NEA Proposal

# 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 Tiles

As the player moves between tiles, the narrator will describe rooms slightly differently depending on what the player has already done there. For example, if the player has already taken an item from a tile, 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 tile the player starts on, but will expand as they move around to reveal more. The map will be constructed from ASCII pipe characters, with each tile taking up a 3x3 space – the centre being the tile itself, and the edges and corners being part of the surroundings to show walls.

To see if the player has explored a tile, the program will use a HashMap to store the tiles as keys, and Booleans for the value. If the value is true, then that tile will be displayed on the map.

### Map Example

A cross with many windows

Description automatically generated with medium confidence

Key:

* Tile where the player can move to
  + ▒
* 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

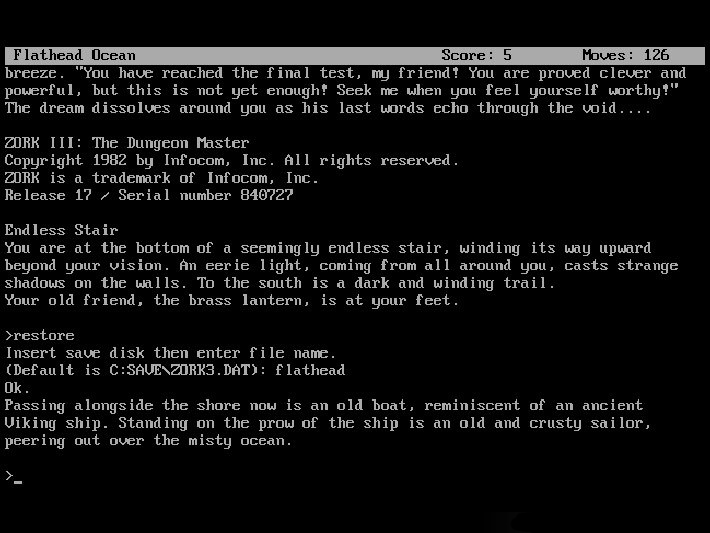
* Follow the PEGI Code

**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 tile would be worth it, as it would make it harder to add to the program later on.