Projektarbeit Sommer 2025

Fachinformatiker für Anwendungsentwicklung

Dokumentation zur schulischen Projektarbeit

Abgabedatum: Hochfeld, den 17.07.2025

Prüfungsbewerber:

Andreas Betz

Deubacher Str. 1

89335 Ichenhausen

Ausbildungsbetrieb:

ABUS Security Center GmbH & Co. KG

Linker Kreuthweg 5

86444 Affing

DND - TEXTADVENTURE

„TEXT-ADVENTURE MIT DND-THEMATIK UND DYNAMISCHER INHALTSERZEUGUNG ÜBER JSON“

DnD-Textadventure

„Text-Adventrue mit DnD-Thematik und dynamischer Inhaltserzeugung über JSON“

1. Einleitung

1.1. Projektumfeld

Das Projekt entstand im Rahmen der Projektwoche an der Berufsschule 7 und wurde unter schultypischen Rahmenbedingungen umgesetzt – so stand uns beispielsweise ein Computerraum zur Verfügung mit einer Workstation pro Schüler, welche je mit einem Bildschirm ausgestattet sind.

Der Auftraggeber stellt somit unser Lehrer Hr. Aicher dar.

1.2. Projektziel

Ziel des Projektes ist ein modular erweiterbares Text-Adventure, welches thematisch auf dem beliebten Tabletop-Role-Playing-Game (TTRPG) „Dungeons and Dragons“ (DnD) basiert.

Das Spiel soll nicht nur der Unterhaltung dienen, sondern auch eine technische Umsetzung einer API-Anbindung und einem dynamisch einsetzbaren Text-Parser darstellen.

1.3. Projektbegründung

Das Projekt stellt ein „Proof of Concept“ – also ein funktionaler Prototyp, welcher demonstrieren soll, dass das zugrunde liegende Konzept auf eine praktikable Art und Weise durchführbar ist.

Dieser Beweis könnte dann als Grundlage für ein vollwertiges, vermarktbares Produkt dienen.

1.4. Projektschnittstellen

Das Endprodukt stellt eine vollständig kompilierte Software, welche mithilfe der „nlohmann::json“-Library Textdateien lokal erstellen & bearbeiten kann.

Des weiteren ist noch eine Schnittstelle zur API von Open5e geplant, um auf offizielle DnD-Daten zugreifen und in das Spiel integrieren zu können.

Die Projektabnahme sowie die abschließende Präsentation erfolgen durch den Auftraggeber Hr. Aicher von der Berufsschule 7.

1.5. Projektabgrenzung

Zu betonen ist, dass dieses Produkt kein Ersatz für eine digitale DnD-Spielumgebung darstellt. Stattdessen handelt es sich um ein eigenständiges Text-Adventure, welches auf der Welt von DnD basiert.

2. Projektplanung

2.1. Projektphasen

Projektphase	Geplante Zeit
Analyse	1h
Entwurf	2,75 h
Implementierung	14,25 h
Abnahme	-
Einführung	-
Dokumentation	6,75 h
Gesamt	24,75 h

Die Phasen „Abnahme“ und „Einführung“ sind für dieses Projekt nicht relevant, da es sich um ein vollständig abgeschlossenes Einzelprojekt handelt, welches nicht in ein bestehendes System integriert wird.

2.2. Abweichungen vom Projektantrag

Die Implementierungsphase nahm mehr Zeit in Anspruch als ursprünglich geplant:

Statt wie vorgesehen am Mittwochvormittag abzuschließen, verlängerte sich die Arbeitszeit um ca. 2,25 Industriestunden auf insgesamt 16,5 Stunden. Die Dokumentation wurde dadurch komprimiert und in 4,5 Stunden fertiggestellt.

Zusätzlich musste die API-Anbindung auf ein funktionales Minimum reduziert werden, da sich der Autor im Aufwand verschätzte.

Im Sourcecode sind die Funktionen für das Abrufen der Open5e Daten zwar vorhanden, jedoch wurden diese nicht aktiv in das Projekt eingebunden und bilden momentan noch keinen Teil des spielbaren Abenteuers.

2.3. Entwicklungsprozess

Für die Umsetzung wurde ein iterativer Prototyping-Ansatz gewählt. In mehreren Entwicklungsschritten wurden einzelne Funktionen getestet, evaluiert und allmählich erweitert.

Dies ermöglicht eine kontinuierliche Verbesserung des Designs und der Funktionalität.

3. Analysephase

3.1. Ist-Analyse

Im Repertoire des Autors befinden sich bereits mehrere Projekte, die mit der gewählten Programmiersprache Text-Parsers und API-Calls implementieren, jedoch wurde von ihm noch nie ein Text-Adventure entwickelt, welches diese Elemente kombiniert und dynamisch mit externen Text-Dateien implementiert.

3.2. Nutzwertanalyse

Das entwickelte System – insbesondere der Text- und JSON-Parser – wurde so konzipiert, dass es künftig als wiederverwendbarer Code dienen kann.

Dadurch reduziert sich der Entwicklungsaufwand für ähnliche Projekte erheblich.

3.3. Anwendungsfälle

Hauptanwendungsfall ist die Verwendung als technisches Grundgerüst für zukünftige Text-Adventures.

Darüber hinaus kann der Parser in anderen Projekten zum Einsatz kommen, bei denen eine Verarbeitung von User-Eingaben erforderlich ist.

3.4. Qualitätsanforderungen

Performance:

Die Anwendung muss auch auf älteren Rechnern zuverlässig ohne merkliche Ladezeiten funktionieren

Usability:

Das Text-Adventure soll intuitiv bedienbar sein – auch für Nutzer*innen* ohne technisches Vorwissen.

Gleichzeitig soll der Quellcode verständlich und wartbar für Entwickler*innen* mit C++-Kenntnissen sein.

Effizienz:

Texteingabe und Dateioperationen sollen mit möglichst wenigen und schnellen Zugriffen ausgeführt werden

4. Entwurfsphase

4.1. Zielplattform

Es wurde sich dafür entschieden, mit C++ auf Windows 11 zu entwickeln.

- Mit C++ können wir unser Qualität-Ziel der Performance mit Leichtigkeit einhalten und die fertige Anwendung wird mit allen notwendigen Abhängigkeiten kompiliert – Dies erleichtert die Verteilung.
- Windows 11 wurde gewählt, das dies der zur Verfügung gestellte Entwicklungsrechner besitzt

Zusätzliche Entscheidungen:

- Entwicklungsumgebung: MSYS2 mit g++-Compiler in der ucrt64-Umgebung
- Bibliotheken
    - nlohmann::json zur Verarbeitung von JSON-Dateien
    - libcurl zur API-Kommunikation
    - C++ Standardbibliothek

4.2. Architekturdesign

Es wurde das MVC-Architekturpattern (Model-View-Controller) verwendet, um die Anwendung in logisch getrennte Bereiche zu gliedern:

- **Model:** verwaltet die Spieldaten (JSON-Dateien)
- **View:** übernimmt die Darstellung des UI
- **Controller:** verbindet Eingabe, Datenverarbeitung und Ausgabe

Diese Trennung ermöglich eine hohe Skalierbarkeit und bessere Wartbarkeit.

4.3. Entwurf der Benutzeroberfläche

Das Spiel wird über eine einfache Kommandozeile bedient. Die Nuter*innen* erleben die Spielwelt ausschließlich über Textein- undausgaben.

Zur Verbesserung der Lesbarkeit und der generellen Erfahrung kommen Leerzeilen und ein Typewriter-Effekt zum Einsatz, bei dem der Text zeichenweise mit kleiner Verzögerung erscheint.

4.4. Datenmodell

Die Datenstruktur basiert vollständig auf JSON-Dateien.

Diese werden beim Start des Spieles kopiert und diese Sitzungskopien, werden dann im Laufe des Spieles verändert, um den aktuellen Spielstand darzustellen.

Es existiert derzeit keine Speicherfunktion, jedoch könnte diese ohne großen Aufwand nachgerüstet werden, indem die Spielstände als eigene JSON-Dateien abgelegt werden.

Es wurde sich für JSON (JavaScript Object Notation) entschieden, da damit schnell und effizient Textoperationen programmiell durchgeführt werden können, ohne dass „Menschenlesbarkeit“ verloren geht.

4.5. Geschäftslogik

Die Anwendung folgt dem MVC-Arichtekturpattern. Dementsprechend haben wir eine View-Klasse, eine Model-Klasse und eine Controller-Klasse.

Das Programm wird über die main-Funktion gestartet.

Dann läuft der Gameloop (ein sich wiederholender Zustand, zu dem ein Spiel immer wieder zurückkehrt, bis es beendet wird) über den Controller, welche Daten vom Modell abfragt, auswertet und anschließend an den Viewer weitergibt, welcher die UI steuert.

Gleichzeitig fängt der Controller auch noch User-Input auf, prüft diese auf Richtigkeit und stößt die passenden Daten-Interaktionen im Modell an. Letztlich wird dann wieder durch den Viewer ausgegeben.

4.6. Maßnahmen zur Qualitätssicherung

Das Produkt wurde durch regelmäßige Prototypen getestet und weiterentwickelt.

Zusätzlich wurde detailliertes Error-Handling eingebaut, welches Abstürze durch invalide Texteingabe verhindern soll.

5. Implementierungsphase

5.1. Implementierung der Datenstrukturen

Die Daten wurden auf drei getrennten JSON-Dateien aufgeteilt:

„baseInteractors.json“

Stellt alle Objekte (im Folgendem „Interaktoren“ genannt) innerhalb des Adventures dar. Gleichzeitig sind dort auch alle Interaktionen mit den Objekten notiert.

Diese Datei wird bei Programmstart kopiert, sodass die Kopie dann beschrieben werden kann.

Alle Interaktionen, die der Spieler zu lesen bekommen kann, werden mit dem Status als Nachläufer versehen. Ihr Value wird als Array mit 2 Inhalten gespeichert:

- Den Text, den der Spieler zu lesen bekommt
- Einen Integer, welche die Interaktion, die mit dem Interaktor geschehen soll, beschrieben ist

```
"jar": {
    "status": "_",
    "use_": ["You try to use the jar. It does not respond.", 0],
    "attack_": ["You hit the jar. It breaks into a million pieces", 2],
    "look_": ["A glass jar, empty", 0],
    "combine_": {
        "salt": [
            "You try to put the jar into the salt ... nothing happens.", 0
        ],
        "monster": [
            "You try to put the monster in the jar. It does not fit.", 0
        ]
    },

    "use_broken" : ["You cut yourself on the sharp glass! Ouch!", 0],
    "attack_broken": ["You try to attack the broken jar. It does not care.", 0],
    "look_broken": ["A broken jar, sharp and dangerous", 0],
    "combine_broken": {
        "salt": [
            "You pour the salt into the broken jar. It does not hold it.", 0
        ],
        "monster": [
            "You try to put the monster in the broken jar. It does not fit.", 0
        ]
    }
},
```

„baseVerbs.json“

Stellt eine Reverse-Lookup-Map dar, welche etwaige Verben, die der User ggf. eingeben könnte, zu einer bestimmten Interaktiont zuweist.

So kann ein User bspw. Sowohl „use“ als auch „utilize“ eintippen und es wird dieselbe Interaktion am Interaktor ausgelöst

```
{
  "use" : "use",
  "utilize" : "use",

  "attack": "attack",
  "break" : "attack",
  "destroy" : "attack",
  "smash" : "attack",
  "fight" : "attack",
  "hurt" : "attack",

  "examine": "look",
  "investigate": "look",
  "watch": "look",
  "inspect" : "look",
  "study" : "look",

  "combine" : "combine",
  "mix" : "combine",
  "blend" : "combine",
  "fuse" : "combine",
  "merge" : "combine"

}
```

Einschränkung:

Jedes mögliche Verb, das auf der Map als „Value“ vorhanden ist, muss auch auf jedem Objekt bei den „baseInteractors“ vorhanden sein, auch wenn dies nicht immer sinnvoll ist.

Grund:

- Der Parser überprüft u.a. anhand der Verben-Liste, ob es sich um einen gültigen Satz handelt

- Handelt es sich um einen gültigen Satz, versucht das Programm, den passenden Wert in der „Interactors“-Datei auszulesen. Existiert dieser nicht, so wirft das Programm eine Fehlermeldung.

„combine.json“

Bestimmte Interaktionen verändern oder kombinieren zwei Interaktoren miteinander.

Passiert dies, so werden die zwei Interaktoren mit einem kombinierten Interaktor von der „combine.json“ ersetzt

Auslesen und Bearbeiten dieser JSON-Files erfolgt über die Klasse „Model.h“:

Diese nutzt einen simplen File-Stream, um die JSON-Dateien auszulesen und dann in „nlohmann::json“-Variablen zu speichern.

Es ist möglich JSON-Dateien zu erstellen, anhand von Keys zu bearbeiten und auch einzelne „Key-Value“-Pairs zu löschen.

Beim Ausführen der Interaktionen an einem Interaktor wird immer der Value des Keys „Status“ angehangen. Somit können wir mehrere Zustände eines Interaktores in einem Objekt speichern und den richtigen Inhalt anhand des Status auslesen.

5.2. Implementierung der Benutzeroberfläche

Es handelt sich um ein klassisches Text-Adventure, somit läuft alles über die Kommandozeilen-Ausgabe. Hierfür nutzen wir den „Input-/Output-Stream“ der Standardbibliothek von C++.

Dieser öffnet standardmäßig die Kommandozeile, welches uns als Benutzeroberfläche genügen soll. Die Print-Logik befindet sich komplett in der Klasse „View.h“ und erhält den Ausgabetext von der Klasse „Model.h“.

Der Ausgabetext kommt als „json“-Datei – ein Datei-Format, welches von der „nlohmann::json“ Library bereitgestellt wird – in „View.h“ an und wird dort in ein String-Array von Größe zwei umgewandelt.

Letztlich wird die erste Position des Arrays durch einen TypeWriter-Effekt an die Konsole ausgegeben.

Den TypeWriter-Effekt wird erzeugt, indem wir jedes Zeichen einzeln printen und denn Thread mithilfe der „Thread“-Standardbibliothek für ein paar Millisekunden pausieren. Zur korrekten Darstellung der Sekunden benutzen wir die „Chrono“-Standardbibliothek.

```
void typeWrite(const std::string& text, int delayMs = 50) {
    for (char c : text) {
        std::cout << c << std::flush;
        std::this_thread::sleep_for(std::chrono::milliseconds(delayMs));
    }
    std::cout << std::endl;
}
```

5.3. Implementierung der Geschäftslogik

Das Programm startet über die main-Funktion, die in einer eigenen .cpp File gespeichert ist.

Von dieser .cpp-File starten wir den GameLoop im Controller, welcher sich in einer Header-File namens „Handler.h“ befindet.

Bei Start des Loopes wird zuerst die JSON-Dateien kopiert, um die laufende Spielsession zu tracken.

Die Handler.h stellt den Controller des „MVC“-Architekturpatterns dar. Hier läuft einmal der Loop, welcher den User immer wieder Informationen gibt und auf User-Input wartet.

Der User-Input wird durch einen stringstream Wort für Wort in einem String-Vector gespeichert. Der Input wird dann auf das Übereinstimmen der simplen Satzstruktur („Verb-Subjekt“ oder „Verb-Subjekt-Präposition-Objekt“) geprüft.

Ist dieser richtig, wird aus der JSON-Datei das jeweilige Objekt ausgelesen und geprinted. Je nachdem, welchen Status das Objekt mitgibt, werden dann noch Aktionen wie das Ändern des Interaktor-Status, das Kombinieren von Interaktoren oder das Beenden des Spieles ausgeführt.

6. Fazit

6.1. Lessons Learned

Ein inkrementeller Prototypen-Ansatz ist für ein Ein-Entwickler-Projekt sehr geeignet. Jedoch stand der Autor vor dem Problem, dass das funktionale Ziel des Endprodukts nicht konkret genug gesetzt wurde.

Dies führte dazu, dass die API-Anbindung ausgelassen werden musste und die Zeitplanung nicht ganz aufging.

6.2. Ausblick

Das System bietet eine stabile Grundlage für komplexere Text-Adventures oder ähnliche Projekte.

Künftige Erweiterungen könnten z.B. beinhalten:

- Speichersystem für Spielstände
- Einbindung einer API, um zufälligen Inhalt für das Abenteuer zu generieren
- Unterstützung komplexere Satzstrukturen und eine stärkere Grammatikprüfung

7. Literaturverzeichnis

C++ Standardbibliothek:

Cppreference: std::cin, std::wcin. o.D. https://en.cppreference.com/w/cpp/io/cin.html, letzter Zugriff: 17.07.2025

Cplusplus: std::vector. o.D. https://cplusplus.com/reference/vector/vector/, letzter Zugriff: 17.07.2025

Cplusplus: std::stringstream::stringstream. o.D. https://cplusplus.com/reference/sstream/stringstream/stringstream/, letzter Zugriff: 17.07.2025

Cplusplus: std::fstream. o.D. https://cplusplus.com/reference/fstream/fstream/, letzter Zugriff: 17.07.2025

GeeksforGeeks: How to iterate through a String word by word in C++. 2022. https://www.geeksforgeeks.org/cpp/how-to-iterate-through-a-string-word-by-word-in-c/, letzter Zugriff: 17.07.2025

Externe Libraries:

libcurl: libcurl – your network transfer library. o.D. https://curl.se/libcurl/, letzter Zugriff: 17.07.2025

nlohmann::json: JSON for Modern C++. 2025. https://json.nlohmann.me/, letzter Zugriff: 17.07.2025

Entwicklungsumgebung:

MSYS2: MSYS2. o.D. https://www.msys2.org/, letzter Zugriff: 17.07.2025

APIs:

Open5e: API DOCS. o.D. https://open5e.com/api-docs, letzter Zugriff: 17.07.2025

Bilder:

Berufsschule 7: BS7-Logo. o.D. https://www.youtube.com/channel/UCH5l88H-wYN9-_Xsl63oo2A, letzter Zugriff: 17.07.2025

Screenshots aus dem Sourcecode.