



# **DEVELOPING A TEXT-BASED GAME WITH MODERN TOOLS**

Lappeenranta–Lahti University of Technology LUT

Degree Programme in Software Engineering, Bachelor's thesis

2024

Oskari Suonpää

Examiner: University lecturer Erno Vanhala (D.Sc. Tech.)

# Tiivistelmä

Lappeenrannan–Lahden teknillinen yliopisto LUT

LUTin insinööritieteiden tiedekunta

Tietotekniikan koulutusohjelma

Oskari Suonpää

## **Tekstipohjaisen pelin kehittäminen nykyaikaisilla työkaluilla**

Kandidaatintyö

2024

42 sivua, 13 kuvaa, 0 taulukkoa and XX liitettä

Tarkastaja: Yliopisto-opettaja Erno Vanhala, TkT

Avainsanat: pelimoottorit, tekoälytyökalut, tekstipelit, pelinkehitys

Tässä kandidaatin työssä tarkastellaan modernien työkalujen, kuten pelimoottoreiden ja tekoälyteknologioiden, käyttöä tekstipohjaisen pelin kehittämisessä. Työn keskeinen tavoite on arvioida erityisesti Godot-pelimoottorin, ChatGPT ja GitHub Copilotin hyödyllisyyttä pelinkehitysprosessissa. Työssä kehitettiin prototyyppi tekstipohjaisesta pelistä, jossa näitä työkaluja käytettiin tarinankerronnan, ohjelmoinnin ja pelimekaniikkojen toteuttamisessa.

Kehitysproessin aikana havaittiin, että modernit työkalut voivat merkittävästi tehostaa pelinkehitystä ja madaltaa aloittamiskynnystä erityisesti uusille kehittäjille. Godot osoittautui monipuoliseksi ja käyttäjäystävälliseksi pelimoottoriksi, kun taas ChatGPT tarjosi tukea tarinankerronnassa ja koodauksen neuvoissa. GitHub Copilot puolestaan nopeutti koodin kirjoittamista tarjoamalla reaaliaikaisia koodiehdotuksia ja auttamalla toistuvissa ohjelmointitehtävissä.

Tulokset osoittavat, että vaikka modernit työkalut voivat helpottaa pelinkehitysprosessia, niiden käyttöön liittyy myös haasteita, kuten tarve pysyä ajan tasalla jatkuvasti kehittyvän teknologian kanssa. Työ tuo esiin tärkeitä näkökulmia ja suosituksia, jotka voivat auttaa sekä uusia että kokeneita kehittäjiä hyödyntämään nykyaikaisia työkaluja pelinkehityksessä tehokkaasti ja luovasti.

# Abstract

Lappeenranta–Lahti University of Technology LUT

LUT School of Engineering Science

Degree Programme in Software Engineering

Oskari Suonpää

## **Developing a Text-Based Game with Modern Tools**

Bachelor's thesis

2024

42 pages, 13 figures, 0 tables and XX appendices

Examiner: University lecturer Erno Vanhala (D.Sc. Tech.)

Keywords: game engines, AI tools, text-based games, game development

This bachelor's thesis examines the use of modern tools, such as game engines and AI technologies, in the development of a text-based game. The primary objective of this study is to evaluate the effectiveness of the Godot game engine, ChatGPT, and GitHub Copilot in facilitating the game development process. A prototype text-based game was developed, utilizing these tools to assist in narrative creation, programming, and implementing game mechanics.

Throughout the development process, it was observed that modern tools can significantly streamline game development and lower the barrier to entry, particularly for new developers. Godot proved to be a versatile and user-friendly game engine, while ChatGPT provided valuable support in narrative development and coding advice. GitHub Copilot, on the other hand, accelerated the coding process by offering real-time code suggestions and assisting with repetitive programming tasks.

The results demonstrate that while modern tools can greatly simplify the game development process, there are also challenges, such as the need to stay updated with rapidly evolving technologies. This thesis offers important insights and recommendations that can help both new and experienced developers effectively and creatively leverage modern tools in game development.

## Acknowledgments

I would like to express my deepest gratitude to my supervisor, Erno, for his patience, continuous support, insightful comments, and invaluable guidance throughout this project.

# Symbols and abbreviations

## Abbreviations

AAA	A term used to describe high-budget, high-profile games produced by large, well-known publishers, often characterized by high production values, extensive marketing, and significant commercial expectations
AI	Artificial intelligence
AR	Augmented reality
ASCII	American Standard Code for Information Interchange
CSS	Cascading style sheets
GPT	Generative pre-trained transformer
IDE	Integrated development environment
litRPG	Literary role-playing game
ML	Machine learning
MMORPG	Massively multiplayer online role-playing game
MUD	Multi-user dungeon
NLP	Natural language processing
NPC	Non-player character
permadeath	permanent death, a game mechanic where the game starts over if the player's character dies
UI	User interface
VR	Virtual reality

# Contents

Tiivistelmä

Abstract

Acknowledgments

Abbreviations

Table of Contents 6

Figures 8

1 Introduction 9

2 Introduction to Text-Based Games 12

2.1 Classic Text-Based Games . . . . . 12

2.1.1 The Oregon Trail (1971) . . . . . 12

2.1.2 Colossal Cave Adventure . . . . . 13

2.1.3 Zork: The Great Underground Empire . . . . . 14

2.2 Multi-User Dungeons . . . . . 15

2.2.1 Multi-User Dungeon the Game . . . . . 15

2.2.2 Achaea, Dreams of Divine Lands . . . . . 15

2.3 ASCII Art as Graphics . . . . . 16

2.3.1 Rogue: Exploring the Dungeons of Doom . . . . . 17

2.3.2 Dwarf Fortress . . . . . 17

2.4 Modern Interactive Fiction . . . . . 18

2.4.1 Fallen London . . . . . 18

2.4.2 A Dark Room . . . . . 19

3 Introduction to Modern Game Development Tools 21

3.1 Game Engines . . . . . 21

3.1.1 Ren'Py . . . . . 21

3.1.2 Godot . . . . . 23

3.1.3 Unity . . . . . 24

3.1.4 Unreal Engine . . . . . 25

3.1.5	Twine . . . . .	26
3.2	Artificial Intelligence Tools . . . . .	26
3.2.1	ChatGPT-4 . . . . .	26
3.2.2	Gemini . . . . .	27
3.2.3	GitHub Copilot . . . . .	28
4	Research Methodology . . . . .	30
4.1	Design Science Research Methodology . . . . .	30
4.2	Problem Identification and Motivation . . . . .	30
4.3	Definition of Objectives for a Solution . . . . .	30
4.4	Design and Development . . . . .	30
4.5	Demonstration . . . . .	31
4.6	Evaluation . . . . .	31
5	Development of the Prototype Game . . . . .	32
5.1	Idealizing the game . . . . .	32
5.1.1	Concept . . . . .	32
5.1.2	Gameplay Mechanics . . . . .	32
5.1.3	UI . . . . .	33
5.1.4	Selected tools . . . . .	33
5.2	Development . . . . .	33
5.2.1	UI Setup . . . . .	34
5.2.2	Input Handling and Command Processing . . . . .	34
5.2.3	Location Management . . . . .	34
5.2.4	Event Management . . . . .	35
5.3	Utilization ChatGPT and GitHub Copilot . . . . .	36
5.3.1	ChatGPT . . . . .	36
5.3.2	GitHub Copilot . . . . .	36
6	Discussion . . . . .	38
7	Conclusions . . . . .	40
	References . . . . .	41

## List of Figures

- 1 *The Oregon Trail* (1990 version).
- 2 *Achaea* (2024 browser version).
- 3 *Rogue* (UNIX 5.2.1 version).
- 4 *Dwarf Fortress* (Classic version).
- 5 *Fallen London*.
- 6 *A Dark Room*.
- 7 Ren'Py.
- 8 Ren'Py's main script (script.rpy).
- 9 Godot.
- 10 Unity.
- 11 Unreal Engine.
- 12 Asking ChatGPT-4o to generate code.
- 13 Asking GitHub Copilot to generate code.

# 1 Introduction

The evolution of video game development has been significantly influenced by continuous advancements in technology. From the inception of text-based games in the early days of gaming to today's high-budget, high-profile AAA games with life-like, almost photographic graphics produced by large studios, the tools available to game developers have evolved at an unprecedented rate. These advancements have lowered barriers to entry, enabling more aspiring developers to create their dream games without needing extensive low-level programming skills or writing their own game engines; instead, developers can utilize publicly available frameworks or game engines and focus more on the creative aspects of game development. (Gregory, 2014; Tekinbas & Zimmerman, 2004.)

The integration of artificial intelligence (AI) has also facilitated various aspects of game development. AI tools assist developers by providing solutions to challenges in programming, storytelling, and graphics, thereby streamlining the development process. For instance, AI-driven procedural content generation and machine learning (ML) algorithms have been leveraged to automate repetitive tasks, generate content dynamically, and personalize player experiences, enhancing both the efficiency and creativity of game development. (Yannakakis & Togelius, 2018.)

Although text-based games might seem obsolete compared to AAA titles, they continue to play a significant role in today's gaming landscape. Text-based games, which rely heavily on narrative and player imagination, offer unique benefits such as lower development costs and simpler mechanics, making them an ideal starting point for new developers. Additionally, these games serve as fertile ground for experimentation with game design and storytelling techniques, fostering strong communities of players and developers who appreciate the rich, immersive stories crafted without the need for advance graphics. (Montfort, 2003; Montfort & Bogost, 2009.)

Understanding the utilization of game engines and AI tools is crucial for the future of game development. As technology continues to advance, comprehending these tools can enable developers to create more efficient and innovative games. Game engines like Godot, Unity, and Unreal Engine provide robust platforms for game development, offering pre-built functionalities that save time and effort (Gregory, 2014.) AI tools, including procedural content generation and natural language processing (NLP) algorithms, enhance game design by automating repetitive tasks, generating content dynamically, and personalizing player experiences (Yannakakis & Togelius, 2018.) A comprehensive study of these tools will benefit individual developers and contribute to the overall growth and innovation within the gaming industry.

To explore the practical application of game engines and AI tools, a prototype text-based game was developed to assess their utility in practice. This project involved using the popular game engine Godot to manage the game's core mechanics and employing AI tools like ChatGPT and GitHub Copilot to assist with tasks such as narrative generation and debugging. The development process provided valuable insights into the practical benefits and challenges of using these modern tools in game development.

The primary objective of this thesis is to examine the modern tools—particularly game engines and AI technologies—currently available for video game development, understand how they can be utilized to develop a prototype text-based game, and analyze the benefits they offer to game developers. By delving into these aspects, this thesis aims to guide both new and experienced developers interested in leveraging the latest technologies to create games.

This research aims to address the following questions:

1. What are text-based games?
2. What modern tools are currently available for video game development?
  - How can these tools be utilized to develop a text-based game?
  - What benefits do these tools offer to game developers?

## Structure of the Thesis

Chapter 2 provides detailed explanations of text-based games, including different types such as text adventures, multi-user dungeons (MUDs), and games that use ASCII symbols to replace traditional graphics, as well as some modern examples. Chapter 3 introduces modern tools for video game development, covering game engines like Unity, Godot, Unreal Engine, and Twine, and AI tools like ChatGPT-4 and GitHub Copilot. Chapter 4 details the research methodology employed in this thesis, specifically the Design Science Research Methodology (DSRM), and describes its application in developing the prototype text-based game. Chapter 5 discusses the development of the prototype game, focusing on the use of the selected tools and their effectiveness. Chapter 6 provides a discussion of the key findings, compares them with existing literature, and answers the research questions posed at the start of the thesis. Chapter 7 concludes the thesis, offering a summary of the results and suggestions for future work.

Chapter 2 provides detailed explanations of text-based games, including different types such as text adventures, multi-user dungeons (MUDs), and games that use ASCII symbols to replace traditional graphics, as well as some modern examples. Chapter 3 introduces modern tools for video game development, covering game engines like Unity, Godot, Unreal Engine,

and Twine, and AI tools like ChatGPT-4 and GitHub Copilot. Chapter 4 details the research methodology employed in this thesis, specifically the Design Science Research Methodology (DSRM), and describes its application in developing the prototype text-based game. Chapter 5 discusses the development of the prototype game, focusing on the use of the selected tools and their effectiveness. Chapter 6 provides a discussion of the key findings, compares them with existing literature, and answers the research questions posed at the start of the thesis. Chapter 7 concludes the thesis, offering a summary of the results and suggestions for future work.

## 2 Introduction to Text-Based Games

Text-based games are a genre where the world and narrative are represented primarily through text or ASCII art. These games often offer narrative-driven experiences in which the player makes decisions that influence the outcome of the story. Despite their simplicity compared to modern AAA titles, text-based games laid the foundation for interactive fiction and continue to thrive, particularly in niche gaming communities. In this chapter, we will explore classic text-based games, MUDs, the use of ASCII art as graphics, and modern interactive fiction.

### 2.1 Classic Text-Based Games

Classic text-based games provided the foundation for many game genres today, offering immersive narratives and challenging gameplay via simple text interfaces. These games were limited by the technology available at the time, yet they capitalized on creative storytelling and decision-making mechanics.

#### 2.1.1 The Oregon Trail (1971)

*The Oregon Trail*, developed by Don Rawitsch, Bill Heinemann, and Paul Dillenberger in 1971, is an educational text-based strategy game written in BASIC. Set in 1847, the player assumes the role of a wagon master leading a party of settlers from Independence, Missouri, to Oregon City, Oregon, via the Oregon Trail, a historically significant route. The game was notable for blending entertainment with education, teaching players about the challenges faced by pioneers, and it became a popular tool in American schools. (Bouchard, 2017.)



Figure 1: *The Oregon Trail* (1990 version).

The game begins in Independence, Missouri, where the player plans for the journey ahead, deciding when to set out and what supplies to buy from a merchant. The journey on the Oregon Trail takes 16 two-week turns, with the player deciding what to do on each turn based on information given by the game, as shown in Figure 1. The options include hunting, continuing the journey, and stopping at a fort. After each turn, the player chooses how to ration the food among the settlers. Between turns, there is a chance for misfortunes, such as losing supplies. This blend of decision-making, management, and random events set the foundation for future strategy games. (Bouchard, 2017.)

### 2.1.2 Colossal Cave Adventure

*Colossal Cave Adventure*, also known as *ADVENT* or *Adventure*, was developed by Will Crowther and later expanded by Don Woods in the late 1970s, written in FORTRAN IV. This game is considered the progenitor of both the text adventure and adventure game genres. It established many conventions that would define the genre, such as the use of text-based commands to interact with the environment. (Reed, 2021a.)

“You are standing at the end of a road before a small brick building. Around you is a forest. A small stream flows out of the building and down a gully.”  
(*Colossal Cave Adventure*’s opening text.)

The game begins with the player standing at the end of a road before a small brick building. The player must navigate through various locations within the world, using text commands

to interact with the environment. Commands are typically two words, such as “go north” or “take lamp”. *Colossal Cave Adventure* was groundbreaking in its ability to immerse players in a vast, fictional world using nothing but text, and it paved the way for later text-based adventure games like *Zork*.

### 2.1.3 Zork: The Great Underground Empire

*Zork: The Great Underground Empire*, also known as *Dungeon*, was developed by Tim Anderson, Marc Blank, Bruce Daniels, and Dave Lebling in the late 1970s. Written in MDL, the game brought the text adventure genre to a larger audience following the early 1980s personal computer revolution. Similar to *ADVENT*, it sets the player in an underground world in search of treasure while encountering challenges such as puzzles, traps, and hostile creatures.

“West of House You are standing in an open field west of a white house, with a boarded front door. There is a small mailbox here.

>open mailbox

Opening the small mailbox reveals a leaflet.

>read leaflet

(Taken)

“WELCOME TO ZORK!

ZORK is a game of adventure, danger, and low cunning. In it you will explore some of the most amazing territory ever seen by mortals. No computer should be without one!” (Zork: *The Great Underground Empire*’s opening text and usage of commands.)

Similar to *ADVENT*, the game begins with the player standing in an open field west of a white house, with a boarded front door. The player must navigate through various locations within the world, using text commands to interact with the environment, as shown in the above quote. Commands are typically two words, such as “open mailbox” or “take leaflet”, while more complex commands can also be made, such as “what is a grue” or chained commands like “open mailbox and take leaflet”. (Reed, 2021b.)

## 2.2 Multi-User Dungeons

Multi-User Dungeons (MUDs), named after the game *Multi-User Dungeon*, are multiplayer real-time games. They are typically text-based, where players can interact with each other and the environment in a persistent shared world.

### 2.2.1 Multi-User Dungeon the Game

*Multi-User Dungeon*, referred to as *MUDI* and *British Legends*, was developed by Roy Trubshaw and Richard Bartle in the late 1970s. Initially written in MACRO-10 and later in BC-PL/MUDDL, it is the first-ever multiplayer text-based game and is considered the progenitor of the MUD and massively multiplayer online role-playing game (MMORPG) genres. The game brought players into a persistent and shared world where they could interact in real-time.

“Welcome! By what name shall I call you? \*Levensius

This persona already exists - what’s the password?

\*Password

Yes!

Your last game was today at 9:21:38.

Hello again, Levensius the champion!

Narrow road between lands.

You are standing on a narrow road between The Land and whence you came. To the north and south are the small foothills of a pair of majestic mountains, with a large wall running around. To the west, the road continues, where in the distance you can see a thatched cottage opposite an ancient cemetery. The way out is to the east, where a shroud of mist covers the secret pass by which you entered the Land.

” (*MUDI*’s opening text.)

The game starts with the player standing on a narrow road, offering a choice of directions to explore. Players navigate through this world by typing text commands, similar to those in *Zork* and *ADVENT*, like “north” or “get sword”. (Reed, 2021c.)

### 2.2.2 Achaea, Dreams of Divine Lands

*Achaea, Dreams of Divine Lands*, also known simply as *Achaea*, was developed by Iron Realms Entertainment in the late 1990s. Initially written in the Hourglass language and later

ported to Vortex and then the Rapture Engine using C/C++ and Lua, it is a MUD game offering players a rich role-playing setting. Players can choose the gender, race, and class of their character, become a city or house leader, captain a ship, or pursue a variety of other roles.

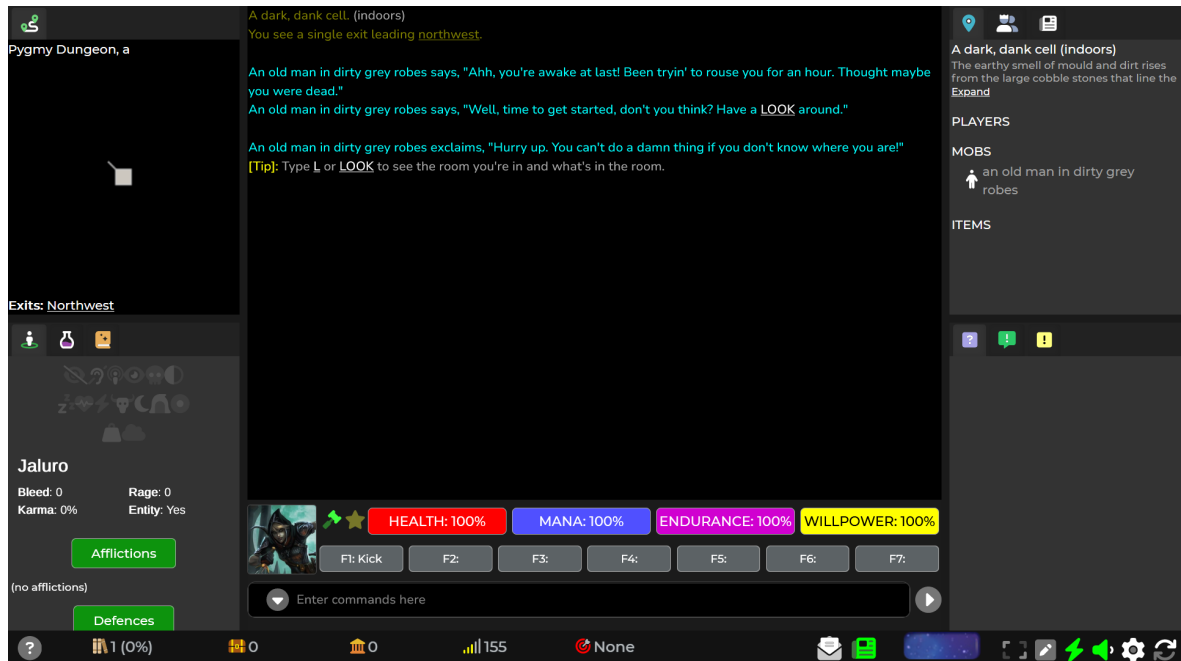


Figure 2: *Achaea* (2024 browser version).

The game starts with the player in a dark, dank cell, where a NPC talks to the player, informing them about their current situation. This serves as an introduction to how the game is played, with occasional tips informing the player what to do. Command words are highlighted and can be clicked to perform the action without typing the commands, although the option exists. The UI of the game, shown in Figure 2, has various elements outside of the main “history” where the game narrates the world, including player statistics, inventory, map, information about the location, and global communication. (Reed, 2021d; Iron Realms Entertainment, n.d.)

### 2.3 ASCII Art as Graphics

While most text-based games represent the world through narrative descriptions, some games use ASCII art to create rudimentary visual representations. ASCII art became popular in text-based games due to its simplicity and the limitations of early computer hardware. By using characters from the American Standard Code for Information Interchange (ASCII), developers could represent environments and characters with minimal resources.

### 2.3.1 Rogue: Exploring the Dungeons of Doom

*Rogue: Exploring the Dungeons of Doom*, also known simply as *Rogue*, was developed by Michael Toy and Glenn Wichman in the early 1980s and initially written in C. As the progenitor of the “roguelike” game genre, it features procedurally generated dungeons where the player fights monsters and collects treasures, with each playthrough being unique. The game also includes the “permanent death” (permadeath) mechanic, where if the player character dies, the game starts over.



Figure 3: *Rogue* (UNIX 5.2.1 version).

Figure 3 represents one of the possible dungeons generated by the game. The player controls a character represented by the “@” symbol in a dungeon that is unique in each playthrough. The dungeon uses different ASCII symbols to represent various elements in the world, for example: periods “.” for floor tiles, dashes “-” and vertical bars “|” for walls, and plus signs “+” for doors, while monsters are represented by capital letters like “S” for snake and “Z” for zombie. The player character is controlled with keyboard inputs. (Craddock, 2016.)

### 2.3.2 Dwarf Fortress

*Dwarf Fortress*, also known as *Slaves to Armok: God of Blood Chapter II: Dwarf Fortress*, was developed by brothers Tarn and Zach Adams of Bay 12 Games. First released in 2006 and written in C/C++, the game features a highly detailed simulation of a dwarven colony.



Figure 4: *Dwarf Fortress* (Classic version).

The player begins the game with a group of seven dwarves tasked with creating a thriving fortress in a hostile world, simulated in intricate detail. The world is presented as ASCII letters and symbols, as shown in Figure 4, with each identifying a specific object, creature, or tile. The gameplay revolves around managing resources, constructing buildings, and ensuring the survival of the colony against numerous threats. (Reed, 2021e; Bay 12 Games, 2024.)

## 2.4 Modern Interactive Fiction

Modern interactive fiction has evolved to include more sophisticated interfaces and narrative techniques, often incorporating visual and auditory elements to enhance the player experience. This section will introduce two modern examples of primarily text-based games; *Fallen London* and *A Dark Room*. On an additional note and not separately introduced, visual novels, a popular subgenre, combine text-based storytelling with static or animated images.

### 2.4.1 *Fallen London*

*Fallen London*, originally released as *Echo Bazaar*, was developed by Failbetter Games and launched in 2009. Written in HTML and JavaScript for the client side, and using React, TypeScript, and NodeJS for the 2019 redesign, the game is set in an alternate 1889, where the player explores an underground version of London, rich with Victorian Gothic themes, interacting with a variety of characters and uncovering numerous stories.

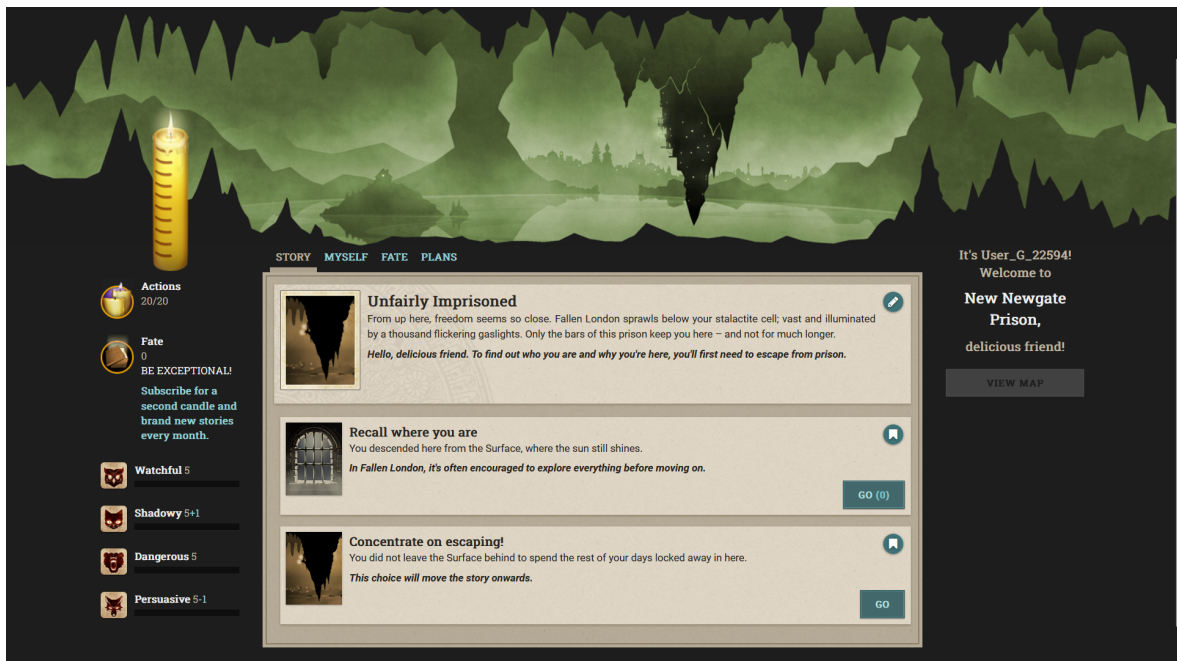


Figure 5: *Fallen London*.

The game begins with the player imprisoned in a cell within New Newgate, London, with the goal of escaping. Overcoming challenges in the game improves the skills of the player, determining the success rate of various actions. The game is choice-based, with the player choosing one of the presented actions at a time, each choice having the possibility for a branching story path. The choices are presented as clickable buttons, as shown in Figure 5. (Reed, 2021f; Failbetter Games, 2024.)

#### 2.4.2 A Dark Room

*A Dark Room*, developed by Doublespeak Games, is a minimalist text-based game. Written in HTML and JavaScript, it begins with the player stoking a fire in a cold, dark room, gradually revealing more complex mechanics and a compelling narrative as the game progresses.

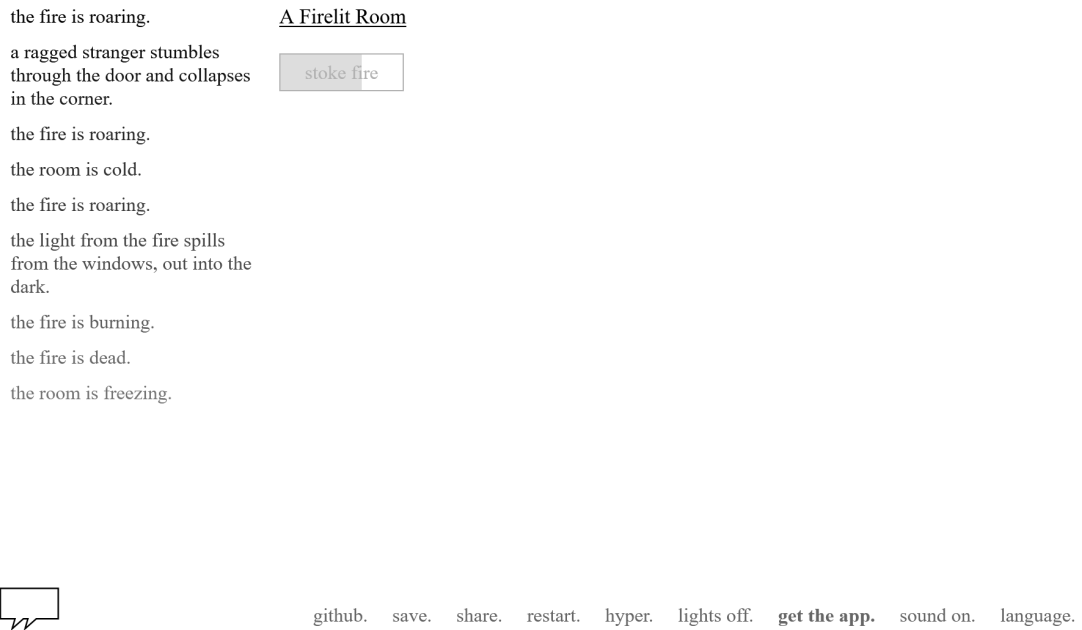


Figure 6: *A Dark Room*.

The game starts with the player in a cold, dark room, tasked with stoking a fire. As the fire grows, so does the player's understanding of the world around them, shown at the left side of Figure 6. Performing actions is done through clickable buttons. As the game progresses, the player must manage resources, build structures, and interact with other characters. (Doublespeak Games, n.d.; GitHub, 2024a.)

## 3 Introduction to Modern Game Development Tools

The field of game development has undergone significant advancements with the advent of modern tools and technologies. This chapter explores several game engines and artificial intelligence tools.

### 3.1 Game Engines

Game engines are software frameworks designed to facilitate the development of video games. They provide a suite of tools and functionalities that allow developers to create interactive experiences without having to build everything from scratch. This section introduces several game engines.

#### 3.1.1 Ren'Py

Ren'Py is a widely used visual novel engine that offers an accessible and robust platform (Figure 7) for creating interactive stories. Developed with the Python programming language, Ren'Py simplifies game development by providing a straightforward scripting language and an extensive library of tools. It allows both novice and experienced developers to design complex narratives with branching storylines, rich multimedia integration, and intricate character interactions. The engine supports various multimedia formats, including images, audio, and video, making it versatile for different types of storytelling. Additionally, Ren'Py's cross-platform compatibility ensures that games developed on this engine can be played on multiple devices, including Windows, macOS, Linux, and mobile platforms, thus broadening the audience reach for developers.

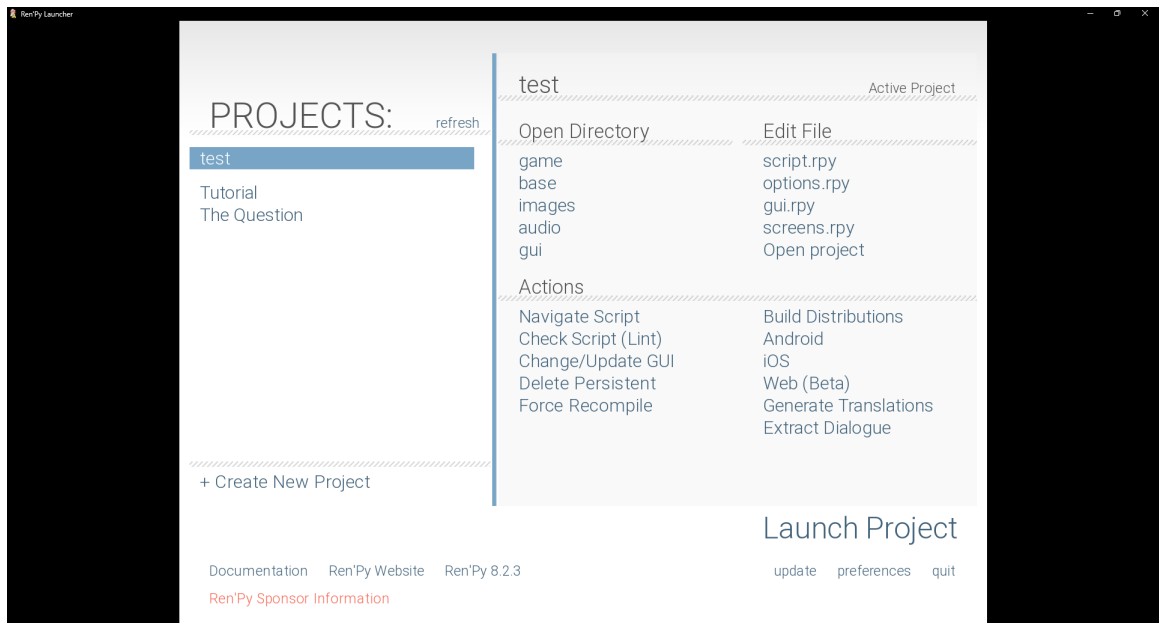


Figure 7: Ren'Py.

```

1 # The script of the game goes in this file.
2
3 # Declare characters used by this game. The color argument colorizes the
4 # name of the character.
5
6 define e = Character("Eileen")
7
8
9 # The game starts here.
10
11 label start:
12
13     # Show a background. This uses a placeholder by default, but you can
14     # add a file (named either "bg room.png" or "bg room.jpg") to the
15     # images directory to show it.
16
17     scene bg room
18
19     # This shows a character sprite. A placeholder is used, but you can
20     # replace it by adding a file named "eileen happy.png" to the images
21     # directory.
22
23     show eileen happy
24
25     # These display lines of dialogue.
26
27     e "You've created a new Ren'Py game."
28
29     e "Once you add a story, pictures, and music, you can release it to the world!"
30
31     # This ends the game.
32
33     return
34

```

Figure 8: Ren'Py's main script (script.rpy).

The flexibility and power of Ren'Py's core scripting capabilities extend beyond, as demonstrated in its main script file (Figure 8), offering extensive customization through Python programming. This allows developers to implement unique game mechanics, sophisticated user interfaces, and advanced features that enhance the player's experience. The Ren'Py community further enriches the engine's ecosystem by contributing a plethora of resources, such as tutorials, assets, and modules, fostering a collaborative environment for learning and innovation. Moreover, Ren'Py's open-source nature ensures continuous improvement and

adaptation to emerging technologies and trends within the visual novel and interactive fiction genres. As a result, Ren'Py stands as a significant tool in the landscape of interactive storytelling, empowering creators to bring their narratives to life with creativity and technical proficiency. (Rothamel & contributors, 2024.)

### 3.1.2 Godot

Godot is a versatile and powerful open-source game engine recognized for its comprehensive toolset (Figure 9) and ease of use in game development. Designed to accommodate both 2D and 3D game creation, Godot offers a flexible and user-friendly environment that supports a wide range of game genres and styles. Its scene system, which uses a node-based architecture, allows developers to create complex game structures with reusable components, enhancing efficiency and scalability in game design. Additionally, Godot's IDE includes a rich array of features such as a visual editor, scripting language (GDScript), and support for C#, providing developers with the flexibility to choose their preferred coding approach.

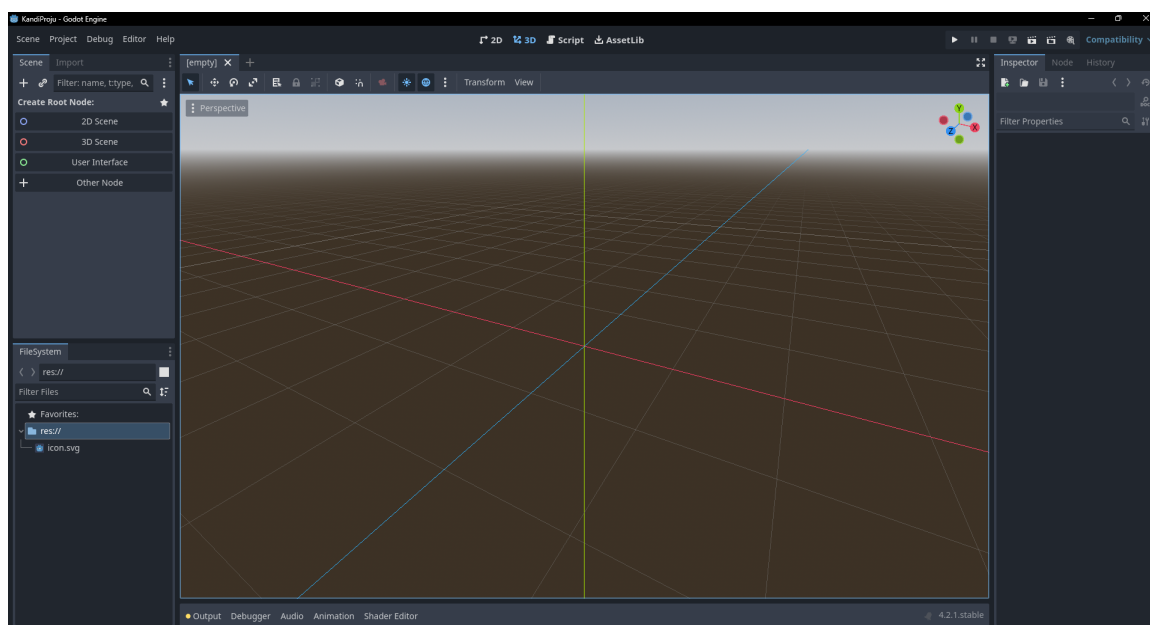


Figure 9: Godot.

Godot excels in its cross-platform capabilities, enabling developers to export their games to various platforms, including Windows, macOS, Linux, Android, iOS, and HTML5. This broad compatibility ensures that games developed with Godot can reach a wide audience. The engine's open-source nature encourages continuous improvement and innovation, driven by contributions from an active community. This collaborative ecosystem provides extensive resources, such as tutorials, plugins, and assets, which support developers at all skill levels in realizing their creative visions. Furthermore, Godot's commitment remain free and open-source underscores its mission to democratize game development, making it an invaluable

asset in the evolving landscape of digital game creation. (Linietsky, Manzur & contributors, 2024.)

### 3.1.3 Unity

Unity is a leading game development platform renowned for its robust capabilities and flexibility (Figure 10) in creating both 2D and 3D interactive experiences. As a comprehensive game engine, Unity supports a wide range of applications beyond gaming, including simulations, VR, AR, and real-time 3D animation. The engine's versatility is facilitated by its component-based architecture, which allows developers to build complex game environments and interactive elements through reusable components. Unity's IDE offers powerful features such as a visual editor, real-time debugging, and extensive asset management, all of which streamline the development process. Furthermore, Unity supports multiple scripting languages, including C#, C/C++, and Rust, which provide developers with the flexibility to implement sophisticated gameplay mechanics and custom features.

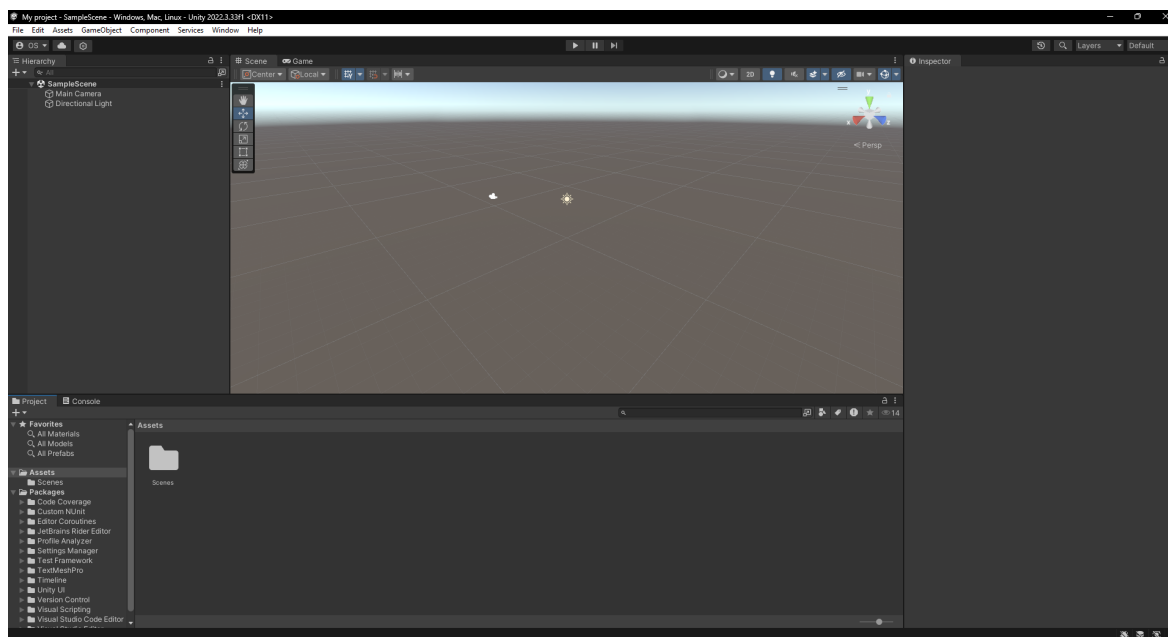


Figure 10: Unity.

One of Unity's most significant strengths lies in its cross-platform compatibility, enabling developers to deploy their projects across a myriad of platforms including Windows, macOS, Linux, iOS, Android, and various game consoles. This expansive reach ensures that creations developed in Unity can access a broad audience. The engine's extensive ecosystem, supported by a large community and a wealth of resources, including tutorials, plugins, and asset stores, empowers developers at all levels to enhance their projects and accelerate their workflows. Unity's ongoing commitment to innovation is evident in its continuous updates and the introduction of cutting-edge features, such as advanced graphics rendering, machine

learning integration, and enhanced collaboration tools. These advancements position Unity as a critical tool in the ever-evolving landscape of digital content creation, fostering creativity and technical excellence among developers. (Unity Technologies, 2024.)

### 3.1.4 Unreal Engine

Unreal Engine, developed by Epic Games, is one of the most powerful and versatile game engines (Figure 11) available today, widely acclaimed for its high-fidelity graphics and extensive feature set. Originally designed for first-person shooters, Unreal Engine has evolved to support a wide variety of game genres and applications, including VR, AR, and film production. The engine's robust rendering capabilities, powered by its state-of-the-art graphics pipeline, allow developers to create visually stunning and highly detailed environments. Unreal Engine's Blueprint visual scripting system provides a node-based interface for designing game logic without requiring extensive programming knowledge, thereby making advanced game development accessible to a broader range of users.

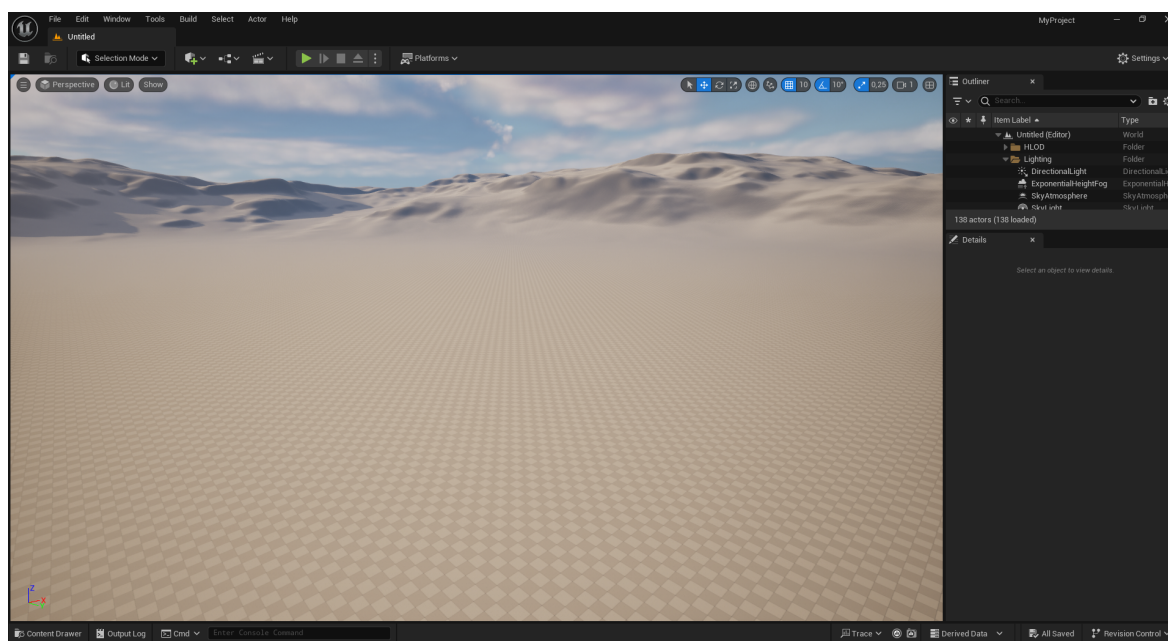


Figure 11: Unreal Engine.

In addition to its superior graphical capabilities, Unreal Engine excels in its cross-platform functionality, enabling developers to deploy their creations across multiple platforms such as Windows, macOS, Linux, iOS, Android, and major gaming consoles. This versatility ensures that games and applications developed with Unreal Engine can reach a diverse and expansive audience. The engine's open-source model encourages collaboration and continuous innovation, with contributions from a vibrant community of developers and enthusiasts. Unreal Engine also offers extensive resources, including comprehensive documentation, tutorials, and an asset marketplace, which support developers at all skill levels in bringing their

creative visions to life. Epic Games' commitment to advancing Unreal Engine through regular updates and the integration of cutting-edge technologies, such as real-time ray tracing and machine learning, positions it as a premier tool in the field of interactive digital content creation. (Epic Games, 2024.)

### 3.1.5 Twine

Twine is an open-source tool for creating interactive fiction, renowned for its simplicity and flexibility, making it an ideal platform for both novice and experienced writers to develop non-linear narratives. Unlike more complex game development engines, Twine allows authors to focus on storytelling by providing a user-friendly interface that requires no programming knowledge. Through its visual story map, users can easily design branching storylines and interconnected passages, enabling the creation of intricate and engaging narratives. Twine supports various formats, such as HTML, which facilitates seamless integration of multimedia elements, including images, audio, and video, thus enhancing the storytelling experience.

In addition to its ease of use, Twine offers extensive customization capabilities through the incorporation of CSS and JavaScript, allowing developers to tailor the appearance and functionality of their stories. This flexibility enables the creation of unique and interactive experiences that go beyond traditional text-based fiction. Twine's open-source nature fosters a collaborative and supportive community, which contributes a wealth of resources such as tutorials, templates, and extensions that help users enhance their projects. Furthermore, Twine's cross-platform compatibility ensures that stories can be published and accessed across various devices and operating systems, thereby reaching a wide audience. As a result, Twine stands as a significant tool in the field of interactive storytelling, empowering creators to craft compelling narratives with technical and creative freedom. (Fiction Technology Foundation, 2024.)

## 3.2 Artificial Intelligence Tools

AI tools have revolutionized various fields by offering advanced capabilities in NLP, ML, and decision-making. This section introduces several AI tools.

### 3.2.1 ChatGPT-4

ChatGPT-4, developed by OpenAI, represents a significant advancement in NLP and AI. As the latest iteration in the GPT series, ChatGPT-4 builds upon its predecessors with enhanced capabilities in understanding and generating human-like text. Utilizing a transformer architecture, ChatGPT-4 is trained on a diverse and extensive dataset, allowing it to gener-

ate coherent, contextually relevant, and highly sophisticated responses across a wide range of topics, as demonstrated in Figure 12. This makes it an invaluable tool for applications such as conversational agents, content generation, and language translation, where nuanced understanding and generation of text are crucial.

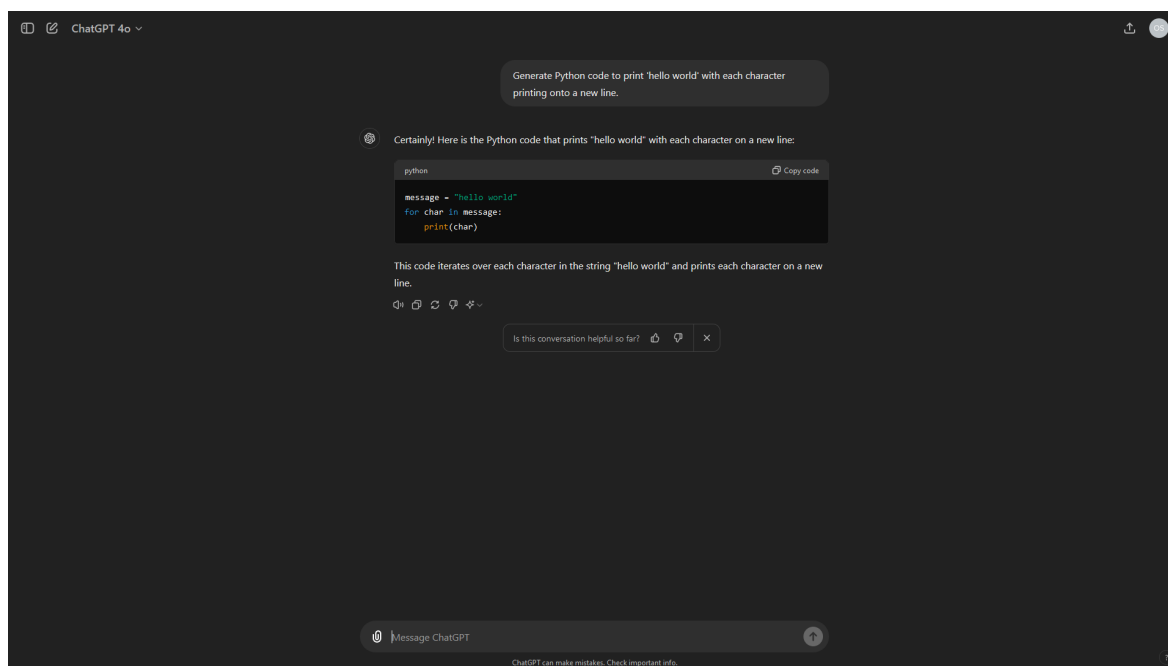


Figure 12: Asking ChatGPT-4o to generate code.

Beyond its core capabilities, ChatGPT-4 integrates advanced features such as improved contextual awareness and the ability to handle more complex queries and tasks. It excels in maintaining context over longer conversations, providing users with more accurate and relevant responses. The model also supports fine-tuning, enabling developers to customize it for specific applications and domains. OpenAI has prioritized ethical considerations in the development of ChatGPT-4, incorporating guidelines and mechanisms to mitigate biases and ensure safe and responsible AI use. The broad applicability of ChatGPT-4, combined with its sophisticated language understanding, positions it as a leading tool in the field of AI-driven text processing and generation. (OpenAI, 2024.)

### 3.2.2 Gemini

Gemini, developed by Google DeepMind, represents a cutting-edge advancement in AI and ML. As an AI model, Gemini is designed to integrate advanced language processing capabilities with deep reinforcement learning, enabling it to excel in a wide range of complex tasks. Unlike traditional AI models that rely primarily on supervised learning from large datasets, Gemini combines this with reinforcement learning techniques to adapt and improve its performance through interaction with dynamic environments. This hybrid approach allows Gemini to perform exceptionally well in both understanding and generating human-like

text and in making decisions based on contextual cues, making it a versatile tool for applications such as conversational AI, decision support systems, and real-time problem-solving.

One of the distinguishing features of Gemini is its ability to maintain high levels of contextual awareness and coherence over extended interactions. This capability is crucial for applications requiring sustained and meaningful engagement, such as virtual assistants, customer service bots, and educational tools. Furthermore, Gemini's architecture is designed to be scalable and adaptable, allowing it to be fine-tuned for specific domains and tasks, enhancing its relevance and effectiveness in specialized applications. Google DeepMind has also emphasized ethical AI development in Gemini, implementing robust measures to ensure transparency, fairness, and the mitigation of biases. As a result, Gemini stands as a leading innovation in the AI landscape, driving forward the integration of advanced language understanding and interactive decision-making capabilities. (Google, 2024.)

### 3.2.3 GitHub Copilot

GitHub Copilot, developed by GitHub in collaboration with OpenAI, represents a groundbreaking advancement in software development tools. This AI-powered coding assistant leverages the OpenAI Codex to provide real-time code suggestions, completions, and even generate entire functions based on the context of the code being written. Integrated directly into popular code editors like Visual Studio Code, as demonstrated in Figure 13, GitHub Copilot is designed to enhance developer productivity by automating routine coding tasks and offering intelligent code snippets, thereby allowing developers to focus on more complex and creative aspects of software development. This integration exemplifies the practical application of AI in programming, significantly reducing the time and effort required for coding and debugging.

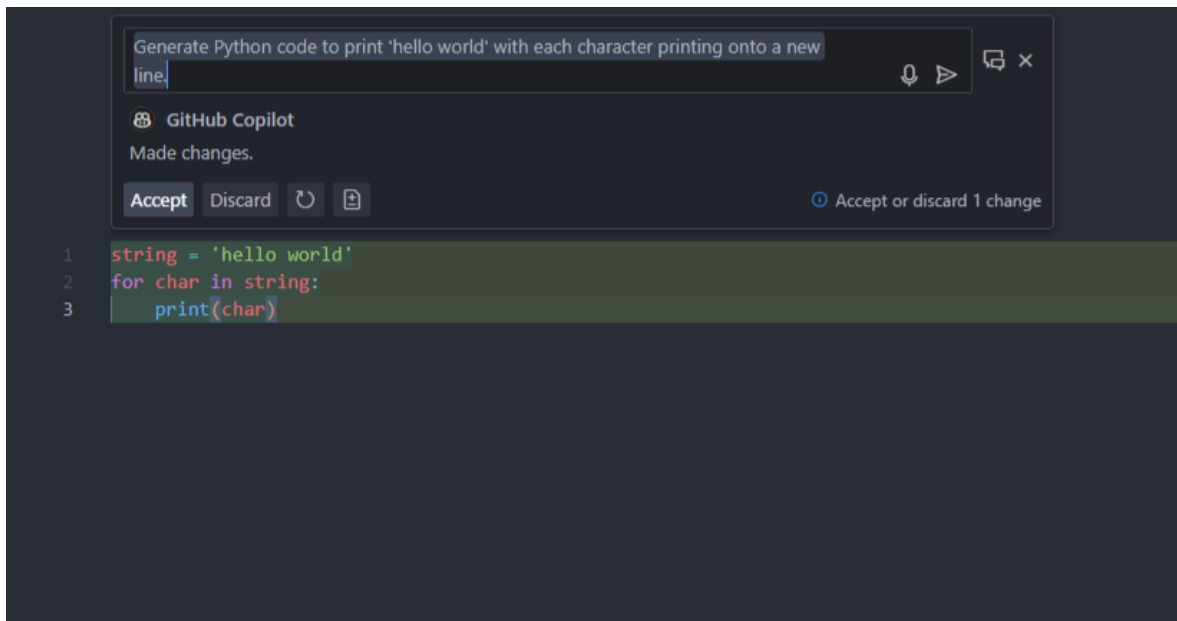


Figure 13: Asking GitHub Copilot to generate code.

Beyond its core functionality, GitHub Copilot is capable of understanding and generating code in multiple programming languages, making it a versatile tool for a diverse range of development environments. It continuously learns from the vast amount of publicly available code on GitHub repositories, improving its suggestions and adapting to the unique coding styles of individual developers. GitHub has also implemented robust measures to address potential ethical and security concerns, ensuring that Copilot adheres to privacy standards and does not inadvertently suggest insecure or inappropriate code. As a result, GitHub Copilot not only boosts efficiency and innovation in the development process but also maintains the integrity and security of the codebase. This advanced AI tool underscores GitHub's commitment to leveraging AI to transform the landscape of software development, making coding more accessible and efficient. (GitHub, 2024b.)

## 4 Research Methodology

This chapter defines the research methodology used in this thesis, focusing on the development and evaluation of a text-based game using modern tools. The chosen methodology is the Design Science Research Methodology (DSRM), which provides a structured framework for addressing the research problem and achieving the thesis' objectives.

### 4.1 Design Science Research Methodology

DSRM is a recognized approach in information systems research, designed to create and evaluate outcomes that solve identified problems. It features six steps: problem identification and motivation, definition of objectives, design and development, demonstration, evaluation, and communication. This methodology is suited for this thesis as it emphasizes the creation and assessment of technological solutions. (Peppers et al., 2007.)

### 4.2 Problem Identification and Motivation

The main problem addressed in this thesis is the need for efficient tools to facilitate the development of video games, text-based games in this case. Despite advancements in game development technologies, a gap persists in accessible, cost-effective tools that allow new developers to create engaging games without requiring extensive programming knowledge. This thesis aims to bridge this gap by evaluating the combined utility of Godot, ChatGPT, and GitHub Copilot in the development process.

### 4.3 Definition of Objectives for a Solution

The objectives of this thesis are:

1. To assess the effectiveness of Godot as a game engine for text-based game development.
2. To evaluate the utility of ChatGPT in both storytelling and game structure.
3. To determine the contribution of GitHub Copilot in facilitating coding tasks.

These objectives align with the broader goal of identifying and promoting tools that lower the barriers to entry for new game developers.

### 4.4 Design and Development

The design and development phase involved creating a prototype text-based game using the following selected tools:

- **Godot:** Chosen particularly for its open-source nature and promise to remain free of charge, and Python-like scripting language GDScript, making it suitable for rapid prototyping.
- **ChatGPT (version 4o):** Utilized for narrative generation, helping to craft the game's story and dialogue, and general help with game structuring and coding.
- **GitHub Copilot:** Employed to assist with coding tasks, providing real-time code suggestions and completions based on the context.

The game concept centers around a student who awakens in an underground shrine and must navigate a dungeon as the chosen champion of a deity named Kaelix who bestowed his powers to them. The gameplay mechanics include text-based commands for navigation and interaction, inspired by classic text adventures with the addition of “role-playing game” elements of titles like Pokémon and Final Fantasy. Additionally featuring a rather unique ‘core swap’ mechanic that would allow the student to transform between the ‘cores’ they have gathered during their journey, thus allowing them to use the powers of the various monsters inhabiting the world.

## 4.5 Demonstration

The demonstration phase of this project involved creating a working prototype of the text-based game using the selected tools. The prototype allowed for practical evaluation of how well Godot, ChatGPT, and GitHub Copilot could be integrated into the development process. The game was tested for functionality, user experience, and overall performance, ensuring that the core mechanics and narrative elements worked seamlessly together.

## 4.6 Evaluation

The evaluation of the prototype focused on several key criteria: usability, efficiency, and effectiveness of the tools used. Usability testing was conducted by me. Efficiency was measured by the potential time saved using AI tools like ChatGPT and GitHub Copilot for coding and narrative generation. Effectiveness was assessed by the overall quality and functionality of the game, with specific attention to how well the tools supported the development goals.

## 5 Development of the Prototype Game

This chapter explores how modern tools (Godot, ChatGPT, and GitHub Copilot) can be utilized in developing a prototype of a text-based game. The prototype, while incomplete, serves as a tool for evaluating the used tools from the perspective of someone relatively new to game development.

### 5.1 Idealizing the game

This section delves into the process of idealizing the game, including the overall concept, gameplay mechanics, narrative structure, and the UI design. While also including brief reasoning for why the selected tools were chosen.

#### 5.1.1 Concept

The game's concept draws inspiration from the 'transmigration' and 'litRPG' genres, which are prevalent in web novels. The narrative is structured as a text adventure with RPG elements. Initially based on a draft novel authored by me, the story was adapted to better suit the interactive nature of a game, with significant revisions made with the assistance of ChatGPT. The roughly planned story for the game goes as follows: The player takes the role of a student who wakes up in an unfamiliar underground shrine connected to a larger dungeon complex. In the shrine, the player interacts with a statue of the deity Kaelix, who offers them a choice: return to their own world or accept the task of defeating a great evil in exchange for his power. Accepting Kaelix's quest sends the player on a journey of survival through the dungeon and the world beyond, culminating in a final confrontation with the main antagonist.

#### 5.1.2 Gameplay Mechanics

The game was intended to be played through textual inputs, simple commands as in *ADVENT*, with the game narrating the results to the player. The player would use "go" command to navigate between the locations representing the world, each having objects that the player could interact with through commands like "look", "take", and "interact".

The "look" command would provide additional information about the locations and the objects within, adding hints and lore for the player to discover. The "take" command would allow player the possibility of picking up items found in the locations. The "interact" command would allow the player to interact with objects, giving the possibility of adding levers or buttons.

Each location in the game was designed with the potential to trigger either a cutscene or a

combat encounter event. These events could be triggered upon the player entering or exiting a location or interacting with specific objects within the environment. “wait” command was planned to be a way to trigger random combat encounters within locations.

The cutscene events were planned to be a way for predetermined scenes with the player needing to use specific commands to progress in the story. Some cutscenes have choices with each branching the story in a different direction.

The combat encounter events were planned to be turn-based, inspired by the Pokémon and early Final Fantasy titles, the player and enemies having stats, moves, and spells. Determined by the speed stat of the player and the enemies, turns would be taken with the player choosing what they would want to do like use a move, item, spell, or run from combat, while the enemies would randomly use either a spell or a move.

### 5.1.3 UI

The UI was intentionally designed to be straightforward, consisting of two primary sections: one dedicated to displaying the game’s textual output and the other for capturing the player’s input. To improve readability of the text, colors were planned to be used to differentiate various types of game output, and certain words being emphasized with less than (<) and greater than (>) signs.

### 5.1.4 Selected tools

Godot was selected as the game engine primarily due to its ‘forever free’ and open-source nature, which made it an attractive option for this project. Additionally, my familiarity with Python sold the choice as GDScript being Python-like made it more suitable for fast prototyping. While the other introduced game engines would have suited the purpose, the choice was more of a personal preference.

For AI tools, ChatGPT-4 and GitHub Copilot were chosen as I already had paid subscription for both of them, and familiarity on how to use them, also what are their strengths and weaknesses. To use GitHub Copilot while programming, Visual Studio Code was chosen as an external editor that could be used with Godot.

## 5.2 Development

Development of the prototype game began by setting up Visual Studio Code as the external editor for Godot, with GitHub Copilot as an extension. Following a tutorial series on Youtube (jmbiv, 2021.) for a few videos, the game diverged to implement the planned features.

### 5.2.1 UI Setup

The first task was to set up the UI for the game, which included separate sections for the game's output, dubbed game info, and the player's input. The game info section was created using a scroll container node, allowing players to scroll through a limited history of the game's output. A label node, user input, was made a separate instanceable scene to display what the player had written. Likewise, a rich text label node, response, was made a scene to display what the game responded, using BBCode formatting to allow coloring the text. The player input was made using a line edit node that allowed typing.

### 5.2.2 Input Handling and Command Processing

To handle what the player inputted, a command processor, dubbed input manager, was created. The line edit node was then connected to the input manager with an “`_on_text_submitted(new_text: String)`” signal, allowing submitted text to be easily sent to be processed.

The script attached to the line edit node was made fairly simple, having a built-in method to grab the focus, “`grab_focus()`”, within the “`_ready()`” function to let the player get to typing without separately selecting the input field once the game started. Additionally, another “`_on_text_submitted`” signal was added to the script to allow the field to be cleared after being submitted with the built-in “`clear()`” method.

The command processor itself handled the raw submitted input with a “`_parse_input(input: String)`” function to separate the main command and possible arguments from the input. The parsed input was then sent to the “`_process_input(command: String, args: Array)`” function to be handled. Depending on the state of the game, the input would be handled by either the command processor itself or an active event that requires user input. Depending on the game state, the available commands would differ.

### 5.2.3 Location Management

Location manager was made a class to manage location related functionalities. The location manager would be initialized by a path to a floor resource file that acted as a collection of locations, while also having a specific entry and exit locations for if or when other floors would be added to the game. The location manager itself serving as a way for the command processor to access things (items, environmental elements, and events) of the specific location, while also handling movement between locations.

Locations themselves were made as resources. Them being resources, allowed new locations to be created within the Godot editor, while also allowing their data to be modified by exposing variables with the “`export`” keyword. Each location had string variables for name

and description, and data structures for exits, environmental elements, items, and events, and methods tied to them.

Likewise, exits were made as resource to allow the possibility for different types of connections between the locations, however, only simple two-way exits were implemented. Exits had variables for the two locations that it connected together that were set in a method in the location itself to allow easy access to the connected locations for movement purposes.

Environmental elements were resources that acted as objects of interest that the player could interact with to get a bit more detail out of the location they were in. Some elements were given the possibility to be triggers to events when interacted with.

Additionally, item resources also were added to locations and as enemy drops to be something for the player to collect. One type of item was a consumable healing item that could be used to heal the player. Another type of item was a note item that provided player information.

#### 5.2.4 Event Management

Event manager was made a class to manage event related functionalities and serve as a bridge with the other managers of the game and the event resource. It handled the connecting and disconnecting the various signals used for the event to communicate with the manager that would then interact directly with other managers such as the location manager in case the location would change during the event, or let other managers such as input manager to determine how the input should be handled.

Events themselves were split into two categories: cutscenes and combat encounters, both being made as resources. The cutscenes were made as set of steps that would do differing things depending on the type of the step, majority being just variations of narration either through plain text or as dialogues from characters. Some steps would ask for the player to use a specific command for them to proceed or give them a choice with the possibility of affecting the story by branching it.

Combat encounters were made as turn based combat events where the player would encounter with an enemy. Their action order was determined by their stats. During their turn, the player or the enemy would choose a move or a spell from their available move and spell pools to either directly attack their opponent, afflict them with negative status effects, or give themselves positive status effects. The event would end when either the player or the enemy was defeated.

### 5.3 Utilization ChatGPT and GitHub Copilot

AI tools ChatGPT and GitHub Copilot were integral throughout the development of the prototype game, aiding with narrative building, planning, and the implementation of various features, alongside helping with debugging. However, the experience was not without challenges, particularly due to the limitations in the AI tools' knowledge and integration capabilities.

#### 5.3.1 ChatGPT

ChatGPT played a pivotal role in assisting with narrative development and providing coding advice. It was especially helpful in brainstorming story elements, refining character dialogue, and structuring the game's branching story paths. For example, ChatGPT contributed to the conceptualization of the Sanguine Slime combat encounter and its associated cutscene, providing ideas on how to create tension and present the critical choice at the scene's climax.

However, one limitation encountered was that ChatGPT's knowledge was based on information available up until 2023. This meant that some of the advice provided, especially regarding newer features or updates in Godot or other relevant technologies, was outdated. Consequently, not all code suggestions worked seamlessly, requiring additional adjustments and manual fixes. Despite these challenges, ChatGPT proved to be a valuable brainstorming partner, offering creative solutions even when the technical details needed further refinement.

#### 5.3.2 GitHub Copilot

GitHub Copilot significantly sped up the coding process by providing real-time code suggestions as development progressed. Integrated with Visual Studio Code, Copilot was particularly useful for writing repetitive code segments, such as command parsing functions and event handlers, which saved considerable time.

Interestingly, Copilot also contributed to the narrative aspects of the game. While directly editing the resource files within Visual Studio Code, Copilot generated dialogue and narrative text for some of the cutscenes. This included snippets of dialogue for the critical scenes, such as the player's encounter with the Sanguine Slime and the subsequent cutscene. By predicting the context and suggesting appropriate text, Copilot helped streamline the process of filling in narrative details, even though these suggestions often required refinement to align with the game's tone and style.

Like ChatGPT, Copilot also faced challenges due to its training on older data. Some of the code it generated was either outdated or not fully compatible with the current version

of Godot, leading to issues that required manual debugging and tweaking. Despite these hurdles, Copilot's ability to quickly produce boilerplate code and even assist with narrative elements made it a valuable tool in the development process.

## 6 Discussion

This chapter addresses the research questions posed at the start of the thesis, drawing from the findings of the development process and reflecting on the role of modern tools in creating text-based games.

### Research Question 1: What are text-based games?

Text-based games, though seemingly outdated compared to graphical AAA games, continue to provide a unique gaming experience by heavily relying on narrative-driven gameplay and player imagination. The findings of this project reaffirm that text-based games remain relevant, particularly in niche communities where storytelling and simplicity are appreciated. The prototype game developed as part of this thesis highlights the potential of text-based games as a foundation for exploring game mechanics and narratives without the complexity of advanced graphics. By focusing on these elements, text-based games allow developers to experiment with story arcs and gameplay mechanics in a more flexible and accessible environment.

### Research Question 2: What modern tools are available for video game development?

Modern tools like Godot, ChatGPT, and GitHub Copilot have become game-changers for independent developers, lowering the barriers to game development. These tools provide significant support in terms of both the technical aspects (like coding and game engine functionalities) and the creative process (such as storytelling and procedural content generation). Godot, for example, offers a robust, user-friendly environment for building games, while AI tools like ChatGPT assist with narrative development and coding suggestions. The findings suggest that these modern tools not only accelerate the development process but also enable developers to focus more on creativity by reducing the technical burden.

### Research Question 3: How can these tools be utilized to develop a text-based game?

The development of the prototype text-based game demonstrates how these modern tools can be effectively integrated into the game development process. Godot served as the core platform for managing game mechanics and user interface elements, while ChatGPT provided valuable assistance in generating dialogues and storylines. GitHub Copilot, integrated with Visual Studio Code, greatly sped up the coding process by suggesting code snippets and

helping debug issues. These tools work together to streamline development, allowing even a relatively inexperienced developer to create a functioning game prototype.

#### **Research Question 4: What benefits do these tools offer to game developers?**

The primary benefit of these tools is the efficiency they bring to the game development process. For instance, ChatGPT helped generate narrative content and provide coding advice, thus saving time and reducing the need for extensive research or trial-and-error coding. GitHub Copilot's real-time coding suggestions allowed for faster iteration and debugging, while Godot's open-source nature and comprehensive features provided an accessible yet powerful platform for game development. Despite some limitations, such as the need to manually adjust AI-generated content or code, these tools enable developers to produce more creative work in less time, thereby enhancing productivity.

## 7 Conclusions

The project successfully explored the utility of modern tools such as Godot, ChatGPT, and GitHub Copilot in developing a text-based game, highlighting their strengths and limitations. The combination of these tools significantly lowered the barriers to entry for game development, allowing for creative exploration without requiring extensive programming knowledge.

Godot proved to be a robust game engine with its open-source nature and user-friendly environment, making it ideal for text-based game development. ChatGPT and GitHub Copilot enhanced the development process, assisting with narrative creation and code generation. However, the project also demonstrated the importance of staying updated with rapidly evolving AI tools and technologies, as some of the suggestions from ChatGPT and Copilot were outdated or required further manual adjustments.

Looking ahead, future work can expand on the findings of this thesis by integrating more advanced AI models, developing complex game mechanics, or deploying the game across various platforms. Engaging a broader community of developers and players could also provide valuable feedback, improving usability and gameplay experiences.

Overall, the tools used in this project offer promising opportunities for both new and experienced developers, allowing them to focus more on creativity by reducing technical complexity.

## References

- Bay 12 Games (2024). *Dwarf Fortress*. [Webpage]. [Accessed: 2024-06-23]. URL: <http://www.bay12games.com/dwarves/>.
- Bouchard, R. P. (2017). *How I Managed to Design the Most Successful Educational Computer Game of All Time*. [Webpage]. [Accessed: 2024-06-21]. URL: <https://medium.com/the-philipendium/how-i-managed-to-design-the-most-successful-educational-computer-game-of-all-time-4626ea09e184>.
- Craddock, D. L. (2016). *Procedural Dungeons of Doom: The Making of Rogue - Chapter 2*. [Webpage]. [Accessed: 2024-06-21]. URL: [https://episodiccontentmag.com/2016/06/10/rogue\\_chapter2/](https://episodiccontentmag.com/2016/06/10/rogue_chapter2/).
- Doublespeak Games (n.d.). *A Dark Room*. [Webpage]. [Accessed: 2024-06-23]. URL: <https://adarkroom.doublespeakgames.com/>.
- Epic Games (2024). *Unreal Engine*. [Webpage]. [Accessed: 2024-06-24]. URL: <https://www.unrealengine.com/en-US>.
- Failbetter Games (2024). *Fallen London*. [Webpage]. [Accessed: 2024-06-23]. URL: <https://www.failbettergames.com/games/fallen-london>.
- Fiction Technology Foundation (2024). *Twine*. [Webpage]. [Accessed: 2024-06-24]. URL: <https://twinery.org/>.
- GitHub (2024a). *A Dark Room*. [GitHub repository]. [Accessed: 2024-06-23]. URL: <https://github.com/doublespeakgames/adarkroom>.
- GitHub (2024b). *GitHub Copilot*. [Webpage]. [Accessed: 2024-06-25]. URL: <https://github.com/features/copilot>.
- Google (2024). *Gemini*. [Webpage]. [Accessed: 2024-06-25]. URL: <https://gemini.google.com/app>.
- Gregory, J. (2014). *Game engine architecture*. CRC Press. ISBN: 978-1-4665-6001-7.
- Iron Realms Entertainment (n.d.). *Achaea*. [Webpage]. [Accessed: 2024-06-22]. URL: <https://www.achaea.com/front>.
- jmbiv (2021). *How to Make a Retro Text Adventure Game in Godot*. [YouTube Playlist]. [Accessed: 2024-07-14]. URL: [https://youtube.com/playlist?list=PLpwc3ughKbZfkSPko3azFD4dd4IHSsi=ne34bvD1zXr\\_xiUC](https://youtube.com/playlist?list=PLpwc3ughKbZfkSPko3azFD4dd4IHSsi=ne34bvD1zXr_xiUC).
- Linietsky, J., Manzur, A. & contributors (2024). *Godot*. [Webpage]. [Accessed: 2024-06-24]. URL: <https://godotengine.org/>.
- Montfort, N. (2003). *Twisty Little Passages: An Approach to Interactive Fiction*. MIT Press. ISBN: 9780262297110.
- Montfort, N. & Bogost, I. (2009). *Racing the Beam: The Atari Video Computer System*. MIT Press. ISBN: 9780262012577.
- OpenAI (2024). *ChatGPT*. [Webpage]. [Accessed: 2024-06-25]. URL: <https://chatgpt.com/>.

- Peppers, K., Tuunanen, T., Rothenberger, M. A. & Chatterjee, S. (2007). A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems* 24(3), pp. 45–77. DOI: 10.2753/MIS0742-1222240302.
- Reed, A. A. (2021a). *1976: Adventure*. [Webpage]. [Accessed: 2024-06-21]. URL: <https://if50.substack.com/p/1976-adventure>.
- Reed, A. A. (2021b). *1977: Zork*. [Webpage]. [Accessed: 2024-06-21]. URL: <https://if50.substack.com/p/1977-zork>.
- Reed, A. A. (2021c). *1980: MUD*. [Webpage]. [Accessed: 2024-06-22]. URL: <https://if50.substack.com/p/1980-mud>.
- Reed, A. A. (2021d). *1997: Achaea*. [Webpage]. [Accessed: 2024-06-22]. URL: <https://if50.substack.com/p/1997-achaea>.
- Reed, A. A. (2021e). *2006: Dwarf Fortress*. [Webpage]. [Accessed: 2024-06-23]. URL: <https://if50.substack.com/p/2006-dwarf-fortress>.
- Reed, A. A. (2021f). *2009: Fallen London*. [Webpage]. [Accessed: 2024-06-23]. URL: <https://if50.substack.com/p/2009-fallen-london>.
- Rothamel, T. & contributors (2024). *Ren'Py*. [Webpage]. [Accessed: 2024-06-24]. URL: <https://www.renpy.org/>.
- Tekinbas, K. & Zimmerman, E. (2004). *Rules of play: game design fundamentals*. MIT. ISBN: 0-262-24045-9.
- Unity Technologies (2024). *Unity*. [Webpage]. [Accessed: 2024-06-24]. URL: <https://unity.com/>.
- Yannakakis, G. & Togelius, J. (2018). *Artificial Intelligence and Games*. Springer. ISBN: 9783319635187.