Noël Joost & Elena Gerber

Development of an LLM-first 2D Videogame

Using Language Models to Create Dynamic Game Worlds

Bachelor's Thesis

OST – Eastern Switzerland University of Applied Sciences
Campus St. Gallen

Supervision

Prof. Dr. Mitra Purandare, Benjamin Plattner

13. June 2025

Abstract

In recent years, Large Language Models (LLMs) have grown in popularity due to their ability to understand and generate human-like text. Their use has expanded far beyond traditional applications, with growing interest in interactive media and game development. This project aims to explore how LLMs can be integrated into video games to generate dynamic content that makes gameplay more engaging, immersive, and interactive. To demonstrate this, a 2D top-down role-playing game (RPG) was developed, serving as both a prototype and testing platform for LLM integration. At the start of each game, an LLM generates a map, quests, Non-Playable-Characters (NPCs) and items. During the game, NPCs interactions are generated by the LLM. The NPCs respond to natural input from the player with LLM-generated answers. This results in a unique experience for each game played. Selected testers showed positive reactions, though issues like inconsistent quests or misplaced items. Overall, the project shows the high potential of using LLMs to make games more flexible and replayable.

Keywords: Game Design, Artificial Intelligence, Software.

Executive Summary

Initial Situation

Traditional role-playing video games (RPGs), where players assume the role of a character in a fictional world, often rely on static dialogues and storylines. This can make replaying the game feel repetitive and less engaging, as the story and conversations remain the same. In contrast, tabletop RPGs thrive on the creativity of the human game master, who continuously invents new settings and stories and allows players to interact naturally with the world. With the rise of large language models (LLMs), new opportunities are emerging for digital games. A key strength of modern LLMs is their ability to generate human-like text in response to natural language input. Integrating LLMs into games enables live, context-sensitive responses to player actions and dialogue, bringing some of the freedom and spontaneity of tabletop RPGs into video games. This project aims to develop a 2D RPG with a top-down view that combines traditional gameplay with the creative power of LLMs. At the start of each game, an LLM generates a unique world. During gameplay, characters respond to the player in real time, meaning no playthrough is the same. For developers, this moves focus away from scripting fixed content to designing systems and prompts that enable the AI to take over parts of the storytelling.

Figure 1: Interaction with an NPC

Approach / Technology

The game is developed in Unity using a 2D pixel art style and includes typical RPG elements such as story, exploration, non- playable characters (NPC), battles, items and quests. Game content is partly generated using LLMs and partly built from predefined assets. For example, character parts like heads, bodies and clothes are defined in advance, and the LLM selects from these to create NPCs. Similarly, the map is built from predefined 30x30 tiles depicting elements like forests, plains and villages, which the LLM assembles into a complete world. The generation process starts with a prompt defining the map, followed by prompts for quests, NPCs and items. NPC appearance and battle skills are generated in separate prompts to keep outputs manageable. For simpler tasks like NPC visuals, skills and interactions, we use OpenAI's cost- efficient GPT-4o-mini. For tasks requiring more consistency and depth, such as map and quest generation, we use the o3-mini model.

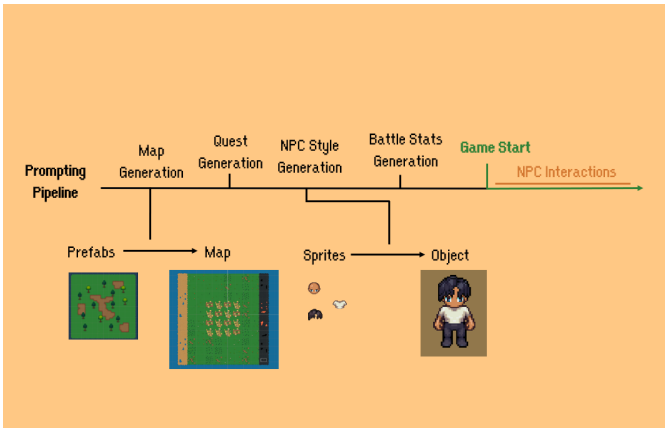

Figure 2: Use of LLM in the game

Result

We built a functional top-down 2D RPG with LLM-supported elements that generates new content on each play through. Our key finding is that the quality of generated content depends heavily on the context provided. Without sufficient details, quests and characters often become repetitive or inconsistent. For example, NPCs might block quest progression, or items can appear in incorrect locations. While these issues occasionally caused confusion, the eight people we selected to test our game generally responded positively, appreciating the dynamic nature of the experience. A major challenge we encountered was balancing the amount and complexity of prompts with system performance. Longer or more detailed prompts improve output coherence but increase the risk of missing details. Splitting input into smaller prompts helps but raises processing time. A promising approach is sending multiple prompts simultaneously for parallel processing, which could improve content accuracy and reduce wait times. Overall, this project indicates that a thoughtful combination of AI and traditional techniques can open exciting new possibilities for creating more dynamic and imaginative games in the future.

Figure 3: Combat

Contents

Part I

Technical Report

Chapter 1

Introduction

1.1 Problem definition

Traditionally, most games feature static dialogues and predefined sequences that players must follow. This makes it difficult to incorporate elements that are not deterministic. By using large language models (LLMs), we can design a game that adapts in real time to player input, creating a unique and immersive gameplay experience.

1.2 Vision

Our vision is to explore the potential of integrating large language models (LLMs) into video games to enable truly dynamic and player-driven experiences. We aim to create a game world that is not only responsive but also expressive—where every player interaction feels meaningful, and every NPC can become a believable character with its own personality and story. Through this, we strive to push the boundaries of narrative design and interactivity, laying the groundwork for a new generation of games that blur the line between authored content and emergent storytelling.

1.3 Goals

1. **Research and Evaluation:** Survey existing technologies, LLM-based game approaches, and relevant literature.
2. **Prototype Development:** Build a 2D video game.
    - The game includes:
        - A quest system
        - A combat system

    - Items
    - A map on which players can move their character
    - An appealing and intuitive user interface
  - The game will be tested by end users.
3. **LLM Integration in Gameplay:**
  - NPC dialogues are generated using an LLM.
  - NPC personalities are created using an LLM.
  - Safeguards are implemented so that:
    - Players cannot convince NPCs to complete quests automatically or break character.
    - Malicious inputs are intercepted (optional).
  - NPC attributes are output in a structured format by the model.
  - NPCs use predefined sprite sheets.
4. **LLM-Driven Combat System:**
  - Abilities are generated via LLM.
  - Abilities are output in a structured format.
5. **Dynamic Item Generation:**
  - Items are created using an LLM.
  - Icons for the items are generated using an image generation AI.
6. **Map Creation:**
  - The map includes interactable objects.
  - Map layout may be generated by an LLM (optional).
  - Environmental changes (e.g., weather effects) may be generated by an LLM (optional).
7. **AI Game Controller (Optional):**
  - The AI knows the game state.
  - Can provide hints on request.
  - Can guide the player through the world.
  - Can assist the player in combat.

1.4 Basic condition

This project is a bachelor thesis with a total workload of 720 hours. The time is divided equally between two team members, corresponding to 12 ECTS credits per person.

Chapter 2

Existing solutions

This chapter provides a brief overview of existing technologies and approaches that integrate Large Language Models (LLMs) or similar AI techniques into video game development.

2.1 AI-Assisted dialogue and narrative systems

Several studios and research projects have started integrating LLMs to dynamically generate dialogues and narratives. One prominent example is Latitude's AI Dungeon, which uses OpenAI's GPT models to create fully open-ended storytelling experiences based on player input [1]. The game demonstrates how LLMs can respond flexibly to natural language input, though it also highlights challenges related to content control and coherence.

Similarly, Hidden Door, a startup focused on narrative AI, aims to provide safe, moderated interactive storytelling using fine-tuned LLMs [2]. Their platform allows players to co-create stories with AI-driven characters while maintaining narrative consistency and appropriateness.

2.2 Procedural content generation

Beyond dialogue, LLMs and generative AI are also being explored for broader content creation. OpenAI's GPT-4, for instance, has been tested for quest generation, item descriptions, and character backstories in both hobbyist and experimental academic projects. Research by [3] explores how generative models can enhance procedural content generation (PCG) by producing design elements that adapt to player behavior and style.

2.3 AI as game engine components

Cutting-edge prototypes such as **Oasis** by Decart and **GameNGen** by Google Research push the boundaries further by using AI to simulate entire game states. These systems replace traditional game logic with neural networks trained to predict visual and logical outcomes frame by frame. While still experimental and performance-intensive, they represent an emerging paradigm where AI acts not just as content generator but as the core simulation engine [4, 5].

2.4 LLMs in commercial development tools

Large game engines like Unity and Unreal are also beginning to integrate AI-driven tools. Unity's "Muse" and "Sentis" toolkits provide generative capabilities and neural inference directly within the editor, allowing developers to prototype or fine-tune dialogue and environment features more efficiently [6]. While these are not full LLM integrations, they indicate a trend toward embedding intelligent tools into standard development workflows.

2.5 Limitations and open challenges

Despite promising developments, the use of LLMs in games still faces several key challenges:

- **Control and safety:** Ensuring generated content adheres to tone, lore, and ethical standards remains a major concern.
- **Performance:** LLM inference can be computationally expensive, especially in real-time settings.
- **Coherence:** Maintaining long-term consistency in narrative and behavior is non-trivial.

Addressing these issues is critical for the widespread adoption of LLMs in mainstream game development.

Chapter 3

Implementation Concept

This chapter describes how the implementation was approached and which steps were taken throughout the development of the game.

3.1 Knowledge gathering

At the beginning of the project, it was important to build a solid knowledge base. We analyzed existing solutions, technologies, and tools relevant for game development, with a particular focus on 2D RPGs.

3.2 Requirements specification

Before evaluating technologies or starting implementation, we defined both functional and non-functional requirements. Functional requirements included core features such as dialogue systems, quest system, combat system, and data persistence. Non-functional requirements focused on performance, and ease of development.

3.3 Evaluation

Based on the defined requirements, various game engines, storage mechanisms, and LLM APIs were evaluated. We focused on tools that matched our needs both for the current prototype and potential future expansions. Unity was selected as the main engine due to its 2D capabilities and support for C#, which simplifies HTTP API calls and tool integration.

3.4 Implementation

3.4.1 Map generation

In the context of our game, it is imperative that we devise a world map that facilitates player exploration. It is imperative that this map remains dynamic; however, it must exhibit consistency across the duration of a single game session. This implies that the map should not be subject to alteration during game play. To this end, a Large Language Model (LLM) is employed for procedural content generation.

TileMap structure

A TileMap constitutes a widely used method for map generation. It functions by placing tiles on a grid, where each tile constitutes a predetermined graphic or sprite. The grid can consist of multiple layers, allowing for visual and interactive complexity.

In the present project, the following layers are employed:

- **Ground** – The base terrain layer.
- **Collision** – Defines which tiles are walkable and which are blocked.
- **GroundTransition** – Enables the creation of smooth transitions between different terrain types.
- **Decoration** – Visual additions such as trees, rocks, or flowers.
- **DecorationWalkBehind** – Decorative elements that the player can walk behind, such as large trees or cliffs.

The combination of these layers results in the creation of a rich, multi-dimensional map that evokes a sense of vitality and dynamism.

Initial approach: Tile-by-tile generation

The initial approach adopted was to generate the map tile by tile. To this end, a 20x20x5 array was constructed and value IDs were assigned to each tile (per layer). This array, along with a prompt, was then sent to the LLM, which returned modified values. These values were then interpreted by an algorithm and applied back to the map grid.

For this process, we employed GPT-4-mini, and while the outcomes were satisfactory, we encountered multiple issues:

- The 20x20 tiles were traversed at an accelerated rate, necessitating frequent additional API calls.
- Tile transitions (e.g., from grass to sand) were often visually inconsistent.
- Frequent API calls lead to higher costs, which directly contradicts our non-functional requirements.
- Visible loading times between generated tiles were not acceptable for a smooth player experience.

Consequently, a decision was taken to revise the approach.

Improved approach: Generation via PreFabs

The approach was transitioned to a higher-level generation system utilising PreFabs. These are larger, pre-designed tile chunks (e.g., 10x10 tiles) that are more visually coherent and reduce complexity.

The allocation of values to these prefabs is then facilitated, with the end goal of providing these values to the LLM. Rather than generating individual tiles, the LLM now operates on a 20x20 array of prefab values, which are then placed on the map using our placement algorithm.

Advantages:

- The capacity for significantly larger worlds: Utilising 20x20 $\times$ 30x30 prefabs enables the generation of maps that are, in effect, 900 times larger.
- Smoother transitions between areas.
- Fewer API calls, reducing latency and cost.

Prefab creation

The following three options were considered for prefab creation:

1. Predefined manually
2. Generated by LLM ahead of time
3. Generated at game start

Runtime generation was ruled out due to the extended loading times (the generation of 30+ prefabs requires numerous API calls).

Additionally, we dismissed the use of LLM-generated prefabs in advance due to their susceptibility to the same transition issues and inconsistencies as the tile-by-tile method.

Consequently, we currently utilize manually created prefabs, which provide us with complete control over:

- Layout
- Visual quality
- Logical consistency

This method offers the best combination of control, quality, and performance.

interactables

Generated maps include interactable elements such as:

- NPCs
- Resources
- Items

Resources are placed randomly and can be collected by the player (see section: *Items*). NPCs and quest-related items are placed according to quest logic (see section: *Quests*).

3.4.2 Combat system overview

The combat system is designed as an Active Time Battle (ATB) System. Each character has a set of core stats that influence turn speed, damage, and defense. The player selects an attack and then chooses a target. Enemies behave similarly and follow the same rules, taking actions when their turn is ready.

Objective

The goal is to strategically use your abilities and manage turns to reduce all enemy health to zero and win the battle.

Example: Pokémon In Pokémon, the objective is to faint all opposing Pokémon. Each trainer can hold up to six Pokémon, and the battle continues until one side has no remaining Pokémon.

Our implementation Our system shares a similar objective but supports simultaneous turn progression through the ATB system, increasing urgency and requiring real-time strategy. Instead of a Team of six Pokémon, our system uses up to four characters.

AI-Generated content

Both attacks and character stats are generated by the AI. These stats can be modified with items, which can be collected. This allows for varied combat encounters and unique combinations of abilities for both players and enemies.

Example: Pokémon In Pokémon, stats and moves are manually designed and balanced. Pokémon can level up and learn new moves, but their combinations are fixed.

Our implementation Stats and attacks are procedurally generated using LLMs, which provides unique encounters and dynamically balanced challenges.

Character stats overview

Each character in the game has a detailed set of stats that define their performance in combat, resource management, and responsiveness.

Example: Pokémon HP, Attack, Defense, Special Attack, Special Defense, Speed.

Our implementation Health & Mana

- **maxHealth (Int)** – The maximum amount of health the character can have
- **currentHealth (Int)** – The current health of the character
- **maxMana (Int)** – The maximum amount of mana
- **currentMana (Int)** – The current mana available

Initiative & Precision

- **Initiative (Int)** – Determines how quickly the character gains turns
- **Precision (Int)** – Represents the chance of landing a critical hit

Element stats

- **PhysicalStats** – Physical attacks
- **FireStats** – Fire attacks
- **WaterStats** – Water attacks
- **EarthStats** – Earth attacks
- **AirStats** – Air attacks
- **PoisonStats** – Poison attacks
- **DefenseStats** – Defense against physical attacks
- **MagicDefenseStats** – Defense against magic attacks

Stats type

- **BaseAttackStat (Int)** – Strength for the corresponding Element
- **AttackStatBuff (Int)** – Multiplier for the Strength

Turn order system

The turn order determines which character acts when. The dynamic system allows fast characters to act more often.

Example: Pokémon Turn order is based on Speed stat. Higher Speed Pokémon move first. Ties are broken randomly.

Our implementation Turn order formula

$$\text{Time} = \frac{\text{ActionCost}}{\text{Initiative} \times \left(1 + \frac{\text{statModifier}}{2}\right) + \text{TurnCount}} \tag{3.1}$$

Each character's time is calculated and placed in a sorted list for action order.

Modifiers explained

- **Stat Modifier:** Each stage = +0.5 (e.g., +2 stages = +1.0)

If speed is altered mid-combat, the character's turn is recalculated.

Example setup and turn calculation

Name	Initiative	Modifier	ActionCost
NPC 1	100	0	1000
NPC 2	50	0	1000
NPC 3	85	4	1000

Table 3.1: Turn calculation example

Name	Turn 1	Turn 2	Turn 3	Turn 4
NPC 1	10	20	30	40
NPC 2	20	40	60	80
NPC 3	3.92	7.84	11.76	15.68

Table 3.2: Turn calculation example

NPC 3 has the highest effective initiative due to a +4 stage modifier, resulting in much faster actions. With a calculated time of only ≈ 3.92 units for their first turn, NPC 3 acts significantly more frequently than NPC 1 and NPC 2. NPC 1 and NPC 2, with lower initiatives and no modifiers, follow at regular intervals of 10 and 20 time units, respectively. The characters are placed in a priority queue sorted by their accumulated time, and the one with the lowest value acts next.

This system ensures dynamic and stat-dependent action order that updates continuously during combat.

Damage calculation

Example: Pokémon Damage is calculated based on level, attack, defense, and type effectiveness using the following formula:

$$\text{Damage} = \left(\frac{2 \times \text{Level}}{5} + 2\right) \times \frac{\text{Attack}}{\text{Defense}} \times \text{BasePower} \times \text{Modifier} \tag{3.2}$$

- **Modifier** includes type effectiveness, STAB (Same-Type Attack Bonus), critical hits, and a random factor.

Our implementation

$$\text{Damage} = \text{BasePower} \times \left(\frac{\text{AttackStat}}{\text{DefenderStat}}\right) \times \text{CritMultiplier} \times \text{Variance} \tag{3.3}$$

- **CritMultiplier:** 1.5 if critical, else 1.0
- **Variance:** Random between 0.85 and 1.0

With buffs/debuffs:

$$\begin{aligned}\text{Damage} = \big[\text{PowerWithBuff} + \text{FlatRaise} \times \big(&\frac{\text{AttackStat} \times \text{StateRaise}_{\text{Attacker}}}{\text{DefenderStat} \times \text{StateRaise}_{\text{Defender}}}\big) \\ &\times \text{CritMultiplier} \times \text{Variance}\big] - \text{FlatReduction}\end{aligned} \tag{3.4}$$

Attack definition

- **Name (String)** – Display Name
- **Description (String)** – Description of the attack
- **Power (Int)** – Base damage
- **ManaCost (Int)** – Resource cost
- **Element (Enum)** – Air, Fire, Earth, Water, Poison, Physical
- **Accuracy (Float)** – Between 0 and 1, modifies crit chance
- **AdditionalEffects (List)** – Optional effects

Each character has 4 active attacks. New attacks can be learned with items.

Buffs and debuffs

Buffs and debuffs share the same structure, differing only in effect:

- **Name (String)** – Effect name
- **Icon (Image)** – Display icon
- **Description (String)** – Description of effect
- **Strength (Int)** – Effect strength
- **Duration (Int)** – Duration in turns
- **isBuff (Bool)** – True if Buff, false if Debuff
- **BeforeAttackEffect** – Trigger before attacking
- **AfterAttackEffect** – Trigger after attacking

- **BeforeTurnEffect** – At the start of turn
- **AfterTurnEffect** – At the end of turn

Calculation order

1. Select Attack
2. Apply *BeforeAttackEffect*
3. Calculate Crit:
$$\text{isCrit} = \text{Random} \leqslant (\text{Precision}_{\text{Attacker}} \times \text{Accuracy}_{\text{Attack}}) \tag{3.5}$$
4. Modify Stats (Buffs/Debuffs)
$$\text{effectiveAttackStat} = \text{BaseAttackStat}_{\text{Attacker}} \times \text{Buff}_{\text{Attacker}} \tag{3.6}$$
$$\text{effectiveDefenseStat} = \text{BaseDefenseStat}_{\text{Defender}} \times \text{Buff}_{\text{Defender}} \tag{3.7}$$
5. Calculate Raw Damage
6. Apply Damage to Target
7. Trigger Additional Effects
8. Apply *AfterAttackEffect*
9. Apply *BeforeTurnEffect* and *AfterTurnEffect* on turn end
10. Decrease Buff/Debuff durations

3.4.3 NPC generation

NPCs are procedurally generated in three main stages: personality, appearance, and battle data. This multi-step generation ensures depth and variety for each character.

Personality generation

Each NPC is assigned a background story and a unique personality profile. An important attribute is the **Personality Affection** value, which represents how much the NPC likes the player. This value can change dynamically:

- Starts at a random base level.
- Increases through completed quests and positive interactions.
- If the value exceeds 50, the NPC may be recruited into the player's team.

Personality also affects dialogue and quest logic, enabling unique storylines and emotional attachment.

Appearance generation

NPCs are visually composed using predefined presets, selected according to a generated profile. The following modular categories define their look:

- **Gender**
- **SkinColor**
- **HairStyle, HairColor**
- **ShirtType, ShirtColor**
- **PantsType, PantsColor**
- **SocksType, SocksColor**
- **ShoesType, ShoesColor**

This system allows for consistent but varied character appearances and supports layering and equipment changes.

Battle data generation

The last step in NPC creation is defining their combat parameters. These include:

- Base stats (e.g., Health, Mana, ElementStats)
- Four active attacks from the available move pool

Additionally, each NPC receives a **First Name** and **Last Name**, which are randomly generated.

Together, these layers ensure that every NPC is context-aware, functionally integrated into the world, and provides dynamic interaction potential.

3.4.4 Quest system

Quests provide structure, motivation, and narrative progression in the game world. Each quest is composed of essential metadata and a sequence of steps that define the required actions for completion.

Quest metadata

Each quest contains the following attributes:

- **QuestName (String)** – The title of the quest
- **QuestDescription (String)** – A brief narrative or instruction describing the objective

- **QuestGiver (NPC)** – The NPC that assigns the quest
- **IsStarted (Bool)** – Indicates whether the player has accepted the quest
- **IsCompleted (Bool)** – Indicates whether the player has completed all quest steps
- **QuestType (Enum)** – Specifies the quest category (e.g., Main, Side, Faction, Timed)

Quest steps

Each quest consists of one or more ordered steps. A quest step represents a specific task that the player must complete. The steps are defined as follows:

- **QuestId (Reference)** – The identifier of the parent quest
- **QuestStepNumber (Int)** – The sequential order of the step
- **QuestStepName (String)** – A short name or title of the task
- **QuestStepDescription (String)** – Detailed instruction or narrative for the step
- **NPCId (Optional)** – ID of the NPC involved in the step (e.g., talk or fight target)
- **ItemId (Optional)** – ID of the required or resulting item
- **Action (Enum)** – Type of interaction required: *Battle*, *Item*, or *Talk*
- **Completed (Bool)** – Marks whether this step has been fulfilled

Quests may branch, repeat, or trigger other quests. This flexible structure supports both linear and dynamic storytelling. These Quests are generated at the start of the game.

3.4.5 Item system

Items are used to support gameplay across exploration, combat, and character development. Items are categorized into three main types with distinct usage contexts and effects.

Item structure

Each item includes the following core attributes:

- **Id (Int)** – Unique identifier for the item
- **Name (String)** – Name of the item
- **Description (String)** – Descriptive text explaining the item's function
- **Category (Enum)** – Defines item type: *QuestItem*, *BattleItem*, or *StatEnhancer*
- **IconPath (String)** – File path or reference to the item's icon

- **EffectJson (JSON)** – Contains effect details (only applicable to BattleItems and StatEnhancers)

Item categories

Quest items Quest items are used exclusively for quest progression. They are placed in the world and collected through interactions. These items do not affect stats or combat but serve as triggers in quest steps.

Battle items Battle items can be used during combat and affect the current state of characters. Common effects include:

- Healing health or mana
- Raising stats temporarily (e.g., defense boost for 3 turns)

The actual effect logic is defined within the `EffectJson` field.

Stat enhancers Stat enhancers are applied outside of combat and alter the character's base stats permanently. They can be used to strengthen a character's attack, defense, or other attributes—or even apply penalties. These effects are also defined in the `EffectJson`.

This structured item system enables dynamic inventory management and strategic use of resources during and outside of battle.

AI-Generation The LLM is instructed to generate stronger and weaker items. Items can also have negative effects, but have stronger positive effects in return.

Chapter 4

Technology Evaluation

4.1 Game engine

4.1.1 Godot

Advantages:

- **Native 2D Support:** The engine is primarily designed for 2D games, offering excellent performance

Disadvantages:

- **Uncommon Language:** GDScript is not widely used outside of Godot
- **Smaller Community:** Fewer assets, tutorials, and forum discussions

4.1.2 Unreal Engine

Advantages:

- **Very Powerful:** Suitable for high-end projects with advanced built-in tools

Disadvantages:

- **Focus on 3D:** 2D projects are possible but not the engine's main strength.

4.1.3 RPG Maker

Advantages:

- **Tailored for 2D RPGs:** Comes with built-in systems (combat, dialogues, inventory), enabling a quick start without programming.

Disadvantages:

- **Limited Flexibility:** Custom systems or external API integration are difficult or not feasible.

4.1.4 Unity

Advantages:

- **Excellent 2D Support:** Built-in support for sprites, tilemaps, animations, and UI tools.
- **Popular Language (C#):** Modern, object-oriented, and widely adopted
- **Large community and asset store:** Extensive resources, ready-to-use tools, and support available online.

Disadvantages:

- **Possibly overkill for simple games:** Brings many 3D features that aren't needed for 2D projects

4.1.5 Decision

After evaluating several engines, we decided Unity was the best fit for our project. Using C# makes it easy to connect with LLM services via HTTP API calls, which lets us create dynamic dialogues and game content. Unity's powerful 2D tools match what we need for the game's design. Plus, the large and active community is a huge bonus, especially for developers like us who are new to game development. Having access to tutorials, forums, and shared knowledge really helps smooth out the learning curve.

4.2 Data persistence mechanism

4.2.1 Relational database (PostgreSQL)

Advantages:

- **Powerful querying:** Supports complex queries and maintains data integrity

Disadvantages:

- **Requires server:** Needs server setup and ongoing maintenance
- **Network overhead:** Adds latency unsuitable for single-player games
- **Complexity:** More complicated than necessary for local game saves

4.2.2 JSON files

Advantages:

- **Simple format:** Human-readable and easy to implement
- **Easy debugging:** Straightforward to inspect and fix data issues.

Disadvantages:

- **Limited querying:** No support for complex data relationships

4.2.3 Scriptable objects

Advantages:

- **Excellent 2D Support:** Unity integration: Designed for easy use inside the Unity editor
- **No dependencies:** Does not require extra libraries or setups

Disadvantages:

- **Not for runtime:** Unsuitable for saving dynamic or player data
- **Limited scalability:** Not good for complex or large data sets

4.2.4 SQLite

Advantages:

- **Lightweight:** Embedded database stored in a single local file
- **Efficient queries:** Supports complex queries and relational data
- **No server needed:** Runs fully locally on the user's device

Disadvantages:

- **Integration required:** Needs adding a database library to the project

4.2.5 Decision

After evaluating all options, we decided to use SQLite for our game. It offers the perfect balance between structure and simplicity. Unlike JSON or Scriptable Objects, SQLite supports complex queries and relational data without requiring an external server like PostgreSQL.

4.3 Large language model

4.3.1 Overview

Large language models (LLMs) play a central role in this project by generating dynamic game content such as maps, quests, NPCs, items, and dialogue. The goal was to reduce static scripting and instead create a system that produces rich, variable content for each playthrough. To accomplish this, we evaluated and used different LLMs accessible via APIs.

4.3.2 OpenAI GPT models

Advantages:

- **High-quality outputs:** Produces coherent, creative, and context-aware text.
- **Easy integration:** Well-documented REST APIs with JSON-based responses.
- **Widely adopted:** Large community, tutorials, and tooling support.

Disadvantages:

- **Usage costs:** Frequent API calls can lead to high running costs.
- **Rate limits:** Depending on subscription, throughput may be restricted.
- **Cloud-only:** No local/offline use, which limits portability.

4.3.3 Anthropic Claude

Advantages:

- **Long context window:** Ideal for generating or interpreting complex multi-part prompts.
- **Safer outputs:** Focus on avoiding offensive or harmful content.

Disadvantages:

- **Limited availability:** Fewer API integrations and regional restrictions.
- **Less control:** Slightly harder to steer output formatting compared to GPT.

Claude is well-suited for quests with multiple conditions or state changes, but integration in Unity is more complex due to limited C# tooling.

4.3.4 Mistral

Advantages:

- **Open weights available:** Can be self-hosted for offline use.
- **Fast and efficient:** Optimized for inference and cost-efficient deployment.

Disadvantages:

- **No official API:** Requires third-party hosting or self-deployment.
- **Weaker coherence:** Slightly lower quality for longer or narrative-heavy prompts.

Mistral is a strong candidate for local generation or embedded use in the future (e.g., offline game versions).

4.3.5 Meta LLaMA

Advantages:

- **Open-source model family:** Actively developed and available in many sizes.
- **Self-hosting possible:** No reliance on commercial APIs.

Disadvantages:

- **High hardware requirements:** Running models locally needs GPUs and setup.
- **Less optimized for conversation:** Requires careful fine-tuning for specific tasks.

LLaMA models could be used for future expansion if offline functionality or full AI control is required.

4.3.6 Model selection summary

- **OpenAI GPT:** Best for fast prototyping, reliable outputs, and ease of use.
- **Claude:** Useful for long and structured responses, with safer default behavior.
- **Mistral:** Efficient and cost-effective for simpler tasks, ideal for self-hosting.
- **LLaMA:** Good long-term option for fully local or customized setups.

4.3.7 Decision

For this project, OpenAI's GPT models provided the best trade-off between quality, integration simplicity, and performance. However, support for other LLMs is planned for the future using abstraction layers (e.g., via LLM Hub) to enable flexible backends, reduce costs, and allow offline use in later game versions. We used:

- **GPT-4o-mini** for lightweight tasks (e.g., character names).
- **o3-mini** for complex structures (e.g., maps, quest chains).

4.4 Additional libraries and technologies

SQLite4Unity3d

To integrate SQLite into Unity, we chose to use **SQLite4Unity3d** [1], a lightweight and Unity-friendly library that simplifies working with SQLite databases. It allows us to map C# classes to database tables and perform common operations like saving and loading player data with minimal overhead. While other options exist, such as custom native plugins or raw SQLite wrappers, SQLite4Unity3d offers the best

[1] https://github.com/robertohuertasm/SQLite4Unity3d

balance of ease of use, cross-platform support, and smooth integration with Unity's C# environment, making it a practical choice for our project.

Chapter 5

Outlook

If more time had been available, some known bugs could have been fixed. Additionally, we identified further features during development and based on user feedback, which could be implemented in the future. Chapter 10 provides a detailed overview of these planned features.

Part II

Project Documentation

Chapter 6

Requirement Specifications

6.1 Use case diagram

A Use Case Diagram was created to serve as the basis for the user stories.

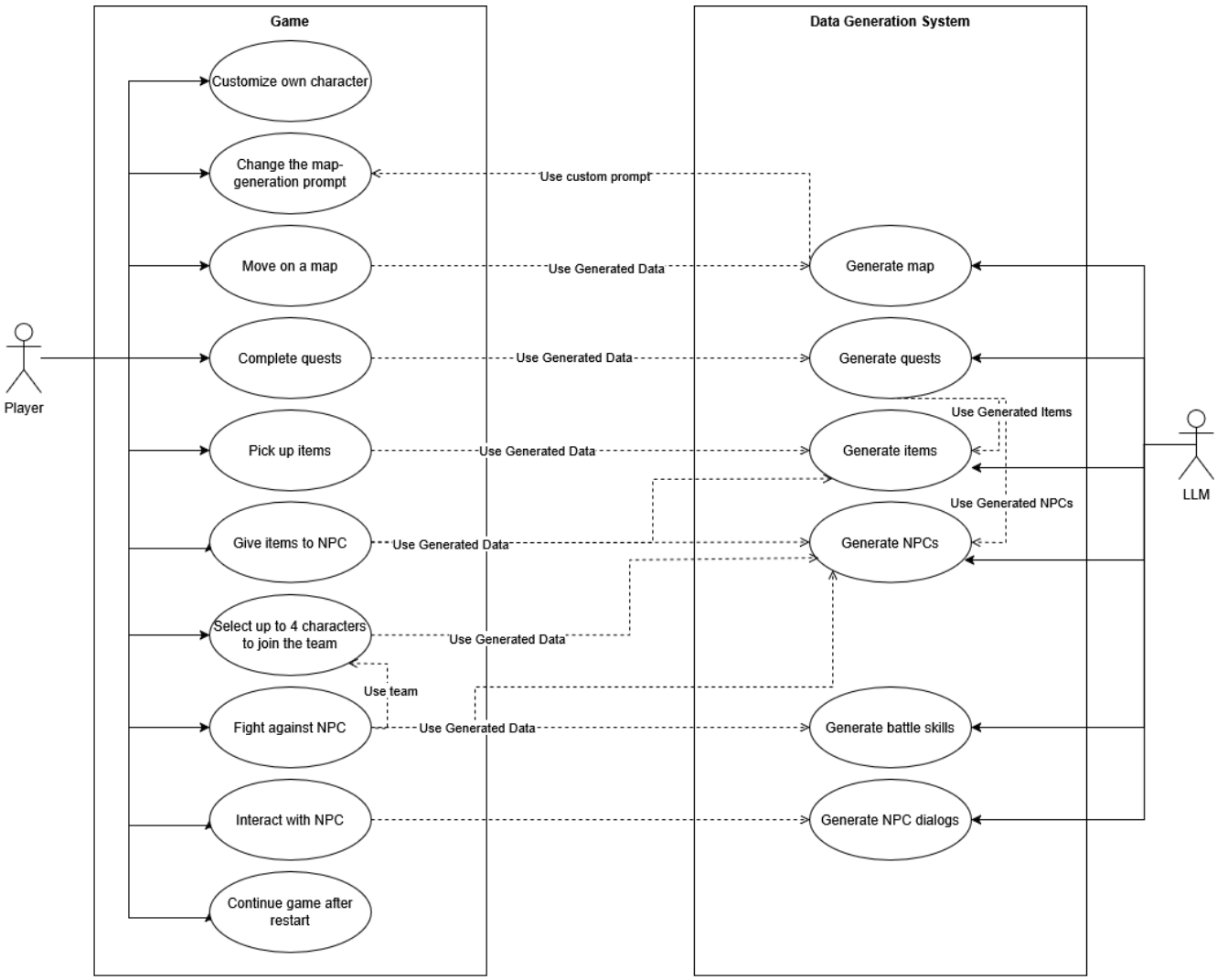

Figure 6.1: Use-Case-Diagram for the AI-Game

6.2 Functional requirements

This chapter briefly explains the epics and user stories, which serve as functional requirements.

6.2.1 Epics

This section describes the epics.

Data Persistency	
Key	AIG-1
Description	This epic ensures game progress and player data are saved and loaded reliably. Players can also delete all saved data to reset the game and start fresh.
Status	Done

Data Generation	
Key	AIG-2
Descirption	This epic involves data generation using a large language model (LLM). Each time the game is reset, new content is generated dynamically. This includes creating new maps, quests, items, battle skills, NPC appearances, and natural NPC responses. Players can also customize the map-generation prompt to influence the generated content. The goal is to provide a varied and highly replayable game world.
Status	Done

Gameplay	
Key	AIG-3
Description	This epic includes all core mechanics through which the player interacts with the game. It covers free player movement, team formation, combat against NPCs, item collection and management, and character customization before starting the game.
Status	Done

6.2.2 User-stories

This section describes the user stories.

Save Game Data	
Key	AIG-4
Beschreibung	As a player, I want my game progress to be saved so I can continue from where I left off.
Status	Done

Delete Game Data	
Key	AIG-5
Beschreibung	As a user, I want to be able to reset the game and delete all game data.
Status	Done

Map Generation Prompt	
Key	AIG-5
Description	As a player, I want new to modify the map-generation prompt.
Status	Done

Map Generation	
Schlüssel	AIG-6
Description	As a player, I want a new map to be generated each time I reset the game.
Status	Done

Quest Generation	
Key	AIG-7
Description	As a player, I want new quests to be generated each time I reset the game.
Status	Done

NPC Style Generation	
Key	AIG-8
Description	As a player, I want NPCs to have varied appearances that are different each time I reset the game.
Status	Done

Item Generation	
Key	FWK-9
Description	As a player, I want new items to be generated each time I reset the game.
Status	Done

Battle Skills Generation	
Key	AIG-10
Description	As a player, I want each character to have generated skills every time I reset the game.
Status	Done

Generated NPC Responses	
Key	AIG-11
Description	As a player, I want NPCs to respond dynamically to my actions so that interactions feel natural.
Status	Done

Player Movement	
Key	AIG-12
Description	As a player, I want to move my character freely in the game world
Status	Done

Team	
Key	AIG-13
Description	As a player, I want to choose up to four characters to create a team, so that I can use them in combat.
Status	Done

Combat	
Key	AIG-14
Description	As a player, I want to fight against NPCs
Status	Done

Items	
Key	AIG-15
Description	As a player, I want to collect items, see what items I collected and give them to NPCs
Status	Done

Character Creation	
Key	AIG-16
Description	As a player, I want to customize my character's appearance before the game starts
Status	Done

6.3 Non-functional requirements

This chapter defines the non-functional requirements.

No.	Requirement	Priority
	General	
NFR-01	The map must load without visible loading times.	Must
NFR-02	The AI must be fair and challenging without being frustrating.	Must
NFR-03	The cost for the Ai should not exceed 0.1 CHF per api call	Must
	Performance	
NFR-04	Loading times after generation must not exceed 5 seconds.	Should
NFR-05	Loading times for game generation must not exceed 30 seconds.	Should
NFR-06	AI decisions must be made within 3 seconds	Must
NFR-07	The frame rate (FPS) must remain stable above 30 FPS.	Must
	Security	
NFR-08	Player and game data must be stored securely.	Must
NFR-09	Anti-cheat mechanisms must prevent the bypassing of NPC personalities.	Must
NFR-10	The AI must not collect or store players' personal data.	Must
NFR-11	The AI must not be used for other purposes.	Must
	Reliability	
NFR-12	The game must provide a readable error message in case of failure.	Must
	Usability	
NFR-13	The user interface must be intuitive.	Must
NFR-14	The controls and interaction with the game world must be intuitive.	Must

No.	Requirement	Priority
NFR-15	Tutorials must clearly explain the game to the player.	Should
Maintainability		
NFR-16	The AI should be modularly extendable or adjustable without restarting the game state.	Should
NFR-17	Logs and analytics must be accessible to developers for quick error resolution.	Must

Table 6.1: Non-functional requirements of the game system

6.4 Landing zones for non-functional requirements

To ensure a structured approach to meeting the non-functional requirements, we define landing zones for each category. These zones include:

- **Minimum (Acceptable)** – The baseline level that must be met.
- **Target (Desired)** – The expected level of performance.
- **Best Case (Optimal)** – The ideal performance goal.

No.	Requirement	Minimum	Target	Best Case
General				
NFR-01	The map must load without visible loading times.	2 sec	1 sec	Instant
NFR-02	The AI must be fair and challenging without being frustrating.	Subjective testing	Balanced across user feedback	Dynamically adaptive AI
NFR-03	The cost for the AI should not exceed 0.1 CHF per API call.	0.1 CHF	0.05 CHF	0.01 CHF
Performance				
NFR-04	Loading times after generation must not exceed 5 seconds.	5 sec	3 sec	1 sec
NFR-05	Loading times for game generation must not exceed 30 seconds.	30 sec	20 sec	10 sec
NFR-06	AI decisions must be made within 3 seconds.	3 sec	1 sec	500 ms
NFR-07	The frame rate (FPS) must remain stable above 30 FPS.	30 FPS	60 FPS	120 FPS

No.	Requirement	Minimum	Target	Best Case
Security				
NFR-08	Player and game data must be stored securely.	Encrypted at rest	Encrypted in transit and at rest	Zero-trust architecture
NFR-09	Anti-cheat mechanisms must prevent the bypassing of NPC personalities.	Basic detection	Advanced detection	AI-driven adaptive anti-cheat
NFR-11	The AI must not be used for other purposes.	None	Policy enforcement	AI governance framework
Reliability				
NFR-12	The game must provide a readable error message in case of failure.	Generic error messages	Specific user-friendly error handling	Specific user-friendly error handling
Usability				
NFR-13	The user interface must be intuitive.	User Tests	Meets accessibility standards	Adaptive UI based on user behavior
NFR-14	The controls and interaction with the game world must be intuitive.	User Tests	Meets industry best practices	Fully customizable controls
NFR-15	Tutorials must clearly explain the game to the player.	Static guides	Interactive tutorials	AI-driven tutorial
Maintainability				
NFR-17	Logs and analytics must be accessible to developers for quick error resolution.	Basic logging	Structured logs with filtering	Structured logs with filtering

Table 6.2: Non-functional landing zone

Chapter 7

Analysis and evaluation

7.1 Domain analysis

Here is the Domain Analysis.

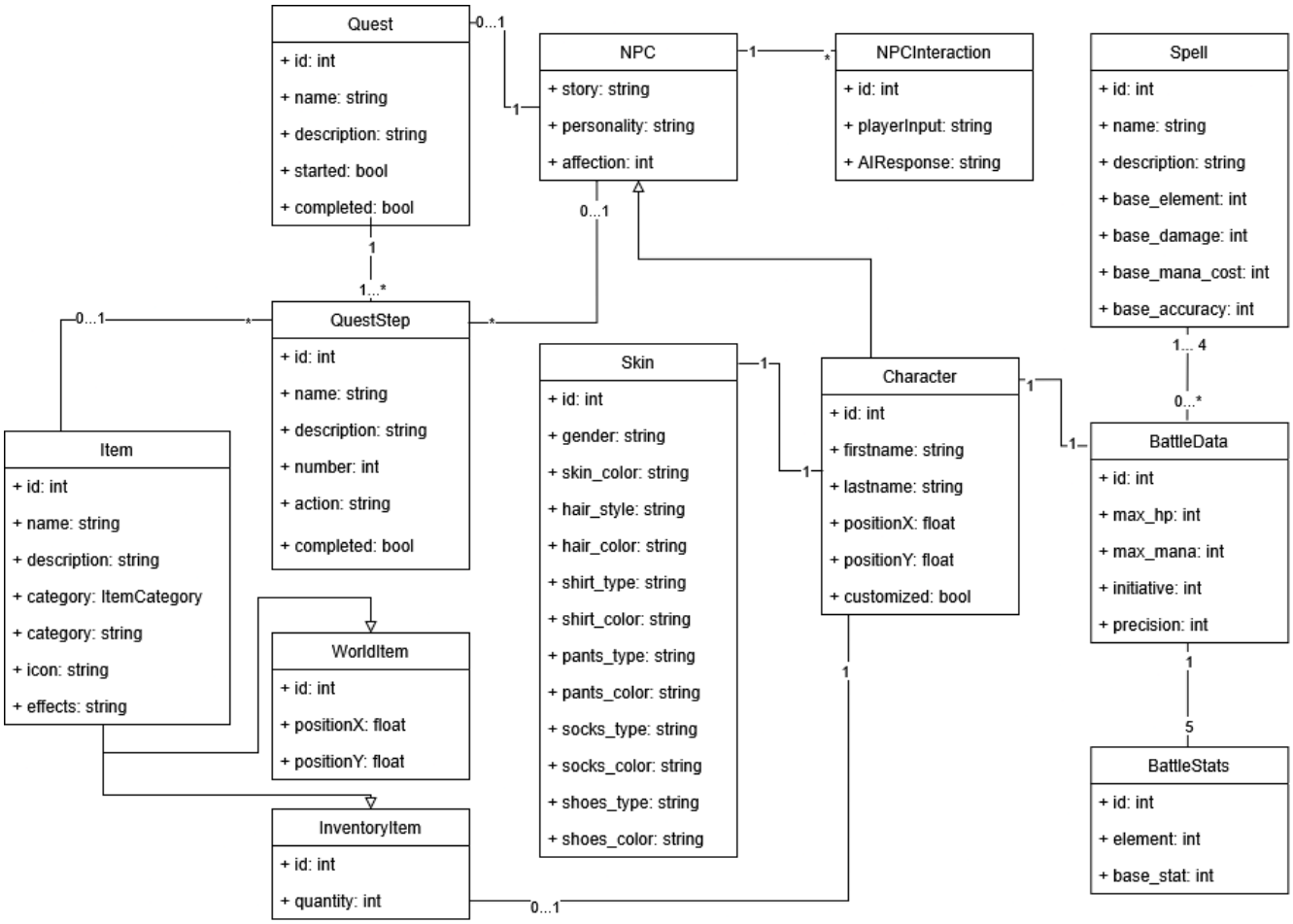

Figure 7.1: Domain Analysis

7.2 Class catalog

7.2.1 Character

The `Character` class is used for playable characters and serves as a base class for NPCs.

7.2.2 NPC

The `NPC` class represents a non-player character in the game.

7.2.3 Skin

The `Skin` class defines the appearance of a character or NPC. It specifies which sprites will be used to visually represent them.

7.2.4 NPCInteraction

The `NPCInteraction` class defines an interaction between the player and an NPC. It holds the player's input (playerInput) and the AI's response (AIResponse).

7.2.5 Item classes

The `Item` class is the base class representing any game item. `WorldItem` inherits from Item and represents items that exist in the game world at specific positions (positionX, positionY) on the map. `InventoryItem` also inherits from Item but represents items that have been collected by the player and stored in their inventory, including an additional quantity attribute.

7.2.6 Quest

The `Quest` class represents a mission in the game. The goal of the game is to complete these quests.

7.2.7 QuestStep

The `QuestStep` class describes a single step within a Quest. A QuestStep always represents exactly one specific action: either talking to an NPC, collecting an item, giving an item to an NPC, or defeating an NPC.

7.2.8 BattleData

The `BattleData` class describes all relevant information that is required and exchanged during a battle. These include, for example, health points (HP), attack power, defense, initiative, and magic resistance. The class serves as the central data structure for the sequence and logic of the battle.

7.2.9 BattleStats

The `BattleStats` class represents the basic values of a unit. Each entry describes an element (e.g., fire, water, etc.) and the corresponding base stat, i.e., the attribute value of the unit for this element. A unit can have several such status entries, which determine its strengths and weaknesses against different types of spells.

7.2.10 Spell

The `Spell` class describes a spell that a unit can cast in combat. A spell has a unique id, a name, and a description that explains the spell's effect or function. The spell is assigned to a specific base element and causes base damage by default. In addition, it costs a certain amount of mana (base mana cost) and has a base accuracy, which indicates the probability of the spell applying critical damage. Each spell can be used by multiple units.

7.3 Traditional approach

In traditional game development, much of the content, such as dialogue, explanations, hints, and background information, is written and programmed manually. This includes:

- Conversations with characters
- Help texts
- Lore explanations
- Reactions to player behaviour

These elements are typically static and predefined, with fixed structures and limited possibilities.

Even in sandbox games like *Hitman 3*, where players have significant freedom, responses are still tied to predefined triggers and scripts. While the illusion of freedom is created, the underlying logic remains rule-based.

7.3.1 Advantages of the traditional approach

- **High level of control over content:** Developers define every line of dialogue and behaviour.
- **Consistent quality:** Manual creation ensures coherent tone, style, and immersion.
- **Lower risk of errors:** All scenarios are predefined, reducing surprises.
- **Performance:** Static systems are more resource-efficient.

7.3.2 Disadvantages of the traditional approach

- **High manual effort:** Writing, testing, translating, and maintaining content is time-consuming.
- **Limited flexibility:** Creative player input is often ignored.
- **Low dynamism:** Content becomes predictable on repeated playthroughs.
- **Scalability issues:** Manual scripting in vast open worlds becomes infeasible.

7.4 Video games with LLMs: A new dimension of interactivity

Large Language Models (LLMs) bring new possibilities to game design:

- Natural language interaction with characters
- Dynamic quest generation
- Believable, context-aware dialogue
- Adaptive world responses

LLMs enable richer interactions and greater immersion by replacing static systems with dynamic, AI-driven content generation.

7.4.1 Advantages of LLM integration

- **Natural communication:** Players express themselves freely and receive relevant responses.
- **Believable worlds:** NPCs adapt based on player decisions.
- **Replayability:** Content can vary on each playthrough.
- **Reduced workload:** Routine tasks like filler dialogue generation are automated.
- **Scalability:** Open worlds can be populated without manual scripting.

7.4.2 Challenges of using LLMs

- **Unpredictability:** Responses can be off-topic or inappropriate.
- **Consistency issues:** Maintaining internal logic and tone is difficult.
- **Performance demands:** Large models require significant resources.
- **Ethical concerns:** Preventing offensive or biased output is crucial.
- **Lack of structure:** Generated content is hard to integrate into fixed mechanics.

7.5 AI as the game engine: Oasis and GameNGen

Traditional engines like Unity rely on scripted logic and handcrafted assets. In contrast, AI-driven engines generate content in real time using neural networks.

7.5.1 Oasis by Decart and Etched

- Open-world game without a traditional engine
- Trained on Minecraft gameplay to predict and render frames
- **Features:** Real-time generation, image-based world creation
- **Limitations:** Stability and performance issues

GameNGen by Google Research

- Simulates Doom using a diffusion model
- Predicts each frame from player input
- **Features:** Neural simulation, real-time performance
- **Limitations:** Narrow scope, potential inconsistencies

7.6 Conclusion: A turning point in game development

LLMs and neural simulation engines mark a paradigm shift. While traditional development ensures control and stability, AI allows for more dynamic, reactive, and immersive experiences.

Players gain agency, while **developers** shift to curating AI-driven systems. Despite current limitations, the potential for transformation is evident.

7.7 How and why we use AI in our game

We use AI to support content generation, not as a full game engine. Traditional systems still handle rendering, input, animation, and core logic. The AI is used to extend creativity and reduce the need for static scripting, which is time-consuming and limits variability.

7.7.1 Why we use AI

There are several reasons why AI plays a central role in our game development process:

- **Replayability:** AI-generated content ensures that each playthrough is different. This reduces repetitiveness and encourages exploration, even after multiple sessions.

- **Scalability:** Manually creating quests, NPCs, and dialogue for large game worlds is resource-intensive. AI allows us to scale content without linear increases in development time.
- **Creativity boost:** AI provides unexpected or novel ideas that can inspire or enrich the game world in ways that static design may not achieve.
- **Faster iteration:** Instead of writing and testing every piece of dialogue or quest manually, AI can rapidly produce drafts that are context-aware and coherent.
- **Immersion and dynamism:** Interactions with AI-generated NPCs feel more natural and personalized, as their responses reflect the current game state and their unique personality.
- **Developer focus:** By outsourcing repetitive or low-level writing tasks to AI, developers can focus on higher-level game design, balancing, and user experience.

7.7.2 Use cases

- **Quest generation:** The AI proposes context-aware missions that are linked to specific characters and locations within the generated world.
- **Dialogue creation:** Conversations with NPCs are generated in real time based on personality traits, past player choices, and current quest progress.
- **Item and lore generation:** Items, equipment, and world lore are created by AI to add depth and richness to the setting with minimal manual input.
- **Map composition:** The map is built using a predefined set of tiles (e.g., forest, mountain, village) arranged by AI to create natural-looking worlds.
- **NPC design:** Visual appearance, names, backstories, and battle attributes are generated to ensure character diversity and consistency with world and quest logic.

7.7.3 Why we avoid full AI engines

- **Need for control:** Traditional logic offers reliability.
- **Player expectations:** Consistency in mechanics is essential.
- **AI is best for creativity:** Dialogue and quests benefit most.
- **Development constraints:** AI engines are still hard to control.

7.7.4 Final thought

LLMs are not just a technical feature—they redefine how games are created and experienced. They offer new opportunities, but also demand thoughtful integration.

Chapter 8

Design

8.1 Architecture

The game is developed using Unity and follows a component-based architecture, which is the default architectural pattern encouraged by the engine. In this model, game objects are composed of modular, reusable components (MonoBehaviours) that define their behavior and data. Each system in the game, such as character customization or npc interaction, is implemented as a set of focused components that work together to form complete functionality. This approach promotes flexibility, separation of concerns, and ease of maintenance, making it well-suited for a 2D character-driven game.

8.1.1 C4-Diagrams

C4 modeling provides an easy-to-understand representation that still contains all the essential information about an architecture. Not all possible C4 diagrams are shown. Below are the System Context Diagram and the Container Diagram. For more information about C4 modeling, see https://c4model.com/.

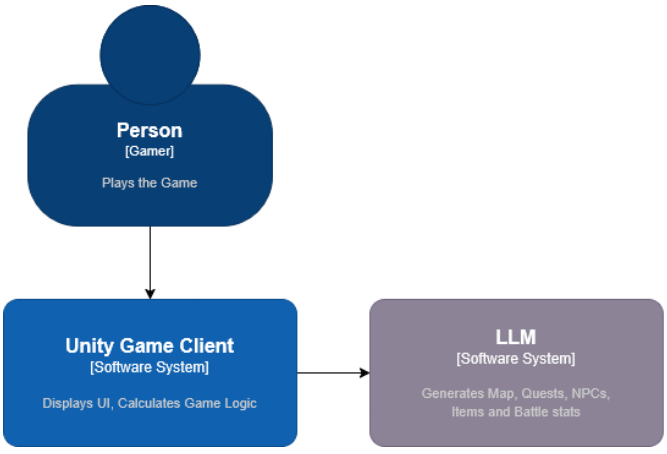

Figure 8.1: C4 System Context-Diagramm

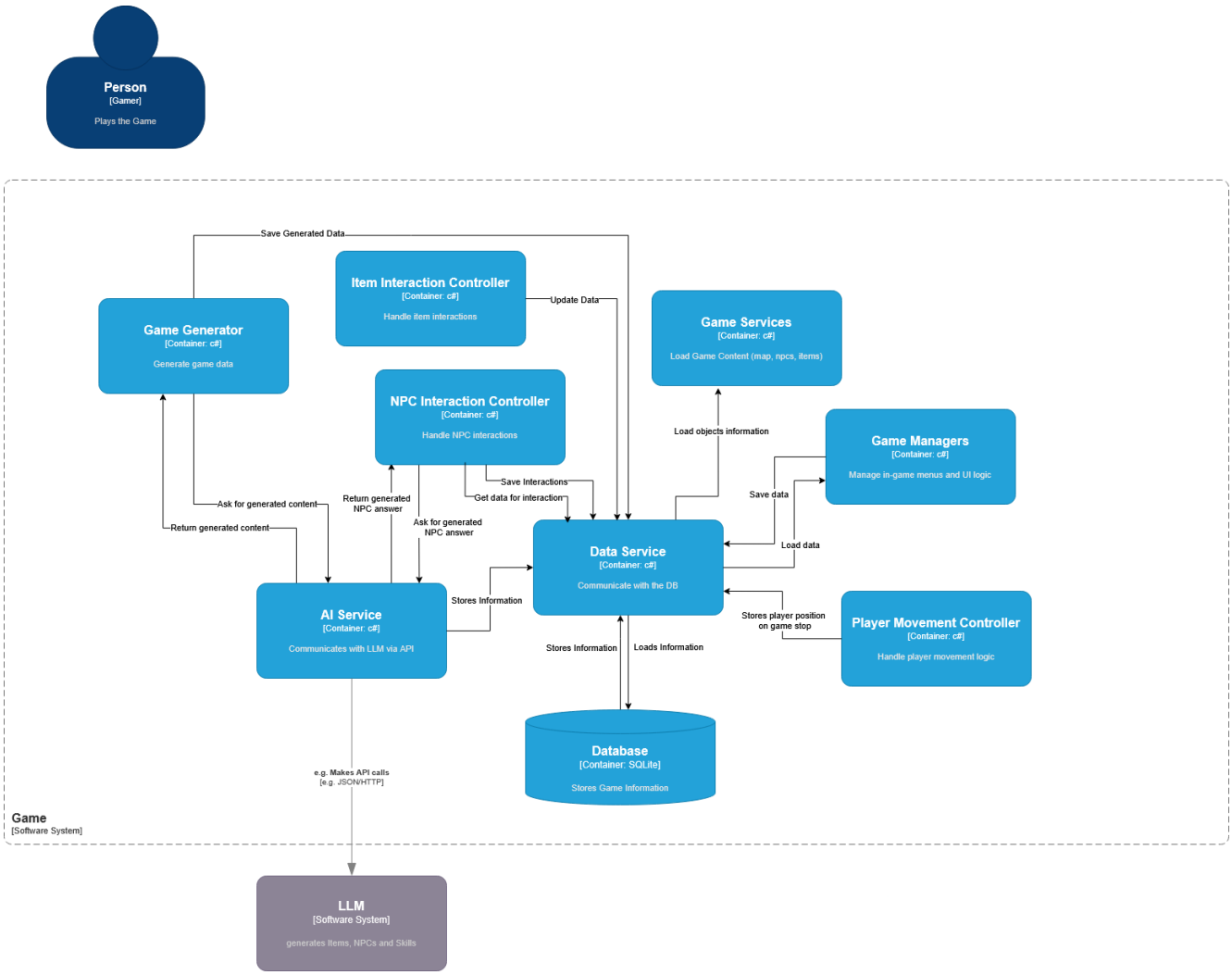

Figure 8.2: C4 Container-Diagramm

8.2 Wireframes

This section provides an overview of the different screens within the game, illustrated using basic wireframes. Each subsection describes the functionality and layout of the key elements in detail.

8.2.1 Main Menu

Figure 8.3 shows the wireframe of the main menu, which is the first screen the player sees when launching the game. It includes four buttons: Play starts a new game or continues an existing one, Settings opens a screen for adjusting options like generation settings or keybinds, Quit exits the game, and Credits displays acknowledgments for the used art assets.

Figure 8.3: Wireframe: Main Menu

8.2.2 Character Creation

Figure 8.4 shows the character creation page, which is displayed the first time a new game is started. The player can customize their own character by selecting options such as hairstyle, shirt, pants, and skin color.

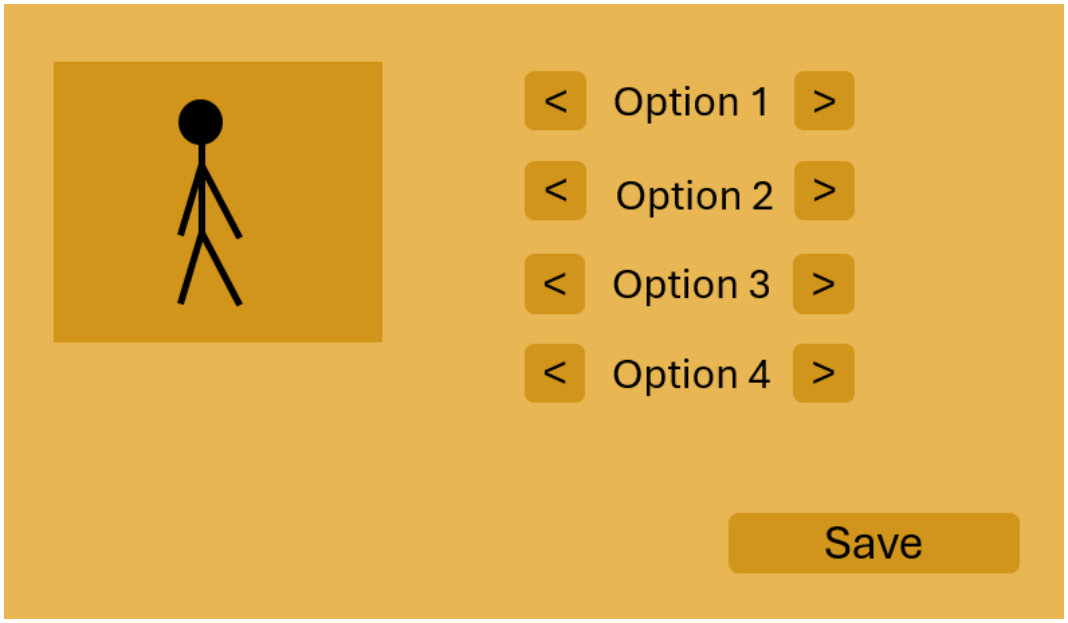

Figure 8.4: Wireframe: Character Creation

8.2.3 NPC Interaction

Figure 8.5 shows how the player interacts with an NPC. When the player presses the interact key (standard: E) while in proximity to an NPC, a dialog window appears. The NPC responds with dynamically generated text from a large language model (LLM), and the player can reply freely by typing their message into the input field.

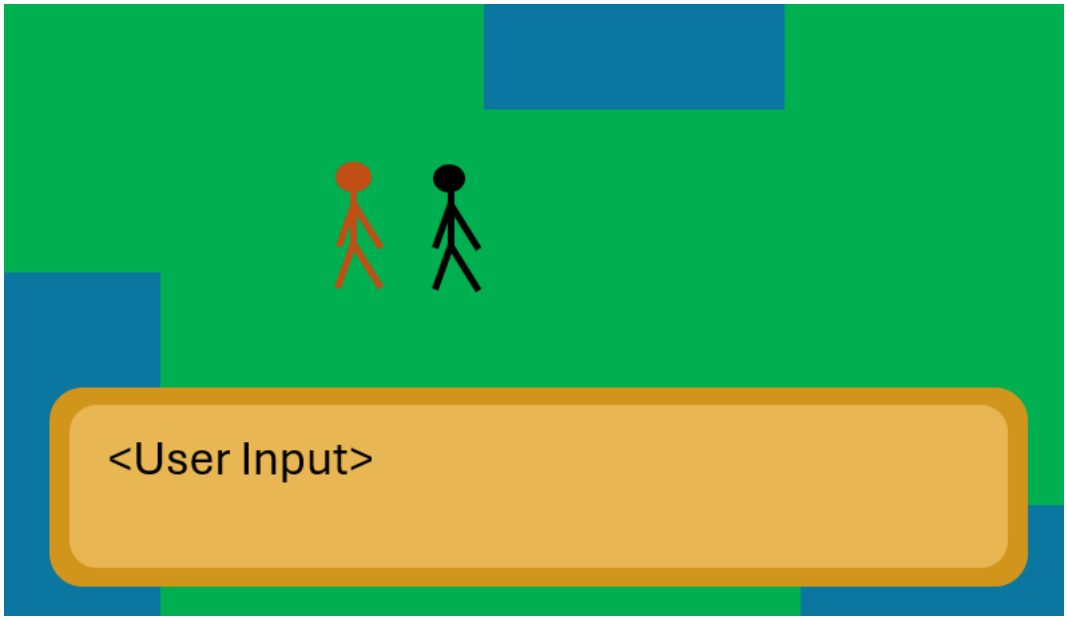

Figure 8.5: Wireframe: NPC Interaction

8.2.4 Item Collection

Figure 8.6 shows how the player can pick up items. Items are generated by the large language model (LLM) and placed on the map. When the player interacts with an item, it is removed from the map and added to the player's inventory.

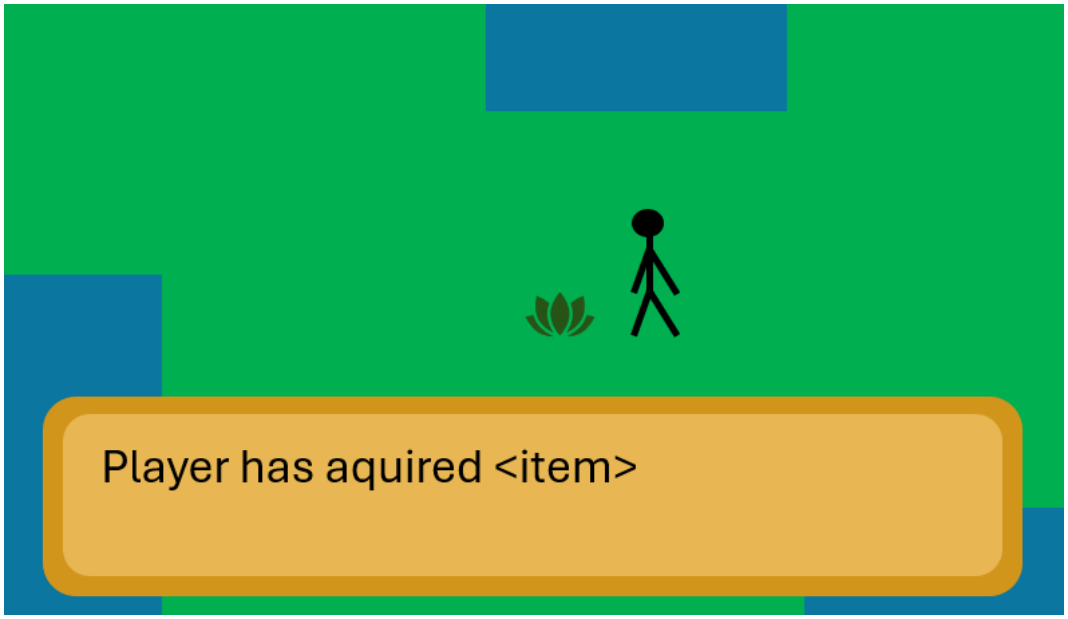

Figure 8.6: Wireframe: Item Collection

8.2.5 Inventory

Figure 8.7 shows the player's inventory. As seen in the image, there are three item categories: materials (used to craft other items), potions (which can be used on characters), and quest items. When an item is selected, the player can view the item's name, icon, and description.

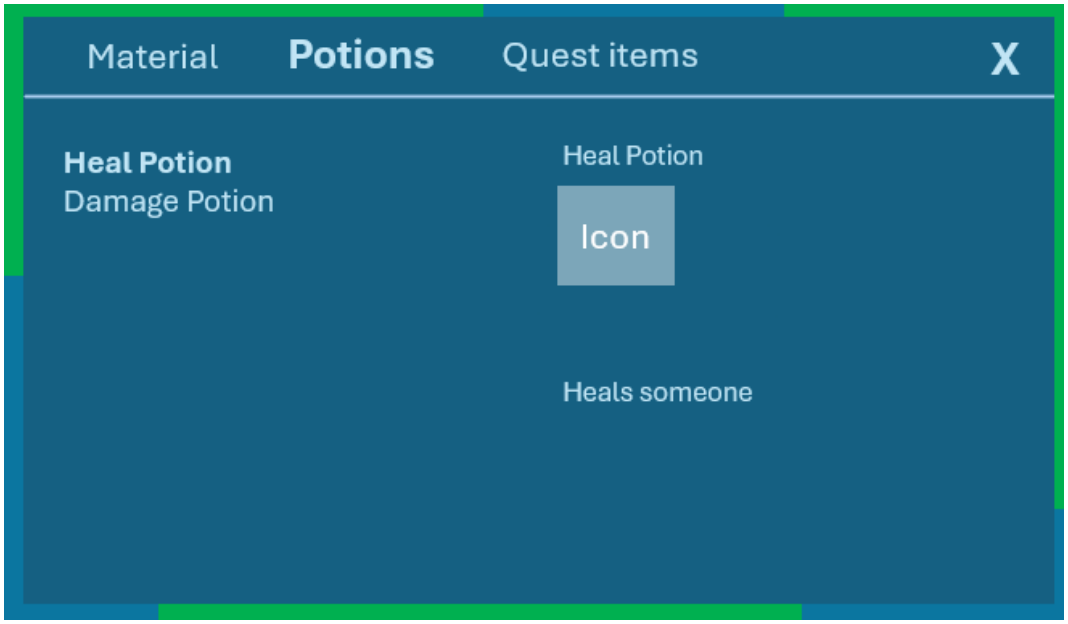

Figure 8.7: Wireframe: Inventory

8.2.6 Quest Menu

Figure 8.8 shows the quest menu. In this interface, the player can view all quests they have started. By selecting a quest, the player can read its description and see the current step that needs to be completed.

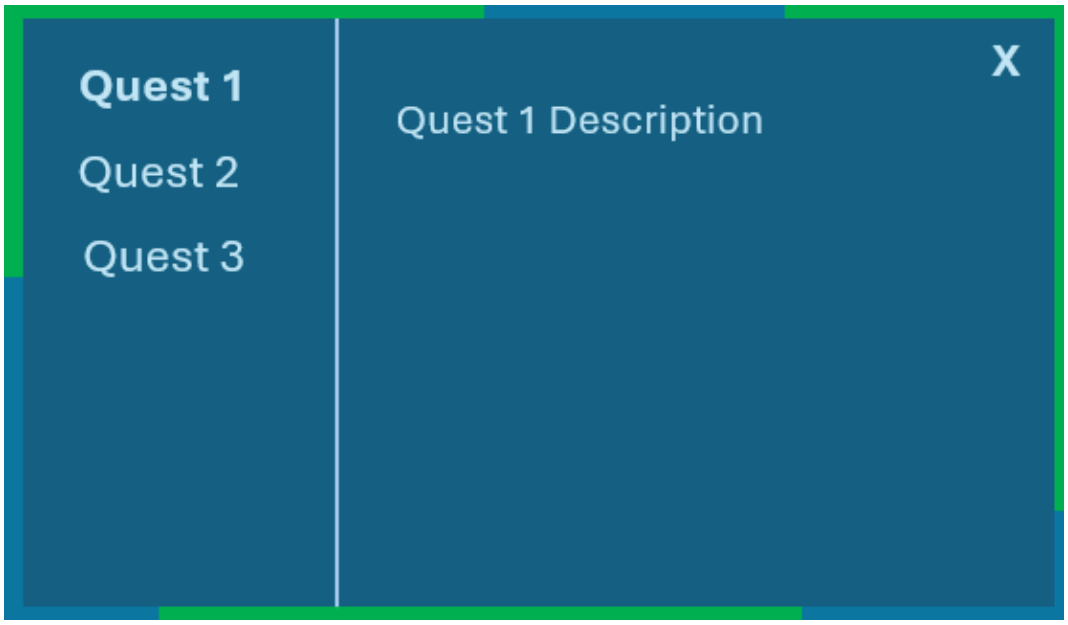

Figure 8.8: Wireframe: Quest Menu

8.2.7 Team Menu

Figure 8.9 shows the team menu. In this interface, the player can choose up to four characters to participate in combat. Only NPCs with a certain level of affection toward the player can be selected. The menu also allows the player to view detailed character information, including attributes and spells. Additionally, the player can select a character as the target for using a potion.

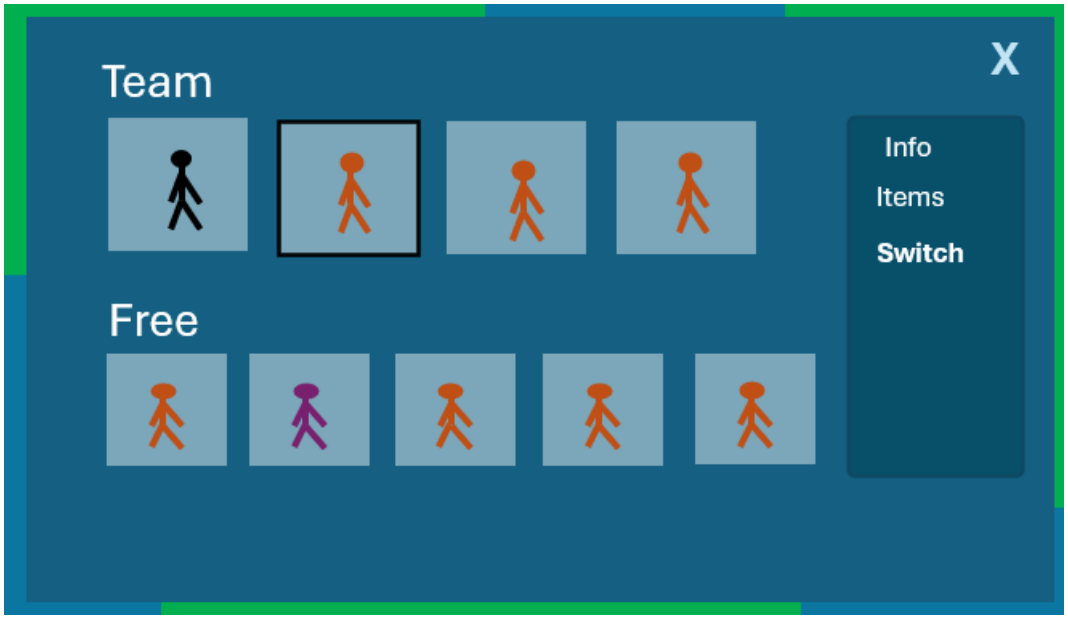

Figure 8.9: Wireframe: Team Menu

8.2.8 Combat

Figure 8.10 shows a combat scene. When the player interacts with an NPC who has negative affection toward them, the game switches to this combat scene. The player and their team are positioned on the left, and the enemy on the right. Combat is turn-based, with turn order based on each character's speed. The turn order is displayed at the top-left of the screen. On the player's turn, they can choose to attack, cast a spell, use a potion, or flee. Spells consume mana points—the more powerful the spell, the more mana it uses. The combat ends when the enemy's health points reach zero, which means the player wins, or when all team members' health points drop to zero, resulting in the player's defeat.

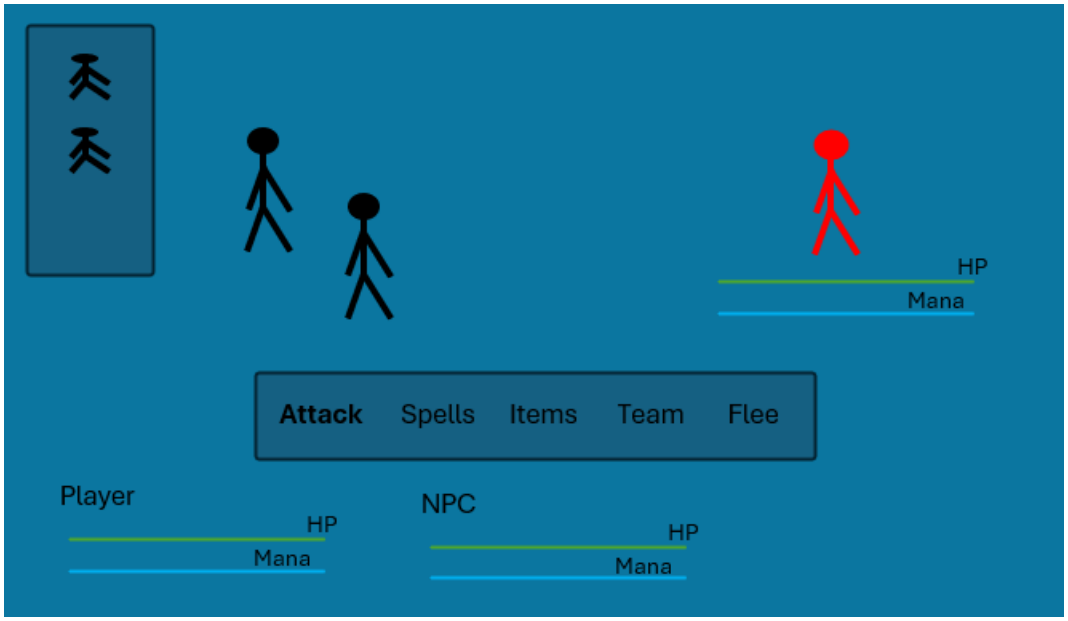

Figure 8.10: Wireframe: Combat

8.2.9 Settings

Figure 8.11 shows the settings page. Here, the user can change keybinds, edit generation prompts, set the API key for the language model (LLM), or reset the game data to start fresh. In this figure, we see how the player can modify the map generation prompt. This prompt is used at the very start of the game to generate the map.

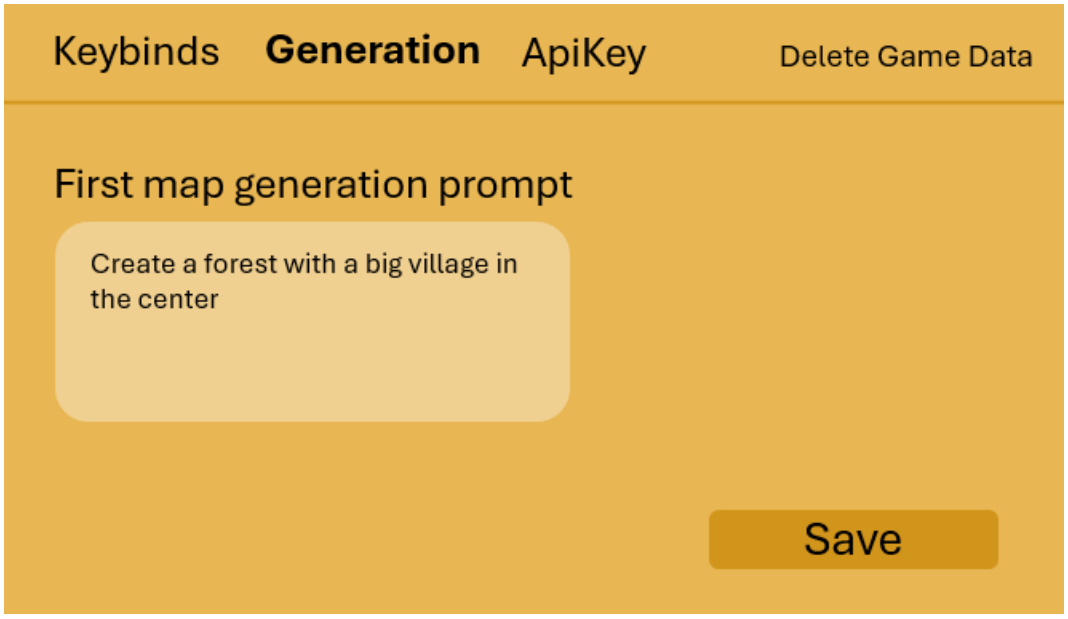

Figure 8.11: Wireframe: Settings

Chapter 9

Implementation und Testing

9.1 Implementation

The game was developed using Unity, a popular game engine that provides a robust scene system. The game consists of seven scenes: the Main Menu Scene, Character Creation Scene, Loading Scene, Game Scene, Combat Scene, Settings Scene, and Credits Scene.

Main Menu

This is the first scene presented to the user. It features a single screen with four buttons: *Play*, *Settings*, *Credits*, and *Quit*. This scene includes minimal logic and primarily serves as a navigation hub for accessing other parts of the game.

Figure 9.1: Main Menu

Character Creation Scene

In this scene, the player can customize their character by selecting from various hairstyles, skin colors, and clothing options. This scene also contains hidden logic, as it is the starting point for generating in-game content.

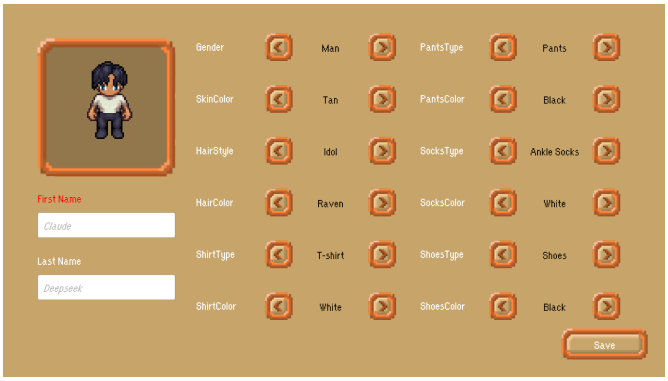

Figure 9.2: Character Creation

Loading Scene

The loading scene is used when game content is still being generated or loaded. It informs the user about the ongoing processes, ensuring they are aware of what is happening in the background before gameplay continues.

Figure 9.3: Loading Scene

Game Scene

This is the main interactive scene of the game. The player can explore the map, interact with NPCs, and collect items.

Figure 9.4 shows how the player can communicate with an NPC by typing a message in the input field. The NPC responds with an AI-generated answer.

Figure 9.4: NPC Interaction

Figure 9.5 shows how the player can pick up items on the map. When collected, the items disappear and are stored in the player's inventory, as seen in Image Figure 9.6.

Figure 9.5: Item Collection

Figure 9.6: Inventory

Figure 9.7 presents the team menu. Here, the player can select up to four characters for battle. By clicking on "Info," more details about each character can be viewed (Figure 9.8).

Figure 9.7: Team Menu

Figure 9.8: Character Information

Figure 9.9 displays the quest menu, where players can see all currently started quests.

Figure 9.9: Quest Menu

Combat Scene

The combat scene is triggered when the player interacts with an NPC having a negative affinity toward them. A battle begins with the player's team on the left and the enemy team on the right. The combat ends when one of the teams has no surviving characters.

Figure 9.10: Combat Scene

Settings Scene

This scene allows the player to adjust game options. They can change keybindings, edit the prompt used for map generation, enter or update their API key, or delete saved game data to start over.

Figure 9.11: Settings

Credits Scene

The credits scene is dedicated to acknowledging the artists whose work was used in the game. All visual assets were sourced from OpenGameArt.org[1], and this scene lists the original contributors of those assets.

9.2 Manuel Tests

Only manual tests were carried out for our game. We regularly checked various game mechanics directly in the running game,in particular the combat system, the effects of spells, status changes of units, and the user interface.

[1] https://opengameart.org

Another important part of the testing was interaction with NPCs. Here, we checked whether dialogues were triggered correctly, quest conditions were met, and markers were displayed correctly on the map. The interplay between quest logic, game progress, and visual feedback was also taken into account.

In principle, it would have been possible to implement automated unit tests. However, due to the high dynamics and strongly procedural structure of much of the game content, such as AI-generated spells, random enemies, or varying events, this would have involved a disproportionately high amount of effort. This content is difficult to test in isolation, as it is closely linked to the context and state of the entire game.

To ensure a stable gaming experience nonetheless, we conducted numerous manual tests with different constellations, game progressions, and extreme situations, paying particular attention to consistency, game flow, and accuracy.

9.2.1 User testing

To evaluate the game demo, user tests were conducted with seven volunteer participants. The testers represented a broad range of ages (22–62) and gaming experience levels, from beginners to hardcore gamers. Feedback was collected using a structured questionnaire with both quantitative ratings and qualitative comments. The results provided valuable insights into game clarity, balance, usability, and technical stability.

9.2.2 User testing feedback

Participant 1
Age: 22
Experience: Hardcore Gamer
Visual Design: Good
Ease of Understanding: Okay
Gameplay Flow: 2
Combat Excitement: 3
Goal Clarity: Partially
Difficulty: Too easy
Controls: 4
Situations of Confusion: Quest could not be turned in
Story Interest: 1
NPC Interactions: 3
Bugs: Items spawned in houses, quest blocked
Overall Rating: 3
Favorite Aspects: –
Suggestions: Sound design
Comments: –

Participant 2
Age: 26
Experience: Experienced
Visual Design: Very good
Ease of Understanding: Okay
Gameplay Flow: 4
Combat Excitement: 2
Goal Clarity: Partially
Difficulty: Easy
Controls: 3
Situations of Confusion: UI hints missing, unclear quest display
Story Interest: 4
NPC Interactions: 4
Bugs: Could not complete quests
Overall Rating: 3
Favorite Aspects: –
Suggestions: UI hints, quest log, map legend
Comments: –

Participant 3
Age: 62
Experience: Beginner
Visual Design: Very good
Ease of Understanding: Somewhat hard
Gameplay Flow: 5
Combat Excitement: 4
Goal Clarity: Partially
Difficulty: Hard
Controls: 1
Situations of Confusion: Did not know how to move or talk
Story Interest: 4
NPC Interactions: 5
Bugs: –
Overall Rating: 4
Favorite Aspects: –
Suggestions: Language selection option
Comments: –

Participant 4
Age: 24
Experience: Hardcore Gamer
Visual Design: Very good
Ease of Understanding: Easy
Gameplay Flow: 5
Combat Excitement: 3
Goal Clarity: Yes
Difficulty: Too easy
Controls: 5
Situations of Confusion: What happens after finishing all quests?
Story Interest: 1
NPC Interactions: 4
Bugs: Combat bug, crash, API key issue on save
Overall Rating: 4
Favorite Aspects: –
Suggestions: More quests, higher difficulty
Comments: –

Participant 5
Age: 28
Experience: Casual Player
Visual Design: Very good
Ease of Understanding: Easy
Gameplay Flow: 4
Combat Excitement: 4
Goal Clarity: Yes
Difficulty: Too easy
Controls: 5
Situations of Confusion: What to do when finished?
Story Interest: 1
NPC Interactions: 5
Bugs: –
Overall Rating: 4
Favorite Aspects: –
Suggestions: –
Comments: –

Participant 6
Age: 28
Experience: Hardcore Gamer
Visual Design: Very good
Ease of Understanding: Very easy
Gameplay Flow: 4
Combat Excitement: 4
Goal Clarity: Partially
Difficulty: Too easy
Controls: 5
Situations of Confusion: Too many spells from the start, only 1v1 fights
Story Interest: 4
NPC Interactions: 5
Bugs: –
Overall Rating: 4
Favorite Aspects: –
Suggestions: Spell system, more variety
Comments: –

Participant 7
Age: 25
Experience: Beginner
Visual Design: Very good
Ease of Understanding: Okay
Gameplay Flow: 5
Combat Excitement: 3
Goal Clarity: Yes
Difficulty: Easy
Controls: 3
Situations of Confusion: –
Story Interest: 3
NPC Interactions: 5
Bugs: Long loading time, initial API bug
Overall Rating: 4
Favorite Aspects: –
Suggestions: –
Comments: –

9.3 Non-functional requirements evaluation

The implementation of the non-functional requirements was evaluated based on observed system behavior, empirical measurements, and qualitative user feedback. Below is a summary of the current fulfillment status:

NFR-01 (Map Loading Time): The map loads instantly with no visible delay, fulfilling the "Best" level of this requirement.

NFR-02 (AI Fairness and Challenge): Based on subjective feedback from user tests, the AI is perceived as fair and balanced, corresponding to the "Good" level.

NFR-03 (AI Cost per API Call): API costs vary: generation calls reach up to 0.1 CHF, while dialogue interactions with NPCs typically remain below 0.1 CHF. Therefore, this requirement is fulfilled at the "Minimum" level, with potential for optimization.

NFR-04 (Loading Time after Generation): Currently, loading time after generation is reported as *X* seconds. If this is 5 seconds, the requirement is met at the "Minimum" level.

NFR-05 (Game Generation Time): Game generation can take up to one minute. This exceeds the defined threshold of 30 seconds and therefore does not currently meet the requirement.

NFR-06 (AI Decision Time): NPC AI decisions are executed in under 1 second under good internet conditions, fulfilling the "Good" level.

NFR-07 (Frame Rate): The system consistently runs at over 120 FPS, fulfilling the "Best" level.

NFR-08 (Data Security): Player and game data are stored in a database. However, no explicit mention of encryption was made, which suggests this requirement is currently fulfilled at the "Minimum" level.

NFR-09 (Anti-Cheat): Anti-cheat mechanisms are implicitly provided by OpenAI's underlying systems, meeting the "Minimum" requirement, though not yet tailored to in-game behavior.

NFR-11 (AI Usage Restriction): The AI is restricted to its intended in-game usage and not repurposed, fulfilling this requirement at the "Minimum" level.

NFR-12 (Error Handling): Generic error messages are currently implemented, satisfying the "Minimum" level. More specific, user-friendly messages could improve this.

NFR-13 (User Interface): The UI has been described as intuitive by test participants, fulfilling the "Minimum" usability requirement.

NFR-14 (Controls and Interaction): Controls were generally understood and intuitive, meeting the "Minimum" level.

NFR-15 (Tutorials): No tutorial system is currently in place. As such, this requirement is not yet fulfilled.

NFR-17 (Logging and Analytics): Basic logging is available, fulfilling the "Minimum" requirement. Structured and filterable logs would improve maintainability.

Summary

Most non-functional requirements are fulfilled at least at the "Minimum" level, with several achieving "Good" or even "Best" status (e.g. frame rate and map loading). Improvement potential exists particularly in the areas of game generation speed, error handling, AI cost efficiency, and user onboarding through tutorials.

Chapter 10

Result and future developement

10.1 Results

As a result of the development process, a fully functional game was created. The core mechanics work as intended and demonstrate the successful integration of LLM-driven systems for content generation, interaction, and game progression.

10.1.1 AI-powered game functionality

The core game features are built around the integration of OpenAI, enabling a dynamic and interactive game play experience by automatically generating various types of content:

- **Map Generation:** Individual maps are created based on text prompts, resulting in diverse structures and visual styles that enhance replayability.
- **Quest:** The LLM generates quests, including specifics actions the player must perform, involved NPCs and required items. NPCs are generated with a backstory and a personnality.
- **NPC Style:** NPC appearances are generated by a LLM using predefined components as hairstyles, skin colors and clothing. The resulting styles align with each NPC's story and personality.
- **Combat Data and Spells:** Attacks and their respective effects are generated using predefined logic combined with AI input, allowing for creative and varied combat scenarios.
- **Dialogue System with NPCs:** The LLM enables natural conversations with NPCs, taking into account both their personality and the current game context.

Figure 10.1 shows how the LLM is used for the game.

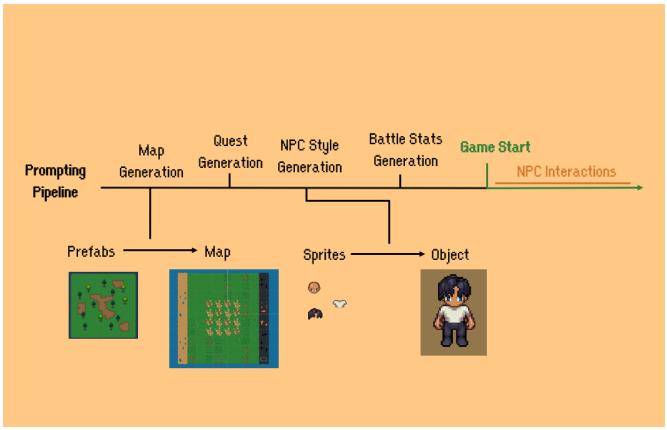

Figure 10.1: LLM Use

Data Persistence: All relevant game data is stored using an SQLite database. This lightweight solution is well-suited for small to medium-sized projects and provides fast and reliable storage for game progress, character data, and map elements.

10.2 Future development

This chapter outlines features and improvements that are planned for future development to further enhance functionality, user experience, and game depth.

10.2.1 Integration of other LLMs

An extension is planned to support additional large language models (LLMs), such as the OST LLM Hub. This would allow for model flexibility and performance comparisons and could enable specific features based on specialized models.

10.2.2 Dedicated LLM for game content

A custom fine-tuned LLM specifically tailored for the game world and its mechanics could be developed. This would enable faster responses, deeper contextual understanding, and unique in-game behavior aligned with the game's narrative and logic.

10.2.3 AI-Generated icons

To enhance the visual identity and variety of the interface, icons for categories or items could be generated dynamically using an image generation AI. This would allow for a more personalized and thematic visual design.

10.2.4 Update of the OpenAI for Unity Project

The existing "OpenAI for Unity" GitHub project is planned to be updated to support the latest API features, optimize performance, and ensure compatibility with newer Unity versions and tool-chains.

10.2.5 Map legend

A visual legend for the map is planned to help users understand the meaning of various symbols, colors, and icons used in the interface. This would improve orientation, especially in complex maps.

10.2.6 Language settings

The addition of language settings would allow users to switch the UI and in-game texts to their preferred language. This would increase accessibility and usability for a broader audience.

10.2.7 Extended quests with multiple objectives

Support for longer quests with multiple objectives is planned. This would make the gameplay more engaging and allow for deeper story progression, branching paths, and more complex mission structures.

10.2.8 Interactive tutorial

An in-game tutorial is planned to guide new players through basic mechanics, navigation, and interactions. This tutorial will be structured, easy to follow, and fully integrated into the game flow.

10.2.9 AI game controller

A planned feature is an intelligent game controller NPC that players can ask for help. This controller could explain mechanics, guide the player, move the character, or simulate a battle to demonstrate gameplay.

10.2.10 Dynamic map behavior

Future quests could dynamically alter the map environment. For example, weather conditions could change based on quest progress, NPCs might disappear or appear, and regions of the map could be locked or unlocked depending on story events.

10.2.11 Regenerating quests after completion

A planned feature is the automatic generation of new quests once all current quests have been completed. This would ensure continued game play and provide long-term engagement by dynamically expanding the game content.

10.2.12 Expandable map on exit

When the player moves beyond the current boundaries of the map, the map will dynamically expand. This would allow for exploration of new areas and the generation of fresh content without predefined limits.

10.2.13 More Prefabs for environmental variety

To increase visual and structural diversity in the game world, additional prefabs are planned for terrain, buildings, and interactive objects. This would help make each play through feel more unique and immersive.

10.2.14 Achievements for milestones

An achievement system is planned to reward players for reaching specific milestones, such as completing quests, discovering areas, or interacting with key NPCs. This would add an extra layer of motivation and progression tracking.

10.2.15 Smarter combat AI

Improvements to the combat AI are planned to make enemy behavior more strategic and context-aware. Smarter decision-making, varied tactics, and adaptive reactions to player actions would make battles more challenging and engaging.

10.2.16 AI-Generated music and sound effects

A planned feature is the integration of a music AI that dynamically generates background music and sound effects based on the current game situation. This would add an additional immersive layer to the game by adapting audio atmospheres to the environment, player actions, or quest progress in real time.

10.2.17 NPC behavior and actions

A planned feature is to extend NPC behavior by allowing them to perform visible in-game actions such as walking around the map, gathering items, or interacting with objects. This would make the world feel more alive and increase immersion by giving NPCs autonomous, context-driven behavior beyond static dialogue.

Chapter 11

Quality Measures

11.1 Quality assessment tools

11.1.1 Linter

SonarLint
SonarLint was used to improve code quality in the Unity project. It helped identify potential bugs, code smells, and maintainability issues in the C# code. Most of the reported issues were fixed during development.

11.1.2 Guidelines

Definition of Done

- All code related to the user story has been implemented and adheres to the coding standards.
- The code has been peer-reviewed and approved by another team member.
- The functionality meets both functional and non-functional requirements.
- All sub-tasks linked to the main task have been completed.
- All necessary documentation has been created and updated.

11.1.3 Git branching and merges

To avoid errors, direct pushes to the main branch are not allowed. After a team member has implemented a feature, a merge request can be created on GitLab and assigned to a reviewer. The reviewer checks the changes and ensures that they meet the Definition of Done. If everything is in order, the merge request is approved by the reviewer.

11.2 Environment

This section describes the technical environment used during development to ensure consistent quality.

- **IDE:** JetBrains Rider[1]
  JetBrains Rider was used as the primary integrated development environment (IDE) due to its strong support for C# and Unity development.
- **Operating System:** Windows 10[2] or Windows 11[3]

  Development was carried out on machines running either Windows 10 or Windows 11 to ensure compatibility with Unity and Rider.
- **Unity Version:** Unity 6[4]
  A consistent Unity version was used across all development environments to prevent version-related issues and maintain build stability.
- **Version Control:**Git[5]
  Git was used for version control.

11.3 CI/CD

11.3.1 Workflow

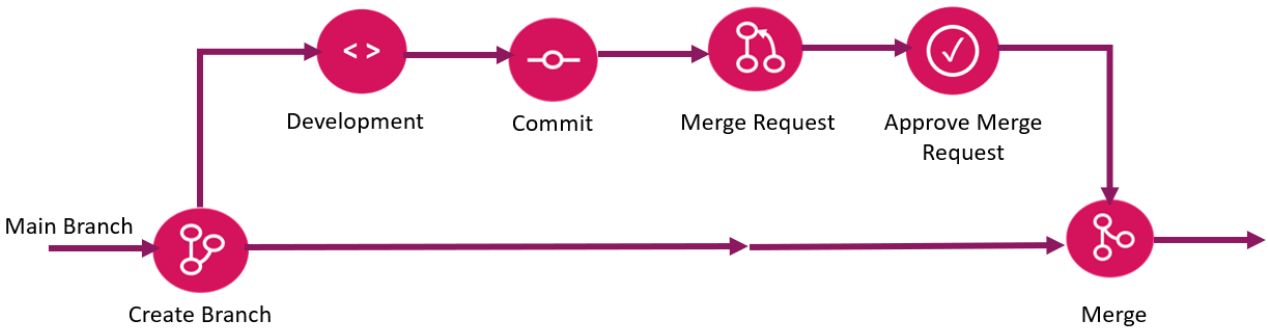

Figure 11.1: Workflow

[1] https://www.jetbrains.com/rider/
[2] https://www.microsoft.com/en-us/software-download/windows10
[3] https://www.microsoft.com/en-us/software-download/windows11
[4] https://unity.com/
[5] https://git-scm.com/

11.4 Communication tools

Discord

Discord is primarily used as a communication tool. It enables communication via voice and text chat and is also used to share screens and discuss issues.

Teams

Teams is used to exchange information with the supervisors and to hold important meetings.

Chapter 12

Software documentation

12.1 Technology stack

This chapter summarizes the key technologies and tools used in the development of the game. It also outlines important prerequisites that must be met in order to run and use the game effectively.

Area	Technology	Description
Game Engine	Unity	2D game development, scene management, animations, and UI
Programming Language	C#	Main language for game logic, API calls, and data processing
LLM Integration	OpenAI (GPT-4o-mini, o3-mini)	Generates dynamic game content (dialogues, quests) via HTTP API
API Integration Tool	OpenAI-Unity (srcnalt)	Unity plugin for accessing OpenAI's API (text and image generation) within the game engine
Database	SQLite	Embedded relational database for save data and persistent info
DB Framework	SQLite4Unity3d	Unity-compatible ORM for easy database operations

Table 12.1: Key technologies used in the game development

12.2 Tool stack

Tool	Usage
JetBrains Rider	IDE for code development
Unity Editor	Integrated environment for game development
DB Browser for SQLite	Graphical tool used for debugging and managing database files
ChatGPT	Text generation, optimization, spell and grammar checking, translation
DeepL	Translation
GitHub Copilot	Assistance with code creation

Table 12.2: Tool Stack

12.3 Installation

This section describes how to set up and run the game.

12.3.1 System requirements

- Operating System: Windows 10 or later
- RAM: Minimum 8 GB
- Storage: At least 4 GB free disk space

12.3.2 Development environment setup

1. Install Unity Hub from https://unity.com/download.
2. Clone the project repository from https://gitlab.ost.ch/llm-game/ai-game.
3. Open the project folder in Unity Editor.
4. Open JetBrains Rider or preferred IDE and open the project solution.

12.3.3 Building and running the game

1. Build the game using Unity's build settings.
2. Run the built game.
3. When the game starts for the first time, open the Settings and enter your **API key** and **Organization name** to enable AI-based features.

Appendix A

Glossary

Term	Description
AI	Artificial Intelligence refers to the capability of machines or software to perform tasks that typically require human intelligence, such as learning, reasoning, or language understanding.
LLM	Large Language Model is a type of AI model trained on large text datasets to generate human-like text and understand natural language input.
RPG	Role-Playing Game is a game genre where players assume the roles of characters in a fictional setting and participate in a narrative-driven experience.
NPC	Non-Playable Character is a character in a game that is not controlled by the player but is part of the game world, usually controlled by the game logic or AI.
Top-Down	A top-down perspective refers to a viewpoint in games where the player looks at the game world from above, often used in strategy and 2D RPG games.
API	Application Programming Interface is a set of routines, protocols, and tools that allow software applications to communicate with each other.
FR	Functional Requirement describes what a system should do to meet user needs.
NFR	Non-Functional Requirement describes how a system should perform its functions, including aspects such as performance, security, and usability.
LaTeX	A typesetting system commonly used for academic and scientific documents.
Scrum	Scrum is an agile project management framework that helps teams develop complex projects through iterative and incremental steps.
RUP	Rational Unified Process is an iterative software development process that divides development into phases and defines best practices for software engineering.

Term	Description
GPT-4o-mini / o3-mini	Two different language models by OpenAI. GPT-4o-mini is optimized for cost efficiency and speed, while o3-mini provides more consistent and deeper outputs.
Prefab	A reusable game object in Unity that stores a preset configuration of components and properties, often used for environments or NPCs.
Buff / Debuff	Temporary effects that increase (buff) or decrease (debuff) a character's combat stats such as attack or defense or reduce health after rounds .
Crit / CritMultiplier	Critical hit – a combat mechanic where attacks deal increased damage. The CritMultiplier defines the strength of this effect.
Variance	A random factor applied to damage calculations, usually to introduce variability and realism. Typically ranges between 0.85 and 1.0.
Tilemap	A grid-based map structure used in 2D games where predefined tiles are arranged to form the game environment.
Turn Count	The number of actions a character has taken in a turn-based system; it influences when they can act again.
FlatRaise / FlatReduction	Static values added to or subtracted from damage calculations to adjust the outcome directly.
STAB	Same-Type Attack Bonus – a damage multiplier applied when a character uses an attack that matches their own element/type.
ActionCost	The time cost of a combat action; affects how quickly the character can act again.
StateRaise	A multiplier applied to stats based on temporary status boosts or reductions (e.g., increased attack or lowered defense).
Prompt	A textual instruction sent to a language model to generate content such as dialogue, quests, or map layouts.
Initiative	A combat value determining how quickly a character acts in turn-based systems; higher initiative means earlier turns.
Hitman 3	A stealth-based video game developed by IO Interactive, where players take on the role of Agent 47 and complete missions using strategic planning, disguises, and creative problem solving. It is often cited as an example of player freedom and open-ended gameplay.
Sandbox Game	A game genre that offers players a high degree of freedom to explore, interact with, and modify the game world, often with minimal structured objectives. Examples include Minecraft, Garry's Mod, and parts of Hitman 3.

List of Figures

List of Tables

Bibliography

[1] Latitude, “Ai dungeon,” 2019, accessed May 2025. [Online]. Available: https://play.aidungeon.io

[2] H. Door, “Hidden door — interactive storytelling with ai,” 2024, accessed May 2025. [Online]. Available: https://www.hiddendoor.co

[3] A. Tavakkoli and M. Lee, “Procedural content generation using large language models: Opportunities and challenges,” *Proceedings of the 2023 FDG Conference*, 2023.

[4] Decart and Etched, “Oasis: An ai-generated open-world game,” 2023, accessed May 2025. [Online]. Available: https://www.decart.com/oasis

[5] G. Research, “Gamengen: Ai-powered game simulation from google research,” 2024, accessed May 2025. [Online]. Available: https://g.co/research/gamengen

[6] U. Technologies, “Unity muse and sentis: Ai tools for game development,” 2023, accessed May 2025. [Online]. Available: https://unity.com/products/unity-muse