Placeholder

[Firmenname] | [Firmenadresse]

[Untertitel des Dokuments]

Betz Andreas

[Jahr]

[1. Einleitung 3](#_Toc203644740)

[1.1. Projektumfeld 3](#_Toc203644741)

[1.2. Projektziel 3](#_Toc203644742)

[1.3. Projektbegründung 3](#_Toc203644743)

[1.4. Projektschnittstellen 3](#_Toc203644744)

[1.5. Projektabgrenzung 3](#_Toc203644745)

[2. Projektplanung 4](#_Toc203644746)

[2.1. Projektphasen 4](#_Toc203644747)

[2.2. Abweichungen vom Projektantrag 4](#_Toc203644748)

[2.3. Ressourcenplanung 4](#_Toc203644749)

[2.4. Entwicklungsprozess 4](#_Toc203644750)

[3. Analysephase 5](#_Toc203644751)

[3.1. Ist-Analyse 5](#_Toc203644752)

[3.2. Nutzwertanalyse 5](#_Toc203644753)

[3.3. Anwendungsfälle 5](#_Toc203644754)

[3.4. Qualitätsanforderungen 5](#_Toc203644755)

[3.5. Lastenheft/Fachkonzept 5](#_Toc203644756)

[4. Entwurfsphase 6](#_Toc203644757)

[4.1. Zielplattform 6](#_Toc203644758)

[4.2. Architekturdesign 6](#_Toc203644759)

[4.3. Entwurf der Benutzeroberfläche 6](#_Toc203644760)

[4.4. Datenmodell 6](#_Toc203644761)

[4.5. Maßnahmen zur Qualitätssicherung 7](#_Toc203644762)

[5. Implementierungsphase 7](#_Toc203644763)

[5.1. Implementierung der Datenstrukturen 7](#_Toc203644764)

[5.2. Implementierung der Benutzeroberfläche 8](#_Toc203644765)

[5.3. Implementierung der Geschäftslogik 9](#_Toc203644766)

[6. Dokumentation 9](#_Toc203644767)

[7. Fazit 9](#_Toc203644768)

[7.1. Soll-/Ist-Vergleich 9](#_Toc203644769)

[7.2. Lessons Learned 9](#_Toc203644770)

[7.3. Ausblick 9](#_Toc203644771)

[8. Literaturverzeichnis 9](#_Toc203644772)

[9. Eidesstattliche Erklärung 9](#_Toc203644773)

[10. Anhang 9](#_Toc203644774)

# Einleitung

## Projektumfeld

* Ausbildungsbetrieb: ABUS Security Center
* Auftraggeber: Maik Aicher, i.A. Berufschule 7

## Projektziel

Das Ziel ist ein erweiterbares Text-Adventure, welches thematisch an dem beliebten Tabletop-Role-Playing-Game (TTRPG) Dungeons and Dragons (DnD) orientiert.

Das Abenteuer soll möglichst einfach zu erweitern sein und mithilfe einer API soll zufälliger Inhalt ausgewählt oder Speicheraufwand auf Seite des Projektes verringert werden

## Projektbegründung

Das Projekt stellt ein „Proof of Concept“ – also ein Prototyp, welcher beweist, dass ein Konzept funktioniert – dar, welches benutzt werden kann, um ein verkaufswertiges Text-Adventure weiter zu entwickeln

## Projektschnittstellen

Das Endprodukt stellt eine vollständig kompilierte Software dar und hat somit technisch nur optionale Schnittstellen.

Z.b. ist eine Schnittstelle zu bspw. Open5e geplant, welche benutzt werden könnte, um das Adventure mit offiziellen DnD-Daten zu füllen.

Das Projekt wird anschließend durch den Auftraggeber Maik Aicher abgenommen und präsentiert.

## Projektabgrenzung

Das Endprodukt ist keine Anwendung, um DnD an einem Gerät oder online zu spielen. Es handelt sich lediglich um ein Text-Adventure, welches mit DnD-Inhalten gefüllt wird.

# Projektplanung

## Projektphasen

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

Abnahme- & Einführungsphase sind nicht relevant für dieses Projekt, da es sich um ein selbstständiges Produkt hat, welches abschließend abgegeben wird.

## Abweichungen vom Projektantrag

Implementierung sollte Mittwoch Vormittag abgeschlossen werden, hat aber ca. 2,25 Stunden länger gedauert, als geplant.

Somit schiebt sich die Implementierung auf 16,5 h und die Dokumentation wurde in 4,5h erstellt.

## Ressourcenplanung

* PC, Schulklassenzimmer (PC-Raum)
* Aufsichtskräfte
* Arbeitszeit (persönlich)
* Elektrizität
* Internetzugriff

## Entwicklungsprozess

Es wurde ein Prototypen-System verwendet, bei dem mehrere Iterationen des finalen Produktes Mockup-artig erstellt und getestet wurden, um weitere Features festzumachen und etwaige Probleme auszumerzen

# Analysephase

## Ist-Analyse

Momentan gibt es noch keinen implementierten Text-Parser, welcher für Text-Adventures verwendet werden kann.

Bisher wurde noch kein Text-Adventure oder ähnliches implementiert hier in der Schule.

## Nutzwertanalyse

Das Projekt kann für zukünftige Produkte hergenommen werden, da der erstellte Text- und JSON-Parser wie eine „Library“ weiterverwendet werden kann.

Somit spart man sich das erneute Entwickeln von ähnlichen Systemen in der Zukunft.

## Anwendungsfälle

Primärziel ist es eine Grundlage für die weitere Entwicklung von Text-Adventures zu bieten.

Der Text-Parser kann aber auch anderswertig weiterbenutzt werden, wenn ein anderes Projekt einen Text-Parser benötigt.

## Qualitätsanforderungen

Performance:

* Das Text-Adventure und der Parser muss selbst auf veralteten Geräten flüssig und ohne Ladezeiten laufen.

Usability:

* Auch Nutzer ohne technisches Vorwissen müssen in der Lage sein, das Textadventure zu spielen
* Der Text-Parser muss für Programmierer mit Vorkenntnissen lesbar und nachvollziehbar sein

Effizienz:

* Das Überprüfen der Eingabe und das Lesen und Schreiben auf den Textdateien, muss mit möglichst wenigen Operationen geschehen

# Entwurfsphase

## 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 Anwendungen wird mit allen notwendigen Abhängigkeiten kompiliert – Dies erleichtert die Verteilung.
* Windows 11 ist zwar nicht die optimale Entwicklungsumgebung für C++, jedoch entwickeln all unsere Entwickler auf Windows und es würde die Erweiterbarkeit behindern, das Projekt auf Linux anzusiedeln

Zusätzliche Entscheidungen:

* Wir nutzen MSYS2 und den g++-Compiler unter der „urct64“ Umgebung, um die Entwicklung mit C++ unter Windows möglichst einfach und klassisch durchzuführen
* Zusätzlich wird die „nlohmann::json“-Library für unser JSON-Dateninteraktionen und die „libcurl“-C-Library für den API-Call verwendet

## Architekturdesign

Das Projekt wurde mit dem MVC-Architekturpattern erstellt.

Dieses trennt Datenmanipulation, UI-Darstellung und dessen Verbindung logisch getrennt und ermöglicht eine besonders hohe Skalierbarkeit.

So ist der View – also was der Endnutzer sieht – und die Control – also die Schnittstelle zwischen View und Datenmanipulation – statisch und bietet einen Rahmen, welches dynamisch auf eine verändertes Model reagieren kann.

Zusätzlich dient das MVC-Pattern der Übersichtlichkeit.

## Entwurf der Benutzeroberfläche

Es handelt sich um eine simple Command-Line-Applikation. Öffnet man das Adventure, wird lediglich eine Bash geöffnet, auf der das Programm läuft.

Das eigentliche Adventure wird rein durch Text Ein- und Ausgabe gesteuert.

Durch Leerzeilen und einem Typewriter-Effekt, bei dem zu lesende Texte allmählich erscheinen, wird eine möglichst angenehme User-Erfahrung angesteuert.

## Datenmodell

Es wird keine Datenbank verwendet. Die Applikation nutzt JSON-Dateien, die beim Öffnen des Spieles für die gestartete Sitzung kopiert 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.  
  
Diese Dateien werden dann im Laufe des Spieles ausgelesen und bearbeitet, um den Verlauf des Abenteuers darzustellen.

Da immer von den Grunddateien kopiert wird, gibt es momentan auch keine Möglichkeit den Spielstand zu speichern.

Dies könnte aber ohne größere Probleme erweitert werden. Dann würde Spielstand in den erstellten JSON-Files gespeichert werden und muss lediglich bei Spielbeginn ausgewählt werden.

## 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 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 geprinted.

## Maßnahmen zur Qualitätssicherung

Das Produkt wurde durch regelmäßige Prototypen getestet und weiterentwickelt.

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

# Implementierungsphase

## Implementierung der Datenstrukturen

Die Daten wurden auf drei getrennten JSON-Dateien aufgeteilt.

* „baseInteractors.json“
  + Stellt alle Objekte innerhalb des Adventures dar. Gleichzeitig sind dort auch alle Interaktionsmöglichkeiten 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 bekomment
    - Einen Integer, welche die Interaktion, die mit dem Interactor geschehen soll, beschrieben ist
* „baseVerbs.json“
  + Stellt eine Reverse-Lookup-Map dar, welche etwaige Verben, die der User ggf. eingeben könnte, zu einer bestimmten Interaktionsmöglichkeit mapt.
  + So kann ein User bspw. Sowohl „use“ als auch „utilize“ eintippen und es wird dieselbe Interaktion am Objekt ausgelöst
  + **Einschränkung:**
    - Jedes mögliche Verb, das auf der Map als „Value“ vorhanden ist, muss auch auf jedem Objekt bei den „Interactors“ 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 wird null zurückgegeben und das Programm wirft eine exception.
* „combine.json“
  + Bestimmte Interaktionen verändern oder kombinieren zwei Interactoren 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.

## 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++.

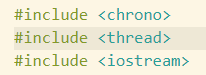
Dieser öffnet standardmäßig „cmd“ unter Windows, welches uns als Benutzeroberfläche genügen soll.

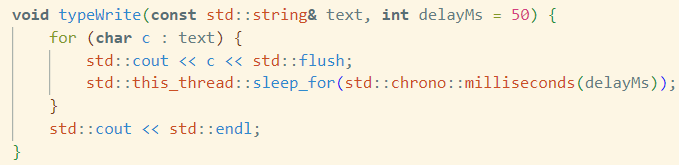
Die Print-Logik befindet sich komplett in der Klasse „View.h“ und erhält den Ausgabetext von der Klasse „Model.h“.

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 geprinted.

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





## 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.

# Fazit

## Soll-/Ist-Vergleich

gay

## Lessons Learned

## Ausblick

# Literaturverzeichnis

# Eidesstattliche Erklärung

# Anhang