Dokumentation Text Adventure

Dany Kanaan BSIT22a

Herr Esper Lernfeld 5

Ich habe mich in diesem Projekt mit einem Text Adventure beschäftigt, welches die Möglichkeit bieten soll, den aktuellen Spielstand zu speichern und bei späterem Neustart des Programmes diesen auch wieder lade zu können.

Angefangen habe ich mit einem Grundgerüst des Programmes und mich vorläufig für 3 Klassen entschieden. Player, Enemy und Room.

Klasse Player

Die Klasse Player speichert Werte des Spielers wie Name, Klasse, Leben und Stärke.

Es gibt 3 Unterklassen, für die sich der Spieler entscheiden kann. Jäger, Krieger und Schurke. Alle haben verschiedene Stärken, Leben und Waffen.

Klasse Room

Die Klasse Room erstellt einen Raum, in welchem wir festlegen, ob der Raum Gegner enthalten soll. Sollte das der Fall sein wird zufällig eine Zahl von 1-3 generiert, welche die Anzahl der Gegner bestimmt. Danach wird wieder durch eine Zufallszahl entschieden welche Gegnerklasse generiert wird. Die erstellten Gegner schreibe ich in einen Vektor innerhalb der Raumklasse, da die Gegner an die Räume gebunden sein sollen.

Die Räume haben außerdem einen Vektor in dem ein Variablenpaar von ENUM und Integer kombiniert und gespeichert werden. Das ENUM gibt die Richtung an, in welcher sich der Raum befindet. Der Integer gibt die Raumnummer an. Diese Raumnummer spiegelt den Raum innerhalb des globalen Vektors<Room> Map wieder. Diese Karte wird in der aktuellen Version statisch erstellt und auch die Zuweisung, welcher Raum Gegner enthält, ist statisch vergeben.

Klasse Enemy

Die Klasse Enemy wird beim Erstellen von Räumen benutzt, um Räume mit Gegnern zu füllen. Die erstellten Gegner werden innerhalb des Raumes in einem Vektor gespeichert.

Es gibt 3 Gegnertypen: Goblin, Skeleton Archer und Oger. Alle haben verschiedenen Lebens- und Stärkewerte.

Befehle

Die meisten Befehle, welche der Benutzer anwenden kann, sind in globalen Variablen gespeichert. Falls z.B. abgefragt werden soll, ob der Benutzer sich bei Start der Anwendung für „Neues Spiel“ oder „Laden“ entscheidet, werden die zwei Variablen durchsucht. Die loadCommands Variable enthält die Befehle „Laden“ und „Lade“, welche der Benutzer in die Konsole schreiben kann, um das Spiel aus der Datei auszulesen. Die Variable startCommands enthält die Befehle „Start“ und „Neu“, welche der Benutzer zum Starten eines neuen Spieles verwenden kann.

Die Eingabe des Benutzers wird immer auf kleine Zeichen formatiert, um sicher zu stellen, dass egal wie dieser die Worte schreibt, das Programm diese verwerten kann.

Programmablauf

Nach dem Start des Spieles kann sich der Benutzer dafür entscheiden, ob er ein neues Spiel starten möchte oder ein vorhandenes Spiel speichern möchte.

Start eines neuen Spieles

- Beim Start eines neuen Spieles wird die Karte erstellt. Danach wird gefragt, wie der Spieler heißen möchte und welche Klasse er gerne hätte.
- Anschließend soll der Benutzer verschiedene Befehle eingeben, welche ihn weiterspielen lassen.
- Das ist jedoch nicht fertig implementiert, da ich mich erst auf das Speichern fokussiert habe und mir gegen Ende die Zeit ausgegangen ist alles umzusetzen. Dazu mehr im Fazit.
- Wählt der Benutzer also speichern wird der aktuelle Spieler und die Karte in zwei verschiedenen Dateien gespeichert.

Speichern des Spieles

- Sollte der Benutzer während seines Abenteuers speichern wollen, so tippt er dies in die Befehlszeile. Das Spiel erstellt, falls noch nicht vorhanden, zwei neue Dateien und speichert in diese die Werte des Spielers bzw. der Karte.
- Die Karte enthält die Werte der Räume und der Gegner, welche sich in diesen Räumen befinden.

Laden des Spieles

- Wählt der Benutzer laden, dann wir der Spielstand und die Karte aus den, sofern vorhanden, zwei Dateien geladen.
- Dazu werden die Lines in der Datei nach und nach ausgelesen und in den Werten des Spielers / der Karte aufgefüllt.
- Sollte die Datei nicht vorhanden sein wird dem Benutzer das auch wiedergegeben.

Fazit & Verbesserungsvorhaben des Projektes

Im Laufe des Projektes habe ich mir neben dem Speichern des Spieles noch mehr Sachen aneignen müssen als vorerst gedacht. Dazu zählt das Auslesen von Befehlen. Aktuell habe ich mehrere Strings erstellt, welche die Befehle beinhalten. Raussuchen ob es den Befehl in der Liste tue ich mit Befehlliste.find(Befehl)! = string::npos. Das Problem jedoch: Auch wenn nur ein Buchstabe eingegeben wurde, welcher sich in dem Befehlsliste String befindet, gilt das Wort als gefunden.

Um das zu lösen wollte dich die Strings in Vektoren<string> umwandeln, um diese mit den einzelnen Worten zu füllen. Das hat mir kurz vor Beendigung des Projektes jedoch noch mehr Fehler in den Raum geworfen als ich beheben / umsetzen konnte. Ich habe die Vektoren mit einer For-Schleife durchlaufen und gesucht, ob die Worte übereinstimmen. Jedoch hat das nicht ganz funktioniert und Befehle wurde nicht mehr richtig erkannt bzw. wurde zu einem späteren Zeitpunkt noch ein Fehler verursacht, obwohl ich schon weitere Eingaben tätigen konnte. Den Code habe ich nicht verworfen, jedoch nicht mehr in meine Abgabe miteinbezogen.

Auch funktioniert neben dem Speichern, Laden und Abfragen nicht viel mehr. Ich hätte gerne noch die Räume um die Logik erweitert, welcher Raum gerade der Aktive ist, das wir Räume passieren können und gegen Gegner kämpfen können. Jedoch hat das alles mehr Zeit in Anspruch genommen als für dieses Projekt vorgesehen war. Dementsprechend können wir das Spiel auch nicht beenden. Vorgesehen war, dass wenn der Boss am Ende der Map besiegt oder der Spieler alle seine Lebenspunkte verliert, das Spiel vorbei sein sollte.

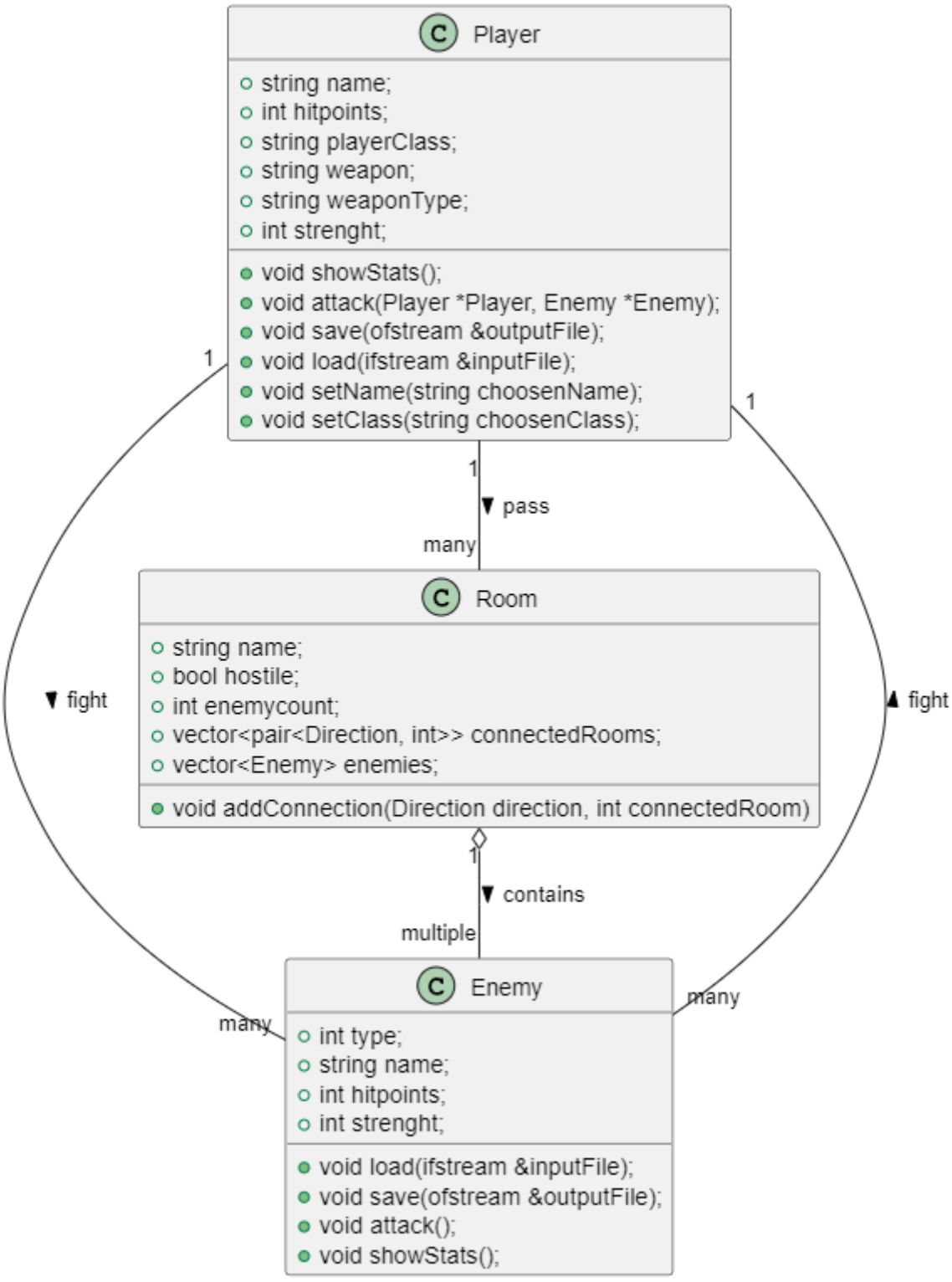
Ich musste leider merken das in C++ manche Sachen mehr Zeit in Anspruch nehmen als angenommen. Vor allem bei Dingen, von denen ich noch nicht ganz eine Idee hatte, wie ich dieses umsetzen möchte. Dazu kommt das ich oftmals mehr Dinge einbauen wollte, wie ich in der Zeit geschafft hätte. Viele Dinge habe ich nochmals verworfen oder nicht mit ins Spiel aufgenommen. Auch das Debuggen des Projektes hat mehr Zeit gekostet als ich gehofft habe.

Trotzdem bin ich zufrieden, dass die Speicher- und Ladelogik so funktioniert wie ich mir das erhofft habe und denke, dass ich zukünftig nebenbei an dem Projekt weiterarbeite, um es irgendwann fertig zu stellen.

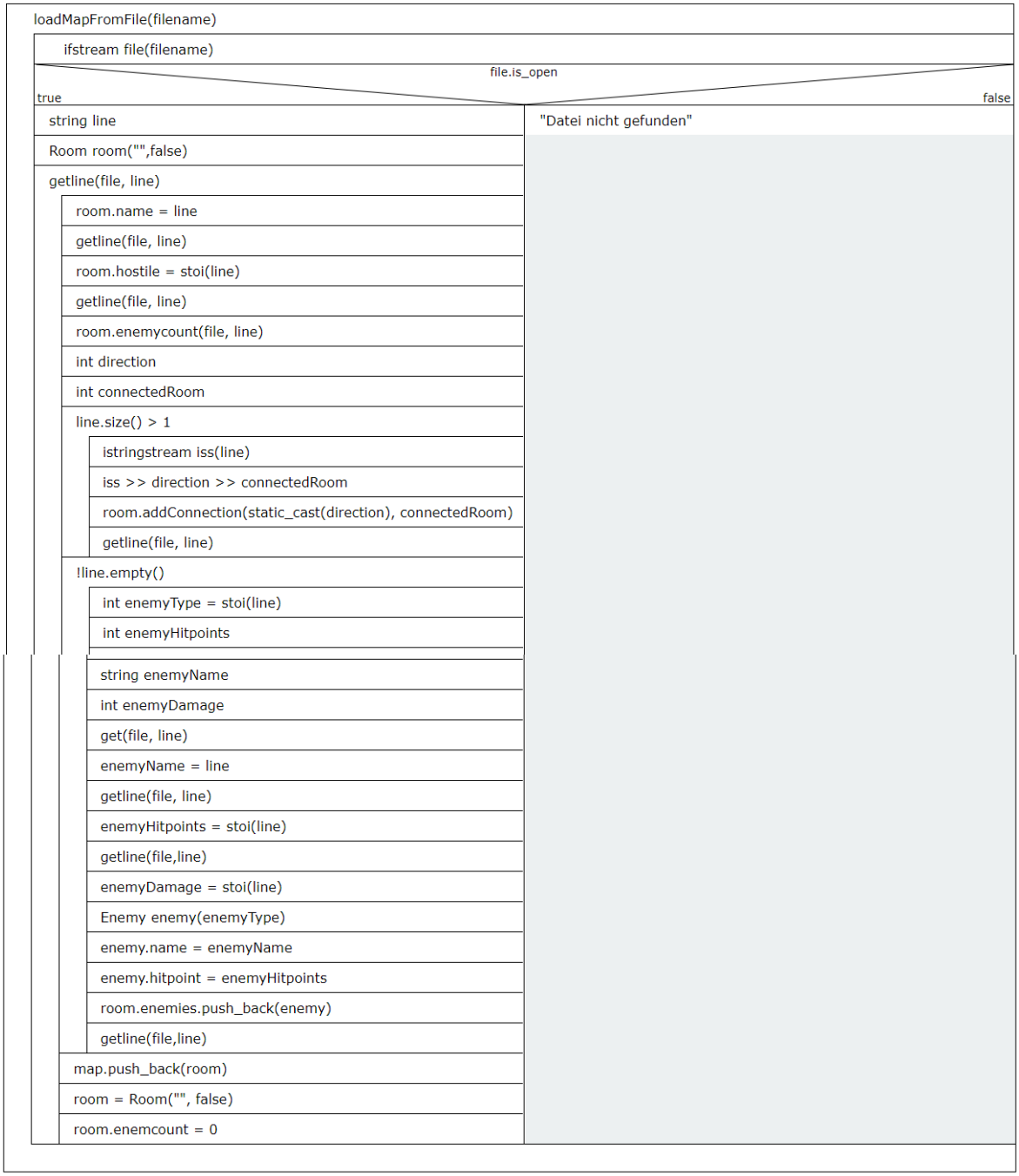
Das Projekt befindet sich in dem Git-Hub Repository:

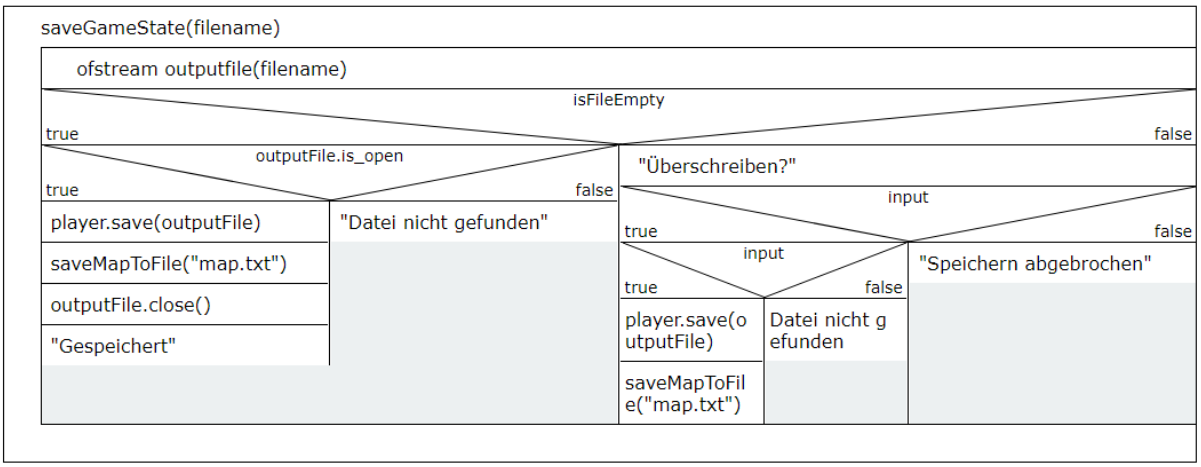
https://github.com/promisedland-hub/TextAdventure

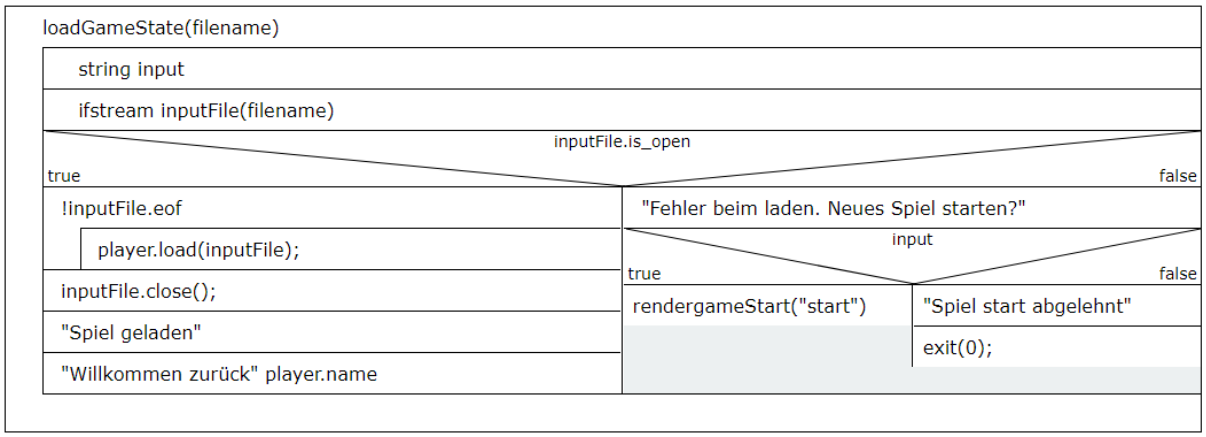
UML-Diagramm:

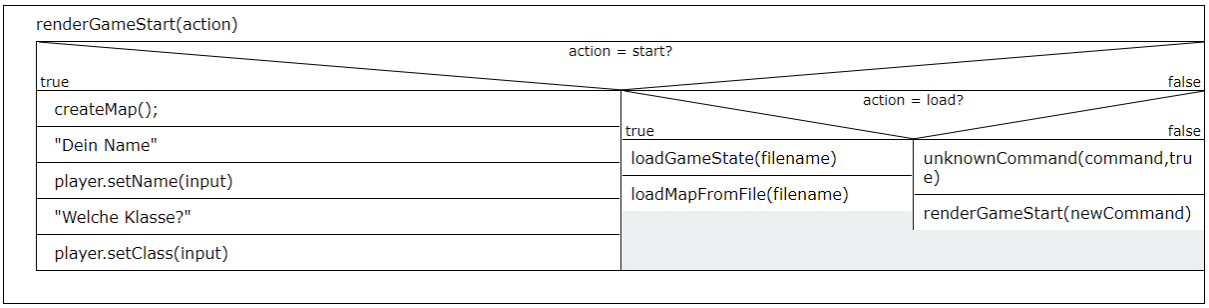


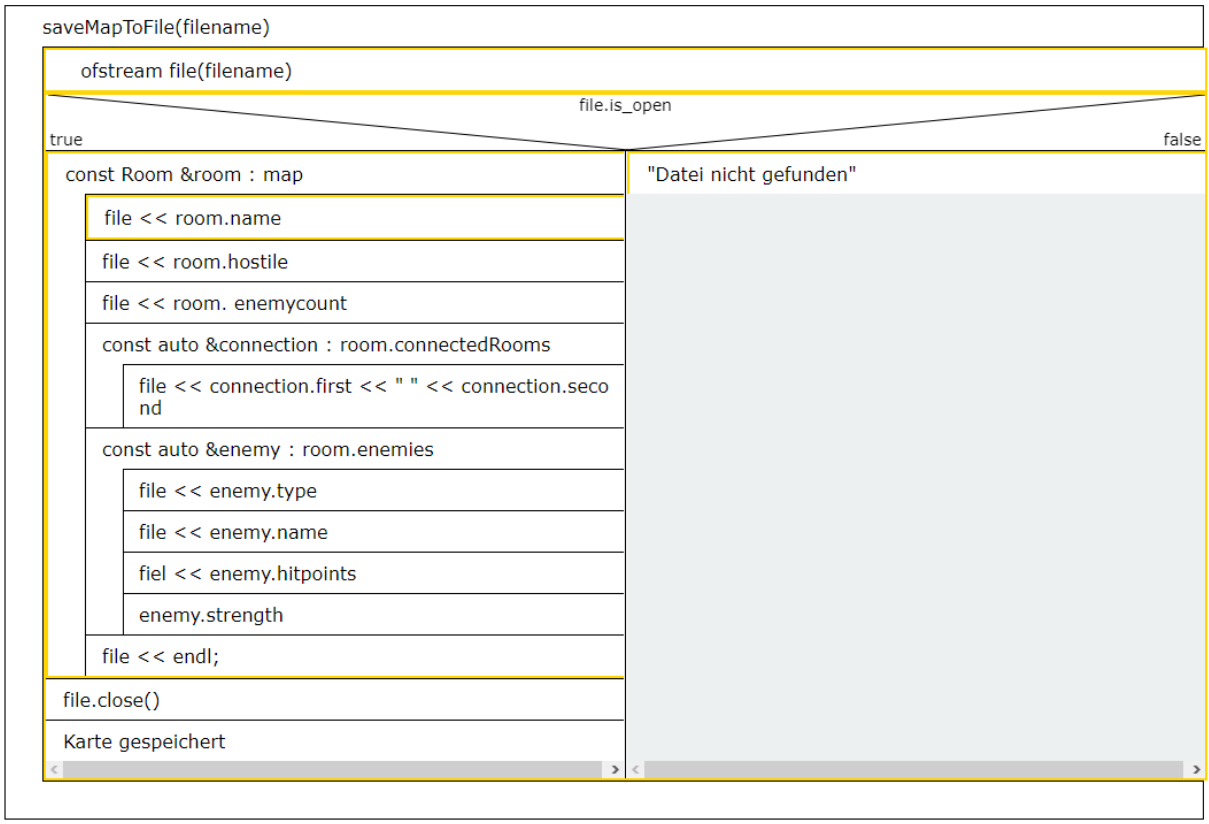
Struktogramme




saveGameState(filename)
ofstream outputfile(filename)
isFileEmpty
true
false
outputFile.is_open
true
false
player.save(outputFile)
saveMapToFile("map.txt")
outputFile.close()
"Gespeichert"
"Datei nicht gefunden"
"Überschreiben?"
input
true
false
input
true
false
player.save(outputFile)
saveMapToFile("map.txt")
Datei nicht gefunden
"Speichern abgebrochen"


loadGameState(filename)
string input
ifstream inputFile(filename)
inputFile.is_open
true
false
!inputFile.eof
player.load(inputFile);
inputFile.close();
"Spiel geladen"
"Willkommen zurück" player.name
"Fehler beim laden. Neues Spiel starten?"
input
true
false
rendergameStart("start")
"Spiel start abgelehnt"
exit(0);


renderGameStart(action)
action = start?
true
false
createMap();
"Dein Name"
player.setName(input)
"Welche Klasse?"
player.setClass(input)
action = load?
true
false
loadGameState(filename)
loadMapFromFile(filename)
unknownCommand(command,true)
renderGameStart(newCommand)


saveMapToFile(filename)
ofstream file(filename)
file.is_open
true
false
const Room &room : map
file << room.name
file << room.hostile
file << room. enemycount
const auto &connection : room.connectedRooms
file << connection.first << " " << connection.second
const auto &enemy : room.enemies
file << enemy.type
file << enemy.name
fiel << enemy.hitpoints
enemy.strength
file << endl;
file.close()
Karte gespeichert
"Datei nicht gefunden"