1 Einführung

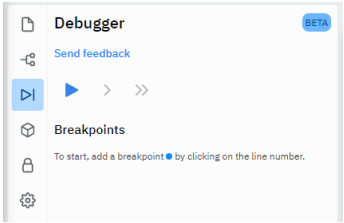
> "Bad programmers worry about the code. Good programmers worry about data structures and their relationships."
>
> — Linus Torvalds

Die Idee hinter dieser Ideensammlung ist es, dir Hinweise auf Datenstrukturen und die Möglichkeiten der Modularisierung aufzuzeigen. Der Weg zum weniger schlechten Programmierer ist gar nicht so lang. Ein paar Hinweise können dich weit bringen. Und am Anfang musst du dich bemühen, diese auch umzusetzen. Das ist ungewohnt, aber nicht wirklich schwer.
Die Beispiele versuchen sich am Grundgerüst zu orientieren. Aber es sind keine fertigen Blöcke, die man einfach so in sein Spiel einkopieren kann. Sie zeigen nur einen rudimentären Ansatz, wie man die angesprochene Idee umsetzen könnte. Das muss weder die eleganteste noch die beste Lösung sein und sie muss auch nicht zu deinem bisher vorhandenem Spiel passen. Die Beispiele sind auch kurz und knapp. An vielen Stellen fehlen unter anderem Prüfungen, ob die jeweiligen Dinge Sinn machen.

1.1 Gebrauchsanweisung

Lies den Text und die Beispiele. Versuche die Programmbeispiele zu verstehen. Dabei hilft es, den Quelltext selber einzutippen und dann schrittweise Änderung für Änderung laufen zu lassen. Selber tippen ist wichtig, weil man dabei genauer lesen muss und dabei bereits mehr versteht. Um es ganz zu verstehen und zu begreifen, hilft es, dich solange mit deinem Programm zu beschäftigen, bist du es wirklich voll und ganz verstanden hast.

Um an unklaren Stellen Einblick zu nehmen, kannst du dir mit `print()` ansehen, welche Werte die Variablen annehmen. Alternativ und eleganter kannst du dafür einen Debugger verwenden. In repl.it ist einer eingebaut, der Werkzeuge enthält, mit denen du dir den inneren Zustand deines Programms bequem ansehen kannst.



Du setzt Haltepunkte durch einen Klick auf Zeilennummern. Anschließend startest du das Programm durch einen Klick auf das blaue Dreieck. Nun stoppt das Programm am ersten Haltepunkt und der Debugger zeigt dir den aktuellen Zustand. Von hier kannst du schrittweise durch dein Programm laufen oder das Programm bis zum nächsten Haltepunkt fortsetzen.

 © 2022 Roland Härter

2 Variablen

Die einfachste Datenstruktur ist noch gar keine Struktur. Es ist die Verwendung einer einfachen Variable, um irgendeinen Zustand im Spiel zu speichern und zu nutzen.
Der Computer merkt sich Dinge in Variablen. Das kann man sich wie einen Eimer mit einem Etikett vorstellen. Eine Variable in Python ist ein Wort ohne Anführungszeichen. Dies ist der Name der Variable. Dieser kannst du mit einem einfachen Gleichheitszeichen = einen Wert zuweisen. Du füllst also etwas in den Eimer. Dabei sorgt der Python-Interpreter automatisch dafür, dass die Variable den richtigen Typ hat. Er wählt sozusagen automatisch die richtige Sorte Eimer für dich aus.

2.1 Beispiel „Leben“

Du kannst deinem Spieler eine begrenzte Zahl an Leben gewähren. Dazu musst du als Entwickler deines Spiels mitzählen können, wie viele Leben der Spieler hat.

```
# ...
leben = 3
# ...
while das_spiel_laeuft == True:
   # ...
   if boese_dinge_passieren() == True:
      leben -= 1
      print ("Du hast noch {} Leben.".format(leben))
   # ...
   if leben < 1:
      print ("Du bist gestorben. Das Spiel ist aus.")
      das_spiel_laeuft = False
   # ...
```


© 2022 Roland Härter

3 Funktionen

Wenn du einen bestimmten Programmcode mehr als ein- oder höchstens zweimal brauchst, wird es Zeit für eine Funktion. Eine Funktion ist ein Stück Programmtext, den du immer wieder aufrufen kannst. Funktionen können Parameter haben, die beim Aufruf mitgegeben werden. Funktionen können auch einen Rückgabewert haben. Das kann in Python alles vom einfachen `True` und `False` bis hin zu komplexen Datenstrukturen sein.
Auch wenn dein Programmtext unübersichtlich wird, ist es eine gute Idee, darüber nachzudenken, ob du nicht einen Teil der Unordnung in eine Funktion packen möchtest. Die Mühe lohnt sich, belohnt dich mit einem aufgeräumteren Programmtext, den dein zukünfiges ich in ein paar Wochen auch noch gut verstehen kann.

3.1 Beispiel „Böse Dinge passieren…“

Hier im Beispiel eine parameterlose Funktion, die auf globale Variablen zugreift.

```
# ...
def boese_dinge_passieren():
   if drache_erwacht == True and current_room == "Drachenhort":
      print ("Der Drache öffnet ein Auge und sieht dich hungrig an.")
      print ("Er öffnet sein Maul und ein Flammenstrahl schießt hervor.")
      print ("Du kannst nicht ausweichen.")
      return True
   elif current_room == "Drachenhort":
      print ("Der Drache scheint zu schlafen.")
   else:
      print ("Es riecht streng. Nach großer Gefahr.")
   return False

# ...
```

4 Schnelles Spiel

Die Uridee des ruhigen Erkundens und sorgfältigen Knobelns ist das eine. Man kann auch ein schnelles Textadventure gestalten. Dazu muss man die Spieldauer messen und es gilt: Je schneller, desto besser.

```
import time
# ...
startzeit = time.time()  # Die Zeit seit 1.1.1970 Mitternacht in Sekunden
while das_spiel_laeuft == True:
   # Hauptschleife
   # ...
# Hinter der Hauptschleife ist das Spiel vorbei
zielzeit = time.time()
dauer = startzeit - zielzeit
# ...
```

Besser lässt sich das ganze mit den hier noch fehlenden Teilen als Funktion implementieren. Die Funktion ruft man dann an allen Stellen auf, an denen das Spiel zu Ende gehen kann. Dabei sollte man zwischen 'Du hast schnell das Ziel erreicht' und 'Du bist schnell gestorben' unterscheiden.

```
def spieldauer(startzeit):
   zielzeit = time.time()
   dauer = zielzeit - startzeit
   if leben > 0:
      pruefe_highscore(dauer) # eine weitere Funktion, die du schreibst...
   else:
      print ("Schnell gespielt und schnell gestorben.")
```

Schnelles Spiel könnte beispielsweise auch in die Richtung eines Escape Rooms gehen. Dann tickt die Uhr, das Wasser steigt, und man muss rechtzeitig den Ausgang finden.

```
# ...
import time
# ...
endzeit = time.time() + (60 * 60) # eine Stunde
while time.time() < endzeit:
   # ...
```

Wenn man die Prüfung als Funktion realisiert, kann man dem Spieler optional einen Hinweis geben, wie viel Zeit ihm noch bleibt.

© 2022 Roland Härter

5 Listen

Listen sind eine einfache, intuitive Datenstruktur. Mit der Kleinigkeit, dass die Listenelemente ab 0 gezählt werden.

5.1 Beispiel „Dinge sammeln“

Um das Spiel erfolgreich zu beenden, muss der Spieler eine Menge Dinge einsammeln. Diese Dinge müssen zum einen zum Vergleich vorliegen, zum anderen braucht der Spieler eine Art Inventar, in dem die bereits gefundenen Dinge notiert werden. Beides kann man als Liste realisieren.

```
# ...
artifacts = [ 'golden coin', 'silver fork', 'copper knife', 'wooden torch']
simple_inventory = []  # leer zu Beginn des Spiels
# ...
while das_spiel_laeuft == True:
   # ...
   if current_room == "attic":
      if 'copper knife' in artifacts:
         print ("There is a copper knife here.")
         antwort = input("Do you want to take it? ").lower()
         if antwort in ['y','j','yes','ja']:
            simple_inventory.append("copper knife")
            artifacts.remove("copper knife")
   # ...
# Am Ende das Ergebnis ausgeben
if len(artifacts) == 0:
   print ("Du hat alle Schätze gefunden. Hurra!")
else:
   print ("Du hast nicht alle Schätze gefunden.")
   antwort = input("Willst du wissen, was du verpasst hast?")
   if antwort in ['y','j','yes','ja']:
      print ("Diese Dinge hast du verpasst:")
      for ding in in artifacts:
         print("{}, ".format(ding),end="")
   else:
      print ("Du hast die folgenden Dinge gefunden:")
      for ding in in simple_inventory:
         print("{}, ".format(ding),end="")
   print ("\n Vielen Dank für deinen Besuch in Krchhs Schatzhöhlen")
```


© 2022 Roland Härter

6 Zufall

Immer das gleiche Labyrinth mit immer den gleichen Monstern, Prinzen, Schätzen an den gleichen Stellen wird schnell langweilig. Dagegen hilft der Zufall, den es auch im Computer gibt. Ziemlich zufällig zumindest.

```
# ...
import time
import random
random.seed(time.time())    # Den Zufallsgenerator mit der aktuellen
                             # Uhrzeit initialisieren
kommazahl = random.random() # eine Zahl zwischen 0 und 1
zufallszahl = random.randint(0,10)
```

6.1 Beispiel „Zufälliger Startraum“

Hier sind als einfaches Beispiel vier Räume aus einem erweiterten Setting ausgewählt, in denen das Spiel sinnvoll beginnen kann. Dabei wird die Zufallszahl passend zur Länge der Liste mit den möglichen Starträumen generiert.

```
# ...
startraeume = [ 0, 9, 10, 18 ]
current_room = startraeume[random.randrange(len(startraeume))]
# ...
```

6.2 Beispiel „Zufällig verteilte Artefakte“

Wie auch den Startraum kann man die findbaren Gegenstände zufällig verteilen. Hierbei ist es einfacher, die Gegenstände in einem Dictionary mit den Räumen zu verküpfen, anstatt sie bloß in einer Liste zu speichern.
Bei dieser Konstruktion muss man sicherstellen, dass es mindestens so viele Räume wie Artefakte gibt. Es spricht aber technisch nichts dagegen, mehrere Artefakte in einem Raum unterzubringen. Dann sollte der Spieler aber danach suchen müssen.

```
import time
import random
random.seed(time.time())
artifacts_list = [ 'golden coin', 'silver fork', 'copper knife', 'wooden torch']
possible_rooms = ['R1', 'R3', 'R7', 'R8', 'R9', 'R12', 'R15'] # mögliche Räume
artifacts = {}  # leeres Dictionary für Zuordnung Artefakt - Raum
for artifact in artifacts_list:
   zufall = random.randrange(len(possible_rooms))
   artifacts[artifact] = possible_rooms[zufall]
   del possible_rooms[zufall]  # den Raum löschen, nicht doppelt belegen

print ("Kontrolle Zuordnung: ", str(artifacts))
print ("Kontrolle Resträume: ", str(possible_rooms))
```

Man kann die Zuordnung auch anders herum vornehmen, also den Räumen die Artefakte zuordnen.

```
import time
import random
random.seed(time.time())
artifacts_list = [ 'golden coin', 'silver fork', 'copper knife', 'wooden torch']
possible_rooms = ['R1', 'R3', 'R7', 'R8', 'R9', 'R12', 'R15', 'R18']
raumbelegung = {} # leeres Dictionary für Zuordnung Raum - Artefakt
nummer = 0
for artifact in artifacts_list:
   zufall = random.randrange(len(possible_rooms))
   raumbelegung[possible_rooms[zufall]] = artifacts_list[nummer]
   del possible_rooms[zufall]  # den Raum löschen, nicht doppelt belegen
   nummer += 1

print (str(raumbelegung))
```

7 Anwenden von Gegenständen

Wenn es Gegenstände im Spiel gibt, kann man diese auch verwenden.

7.1 Beispiel „Verschließbare Türen“

Im Beispiel „Verschließbare Türen“ kommt der Benutzer nur mit Hilfe des Schlüssels `diamond key` durch diese Tür. Hier ist implizit enthalten, dass es mehrere Schlüssel gibt, von denen jeder nur „seine“ Tür öffnet. Damit wird das Spiel spannender, vor allem, wenn man die Zuordnung Schlüssel – Tür für jeden Lauf zufällig neu bestimmt.

```
# Diese Funktion prüft, ob man einen Schlüssel für den Raum hat.
def check_key(my_simple_inventory, my_current_room, my_direction):
   key = input("Hast du den Schlüssel? ").lower()
   if key in [ "j", "ja", "y", "yes" ]:
      if 'diamond key' in my_simple_inventory:
         return open_the_door(my_current_room, my_direction)
      else:
         print ("Du hast nicht den richtigen Schlüssel")
   else:
      print ("Die Tür ist verschlossen.")
   return False

# ...
```

Hier ist eine Funktion definiert, die explizit alle notwendigen Informationen als Parameter übergeben bekommt. Das ist ein Hinweis auf *Gültigkeitsbereiche*.

7.2 Gültigkeitsbereiche

Grob gesagt gibt es verschiedene Bereiche, in denen Variablen gültig sein können. So sind normalerweise alle Variablen in einer Funktion nur innerhalb dieser Funktion gültig und existent. Sie heißen dann *lokale Variablen*. Wenn eine Variable außerhalb aller Funktionen definiert wird, ist sie eine *globale Variable*. Diese kann man innerhalb von Funktionen lesen, aber nicht verändern. Wenn man das trotzdem tun möchte, muss man die Variable in der Funktion als `global` deklarieren.
Zu recht gelten globale Variablen als schlechter Stil, weil ein Programm schnell unübersichtlich wird und globale Variablen ein interessanter Quell an unerwarteten Seiteneffekten und Fehlern sind.

 © 2022 Roland Härter

8 Veränderliche Welt

Die Spielwelt muss nicht unveränderlich sein. Es ist allerdings ein schmaler Grat zwischen einer attraktiven dynamischen Spielwelt und einer in den Wahnsinn treibenden Hölle.

8.1 Beispiel „Portal erzeugen“

Portale sind einfache Veränderungen an der Spielwelt. Hierbei modifiziert man zur Laufzeit die Dictionaries mit den Richtungen des Spiels.

```
if current_room == 'Thronsaal' and
                        command == 'go west' and west['Thronsaal'] == None:
   if 'ruby rod' in simple_inventory:  # bisher keine Hände
      print ("The ruby rod shivers, then starts to glow red."
      time.sleep(1)
      print ("Suddenly a bright red flash emits and hits the western wall")
      print ("The western wall hums and turns into a billowing mirror")
      west['Thronsaal'] = 'Schatzkammer'
```

8.2 Beispiel „Labyrinthe der Hölle“

In einem extremen Adventure kann man kontinuierlich alle Verbindungen verändern. Dabei sollte man aber durch die Programmlogik dafür sorgen, dass das Spiel irgendwie spiel- und gewinnbar bleibt.

```
raumliste = list('ABCDEFGHIJKLMNOPQRSTUVWXYZ') # 26 Räume billig erzeugt

def generiere_ziel():
   if random.random() > 0.7:  # 30% Wahrscheinlichkeit gibt durchschnitlich
                              # mindestens einen Ausgang pro Raum
      return raumliste[random.randrange(len(raumliste))]
   else:
      return None

west = {}
east = {}
north = {}
south = {}
for raum in raumliste:
   for richtung in east, west, north, south:
      richtung[raum] = generiere_ziel()
```

9 Generierte Welt

Am Anfang macht es viel Spaß, sich Räume, Beschreibungen und Inhalte auszudenken und sie in den Programmtext einzufügen. Aber auf Dauer ist es ermüdend und damit fehlerträchtig, eine große Vielzahl von Räumen zu erschaffen und mit Inhalt zu füllen. Als Programmierer denkt man dann gerne nach, um sich die Routineaufgaben von der Maschine abnehmen zu lassen. Das geht auch hier. Das Ziel ist dabei, viele Räume mit ihren Beschreibungen und Rauminhalten aus einfachen Textdateien zu erzeugen. Es braucht also Raumname, Raumbeschreibung und Rauminhalt. Diese müssen zueinander passen. In einer einfachen Welt gibt es darüber hinaus keine weiteren Randbedingungen, die ein Generator berücksichtigen muss.
Für ein möglichst einfaches Erstellen der Eingabedatei für den Generator sollte das Format dieser Datei einfach und gleichzeitig leicht lesbar sein. CSV, Comma Separated Values, wäre ein solches Format, solange die Beschreibungstexte nicht zu lang und damit unübersichtlich werden.

```
kitchen; The kithen is very large.;plate,cup,wood
living room;The living room is full of enormous furniture.;folio, bird skull
```

Alternativ wäre eine einfache Teilmenge von YAML vorstellbar. Hier trennt `---` die einzelnen Räume und jeder der drei möglichen Bestandteile wird durch `-` eingeleitet.

```
---
- kitchen
- The kitchen is very large. The floor is covered with art nouveau tiles.
Some of them are cracked. There is a large wooden cooker in front of the window.
Next to it, there is firewood in a basket. On the side wall is a sideboard
with plates and cups.
- plate, plate, plate, plate, cup, cup, cup, cup, wood, wood, wood, wood
```

YAML bietet auch die Möglichkeit, die einzelnen Bestandteile zu benennen.

```
---
room: living room
description: The living room is full of enormous furniture. The chic oak parquet
with filigree inlays hardly comes into its own. On one side is a ceiling-high
shelf full of dark leather-bound folios. The titles on the spine, if there were
any, are not visible. On the other side is a massive multi-piece living room
cabinet. Some doors are glazed. Arcane artefacts can be seen behind them.
A bird skull grins at you.
contents: folio, folio, folio, bird skull, staff, rod, wand, hammer, dagger
```

 © 2022 Roland Härter

10 Einfaches YAML selber lesen und schreiben

10.1 Schreiben

Wenn man das Schreiben gelöst hat, ist das Lesen viel einfacher. also beginnen wir mit dem Schreiben.

```
raum = [ 'kitchen', 'living room', 'sleeping room' ]
descr = [ 'A large kitchen with an opulent fridge.',
      'The living room is dominated by a giant TV screen.',
      'The sleeping room is almost empty. Only a tatami lies in the corner.', ]
cont = ['fridge, table, chair, kitchen sink','tv, armchair','tatami' ]
raumdatei = open ("raumdatei.yaml", "w")
for i in range(len(raum)):
   raumdatei.write("---\n")
   raumdatei.write("raum: {}\n".format(raum[i]))
   raumdatei.write("description: {}\n".format(descr[i]))
   raumdatei.write("contents: {}\n".format(cont[i]))
raumdatei.flush()
raumdatei.close()
```

10.2 Lesen

Es geht hier darum, das einfache selbstdefinierte Format einzulesen. Dieses sollte genau nach der eigenen Spezifikation sein. Raute als Kommentarzeichen am Zeilenanfang wird erkannt und Kommentarzeilen werden ignoriert.

```
raumdatei = open ("raumdatei.yaml", "r")
raumliste = []
for zeile_ in raumdatei.readlines():
   zeile = zeile_.strip()
   if zeile.startswith('#'):
      pass
   elif zeile.startswith('---'):
      print("Neuen Raum begonnen.")
   elif zeile.startswith('raum:'):
      token,inhalt = zeile.split(':',1)
      raumliste.append(inhalt)
   else:
      print("Der Rest ist Übung für dich.")
raumdatei.close()
# Ausgabe des bisher eingelesenen
print(str(raumliste))
```

11 Meta-Generator

Es ist eine bekannte Idee, mehr oder weniger sinnvolle lustige Kombinationen von Figuren oder Sätzen zu bilden. Das gibt es als Klappbilderbuch mit geteilten Seiten oder als Programmieridee, wie zum Beispiel `https://nostarch.com/download/samples/BigBook_SampleProject11.pdf`.
Statt nun die Prinzessin auf dem Hexenbesen mit den Eisenfüßen zu erzeugen, kann man aus einer oder mehreren Auswahlmengen Räume samt Beschreibung und Inhalt erzeugen lassen. Diese kann man dann auch für die Türidee in Abschnitt 7.1 verwenden. Kombiniert mit der höllischen Variante wird man dann an der Tür nach irgend einem von den vielen durch den Meta-Generator möglichen Schlüsseln gefragt. Und zwar jedes Mal nach einem anderen. So dass man als Spieler möglichst alle davon sammeln muss. Das wiederum kann man durch ein begrenztes Inventar verschärfen. Dabei darf man aber nicht die Spielbarkeit aus dem Auge verlieren. Es muss also möglich sein, jede Aufgabe zu lösen. Für ein begrenztes Inventar und eine mögliche Menge an Schlüsseln könnte das beispielsweise bedeuten, dass das Inventar gerade groß genug ist, um alle möglichen verschiedenen Schlüssel aufzunehmen. Aber eben nur genau so groß, wie fies. Dazu sollte es dann auch die Möglichkeit von Truhen oder ähnlichem geben, in denen der Spieler sein weiteres Inventar in der Zwischenzeit lagern kann. Eine weiter verschärfte Version wären Bahnhofsschließfächer, für die man Münzen braucht, um diese für eine bestimmte Anzahl an Bewegungen zu reservieren. Natürlich sind jedes Mal andere Münzen gefragt und auch die Dauer, für die man reserviert, ist von der Tagesform des Schließfaches und der gerade geforderten Art von Münzen abhängig.

11.1 Gewichte

In einem Haus gibt es verschieden viele Exemplare von jeder Art Raum. Meistens gibt es eine Küche, zwei Toiletten, ein paar Bäder und recht viele normale Räume. Eine Möglichkeit, eine solche Gewichtung abzubilden, ist die Verwendung einer Liste, in der die zu verteilenden Räume in der gewünschen Verteilung vorliegen. Hier zum Beispiel 10% Küchen, 20% Toiletten, 20% Badezimmer und 50% normale Räume.

```
raumliste = [ kitchen, toilet, toilet, bathroom, bathroom,
          room, room, room, room, room ]
```

Bei zufälliger Auswahl sollten die Räume in der gewünschten Verteilung erzeugt werden. Allerdings darf hierbei die Anzahl der Versuche nicht zu klein sein. Es leuchtet ein, dass sich bei nur drei generierten Räumen keine sinnvolle Verteilung einstellen kann.

12 Biome

Beispielsweise in Minecraft gibt es das Konzept der Biome. Biome bestimmen bestimmte Parameter der Umwelt. Ein Küstenbiom ist ganz anders als das Hochgebirge. Umgesetzt auf dein Textadventure könnte dieses verschiedene Biome je nach Setting enthalten.
In einem Haus sind die Keller anders als das Erdgeschoss oder der Garten. Keller sind dunkler, feuchter, kälter und in Kellern fühlen sich andere Tiere und Pflanzen wohl als auf dem Dachboden.
In einem Labyrinth von Höhlen kann man geologische Faktoren erfinden. So kann es Zonen mit vielen kleinen Höhlen geben oder Bereiche ohne Wasser.

12.1 Datenstrukturen

Wenn ich Biome habe, müssen diese auch einen Platz in meinen Datenstrukturen haben. Als Beispiel hier eine Raumliste mit einem einzigen Raum.

```
raumliste = { 'raumname' : {
         'biom' : 'Keller',
         'beschreibung' : 'Ein kleiner Keller mit einer Kartoffelkiste.',
         'inhalt' : ['kartoffel', 'kiste'],
         'menge' : {
            'kartoffel' : 23,
            'kiste' : 1,
            }
   }
}
```

Ein Raum hat einen Raumnamen. Der ist hier `'raumname'`. Dieser Name ist der Schlüssel in einem Dictionary. Der dazugehörige Wert ist ein weiteres Dictionary mit den Schlüsseln `'biom'`, `'beschreibung'`, `'inhalt'` und `'menge'`. `'inhalt'` zeigt auf eine Liste und `'menge'` auf ein weiteres Dictionary. Man kann in Python beliebige Datenstrukturen erstellen. Allerdings sollte man diese auch in seinem Kopf 'jonglieren' können.

```
if raumliste['raumname']['biom'] == 'Keller':
   print ("Uhh, ein Keller. Es riecht auch so.")
# ...
if raumliste['raumname']['menge']['kartoffel'] < 1:
   print("Es gibt keine Kartoffeln mehr.")
```


© 2022 Roland Härter

13 Zähler und Punkte

Menschen lieben es, ihren Fortschritt sehen zu können. Zahlen, die größer werden und Leisten, die länger werden, begeistern uns. Jedenfalls ab einem bestimmten Alter. Vorher begeistern die schönen Tonnen mehr und die Bedeutung der ganzen Zahlen ist unklar.
Zähler können alles mögliche zählen. Schritte, die man gegangen ist oder Monster, die man gefangen hat. Zähler können immer angezeigt werden oder nur auf Anfrage. Ersteres führt in einem reinen Textaadventure, das einfach zeilenweise Text ausgibt, schnell zu unübersichtlichen Ausgaben. Von daher bietet sich eine Funktion an, die auf Wunsch oder bei bestimmten Ereignissen den Spieler über einen Teil seines inneren Zustands informiert. Manches sollte verborgen bleiben, damit das Spiel spannend bleibt.
Ein mögliches Ereignis könnte in einer Höhlenwelt zum Beispiel das Essen einer bestimmten Flechte oder eines bestimmten Pilzes sein, der den Charakter in einen Zustand höherer Selbstwahrnehmung bringt.

14 Level

Level können zwei Bedeutungen haben. Zum einen der Level in einem klassischen Dungeon. Jeder Abstieg auf eine tiefere Ebene bedeutet den Aufstieg um ein Gewölbelevel. Hier warten größere Schätze, aber auch größere Gefahren auf den Spieler. Zum anderen kann das Sammeln einer gewissen Menge an Schätzen, Monstern, Gold, ... den Aufstieg des Spielers auf einen höheren Spielerlevel auslösen. In einem höheren Spielerlevel bekommt der Spieler erweiterte Fähigkeiten. Aber auch die Qualität und Gefährlichkeit der Gegner sowie die Knifffligkeit der Rätsel kann genauso wie bei einem höheren Gewölbelevel zunehmen.

15 Minigames

Als Herausforderungen für den Spieler kann man auch Minigames implementieren. Mein persönlicher Favorit an der Stelle wären Programmierherausforderungen. Das gab es sogar schon mal in einem Textadventure. Dort musste man knifflige LISP-Programme schreiben, um im Spiel weiter zu kommen.

 © 2022 Roland Härter

16 Escape Room

Die ursprüngliche Idee hinter dem Spiel, dass der Spieler „Ravenswood Manor" erkunden muss, kann man bereits als eine Art Escape Room bzw. Escape House auffassen. Kombiniert mit begrenzter Zeit und begrenzten Ressourcen wird es ein richtiger Escape und dies kann spannend und reizvoll sein.

17 NPC

Man kann ELIZAs einbauen, die mit dem Spieler reden. Dabei können diese eventuell hilfreiche Tipps geben. Vergleichbar beispielsweise mit den preiswerten und den teuren Orakeln in Nethack. Ein Beispiel ware diese:
`https://replit.com/@kunstundtechnik/ELIZA-full`

18 Monster

In einem guten Dungeon sollte es Monster geben, oder nicht?

18.1 Alternatives Setting ohne „böse" Monster

Es ist auf jeden Fall eine Überlegung wert, ob man auch ohne Monster und Kämpfe eine spannendes Spiel gestalten kann. Tetris kommt beispielsweise ohne Monster und Kämpfe aus.

19 Weiterführende Ressourcen

Hier noch ein paar mehr oder weniger zufällig, auf Englisch „by serendipity", gefundene Links. Gestartet habe ich meine Suche mit „textadventure lisp" aufgrund von Abschnitt 15.
`https://eblong.com/infocom/`
`http://textadventures.co.uk/`
`https://www.youtube.com/user/TheMorpheus407`
`http://www.pythonchallenge.com/about.php`

 © 2022 Roland Härter