NEA Proposal

# Contents

### [Overview](#_Overview)

### [Features](#_Features)

#### [Graphical User Interface](#_Graphical_User_Interface)

#### [Login System](#_Login_System)

#### [Leaderboard](#_Leaderboard)

#### [Inventory System](#_Inventory_System)

#### [Chance Actions](#_Chance_Actions)

#### [Adaptive Rooms](#_Adaptive_Rooms)

#### [Adaptive Map](#_Adaptive_Map)

### [Stakeholders](#_Stakeholders)

#### [Internal](#_Internal)

#### [External](#_External)

### [Similar Games](#_Similar_Games)

#### [Zork](#_Zork)

#### [The Hobbit](#_The_Hobbit)

#### [Pokémon Red/Blue/Yellow (1999)](#_Pokémon_Red/Blue/Yellow_(1999))

### [Limitations](#_Limitations)

# Overview

[TITLE] is a classic-style text adventure game, written in Java. It consists of entering commands to perform actions within the game, while the world is continuously described by the program, who is the narrator. There will be loot and weapons to find, enemies to defeat, and puzzles to solve. There will be enemies of varying difficulty to battle against, and a final boss guarding the finish for the game.

# Features

## Graphical User Interface

There will be a relatively simple GUI for the game, where the narrator will talk to the player, and the player will enter commands. At the top of the screen will be the player’s health, and a box where a dice roll animation will be played when actions relying on chance are performed.

## Login System

When the game first loads, there will be a login screen where the user can register/login to an account. When registering an account, (if it is an available username) the program will hash the password and store it with the username in a local file. When logging in, the program will hash the entered password and check it against the one stored, and if they do not match, the user will not be able to log in.

Due to user details being stored in a file on the computer, it will only be local to that machine, and users will have to make new accounts for separate machines.

## Leaderboard

From the main menu, users will be able to access the leaderboard for the machine they are using. Players’ highest scores will be stored with their username and password, and displayed in a list from greatest to least.

## Inventory System

Throughout the game, there will be items the player can pick up. When they do, the item will be added to their inventory – a space where all their items are stored. At any point, players will be able to check their inventory, and also try to use items from it. Some items will be finite (such as food), meaning the quantity decreases when it is used. However, others will be infinite (such as swords) and will have unlimited uses.

When an item is added to the inventory, it will be sorted into an appropriate category (e.g. consumables, weapons), and then sorted alphabetically within their categories based on an insertion sort algorithm.

## Chance Actions

In the game, there will be items and weapons that perform actions based upon chance, similar to the table-top game Dungeons & Dragons.

For example, there could be a sword that has a 25% chance to inflict a critical hit. This would require a roll of at least 15 on a twenty-sided dice. When the player uses the sword, a dice rolling animation would play at the top of the GUI. If the roll was sufficient, the critical hit would be performed.

Other items that would utilise this could be potions, spells, armour, etc. It would also be utilised during attacks from enemies.

## Adaptive Rooms

As the player moves between rooms, the narrator will describe them slightly differently depending on what the player has already done there. For example, if the player has already taken an item from a room, the program will remember and not tell the player about it next time. In addition, if a player has already defeated an enemy, the description will reflect this.

## Adaptive Map

At the top of the GUI, there will be a small map of the world. At the beginning, this map will only show the room the player starts in, but will expand as they move around to reveal more. The map will be constructed from ASCII pipe characters, with each room taking up a predefined number of rows and columns.

To see if the player has explored a room, each Room object will have an “explored” Boolean value. If the value is true, then that room will be displayed on the map. There will also be numbers in each room, with a key on the side giving the names of the rooms.

### Map Example

A diagram of a house

Description automatically generated

Key:

* Wall
  + │ ┌ ─ ┐ ├ ┤ ┼ └ ┘ ┬ ┴
* Door
  + ═ ║

# Stakeholders

## Internal

### Developer (me)

I will be developing the game, and also playing it for testing.

### Younger Players

Younger players are players aged under 35. They will likely be playing as a large proportion of people in that age group play video games often.

#### Requirements

* Intuitive controls

**Why**: as younger players are unlikely to have come across text-based adventure games before, it’s important to make the controls (commands and parameters) easy to understand to prevent players from becoming frustrated and not wanting to play.

**How**: the game will output a list of commands and parameters, along with their descriptions, at the beginning. For each command, there will be a list of acceptable aliases (for example, to pick up an item, a player could either type “pickup”, or “take”, etc.). The game will also try to explain to the player what they typed wrong if there was a syntax error in their command.

* Challenging sections

**Why**: to prevent the player becoming bored, it’s important to have challenging sections. This is especially important for younger players as they will likely be used to modern games that often have very complex problems.

**How**: the game will contain challenging puzzles and enemies that are difficult to defeat.

* Fun combat system

**Why**: as younger players will likely be used to modern games such as first-person shooters, the game will need a combat system that is fun and will keep the player engaged.

**How**: the combat system needs to be a balance between challenging, easy to understand, and luck based. To accomplish this, it will undergo lots of testing with different people, and their views will be recorded, and the game adjusted to help. I will also take inspiration from popular games using a similar system, such as Pokémon.

### Older Players

Older players are players aged 35 and over – this is due to the timeframe when text-based adventures were most popular, as they started to become less so towards 1990, so they are the ones likely to have played them as children. They will likely be playing due to nostalgia.

#### Requirements

* Classic look and feel

**Why**: as older players will most likely be playing due to nostalgia, it is important that the game will look and feel similar to those they would’ve played as children.

**How**: I will take inspiration from original text-based adventure games such as Zork and The Hobbit, incorporating key elements such as old-fashioned fonts and looking like a console.

* Engaging storyline

**Why**: as older players are typically more invested in the story of a game compared to younger players, it’s important to maintain its quality throughout the game to keep them engaged.

**How**: I will plan the entirety of the story out before hand and run it past a small selection of people that would be likely to play the game. I will them make adjustments according to their feedback.

## External

### PEGI

PEGI (Pan-European Game Information) is an organisation that rates games for age suitability based on their content. They will be playing to judge what ages the game is suitable for.

#### Requirements

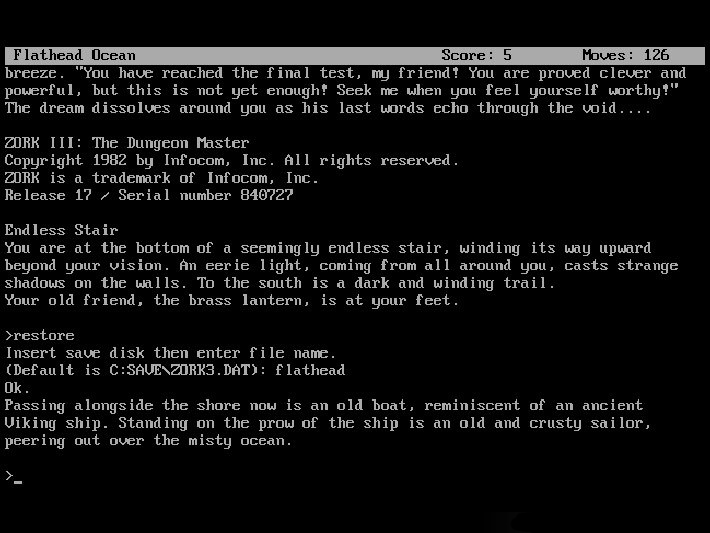
* Follow the PEGI Code

**Why**: all games distributed within the EU must comply with the PEGI Code in order to receive a PEGI age rating. As these age ratings can be highly influential on whether people can play the game or not, it’s important the game complies.

**How**: the game will be in compliance with the applicable data protection and privacy laws. It will also not contain any monetisation methods.

# Similar Games

## Zork



Zork is a text-based adventure game, released in 1977. It was extremely popular, and likely the most famous one to this date. In the game, the player explores the ruined Great Underground Empire and must complete puzzles and move around the map to proceed. To interact with the world, the player enters commands into a command line interface, the computer then responds to them, acting as the story’s narrator. The commands it accepts are very complex, allowing things like “north” and “get lamp”, but also “put the lamp and the sword in the case”.

It followed a linear storyline, meaning that all players would have to complete the same challenges in the same way, in the same order. The entirety of the first title took, on average, two and a half hours to complete.

### Features

#### Complex Command System

The command system in Zork is very complex, allowing players to type commands in virtually plain English. This was very effective as it meant players didn’t have to read any instructions before playing. This makes the game extremely user-friendly and would’ve definitely helped the game in popularity when it was released.

I will not be incorporating this feature into my game, as I do not have the time available that it would take to implement. This will result in my game being less user-friendly, as players will likely have to review a list of commands at the start, so they know what is allowed. To mitigate the effect of this, I will be adding aliases for the commands – common alternatives for the keywords. For example, you could either type “take” or “pickup”, and they would perform the same action.

#### Simple GUI

The user interface in Zork is extremely simple, only being a command line. While this may not be as interesting to those used to modern video games, it keeps the code simple and allows the program to run very efficiently and consume minimal resources.

I will be incorporating this feature into my game, as I like the aesthetics of a classic command line interface, and I believe it could attract lots of older players towards my game due to their nostalgia for it. In addition to this, I do not think the time investment required to create images for each room would be worth it, as it would make it harder to add to the program later on.

#### Point System

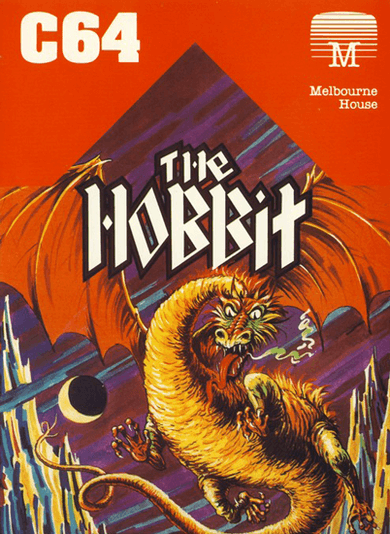
Zork uses a point system to score players. You can get points for solving puzzles, acquiring treasures, performing certain actions, and visiting certain locations. This is useful because it allows the player to gauge how well they have played the game, and compare their score to other players and even compete to get the best one. This point system is not stored anywhere.

The game also keeps track of how many moves a player has taken – again something players can use to compare with friends and even perform speedruns on the game.

These are both features that I would like to incorporate into my game, because I feel like they’re very fun for the player to be able to see and compete with. However, I will be adding to these features by storing the number of points and moves for each player in a global database. These values will then be visible from the starting screen, along with the players’ usernames.

## The Hobbit

A screenshot of The Hobbit for the Commadore 64. In the upper half of the screen, there is simple computer drawing of trees, and a cauldron surrounded by logs. The floor is coloured red.
In the lower half of the screen, there is a command line where the player can enter commands and receive responses.

The Hobbit is a text-based adventure game, released in 1982. Like most adventure games of the time, it is made up of the player exploring the map, collecting treasures, fighting enemies, and solving puzzles. However, the reason this game was so successful was thanks to its illustrations. The majority of locations throughout the game had custom illustrations that the player could see. These were more detailed on the disk version, but simple outlines and filled colours on home computers to save space. These pictures allowed players to explore easily, and feel more engaged with the story.

In addition to this, it also used a very complex parser, using a subset of English called Inglish. This allowed players to type phrases such as “Ask Gandalf about the curious map then take the sword and kill the troll with it”. This made the game very appealing to players as it was so much easier than having to follow a rigid structure like other games.

### Features

#### Illustrations

The Hobbit displays pictures for the vast majority of locations the player can be. This gives the player a much more detailed story to follow, feeling as if they are actually in the world itself. On the disk version of the game, it used detailed illustrations, but on the home computer versions, it used simple outlines that were filled by a solid colour, to save space.

While I would very much like to incorporate this feature, I will not be able to as I believe it is beyond my capabilities to create illustrations for all the map locations and still complete the program to an acceptable level. However, I will be incorporating a feature inspired by this which is the map. The player will have a map in the top corner of their screen, drawn with ASCII characters, showing them and the areas they have already explored. While not being as effective as the illustrations at engaging the player, it will make it much easier to navigate the world as you won’t have to rely solely on memory.

#### Real Time Gameplay

Unlike other text-based adventure games of the time, The Hobbit utilised a real time system, meaning that things were constantly happening in the background. For example, if a player took too long to enter a command, events had the possibility to happen such as “Thorin sits down and starts singing about gold”. This kept players engaged, as they could not simply leave the game at any point, they had to type a command to pause it. It also allowed the player to feel as if they were within the game as it was not all controlled by them.

I will not be incorporating this feature into my game, as I believe it is beyond my capabilities to program this system and still complete the program to an acceptable level. I also think that, while it is an excellent feature, it is not needed for my game as it will still be very fun and engaging without it.

## Deconstructing: The Battle Music of Pokemon Red and Blue - Jason M. YuPokémon Red/Blue/Yellow (1999)

Pokémon was originally released for the EU in 1999 on the Gameboy. It had a simplistic GUI, featuring your character moving around a world and entering battles with creatures called Pokémon, and other Pokémon trainers. This game was extremely successful, and this was mainly due to its gameplay experience over its graphics, which were dated even at its release. However, these simplistic graphics were said to be one of the game’s biggest draws, forcing people to use their imaginations, making it more engaging.

### Features

#### Combat System

While the general gameplay of Pokémon is not something I will be incorporating as it’s very different from a text adventure’s, the combat system it similar. The player is given a list of options to pick from which each perform different actions. For example, it could be to attack the opponent, or perform a special move to give them more power on their next attack, etc. This is similar to a text adventure, except you are given the options instead of having to type them yourself

In addition to this, Pokémon uses a chance-based combat system. This means that, depending on what random number is generated, the attack or defence will be more/less effective. These are both features that I would like to incorporate, as they make the game more complex and prevent the player from becoming quickly bored – having to come up with different tactics as they go. However, I will be adapting this feature by using a dice to determine the random number, and this will be visible to the player. As well, the player will have to type in their actions instead of selecting them from a list.

#### Graphics

During battles, Pokémon displays a standard battle screen. This screen follows the same format every time, only showing different characters and options. This screen is useful to the player as it is extremely user-friendly, showing them all the available options, as well as their health and the opponents health. It also allows the player to see the opponent they are seeing instead of just reading a name or description. This can make it easier to convey the opponent – whether they are a cute Pokémon they should take pity on, or a frightening and challenging one.

I will not be incorporating this feature into my game as, while it would be a nice addition, I feel it does not follow the classic look and feel of a text adventure. In addition to this, I am not confident in my ability to create even pixel art versions of characters to an acceptable standard, while still spending enough time on the programming. However, I don’t think my game will lose too much from not including this feature as the descriptions of the characters will be comprehensive and the health will be visible after every move.

# Limitations

## Simple Commands

As opposed to games like Zork and The Hobbit, [TITLE] will only accept relatively simple commands, such as “pickup” and “use”. This is due to not having enough time to implement a more complex system into the program while still completing the game to an acceptable level.

## Simple GUI

[TITLE] will use a very simple user interface, being a command line with the player’s health, a map, and a dice at the top. I will not be adding a more intricate interface into the game like in Pokémon or even The Hobbit as I prefer the look and feel of a plain UI – with one of the game’s main selling points being nostalgia for games like Zork. I also feel that the main focus of the game is on the story and adventure, not the UI.