November 2024

Praktikum zur Lehrveranstaltung Software Engineering (Grundlagen)
Versuch 3, Labyrinth

Organisation des Praktikumsversuchs: M. Herhold

Inhaltsverzeichnis

1 Ziele dieses Praktikumsversuchs

- Festigung des Vorlesungsstoffes zur Python-Programmierung.
  - durch Benutzung von Tupeln Arrays und Programmschleifen
  - durch die Nutzung und den Entwurf von Funktionen
  - durch Aufteilen von Funktionalität eines Programms auf mehrere Python-Module
- Eigenständiges Entwerfen und Implementieren eines Programms in der Programmiersprache Python.
- Schulung der Teamfähigkeit.

2 Wichtiger Hinweis

Der Zeitaufwand der Aufgabenstellung für Studenten, welche keine Programmiererfahrung haben kann umfangreich werden, da viele Programmierkonzepte und Herangehensweisen erst erlernt werden müssen. Fangen Sie daher rechtzeitig an die Aufgabe zu bearbeiten und verteilen Sie die Teilaufgaben an alle Gruppenmitgliedende[1]. Die Teilaufgaben sind auf Teamarbeit ausgelegt und mit Informationen zu Zeitaufwand und Schwierigkeitsgrad versehen (Rückmeldungen zur Verfeinerung dieser Angaben sind erwünscht).

[1] Alternativ Mitglieder*innen, oder ist es hier nicht angebracht? Video mit Erklärungen. Um das Verständnis der Aufgabe für alle zu ermöglichen, sind in diesem Dokument Personengruppen jeglichen Geschlechts gemeint, ohne dies gesondert zu kennzeichnen.

3 Versuchsaufgabe

Aufgabe des Praktikumsversuches ist es, eine 2D-Karte eines Labyrinths aus einer Datei zu laden und anschließend einen Avatar (Spielfigur) durch das Labyrinth vom Start zum Zielpunkt zu navigieren.

Zwingend zu verwendender Python-Quellcode und die Labyrinth-Daten finden Sie zum Download auf der Webseite zum Versuch.

Es sind die folgenden Teilaufgaben zu lösen.

Wichtig: jede einzelne Teilaufgabe ist in einem separaten Python-Modul (separate Quellcode-Datei) mit jeweils eigener `main`-Funktion und Test-Funktionalität zu implementieren (als Beispiel siehe Datei `gfx_stack.py`). Dies ermöglicht es jedem Gruppenmitglied seinen eigenen Quellcode zu seiner Teilaufgabe zu entwickeln und zu testen. Ein Test-Framework – wie z.B. pytest – zu verwenden, ist für die Versuchsaufgabe nicht notwendig.

3.1 Datenstrukturen festlegen

Arbeitsgrundlage für alle Teilaufgaben.

Öffnen Sie die mitgelieferte Datei `Labyrinth-1.txt` mit einem Texteditor und finden Sie eine geeignete Datenstruktur um die Daten der 2D-Karte dem Programm zur weiteren Verarbeitung zur Verfügung zu stellen. (Vorschlag: 2-Dimensionale Liste)

Legen Sie fest, welche Werte in der Datenstruktur, welche Bedeutung für die Weiterverarbeitung haben sollen. Siehe dazu auch Teilaufgabe 3.3.
(Beispiel: eine Wand im Labyrinth wird durch die Integerzahl `11` dargestellt, weitere Elemente: Gang, besuchter Gang, Avatar, Start, Ziel). Die Festlegung ist wichtig, damit die Zeichenroutine korrekte Farben zeichnet und die Wegfindung Markierungen korrekt erkennt / in die Karte eintragen kann.

Schreiben Sie eine Funktion, welche das Labyrinth auf der Python-REPL[2] (Python-Shell) ausgibt. Diese Funktion kann Ihnen in allen Teilaufgaben zum Debuggen hilfreich sein.

Erzeugen Sie für jede Teilaufgabe geeignete minimale Beispieldaten, welche die Arbeitsgrundlage für die Programmierung der Teilaufgabe als separates Modul darstellen.

- Schwierigkeitsgrad: *einfach*
- Zeitaufwand: *mittel*
- Anzahl Bearbeiter: alle Gruppenmitglieder

3.2 Datei einlesen

Einzulesen ist eine Textdatei, welche ein Labyrinth aus der Vogelperspektive als 2D-Karte abbildet. Dabei stellt jedes einzelne Zeichen in der Datei ein Element (z.B. Wand, Anfangsfeld, Ausgang) im Labyrinth dar.

Die Karte des Labyrinths hat immer eine rechteckige Form, das bedeutet, alle Zeilen in der Datei sind gleich lang. Die Dimension (Länge x Breite) des Labyrinths bestimmt sich aus den Daten in der Textdatei.

Schreiben Sie eine Funktion, welche die Daten aus der Datei in die Datenstruktur (aus Teilaufgabe 3.1) lädt. Dazu müssen Sie die Daten in der Datei analysieren und Ihre Datenstruktur in der korrekten Größe anlegen.

Besondere Aufmerksamkeit sollten Sie dem Zeilenende-Zeichen widmen.

- Schwierigkeitsgrad: *einfach*
- Zeitaufwand: *gering bis mittel*
- Anzahl Bearbeiter: eine Person

[2] Read-Eval-Print, Loop

3.3 Kartendaten normalisieren

Die Daten aus der vorherigen Teilaufgabe sind nun in ein normalisiertes Format zu konvertieren, damit die folgenden Programmteile die Daten unabhängig vom Eingabedatenformat weiterverarbeiten können.

Es gibt 2 unterschiedliche Labyrinth-Daten-Formate (siehe Dateien `Labyrinth-*.txt` im `Quellcodeframework-Labyrinth.zip`). Beide Formate sollen von Ihrem Programm verarbeitet werden können. Um den Algorithmus für die Wegfindung wesentlich zu vereinfachen, ist daher ein Normalisieren der Daten sinnvoll.

Die folgende Tabelle gibt die Bedeutungen der Zeichen wieder:

	Zeichen	Bedeutung	Hinweis
Typ 1			
	W	Wand	der Avatar kann sich nicht auf, oder durch dieses Feld bewegen
	.	Weg	auf dieses Feld kann sich der Avatar bewegen; ASCII-Code: 0x2e
Typ 2			
	X	Wand	
	W	Weg	
Gemeinsamkeiten (von Typ 1 und 2)			
	S	Startpunkt	der Avatar befindet sich zu Beginn auf diesem Feld
	Z	Zielpunkt	der Avatar soll sich zu diesem Punkt durch das Labyrinth bewegen
	Ziffern	Bonuspunkte	noch unbenutzt, als begehbaren Weg betrachten

- Schwierigkeitsgrad: *mittel*
- Zeitaufwand: *mittel*
- Anzahl Bearbeiter: eine Person

3.4 Spielfeld (aka Labyrinth) auf den Bildschirm zeichnen

Schreiben Sie eine Funktion, welche die Karte mit Hilfe der Funktion `gfx_stack.set_pixel()` auf den Bildschirm zeichnet. Jeder Eintrag in der Datenstruktur stellt dabei einen einzelnen Pixel dar. Benutzen Sie unterschiedliche Farben (siehe Datei `gfx_stack.py`) für die unterschiedlichen Elemente der Karte.

Alle Elemente der Karte – der von der Spielfigur begangene Weg, der Start- und Ziel-Punkt, die Wände, etc. – sollen farblich klar unterscheidbar sein. Nutzen Sie die Farben und Farbnamen/Farbindex der Farbpalette `color_dict` aus der Datei `gfx_stack.py`.

- Schwierigkeitsgrad: *einfach*
- Zeitaufwand: *gering*
- Anzahl Bearbeiter: eine Person

3.5 Start und End-Punkt im Labyrinth finden

Schreiben Sie eine Funktion, welche die Start- und die Ziel-Koordinaten findet. Diese Informationen sind wichtig, für die Wegfindung.

- Schwierigkeitsgrad: *sehr einfach*
- Zeitaufwand: *gering*
- Anzahl Bearbeiter: eine Person

3.6 Einfache Wegfindung

Nachdem das Labyrinth geladen ist und die Start- und Zielkoordinaten bekannt sind, soll ein einfacher Algorithmus einen Avatar (Spielfigur) helfen den Weg durch das Labyrinth zum Ziel finden.

Dieser Avatar hat dabei eine Blickrichtung (welche auf dem Startfeld noch unbestimmt ist), je nachdem aus welcher Richtung er kam.

- Beispiele:
  - Der Avatar ging von einem östlichen Feld nach Westen, also schaut er aktuell nach Westen.
  - Der Avatar ging von Süden nach Norden, die neue Blickrichtung ist Norden.

Der Avatar kann sich dabei immer nur auf ein Nachbar-Spielfeld, welches horizontal oder vertikal (niemals schräg) liegt, bewegen. Ähnlich dem Bewegungsmuster eines Turms mit der Reichweite eines Königs beim Schach.

Mit dem Avatar außen um das Labyrinth zu laufen, um zum Zielfeld zu gelangen, ist keine gültige Lösung. (Es befindet sich sozusagen eine zusätzliche Mauer um das Labyrinth.)

Der Algorithmus zur Wegfindung soll eine einfache Strategie verfolgen, welche darin besteht das Spielfeld rechts vom Avatar, mit dem Avatar zu betreten. Ist dies nicht möglich, weil dieses Spielfeld durch eine Wand blockiert ist, dann versucht der Algorithmus gerade aus weiterzugehen. Falls dies ebenfalls nicht möglich ist, soll das Spielfeld links vom Avatar betreten werden. Falls die vorherigen 3 Versuche fehlschlagen, muss der Avatar zurückgehen. Der Avatar verändert bei jeder Bewegung (außer beim geradeaus gehen) die Blickrichtung.

Der begangene Weg soll auf der Karte eingetragen werden, damit er im grafischen Fenster angezeigt wird. Wie Sie die Karte aktualisieren (nach jedem Schritt, oder nach vollständiger Berechnung) bleibt Ihnen überlassen.

Hinweis: Sie erhalten nur Labyrinthe, welche durch den Algorithmus lösbar sind (Labyrinthe ohne Inseln).

- Schwierigkeitsgrad: *hoch*
- Zeitaufwand: *hoch*
- Anzahl Bearbeiter: eine bis zwei Personen (vorzugsweise mit Spaß am Knobeln)

3.7 Hauptprogramm

Schreiben Sie eine Funktion, welche bei Programmstart aufgerufen wird und die Funktionen der Teilaufgaben in der korrekten Reihenfolge (gegebenenfalls wiederholt) aufruft.

Folgender Programmablauf ist denkbar:

1. Datei einlesen
2. eingelesene Daten normalisieren
3. aktuelle Karte auf den Bildschirm zeichnen
4. eine Schleife starten
   a) nächsten Schritt berechnen
   b) aktuelle Karte auf den Bildschirm zeichnen
   c) wiederholt warten und das Bild ohne sonstige Berechnungen zeichnen, um die Wegfindung für Zuschauer zu verlangsamen und damit beobachtbar zu machen.
5. Programmende, wenn die Variable `gfx_stack.stop_prog` den Wert True angenommen hat

Beim Bearbeiten dieser Teilaufgabe werden alle Teilaufgaben zum ersten Mal zu einem gemeinsamen Programm zusammengebaut. Je nach Sorgfalt der Bearbeitung der vorherigen Teilaufgaben läuft dieser Teil nahezu reibungslos oder es erwartet Sie eine zeitraubende Fehlersuche. Planen Sie ausreichend Zeit ein!

- Schwierigkeitsgrad: *gering bis komplex*
- Zeitaufwand: *wenig bis viel*
- Anzahl Bearbeiter: alle Gruppenmitglieder

3.8 Programmdokumentation (Nachbereitung zum Versuchstermin)

Zwei Wochen nach dem Versuchstermin ist eine Programmdokumentation als PDF per E-Mail beim Betreuer einzureichen. Die Programmdokumentation ist Teil der Bewertung der Gruppe und sollte als praktische Übung zum Verfassen von wissenschaftlichen Texten[3] angesehen werden. Umfang, vier bis acht A4-Seiten mit zusätzlichem Deckblatt.

3.8.1 fakultativer Inhalt des Dokuments

- Einleitung oder Einführung oder Aufgabenstellung
- Inhaltsverzeichnis
- Nennung/Beschreibung/Begründung verwendeter Konzepte/Besonderheiten zur Aufgabenlösung
- Programmablauf in sinnvollem Detailgrad und geeigneter Präsentationsweise.
- Quellcode nur als kurze Ausschnitte, entsprechend hervorgehoben und mit Erläuterungen
- Dokumentation von Quellcode-Abschnitten: was und warum es so arbeitet, aber nicht wie es eine Aufgabe löst, dies dokumentiert bereits der Quelltext.
- Analyse der aufgetreten Probleme beim Bearbeiten der Aufgabe
- Verbesserungsvorschläge zur eigenen Arbeit
- Quellenangaben für fremden Quellcode oder Text (sofern notwendig)

3.8.2 Was nicht in die Dokumentation gehört:

- Ich-Form
- kompletter Quellcode des Programms
- Im Detail erklären wie einzelne Funktionen Ihres Programms arbeiten. Gehen Sie davon aus, dass der Quelltext Ihres Programms bereits digital vorliegt und vom Leser verstanden wird.

[3] Dieses Dokument wurde mit GNU Emacs und org-mode verfasst.

3.9 Bonusaufgaben

Diese folgenden Aufgaben sind vollständig optional, zum Spaß[4], und können bei überzeugender Erklärung in der Präsentation zu Bonuspunkten bei der Bewertung führen.

3.9.1 Kommandozeilenparameter

Nutzen Sie Kommandozeilenparameter um die einzulesende Labyrinth-Daten-Datei bei Programmstart anzugeben. Siehe dazu die Quellcodedatei `kommandozeilen_argumente.py`.

- Schwierigkeitsgrad: *einfach*
- Zeitaufwand: *gering bis mittel*

3.9.2 rekursive/iterative Funktionen

Implementieren Sie den Algorithmus zur Wegfindung als iterative / rekursive Funktion. So, dass Sie anschließend beide Varianten (die rekursive und die iterative) in Ihrem Programm nutzen können.

Einige Probleme sind einfacher per Rekursion, andere einfacher als iterativer Vorgang zu formulieren, daher ist dies eine gute Übung.

- Schwierigkeitsgrad: *abhängig von Vorkenntnissen*
- Zeitaufwand: *abhängig von Vorkenntnissen*

3.9.3 anderer Algorithmus zur Wegfindung

Implementieren Sie einen anderen Algorithmus um den Weg durch das Labyrinth zu finden (z.B. den A-Star Algorithmus).

- Schwierigkeitsgrad: *hoch*
- Zeitaufwand: *viel*

3.9.4 Zeitmessung

Messen Sie die Zeiten ihrer Algorithmen um einen Vergleich dieser zu ermöglichen. Finden Sie Messmethoden um sinnvoll vergleichbare Ergebnisse zu erhalten.

- Schwierigkeitsgrad: *mittel*
- Zeitaufwand: *mittel, aber zusätzlich andere Bonusaufgabe benötigt*

[4] Advent of Code – Adventskalender mit anspruchsvollen Programmierrätseln

4 Weitere Hinweise zum Praktikumsversuch

4.1 Benennung von Funktionen und Variablen

Benennen Sie die Funktionen und Variablen in Ihrem Programm selbständig und sinnvoll.

- Funktionsnamen sollten ausdrücken, was eine Funktion macht (nicht wie sie es macht) weitere Informationen und Beispiele.
- Variablennamen sollten ausdrücken, wofür die Variable benutzt wird weitere Informationen und Beispiele.

4.2 Programmierumgebung während des Praktikumstermins

- Hardware: Es steht aus Platzgründen pro Gruppe nur ein einzelner PC zur Verfügung. Auf diesem ist der Quellcode und das fertige Programm zu präsentieren.
- Software (Information, damit Sie vorab die Funktion Ihres Programms testen können):
  - Betriebssystem: GNU Linux
  - Python Version: 3.11
  - `pygame` Version: 2.1
  - Editor: PyCharm Community Edition – Visual Studio Code steht nicht zur Verfügung, Link zu Gründen weshalb
- Datennetz: vorhanden aber leicht eingeschränkt

4.3 Vorbereitung auf den Versuchstermin

Implementieren Sie die Praktikumsaufgabe und Teilaufgaben selbständig und erscheinen Sie als Gruppe mit dem **lauffähigen** Programm zum Versuchstermin. Während des kurzen Versuchstermins reicht die Zeit erfahrungsgemäß nicht aus, ein *fast* fertiges Programm lauffähig zu bekommen (oft sind eine aufwändige Fehlersuche und fehlendes Wissen das Problem). Stellen Sie bei Problemen oder Unklarheiten Fragen *rechtzeitig* per E-Mail an den Betreuer des Versuchs. Für erfahrene Programmierer ist die Aufgabe schnell gelöst, aber unerfahrene Programmierer verlieren erfahrungsgemäß viel Zeit mit Recherche und Fehlersuche. Fangen Sie rechtzeitig mit der Bearbeitung an und verlegen Sie den Versuchstermin an einen entfernteren Zeitpunkt. Das selbständige Lösen von Teilaufgaben ist die beste Vorbereitung auf den schriftlichen Eingangstest.

4.4 Details zum Versuchstermin

- Es erfolgt ein schriftlicher Eingangstest. Das Ergebnis fließt in die Versuchsbewertung ein.
- Während des praktischen Teils erhalten Sie weitere Labyrinth-Karten, welche Ihr Programm korrekt einlesen, darstellen und mit dem Algorithmus lösen können soll. Die neuen Labyrinthe können (viel) größer oder kleiner sein, als das `Labyrinth-1.txt`.
- Stellen Sie Ihr Programm dem Betreuer vor, beantworten inhaltliche Fragen zu Ihrem Quelltext und nehmen Sie vor den Augen des Betreuers Änderungen am Quelltext vor (Prüfung bezüglich KI generierten Quellcodes). Dies ist Pflichtteil zur Bewertung und erfolgt individuell für jeden Studenten.
- Nach erfolgreicher Vorführung des Programms und des Quellcodes kann die Gruppe den Versuchstermin vorzeitig beenden.

4.5 Objektorientierte Programmierung

Um Studenten – ohne Vorkenntnissen im Programmieren – den Einstieg zu erleichtern, ist zum Lösen der Aufgabenstellung keine Objektorientierte Programmierung notwendig und wird im Rahmen des Versuchs nicht verlangt. (OOP ist somit optional.)

5 Arbeits- und Brandschutzhinweise

5.1 Vorbeugende Maßnahmen:

- Die Praktikumsteilnehmer haben sich so zu verhalten, dass Gefahrensituationen und Unfälle vermieden werden.
- Die Befugnis zum Bedienen und Nutzen von Geräten ist auf den zugewiesenen Praktikumsplatz beschränkt.
- Eingriffe in die zum Praktikumsaufbau gehörenden Geräte sind nicht erlaubt.
- Defekte an Geräten oder Gebäudeeinrichtungen sind unverzüglich dem Betreuer mitzuteilen. Betroffene Geräte sind außer Betrieb zu nehmen. Andere Personen sind vor Gefahren zu warnen.
- Den Anweisungen der Praktikumsbetreuer bzw. anderer aufsichtsführender Personen ist unbedingt Folge zu leisten.
- Betriebsfremde dürfen sich nur mit Erlaubnis des Praktikumsbetreuers in den Praktikumsräumen aufhalten.
- Rauchen und Umgang mit offenem Feuer ist nicht gestattet.
- Nach Ende des Praktikums ist der Arbeitsplatz in sauberem und aufgeräumtem Zustand zu hinterlassen.
- Außergewöhnliche Ereignisse bzw. besondere Vorkommnisse sind umgehend dem Betreuer oder dem diensthabenden Assistenten zu melden.

5.2 Verhalten im Falle eines Brandes:

- Beachten der richtigen Reihenfolge: **MELDEN - RETTEN - LÖSCHEN**

5.2.1 Feuer melden:

- Telefonische Brandmeldung:
  - Notruf 112 der Feuerwehr (von jedem Telefon aus möglich)
  - Notruf HA 34515 der Technischen Leitzentrale der TUD
- Deutliche, genaue und vollständige Angaben:
- Wo brennt es?
- Was brennt?
- Angaben zu verletzten oder gefährdeten Personen
- Wer meldet?

5.2.2 Personen retten:

- Erste Hilfe leisten
- Weitere Hilfe organisieren, medizinische Hilfe anfordern
- Gefahrenbereich räumen; Fluchtwege benutzen, keine Aufzüge
- Andere Personen warnen, Sammelplatz (Platz vor Turmeingang zum Barkhausenbau) aufsuchen
- Behinderten und älteren Personen helfen

5.2.3 Löschversuch unternehmen

- Feuerlöscher verwenden (Standorte: Gänge des Barkhausenbau), dabei sich nicht selbst gefährden
- Fenster und Türen schließen, aber nicht abschließen
- Möglichst elektrische Verbraucher abschalten

5.3 Rufnummern für Notfälle:

Helfer	Telefonnummer
Rettungsdienst	112
Polizei	110
TUD-Notruf	34515
Betriebsärztlicher Dienst	36199
Klinikum Friedrichstadt Notaufnahme	0351-480 1938