Projekt DA2005 Programmeringsteknik, ST23

Henry Svedberg

August 20, 2023

Contents

1 Val av uppgift. Hur programmet används

Uppgiften som implementerats är ASCII Art Studio uppgift 2. Filen heter `ASCII_Art_Studio` och har tre klasser vid namn `"ASCII_art"`, `"SessionManager"` samt `"ASCII_UserInterface"` som behöver laddas in med import. Vidare behövs även modulerna PIL, Numpy, os, samt json. Börja således med att ladda in alla som exempelvis:

```
from PIL import Image, ImageEnhance
import numpy as np
import os
import json
from ASCII_Art_Studio import ASCII_art, SessionManager, ASCII_UserInterface
```

Förutsatt att ASCII_Art_Studio.py finns i det nuvarade work directory. Om huvudfilen körs direkt så skall det räcka med att bara köra "run file". Annars om användaren önskar att starta det själv från import så räcker det med att bara skriva: `ASCII_art_UserInterface()` för att direkt komma igång med själva programmet. Då dyker följande meddelande upp i konsolen:

```
Welcome to ASCII Art Studio!
AAS:
```

där `AAS:` innebär att den väntar på att användaren skall mata in kommandon. De kommandon stom stöds är de som specificeras i instruktionerna till uppgiften, det vill säga de kommandon för load, info, render, set, save och quit.

Så exempelvis bör man starta med kommandot `"load image filename (as alias)"` eller `"load image filename as alias"` där `filename` representerar namnet antingen på en fil i sitt nuvarande work directory eller dess sökväg på datorn och "as alias" är frivilligt om man vill döpa filen till något annat i programmet. Flera filer kan läsas in i en session.

Önskar man rendera en fil till ASCII-konst används kommandot `render` för att rendera det nuvarande objektet, eller `render img` där img är namnet antingen på filnamnet eller på dess alias. När en ny bild läses in sätts dess `target_width` som standardvärde till 50 och dess `target_height` justeras automatiskt till en passande höjd, och det är dessa som alltså används när den skrivs ut och motsvarar hur många rader eller kolumner med tecken det skall vara. Det går att själv sätta en egen höjd eller bredd med `set` kommandot, dock kommer den andra alltid att anpassas efter det. Det går att ändra ljusstyrka eller kontrast med `set img brightness <siffra>` och `set img contrast <siffra>`.

För att rendera direkt till konsolen används `render` för att rendera bild som är satt som current, eller `render img (to filename)` ifall man vill spara renderingen i en ny fil som något filnamn. Det går att spara sessionen med `save session <namn>` och därefter ladda den vid senare tillfälle med `load session <namn>`.för att avlsluta programmet skriver man `quit`. Det finns även ett help kommando som printar ut all giltiga kommandon med en kort beskrivning.

Det finns en .py fil vid namn "assertion" som innehåller enhetstester med unittest. Givet att den finns med filen `ASCII_Art_Studio` så skall det gå att bara köra "run file" för att få med alla funktioner då den importerar alla relevanta moduler . Däremot används filerna "grayscale.jpg" samt "slalom.jpg" för testerna: följakltigen måste de vara med i samma work directory och jag lade in dem i samma zip-fil. I assertion.py testas klasserna `ASCII_Art` och `SessionManager` med många olika tester som jag anser övergripande för syftet. Den testar inte varenda funktion som alla hjälpfunktioner, men de viktiga metoderna för resultaten som att ladda filer eller sessioner, ändra attribut, hålla koll på current med mer. Dock hade jag inte någon rikig lösning på att testa klassen UserInterface med unittest då den bygger på att en användare hela tiden har en input att skriva i konsolen. Men i och med att filen ändå testar metoderna som ASCII_UserInterface i sin

tur delegerar till givet giltigt kommando så fick jag testa UserInterface mycket själv istället för att förlita mig på unittester gällande just giltiga kommandon och ansåg att det ändå generellt täcker de relevanta aspekterna av testningen.

2 Programstruktur

Som tidigare nämnt finns det tre klasser. Dessa klasser har olika ansvarsområden. Som programmet är strukturerat är det meningen att en användare skall använda sig av klassen ASCII_UserInterface och alltså inte direkt av de andra klasserna. Däremot så grundar sig ASCII_UserInterface på de andra klasserna för att fungera.

Klassen `ASCII_art` utgår från enskilda bilder som egna objekt. Klassen har en load metod som används när ett objekt instansieras, varpå bildens relevanta egenskaper sparas som attribut; exempelvis som exempelvis `_height, _width, _contrast, _brightness, file_name`. Klassen har då metoder som `resize` för att ändra storlek, `set_img_enhance` för att ändra kontrast eller ljusstyrka, `render` för att kunna rendera antingen till konsol eller till ny fil, samt några funktioner för att konvertera pixlarna till ASCII-karaktärer. Alltså, sådant man vill kunna göra med de enskilda bilderna finns som metod i denna klass. Följakltigen fungerar det att enbart använda den ifall man skulle vilja hantera och rendera enskilda bilder. Exempelvis:

```
ascii_objekt = ASCII_art(filename)
ascii_objekt._resize(new_width=50)
ascii_objekt.set_img_enhance("contrast", 0.5)'
ascii_objekt.render()
```

Dock är inte programmet skrivit med det som syfte. exempelvis hanterar inte denna klass alla felmedelanden eller kontrollerar att all indata är giltig osv. Poängen var bara att visa att det går och hur klassen fungerar.

Klassen `SessionManager` utgör själva sessionen som ett eget objekt. Denna hanterar ingen extern användarinput utan utgår från att den input som kommer är giltig. En session instansieras med attributen `self.members` som är en tom lista som fylls på när bilder laddas samt `self._current` som ändras varje gång en bild laddas eller ändras. Klassen har dels några metoder som utgår från ASCII_art metoderna och som är snarlika till namnet, exempelvis `_load_image, _rener_img, _set_img_enhancet`. Dessa metoder utgår från de tidigare ASCII_art metoderna som nämndes ovan, men anpassat till sessionen. Exempelvis: ifall det finns två bilder laddade i sessionen och en användare har skrivit "render img" eller bara "render" och att detta kommando skickats till sessionen, kommer sessionen först att leta upp bilden med namnet "img" för att därefter använda sig av ASCII_klassens metod för render. Vidare har klassen några metoder som är särskilda för just sessionen som exempelvis `info` för att kunna hålla koll på vilka bilder som är laddade och dess attribut, samt metoder för att spara och ladda sessioner.

Klassen ASCII_UserInterface är användargränssnittet. Dess uppgift är att först kontrollera ifall användaren skriver ett giligt kommando som "load", "set", "render" osv, för att därefter delegera varje kommando vidare till en metod vars specifika syfte är att analysera det kommandot för att kontrollera att längden är korrekt eller att den har rätt nyckelord osv. Exempelvis om en användaren skriver "load image grayscale.jpg" så används methjoden `_handle_load_cmd` och `_handle_load_img_cmd` för att avgöra att det är ett giltigt kommando. Därefter används klassen `SessionManagers` metod för att ladda in en fil beroende på exempelvis om användare skrivit in alias eller inte. Eftersom `SessionManagers` metod för att ladda in en fil i sin tur tar avstamp i `ASCII_Art` klassen så finns det en hierarkiskt struktur i hur metoderna delegeras, men detta är inte ett direkt arv.

Tanken med att strukturera upp det i dessa klasser så här var i ett försök att göra det hyfsat modulärt. Vill man ändra på användergränsnittet och lägga till fler kommandon eller felmedellanden, eller för den delen göra den helt annorlunda, är det relativt enkelt då det inte påverkar de andra delarna. Likaså ifall man vill lägga till mer saker i sessionen eller ifall man vill ändra på hur render ska printas till konsolen är det också relativt enkelt att bara ändra på fåtal ställen utan att allt kollapsar.

3 Övriga reflektioner

Gällande effektivitet och datastrukturer försökte få det att bli som mest effektivt vid själva renderingen då jag skulle gissa att det är där som det blir som mest beräkningsintensivt. Modulen numpy användes för att göra om pixlarna till en np.array, vilket möjliggör vektoriserade operationer. Detta skall i teorin vara snabbare än for-loopar då det möjliggör att man kan utföra operationer på elementen direkt snarare än genom loopar. Jag använder dock explicit en np.vectorise() funktion vid ett tillfälle som jag är medveten om i praktiken är en for loop och inte en "riktig" vektoriserad operation. Men om inte annat gör det koden relativ simpel då det kan göras mycket med få rader kod. Däremot var det ett medvetet val att göra koden lite mer läsbar än att gå på ren effektivitet. Exempelvis de två första raderna i render-metoden är:

```
greyscale_array= self.__normalize(np.asarray(self._image))
 #creating an np.array and normalizing values
int_array = (np.rint(greyscale_array).astype(int))
```

där pixlarna i gråskala först konverteras till en array, och därefter skapar en ny array uifrån den. Sedan vid nästa kod som inte är med här skapas ytterligare en array. Det mest optimala hade nog varit att bara fortsätta metoderna i en lång kedja utan att spara det som två seperata arrayer, men det blir mer läsbart så.

Vad gäller hjälp från internet var det några få funktioner jag kunde implementera med hjälp av vad folk skrivit på StackOverflow. Jag använde även ChatGpt för några saker: främst för att hitta mindre syntaxrelaterade fel som jag inte såg själv i min kod. Den hjälpte till med några specifika funktioner också. När jag bad den om hjälp ställde jag frågorna i stil med "how do i do X in python". Alternativt att jag försökte förklara kontexten något och be den om hjälp om struktur: till exempel "my class is supposed to handle Y, how can i write a function that does Z" etc. I koden har jag kommenterar var jag fått hjälp av StackOverflow eller ChatGpt.