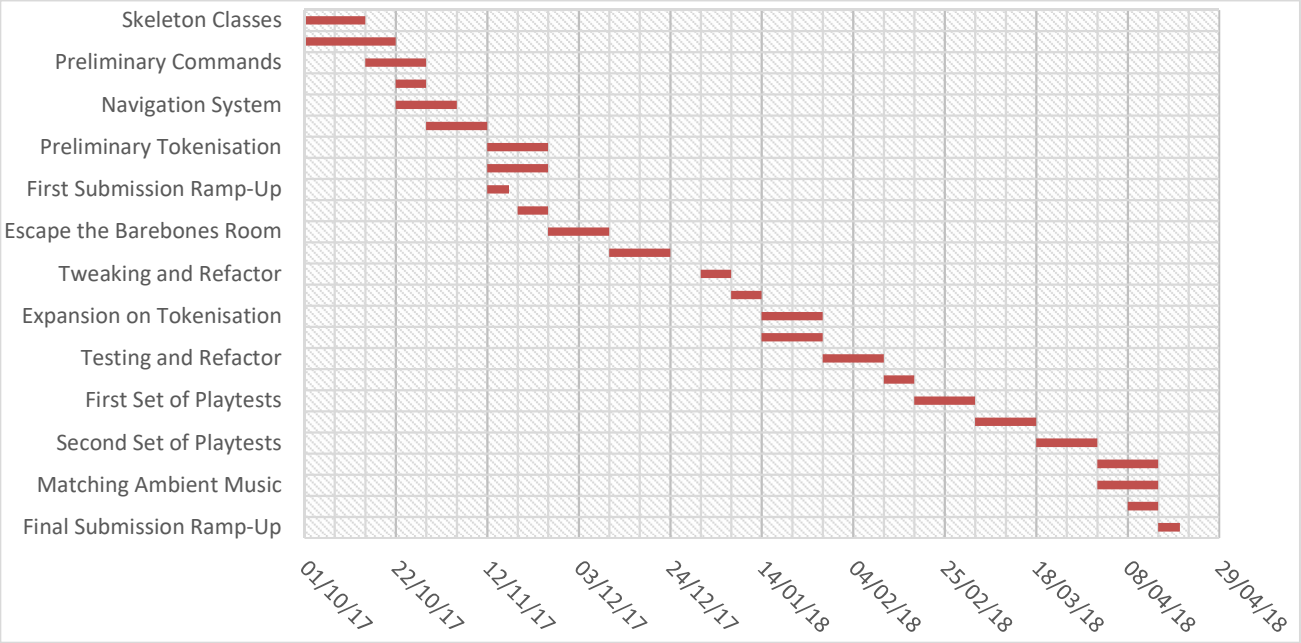
CLASS DIAGRAM



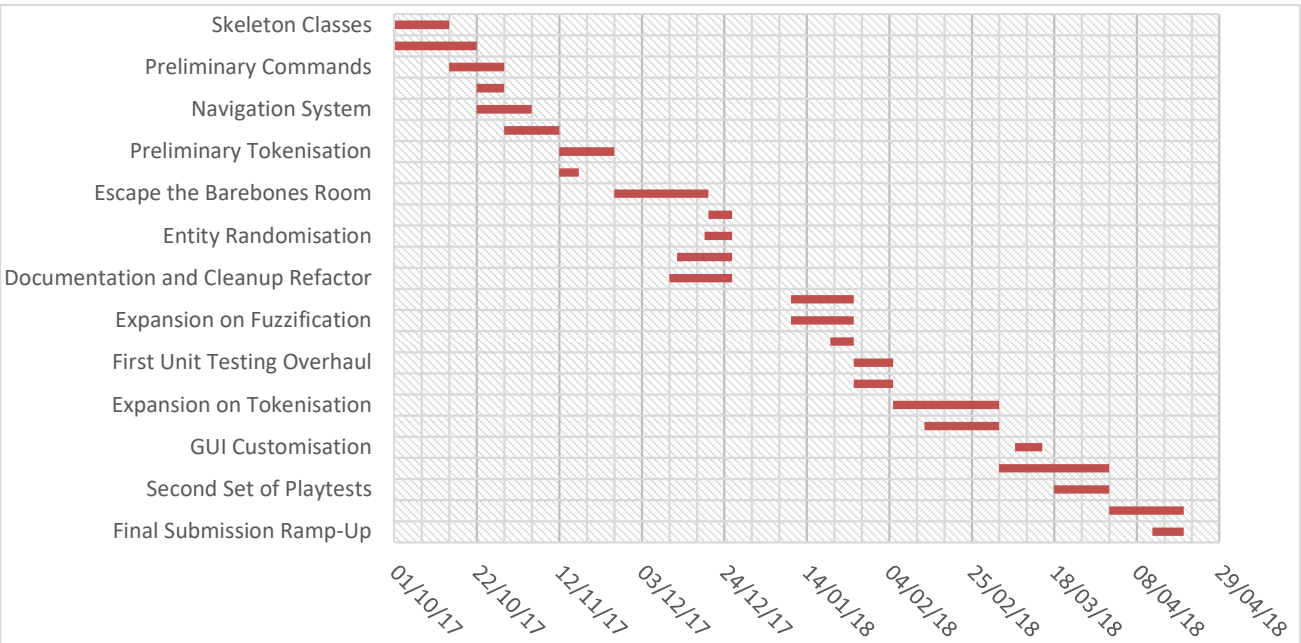
Can be found as TerrificDisposition.png in the Diagrams folder found within the Documentation folder.

SCHEDULING

ORIGINAL SCHEDULE



The initial schedule emphasised working on both the fuzzy and the nlp components while also working on other components in a more paced out structure. It also delineated a very specific time for unit and play testing.



The revised or final schedule shows that the actual schedule became working on separate components at any given time, while also spreading out the documentation and testing during a larger period. There was also a larger break during the Christmas or winter break due to family time than was originally scheduled.

GITHUB

https://github.com/theShanks96/Terrific-Disposition.git