**pyOS**

*Ein interpretiertes Betriebssystem, geschrieben in Pygame*

Von Niklas M. Bugenhagen

Gymnasium Alfeld

Kurs IF2

Abgegeben: 00:00:0000

Inhaltsverzeichnis

Projektidee 1

Erwartungen 1

Projektplanung 1

Lerntagebuch 1

Reflexion 2

Quellenverzeichnis 3

**Projektidee**

Ein grafisches Mock OS, mit Spielen und Programmen, eventuell auch eine eigene Programmiersprache. Einstellungen zur Veraenderung des Aussehens sollen ebenfalls implementiert sein.

**Erwartungen**

Es wird fordernd, da Vieles gleichzeitig gemanaged werden muss. Ich bin jedoch zuversichtlich dass ich der Aufgabe gewachsen bin. Ich werde mich viel mit Optimierung beschaeftigen muessen, mit mehreren Dateien arbeiten und vor allem an der Lesbarkeit meines Codes arbeiten.

**Projektplanung**

Vorerst muss die grundlegende Framework programmiert werden, auf eine Weise die Erweiterungen spaeter vereinfacht. Sobald dieser Grundbaustein steht, werde ich an Einstellungen, Programmen und Spielen arbeiten und diese implementieren. Zuletzt werde ich den Code bereinigen, die Grafik designen und Bugtest durchfuehren.

Framework – bereits fertig

Einstellungen, Programme, Spiele – bis 10.3

Bereinigen, Kommentieren – bis 14.3

Grafik – zwischendurch, bis 10.3

Bugtests – zwischendurch, bis 14.3

**Lerntagebuch**

Vorerst habe ich die Grundlagen programmiert, kompliziert durch den Faktor dass ich mir selbst vorausgesetzt hatte, dass Alles mit der Bildschirmgroesse skalieren soll, welches durch zwei Variablen, scale\_horizontal und scale\_vertical implementiert ist. Das zweite Hindernis war die Implementierung eines „Screen Managers“, da dieser eine wichtige Komponente des Grundprinzips darstellte. Jeder „Screen“ besteht aus einer Class mit 3 Funktionen, die in jedem Screen gleich benannt sind und dieselben Argumente benoetigen. Diese sind run(), draw() und click\_check(). Run() ist fuer die Ausfuehrung von Funktionen zustaendig, click\_check() zum Ueberpruefen ob Knoepfe gedrueckt sind und draw() um den Screen anzuzeigen. Zusaetzlich koennen die Classes natuerlich auch andere Funktionen haben. Der Screen Manager funktioniert, indem in main.py erstmal ein dictionary mit den Namen und Instanzen jeder Screen Class. In der Hauptschleife werden dann mit einer Variable „current\_screen“ im dictionary die Funktionen der Instanz des momentanen Screens ausgefuehrt. Anschliessend habe ich eine generelle Class fuer Knoepfe implementiert, die das Erstellen von Knoepfen in der Zukunft erleichtert hat. Daraufhin kam die Implementierung des Start Menues, welches einen Knopf zum Zurueckkehren zum Desktop sowie einen der zu den Einstellungen fuehrt beinhaltet. Hier war der schwierige Teil, das Start Menue nur anzuzeigen und auf Klicks zu ueberpruefen, wenn es auch geoeffnet ist. Dies wurde mit einer Hitbox und Boolean variable geloest. Ausserdem eine Huerde war die Taskleiste. Diese beinhaltet zwei Listen, die offene Screens und Knoepfe fuer diese beinhaltet. Das Schwierige hier war das Entfernen wenn einer der Screens geschlossen wird, und dann die folgenden Knoepfe auf der Taskleiste nach vorne zu bewegen. Als erstes Programm kam eine simple Version von Flappy Bird, wo die Schwierigkeit das Arbeiten mit mehreren Classes und dessen Variablen war. Ebenfalls problematisch war vorerst die Kollisionserfassung, durch die Skalierung. Als naechstes kam Minesweeper, was mein Verstaendnis von Listen/Matrizen vertiefte, da das Spielfeld als Liste gespeichert wird, und die Implementierung der Anzahl an anliegenden Minen sehr viele for Schleifen benoetigte. Zu diesem Zeitpunkt hatte das gesamte Projekt ca. 750 Zeilen Python Code. Dann kam ein Terminal, was viel Uebung mit Textanzeige sowie Texteingabe in Pygame mit sich brachte. Ebenfalls kam zu diesem Zeitpunkt das wohl komplizierteste Thema des Projekts, „pex“, die eigene Programmiersprache von pyOS, welche komplett im Terminal ausgefuehrt werden kann. Eine eigene interpretierte Programmiersprache zu implementieren war sehr kompliziert und erforderte viel Zeit, wahr jedoch weniger schwierig als erwartet. Zu diesem Zeitpunkt wandte ich auch die Zeit auf, allen Code zu kommentieren und die Lesbarkeit zu verbessern. Das naechste grosse Projekt war die Implementierung eines Text Editors, welches mein Verstaendnis von Pygame Textobjekten weiter forderte, sowie pyOS erlaubt, Programme intern zu schreiben und auszufuehren. Nach und nach kamen mehr Funktionen und Logik, sowie Bugfixes mit pex und anderen Teilen des Projekts. Da pex in pyOS zu interpretieren sehr langsam ist, schrieb ich ausserdem einen Intepreter in Golang. Zu diesem Zeitpunkt besteht das Projekt aus ca. 1800 Zeilen Python Code, sowie ca. 500 Zeilen Go Code. Darauffolgend habe ich dem Editor Syntax Highlighting hinzugefuegt.

**Reflexion**

Abschließend reflektieren Sie Ihre Projektarbeit. Was lief gut? Was hätte besser laufen können? Was waren Ihre größten Herausforderungen und wie sind Sie mit diesen umgegangen? Was haben Sie gelernt? Hatten Sie Freude an Ihrem Lernprozess? Gehen Sie auch auf Ihre eingangs formulierten Erwartungen ein.

**Quellenverzeichnis**

Hier stellen Sie übersichtlich Ihre verwendeten Quellen dar.

Pygame Documentation - <https://www.pygame.org/docs/ref/pygame.html>

Stack Overflow – <https://stackoverflow.com/>

render\_text() Funktion in editor.py beinhaltet abgeanderte Zeilen von ChatGPT