Fernfachhochschule Schweiz
Zürich | Basel | Bern | Brig

Mitglied der SUPSI

2.Semesterarbeit

BattleScribe – Die GitHub Analyse

Aufgabestellungen aus dem Modul EDS – Einführung in Data Science

Inhaltsverzeichnis

1. Abstract

Eine einfache Betrachtung eines GitHub Users und seiner Repositories in Bezug auf die Gewichtung in der Realität. Verwendet wird Python in der PyCharm IDE. Die Voraussetzungen werden beschrieben mit denen Daten von GitHub gewonnen werden können. Nach dem die benötigten Daten ausgearbeitet wurden können diese mit Gephi visualisiert werden und einfache Berechnungen getätigt werden. Die Werte beschreiben das gezeigte Netzwerk und seine Gegebenheiten.

2. Einleitung

BattleScribe [1] ist eine unabhängige Spielerweiterung zu den meisten TableTop-Spielen der Firma Games Workshop [2]. Darin sind Punktekosten für Einheiten enthalten die von Zeit zu Zeit durch Games Workshop publiziert werden. In dieser Arbeit werden die Repositories des GitHub [3] Benutzer BSData betrachtet. Näher eingegangen wird auf die Abbildung in der Realität, verhält sich das Interesse an den verschiedenen Repositories gleich wie zum dazugehörigen Spiel. Teilen sich die Interessen unter den Spielsystemen stark auf. Gibt es Gruppen, die sich in mehreren Systemen bewegen. Können Benutzer erkannt werden, die eine spezielle Rollen haben. Werden verwandte Spielsysteme auch von denselben Usern betreut.
Einfachheitshalber werden in den weiteren Abschnitten BattleScribe als BS, GamesWorkshop als GW, Repositories als Repos und GitHub als GH bezeichnet.

3. Material

Zur Verwendung kamen die IDE PyCharm [4] und Programmiersprache Python [5]. Die Daten wurden mit PyGithub [6] beschafft. Dazu wird ein GH-Account benötigt mit Access Token. Die Anleitung dazu befindet sich in den GH-Docs [3]. Das so generierte Tocken wird von OAuth [7] benötigt und berechtigt bis zu 5000 Requests pro Stunde. Für die Visualisierung der Daten wurde Gephi [8] verwendet.

Vorangehend sollten folgende Pakete installiert werden:

```
pip3 install networkx
pip3 install pygithub
pip3 install requests
```

4. Methoden

Grundsätzlich wurden zwei Frameworks verwendet. Zum einen PyGitHub [3] selbst und NetworkX [9] welches weiterführend nur noch NX genannt wird. GH stellt alle nötigen Methoden bereit, um via ihrem API an die gewünschten Daten zu gelangen. NX stellt die Methoden bereit, um die Daten in einem Graphen darzustellen und Berechnungen anzustellen. Jedoch werden hier die Berechnungen anschliessend in Gephi durchgeführt.

Access Token:

```
import requests
import json

username = 'username' # Your GitHub username
password = 'password' # Your GitHub password

# Note that credentials will be transmitted over a secure SSL connection
url = 'https://api.github.com/authorizations'
note = 'YOUR_APPLICATION'
post_data = {'scopes':['repo'],'note': note }

response = requests.post(
    url,
    auth = (username, password),
    data = json.dumps(post_data),
    )

print("API response:", response.text)
print()
print("Your OAuth token is", response.json()['token'])
```

Mit dem Token kann nun weitergearbeitet werden. Dieses Token sollten nicht verteilt werden. Im Zweifelsfall kann es auf GH unter Developer Settings gelöscht und direkt neu erstellt werden.

GitHub:

```
client = Github("ACCESS_TOKEN", per_page=100)
```

Mit dem Access Token der persönlichen GH Applikation werden die Anfragen authentifiziert. So können über die Instanz bis zu 5000 Anfragen pro Stunde gestellt werden.

```
MAINUSER = 'BSData'
user = client.get_user(MAINUSER)
```

Um den User BSData zu untersuchen wird eine Variable des Users angelegt.

```
repos = [s for s in user.get_repos()]
```

Alle userbezogenen Repos werden in eine Liste gezogen.

Mit diesen Daten werden nun die weiteren Abfragen gestaltet. Hier wird das Interesse auf die Stargazer jedes einzelnen Repos des Users BSData gelegt.

```
repo_stargazers = [s for s in r.get_stargazers()]
```

Pro Repo wird eine Liste der Stargazer erstellt und daraus Knoten und Kanten dem Graphen hinzugefügt.

```
for follower in sg.get_followers():
    if follower.login + '(user)' in g:
        g.add_edge(follower.login + '(user)', sg.login + '(user)', type='follows')
```

Um die Vernetzung unter den Stargazer beurteilen zu können werden die Follower ausgelesen und mit der bestehenden Stargazerlister verglichen. Besteht eine Übereinstimmung wird eine Kante hinzugefügt.

NetworkX:

Die ausgelesenen Daten werden Schritt für Schritt in den erstellten Graphen hinzugefügt.

```
g = nx.DiGraph
```

Der erstellte Graph wird anschliessend mit Knoten und Kanten abgefüllt.

```
g.add_node(user.login + '(user)', type='user')
```

Knoten mit Bezeichnung des Users und typ user hinzufügen.

```
g.add_edge(sg.login + '(user)', r.name + '(repo)', type='gazes')
```

Kante mit der Richtung User → Repo hinzufügen.

```
nx.write_gexf(g, "BSData_Stargezers.gexf")
```

Den Graphen exportieren als GEXF [10] zur Weiterverarbeitung in Gephi.

Gephi:
Die Exportierte GEXF-Datei kann ins Gephi importiert werden. Die Visualisierung muss erst noch einiges an Bearbeitung in Anspruch nehmen. Berechnungen könnten jedoch in diesem Zustand bereit gemacht werden.

Gephi stellt diverse Algorythmen zur Verfügung, um Ordnung in die Rohdarstellung zu bringen.

Verwendet wurden in meinem Beispiel eine Mischung aus Yifan Hu und Force Atlas2. Es werden auch einfache Funktionen angeboten wie Rotate (Rotieren), Noverlap (Keine Überlappung), Expansion (Vergrössern). Die Funktionen und Algorithmen können nach Belieben kombiniert werden und sind je nach Graph unterschiwdlich sinnvoll. Hier muss selbst die richtige Darstellung herausgefunden werden.

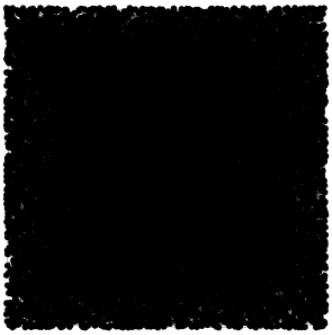

Abbildung 1: Rohdaten in Gephi

Berechnungen:
Degree
Der Wert gibt an wie stark der Knoten eingebunden ist mit Ein- und ausgehenden Kanten.

Out-Degree
Wert für die Ausgehenden Kanten des Knoten.

In-Degree
Wert für die Eingehenden Kanten des Knoten.

Closeness Centrality
Berechnet die kürzesten Verbindungen zu allen anderen Punkten. Der Knoten mit dem höchsten Wert zeigt die kürzesten Verbindungen und gilt als am zentralsten im Netzwerk.

Betweenness Centrality
Berechnet wie gut die Verbindungen des Knotens untereinander verbunden sind. Kann als Verbindungsknoten zwischen untereinander gut verbundenen Netzwerken angesehen werden.

Cluster Coefficient
Er beschreibt, wie stark Nachbarknoten untereinander vernetzt sind.

Eigenvector Centrality
Berechnet wie gut ein Knoten mit den Teilen des Netzwerks mit der besten Konnektivität verbunden ist

5. Resultate

Im bearbeiteten Graphen sind klar die Repos in Gelb und die dazugehörigen User in Rot ersichtlich. Markante User wurden grün gefärbt. Das Repo der Webapplikation wurde blau eingefärbt.

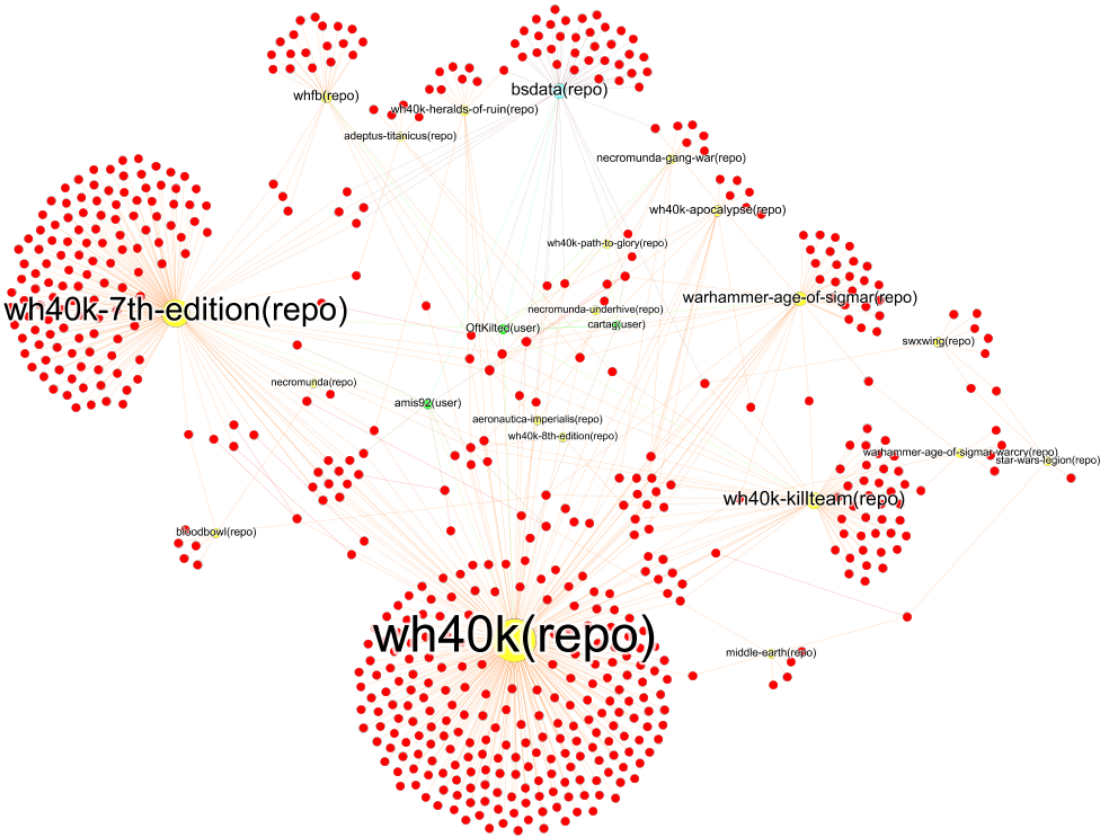

Abbildung 2: Graph Degree mit Beschriftung

Die Daten wurden soweit bearbeitet, dass die Repos der Spielsysteme von GW übrig sind. Klar ersichtlich welchen Systemen am meisten Interesse geschenkt wird. Wh40k ist mit Abstand das grösste Produkt von GW und erhält auch in BS die grösste Aufmerksamkeit. Warhammer Age of Sigmar und Necromunda folgen etwas abgeschlagen dahinter.
Die Gruppen sind klar getrennt und anders als erwartet zeigen nur wenige Interesse an mehr als einem System. Klar zu sehen durch den Abstand der Repos sind die Zwischengruppen, die sich für zwei oder mehr Repos interessieren
Drei User wurden für das Netzwerk als Bindeglieder zwischen den grössten Netzwerken erkannt und grün markiert. Es spielt keine Rolle ob die Spielsysteme verwandt sind oder nicht es können nicht mehr oder weniger User in den Zwischengruppen festgestellt werden.

Die Zahlen sagen dasselbe:

Das Netzwerk ist nicht sehr dicht, es wurde ein durchschnittlicher Grad von 1.212 berechnet.
Diese Darstellung zeigt die Zentralsten User:

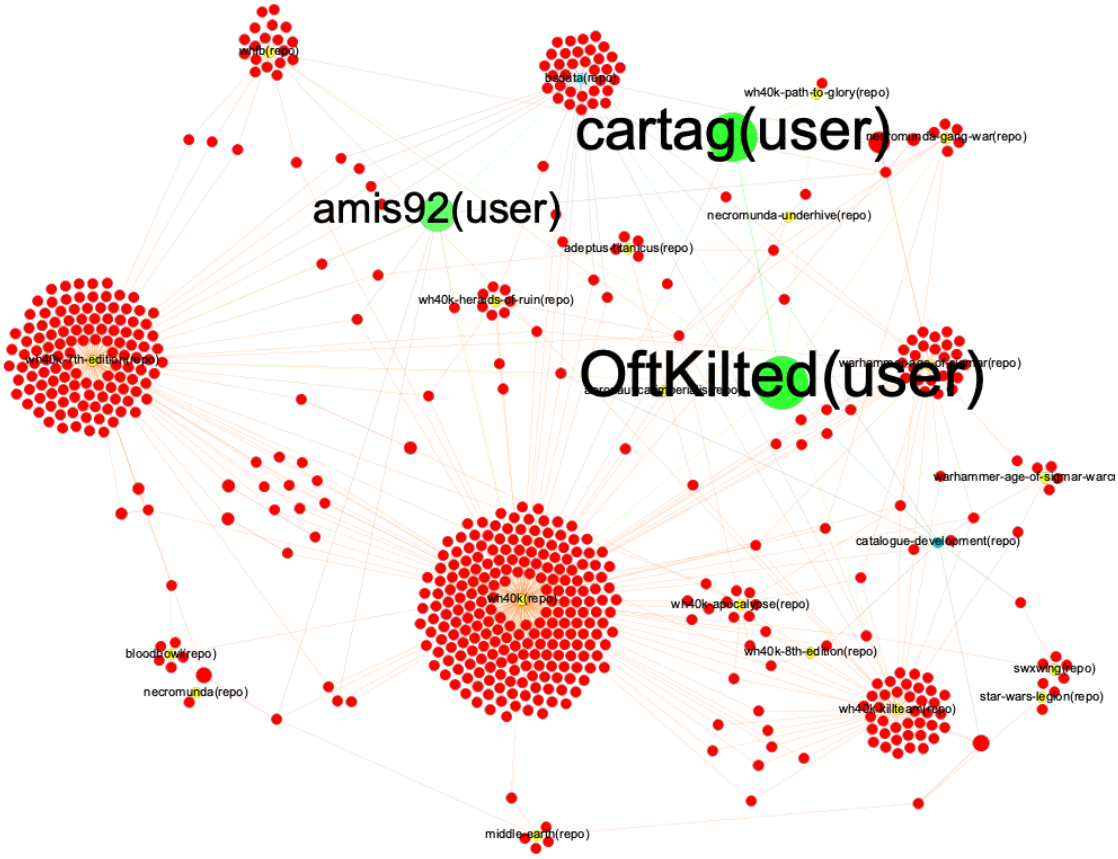

Abbildung 3: Graph Betweenness Centrality komplett

amis92, cartag und OftKilted sind mit nahezu jedem Repo oder User die sehr nahe mit grossen Netzwerken verbunden.
Mit einer Betweenness Centrality von 18 für OftKilted, 17 für cartag und 13 für amis92 sind sie mit Abstand die Zentralsten User

User die in mindestens zwei der grössten Repos verbunden. In der Mitte des Graphen befinden sich User, die mit mindestens drei Repos verbunden sind. Darunter sind ebenfalls die bereits genannten zentralsten User.

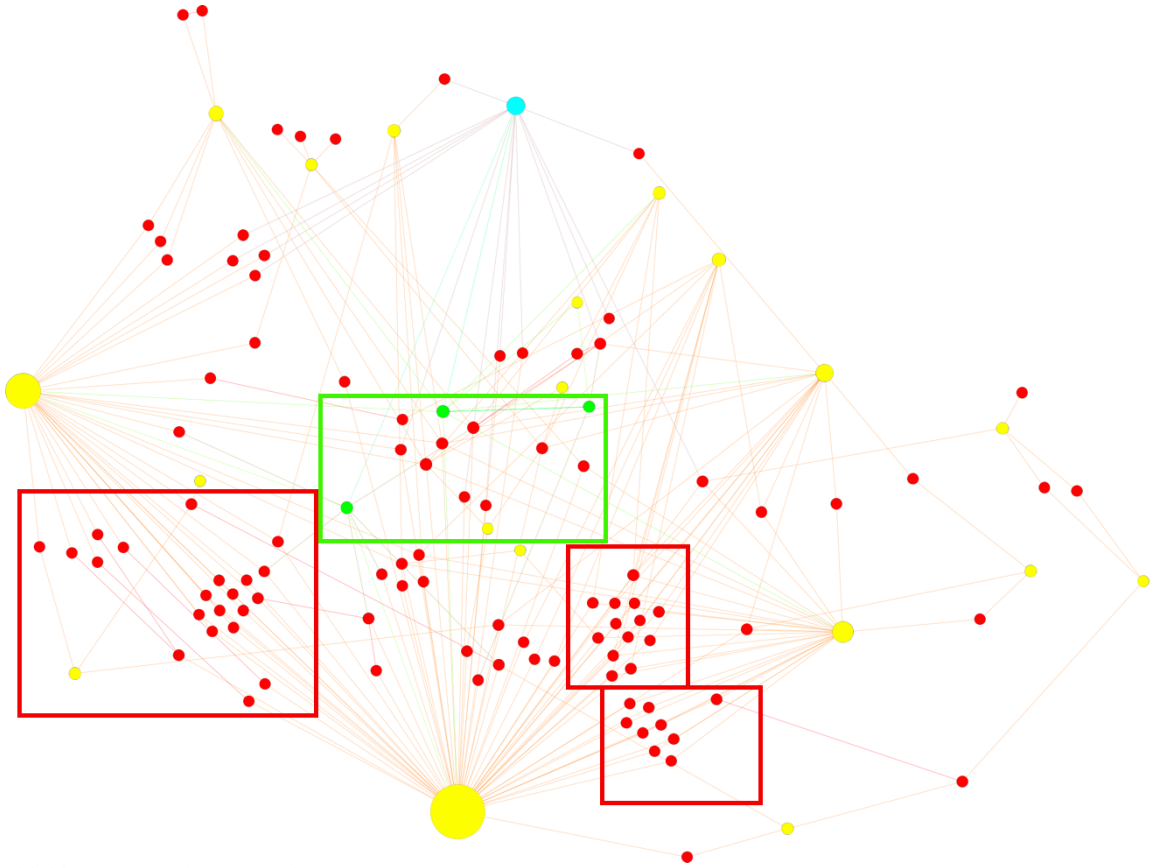

Abbildung 4: Graph Degree ohne Degree 1

6. Diskussion

Die Daten könnten auch durch andere Varianten bereitgestellt werden. Indem eine oder mehrere Knoten und Kanten Listen in CSV exportiert und einzeln oder modular hinzugefügt werden. Werden unterschiedliche Datensätze benötigt ist dies die bessere Variante. In diesem Beispiel hat ein Datensatz direkt als GEXF gereicht, jedoch lässt es einen kleineren Spielraum offen für die Nachbearbeitung.
Die Limitation der Requests auf 5000 pro Stunde kann während der Erarbeitung zu einem Problem werden. Sobald die gewünschten Daten zusammengefasst sind kann die Abfrage gezielter gestellt werden. Für dieses Beispiel reicht die Beschränkung ohne Probleme.

7. Quellen

[1]	BattleScribe https://battlescribe.net/?tab=downloads BattleScribe GitHub https://github.com/BSData
[2]	Games Workshop https://www.games-workshop.com/en-WW/Home
[3]	GitHub https://github.com Docs https://docs.github.com/en
[4]	PyCharm https://www.jetbrains.com/de-de/pycharm/
[5]	Python https://www.python.org
[6]	PyGithub https://github.com/PyGithub/PyGithub
[7]	OAuth https://oauth.net
[8]	Gephi https://gephi.org
[9]	NetworkX https://networkx.github.io/documentation/stable/index.html
[10]	GEXF (Graph Exchange XML Format) https://gephi.org/gexf/format/

8. Anhang

- Beispielcode

```
import networkx as nx
from github import Github

# create a Github instance using username and password or an access token:
# client = Github("user", "password", per_page=100)
client = Github("access_token", per_page=100)

# Define User of interest
MAINUSER = 'BSData'
user = client.get_user(MAINUSER)
repos = [s for s in user.get_repos()]

# create a DiGraph
g = nx.DiGraph()
g.add_node(user.login + '(user)', type='user')

limit = (0, 0)
for r in repos:
    try:
        limit = client.rate_limiting
        print("Rate limit remaining", limit)

        g.add_node(r.name + '(repo)', type='repo')
        repo_stargazers = [s for s in r.get_stargazers()]
        print("Number of stargazers", len(repo_stargazers), r.name)

        for sg in repo_stargazers:
            g.add_node(sg.login + '(user)', type='user')
            g.add_edge(sg.login + '(user)', r.name + '(repo)', type='gazes')

        for i, sg in enumerate(repo_stargazers):
            try:
                for follower in sg.get_followers():
                    if follower.login + '(user)' in g:
                        g.add_edge(follower.login + '(user)', sg.login + '(user)', type='follows')
            except Exception as e:
                print(e)

    except Exception as e:
        print(e)

nx.write_gexf(g, "BSData_Stargezers.gexf")
```