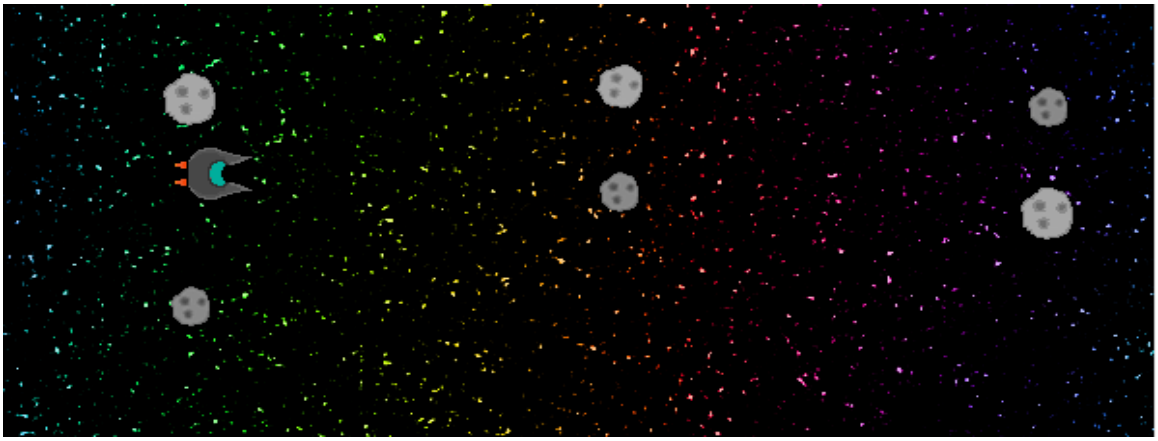
Anna-Freud-Schule Informatikkurs in17G



Bodo Beyer – 18.10.2015

Ein Spiel in Python

Inhaltsverzeichnis

1. Spielidee

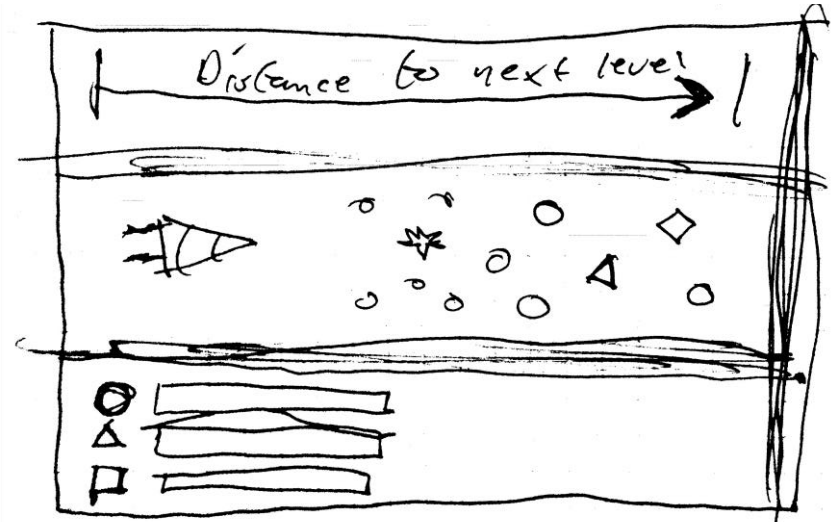


Abb.1:
Die erste Skizze

Die ursprüngliche Spielidee war es, verschiedenartige Objekte mit einem Raumschiff einzusammeln und dabei darauf zu achten, in etwa gleich viele Objekte der verschiedenen Typen zu sammeln. Letztendlich erschien mir dies aber als zu verwirrend und ich habe mich entschieden ein simpleres Spielkonzept umzusetzen.
Im Spiel geht es nun darum, mit seinem Raumschiff umherfliegenden Gesteinsbrocken auszuweichen und Treibstoff einzusammeln. Das Spiel endet, wenn der Treibstoff alle ist.

2. Anleitung

Abb.2: Das Spielfenster

Ziel des Spiels ist es, durch das Einsammeln von Treibstoff möglichst lange zu fliegen und möglichst viele Punkte zu erreichen. Dabei sollte den Gesteinsbrocken ausgewichen werden, da ein Zusammenstoß Treibstoff und Punkte kostet.

Steuerung

Die Spielfigur folgt dem Mauszeiger.

Die **Taste R** startet das Spiel neu.
Die **Taste P** pausiert das Spiel, um das Spiel fortzusetzen muss auf die Spielfigur geklickt werden. Das Spiel kann auch durch klicken angehalten werden.

Objekte im Spiel

Treibstoff	muss eingesammelt werden um die Treibstoffanzeige zu füllen und Punkte zu erhalten.
Gesteinsbrocken	dürfen nicht eingesammelt werden, sonst sinkt die Punktzahl und die Treibstoffanzeige.
Der Spieler	folgt dem Mauszeiger.

Aufbau des Spielfensters

Die Treibstoffanzeige	zeigt an, wie viel Treibstoff noch vorhanden ist. Das Spiel endet, wenn sie leer ist.
Die Punktzahl	gibt an, wie viele Punkte im Spiel erreicht wurden. Sie erhöht sich, je länger man spielt und je mehr Treibstoff eingesammelt wird.
Das Spielfeld	befindet sich über der Treibstoffanzeige und bildet das eigentliche Spielgeschehen ab.

3. Aufbau des Programmes

3.1 Code-Dateien

Hinweis zum Aufbau der einzelnen Dateien (genannt Module):

Alle Module enthalten wichtige Konstanten die am Anfang der Datei zwischen den Importen und der Klassendefinition definiert werden. Konstanten sind, nach den geltenden Python-Stilrichtlinien[1] mit Großbuchstaben bezeichnet.

spiel.py

- ist die Hauptdatei.
- beinhaltet die Klasse Spiel (wird nur 1-mal instanziiert).
- übernimmt die Initialisierung des gesamten Spiels.
- übernimmt die Verarbeitung von Eingaben (Tasten, Maus).
- beinhaltet die Hauptschleife des Spiels.

spieler.py

- beinhaltet die Klasse Spieler (wird nur 1-mal instanziiert).
- übernimmt das Zeichnen der Spielfigur.

spielfeld.py

- beinhaltet die Klasse Spielfeld (wird nur 1-mal instanziiert).
- initialisiert das Spielfeld und stellt dieses bereit.
- übernimmt das Zeichnen des Spielfeldhintergrundes.

statistik.py

- beinhaltet die Klasse Statistik (wird nur 1-mal instanziiert).
- übernimmt das Zeichnen der Treibstoff- und der Punkteanzeige.

zielObjekt.py

- beinhaltet die Klasse ZielObjekt (wird in der Klasse Spiel für jedes ZielObjekt einmal instanziiert)
- übernimmt das Zeichnen der einzelnen ZielObjekte.

3.2 Grafik-Dateien

spieler.png	die Spielergrafik (Raumschiff)
hintergrund.png	der Spielfeldhintergrund
stein0.png	Grafik für einen Gesteinsbrocken
stein1.png	Grafik für einen Gesteinsbrocken
stein2.png	Grafik für einen Gesteinsbrocken
treibstoff.png	Grafik für den Treibstoff

[1] https://www.python.org/dev/peps/pep-0008/#constants

3.3 Programmablauf

Beim Ausführen der Datei spiel.py wird die ein Objekt der Klasse Spiel instanziiert:

spiel.py
```
217 if(__name__== "__main__"):
218     Spiel()
```

Die `__init__` Methode übernimmt die Initialisierung der anderen Programm Komponenten und ruft die Methode `start` auf, welche die Hauptschleife aufruft und tkinter anweist, mit dem Zeichnen des Spielfensters zu beginnen:

spiel.py
```
62  self.hauptschleife()
63  self.fenster.mainloop()
```

In der Hauptschleife sorgt die Anweisung

```
self.fenster.after(10, self.hauptschleife)
```

dafür, dass die Hauptschleife durch tkinter wieder aufgerufen wird. Das erste Argument gibt dabei die Zeit in Millisekunden an, die bis zum Aufruf gewartet wird. In dieser Zeit zeichnet tkinter das Spielfenster. Die Zeit wird so berechnet, dass die Hauptschleife eine bestimmte Anzahl von Aufrufen in der Sekunde erfährt. Diese Anzahl ist durch die Konstante `FPS` festgelegt:

spiel.py
```
10  FPS = 40
```

Die Hauptaufgabe der Hauptschleife ist dabei der Aufruf der beiden Methoden, die das eigentliche Spiel berechnen und anzeigen:

spiel.py
```
89  self.aktualisiere()
90  self.zeichne()
```

Die Methode `aktualisiere` überprüft, ob noch Treibstoff vorhanden ist und beendet gegebenenfalls das Spiel. Ansonsten sorgt sie für den Treibstoffverbrauch und die mit der Zeit steigende Punktzahl. Sie ruft die gegebenenfalls die Methode `erzeugeWelle` auf, welche eine neue Welle[2] von ZielObjekten(Gesteinsbrocken und Treibstoff) instanziiert.

Anschließend ruft sie die Methode `bewegeDich` der einzelnen Zielobjekte, die in der Liste `ziele` gespeichert sind, auf und sorgt so für deren Bewegung. Sie überprüft dabei, ob es zu einer Kollision mit dem Spieler gekommen ist und ruft gegebenenfalls die Methode `zielGesammelt` der Klasse Statistik auf und markiert das Ziel als ungültig.

[2] Mit Wellen werden hier die, in regelmäßigen Abständen erscheinenden, Ansammlungen von ZielObjekten bezeichnet.

Die Methode `zeichne` sorgt für das Zeichnen des Spiels und zeichnet

1. den Spielfeldhintergrund,
2. die einzelnen Zielobjekte,
3. die Spielfigur,
4. und die Treibstoff- und Punkteanzeige.

Diese Reihenfolge ist wesentlich, damit die Objekte in der richtigen Reihenfolge abgebildet werden und zum Beispiel der Hintergrund nicht alle anderen Objekte überdeckt.

In Schritt zwei wird überprüft, ob das ZielObjekt noch gültig, das heißt weder eingesammelt worden ist, noch das Spielfeld verlassen hat, ist und dieses wird dementsprechend gelöscht oder gezeichnet.

3.4 weitere interessante Programmteile

spiel.py

```
 25  # Gewichtung der Zielobjekt-Typen
 26  ZIELOBJEKT_WAHRSCHEINLICHKEITEN = [5, 2]
     }
144  def erzeugeWelle(self):
     }
150  # Waehle den Typ des Zielobjekts unter beruecksichtigung
     der
151  # Gewichtung aus.
152  liste = []
153  for t in range(len(ZIELOBJEKT_WAHRSCHEINLICHKEITEN)):
154                      liste += [str(t)] *
                        ZIELOBJEKT_WAHRSCHEINLICHKEITEN[t]
155  typ = int(random.choice(liste))
```

Dieser Programmteil legt fest, von welchem Typ in neu generiertes ZielObjekt seien soll.

Dazu wird eine Liste erzeugt, die die zwei möglichen Typen entsprechend ihrer Gewichtung enthält. Typ 0 (STEIN) hat hier eine Gewichtung von 5. Typ 1 (TREIBSTOFF) hat eine Gewichtung von 2.

Die erstellte Liste lautet also [0, 0, 0, 0, 0, 1, 1]. Aus dieser Liste wird dann ein zufälliges Objekt ausgewählt, welches als Typ festgelegt wird. So entsteht hier ein Verhältnis von 5 Stein- zu 2 Treibstoffobjekten.

spielfeld.py

```
 6   HINTERGRUND_GESCHWINDIGKEIT = 1
     }
23   def zeichneHintergrund(self):
     }
38   # Aktualisiere die Hintergrundposition
39   self.hintergrund_x -= HINTERGRUND_GESCHWINDIGKEIT
40   if(self.hintergrund_x <=
        0 - self.hintergrundbild.width()):
41           self.hintergrund_x += self.hintergrundbild.width()
42           delta_x = self.hintergrundbild.width()
43   else:
44           delta_x = HINTERGRUND_GESCHWINDIGKEIT * -1
45           # Bewege die Hintergrundteilstuecke
46           self.leinwand.move(self.hintergrund_links,
                                delta_x, 0)
47           self.leinwand.move(self.hintergrund_rechts,
                                 delta_x, 0)
```

Abb.3: Wie der Hintergrund gezeichnet wird.

Dieser Programteil legt fest, wie der Spielfeldhintergrund gezeichnet wird.

Das Spiel zeichnet die Hintergrundgrafik zwei Mal direkt nebeneinander (Abb.3 – die beiden gestrichelten Kästen) und verschiebt beide gleichzeitig (Zeilen `46` und `47`). Wenn der linke Spielfeldhintergrund das eigentliche Spielfeld vollständig verlassen hat, werden beide Hintergründe wieder nach rechts verschoben, so dass der linke Hintergrund das Spielfeld vollständig einnimmt. Da es sich bei der Hintergrundgrafik um eine nahtlos Kachelbare Grafik handelt, entsteht die Illusion einer fortlaufenden Bewegung.

4. mögliche Verbesserungen

Programmfehler

Hinweis:

Diese Liste erhebt keinen Anspruch auf Vollständigkeit und beinhaltet lediglich die Probleme, die ich beim Testen und Programmieren entdeckt habe.

- Nach dem das Spiel pausiert und wieder fortgesetzt wird, entsteht meistens eine neue Welle von Zielobjekten, da die Zeit, die den Abstand zwischen den Wellen misst weiterläuft und somit beim Fortsetzen das Erzeugen einer neuen Welle bewirkt wird.
- Um den Pause Modus zu beenden, muss auf die Spielfigur geklickt werden. Manchmal funktioniert dies nicht einwandfrei und es sind mehrere Klicks erforderlich. Der Grund dafür ist mir nicht bekannt, aber ich vermute, dass es mit der Verarbeitung von Eingaben durch tkinter zusammenhängt.
- Die Abstände die ober- und unterhalb der Treibstoffanzeige entstehen sind nicht beabsichtigt. Da ich vor diesem Projekt weder Erfahrung mit tkinter oder Python hatte, ist es mir nicht gelungen sie zu beseitigen.
- Bei dem Erstellen der Wellen, kommt es häufig dazu, dass sich einzelne ZielObjekte überlagern. Dies sollte durch einen verbesserten Algorithmus in der Methode `erzeugeWelle` verhältnismäßig einfach zu beheben sein.

Sonstige mögliche Verbesserungen

- Einen Namen für das Spiel entwickeln.
- Anzeige der erreichten Punktzahl, nachdem der Treibstoff aufgebraucht ist.
- Eine bessere Grafik für den Treibstoff verwenden. Ich hatte eine Grafik eines Benzinkanisters entworfen, war mit dieser allerdings nicht zufrieden und habe mich daher für die im Spiel verwendete abstrakte Darstellung entschieden.
- Die ZielObjekte könnten sich unterschiedlich schnell bewegen. Ich bin mir noch nicht sicher, was bei einer Kollision von Zielobjekten passieren soll.
- Mehr Arten von ZielObjekten. Zum Beispiel „Power-Ups“.
- Implementierung einer Bewegungsparalaxe[3] für den Spielfeldhintergrund.
- Implementierung von Soundeffekten. Da ich keine Möglichkeit gefunden habe Sounddateien mit den Python Standartbibliotheken wiederzugeben, habe ich darauf verzichtet.
- Animationen. Zum Beispiel für die Triebwerke des Raumschiffs oder rotierende Gesteinsbrocken.

[3] Unter **Bewegungsparallaxe** versteht man in der Wahrnehmungspsychologie den Effekt, der sich optisch ergibt, wenn verschiedene Objekte unterschiedlich voneinander entfernt in einer Landschaft verteilt sind und sich der Beobachter parallel zu diesen Objekten seitlich fortbewegt und dabei in Richtung Horizont blickt. (https://de.wikipedia.org/wiki/Bewegungsparallaxe, abgerufen am 19.10.2015)
Grafisches Beispiel auf Wikipedia vorhanden.

5. Quellen

Arbeiten mit tkinter:

http://usingpython.com/using-tkinter/

http://zetcode.com/gui/tkinter/drawing/

http://effbot.org/tkinterbook/

Arbeiten mit Python:

https://docs.python.org/3/faq/design.html

https://docs.python.org/3/tutorial/

Umsetzung einzelner Programmkomponenten:

(**spiel.py** `istKollidierend` Zeile 183 bis 193)

http://www.gamedev.net/page/resources/_/technical/game-programming/collision-detection-r735

(**spiel.py** `hauptschleife` Zeile 82 bis 106)

http://www.koonsolo.com/news/dewitters-gameloop/