Experiment 1: Hier stelle ich die verschiedenen Diagramme vor, die ich in meiner Arbeit mit Methoden aus der Klasse StatsVis verwendet habe. In meinen Messungen habe ich jedes Experiment 5 mal ausgeführt. Wenn der Exp1.py Code ausgeführt wird, kann es passieren, dass dort Daten überschrieben werden, so dass dort sowohl Fehler als auch andere Daten angezeigt werden, bitte beachten Sie dies. Die Excel Dateien der Messungen enthalten immer 5 Spalten: Generation , Mean Fitness , STD , Min Fitness , Max Fitness plot_from_excel_global_max wird verwendet, da die NEAT-Bibliothek standardmäßig nur die beste Fitness pro Generation speichert und nicht die globale Fitness, die Funktion sorgt dafür, dass die globale Fitness aus der vorherigen abgeleitet und dann geplottet wird. Anschließend wird die Generation für die Tabellen berechnet. Die Zeiten werden in der Datei Exp1Times.txt gespeichert. Am Ende wird ein Graph erstellt, um die Standardabweichung in der Speziation und Normal zu vergleichen.

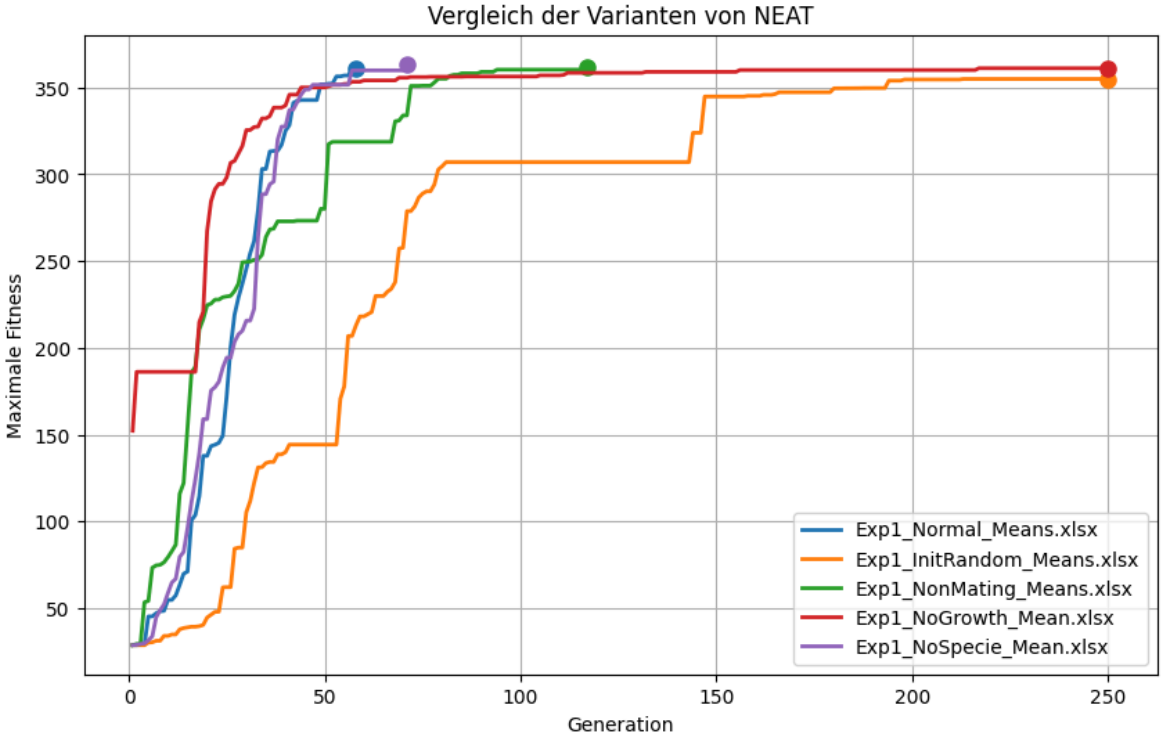
In [3]:
```
import StatsVis
import os
import pandas as pd
import matplotlib.pyplot as plt

local_dir = os.getcwd()
base_path = os.path.join(local_dir, "Excel Results")
```

In [4]:
```
files = [
    f"Exp1_Normal_{i}_stats.xlsx" for i in range(1, 6)
] + [
    f"Exp1_InitRandom_{i}_stats.xlsx" for i in range(1, 6)
] + [
    f"Exp1_NonMating_{i}_stats.xlsx" for i in range(1, 6)
] + [
    f"Exp1_NoGrowth_{i}_stats.xlsx" for i in range(1, 6)
] + [
    f"Exp1_NoSpecie_{i}_stats.xlsx" for i in range(1, 6)
]
filesNormal = [f for f in files if "Normal" in f]
filesInitRandom = [f for f in files if "InitRandom" in f]
filesNonMating = [f for f in files if "NonMating" in f]
filesNoGrowth = [f for f in files if "NoGrowth" in f]
filesNoSpecie = [f for f in files if "NoSpecie" in f]
```

In [5]:
```
file_name_averages_normal = "Exp1_Normal_Means.xlsx"
file_name_averages_init_random = "Exp1_InitRandom_Means.xlsx"
file_name_averages_non_mating = "Exp1_NonMating_Means.xlsx"
file_name_averages_no_growth = "Exp1_NoGrowth_Mean.xlsx"
file_name_averages_no_specie = "Exp1_NoSpecie_Mean.xlsx"

StatsVis.create_average_excel(base_path, filesNormal, file_name_averages_normal,
StatsVis.create_average_excel(base_path, filesInitRandom, file_name_averages_ini
StatsVis.create_average_excel(base_path, filesNonMating, file_name_averages_non_
StatsVis.create_average_excel(base_path, filesNoGrowth, file_name_averages_no_gr
StatsVis.create_average_excel(base_path, filesNoSpecie, file_name_averages_no_sp
```

```
Die Durchschnittswerte wurden in d:\Python Workspace\BA Neat\Excel Results\Exp1_N
ormal_Means.xlsx gespeichert.
Die Durchschnittswerte wurden in d:\Python Workspace\BA Neat\Excel Results\Exp1_I
nitRandom_Means.xlsx gespeichert.
Die Durchschnittswerte wurden in d:\Python Workspace\BA Neat\Excel Results\Exp1_N
onMating_Means.xlsx gespeichert.
Die Durchschnittswerte wurden in d:\Python Workspace\BA Neat\Excel Results\Exp1_N
oGrowth_Mean.xlsx gespeichert.
Die Durchschnittswerte wurden in d:\Python Workspace\BA Neat\Excel Results\Exp1_N
oSpecie_Mean.xlsx gespeichert.
```

In [6]:
```
filesMeans = [
    "Exp1_Normal_Means.xlsx",
    "Exp1_InitRandom_Means.xlsx",
    "Exp1_NonMating_Means.xlsx",
    "Exp1_NoGrowth_Mean.xlsx",
    "Exp1_NoSpecie_Mean.xlsx"
]

column_indices = [0] * 5
StatsVis.plot_from_excel_global_max(base_path, filesMeans, column_indices, "Maxi
```



In [7]:
```
from collections import defaultdict

# Initialisiere ein Dictionary, um die Generationsdaten für jede Version zu samm
version_generations = defaultdict(list)

# Durchlaufe jede Datei und zähle die Generationen
for file in files:
    generations = StatsVis.count_generations(os.path.join(base_path, file))
    # Extrahiere den Namen ohne Pfad und Endung für die Formatierung
    file_name = os.path.splitext(os.path.basename(file))[0]
    # Extrahiere den Versionstyp aus dem Dateinamen
    version = file_name.split('_')[1]

    # Speichere die Anzahl der Generationen für die jeweilige Version
    version_generations[version].append(generations)
```

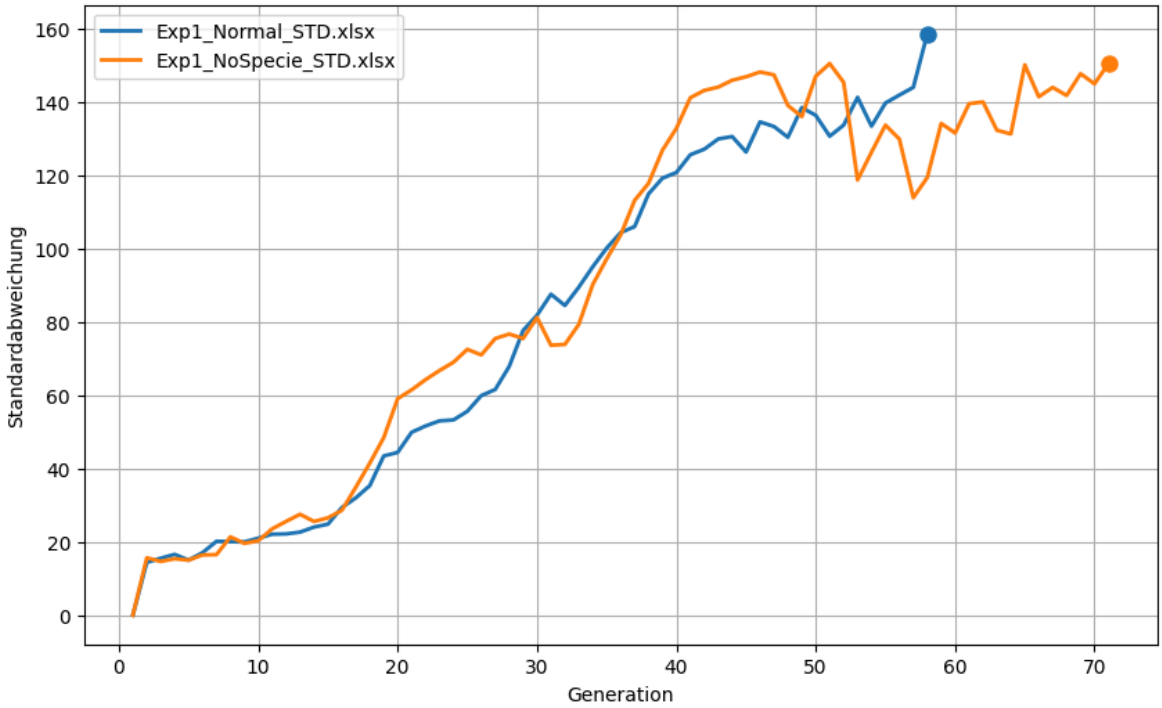
```
    print(f"{file_name} hat {generations} Generationen gebraucht")

# Durchschnitt der Generationen pro Version berechnen und ausgeben
for version, gen_counts in version_generations.items():
    avg_generations = sum(gen_counts) / len(gen_counts)
    print(f"{version} hat im Durchschnitt {avg_generations:.2f} Generationen geb
```

```
Exp1_Normal_1_stats hat 49 Generationen gebraucht
Exp1_Normal_2_stats hat 44 Generationen gebraucht
Exp1_Normal_3_stats hat 57 Generationen gebraucht
Exp1_Normal_4_stats hat 51 Generationen gebraucht
Exp1_Normal_5_stats hat 58 Generationen gebraucht
Exp1_InitRandom_1_stats hat 81 Generationen gebraucht
Exp1_InitRandom_2_stats hat 213 Generationen gebraucht
Exp1_InitRandom_3_stats hat 41 Generationen gebraucht
Exp1_InitRandom_4_stats hat 220 Generationen gebraucht
Exp1_InitRandom_5_stats hat 250 Generationen gebraucht
Exp1_NonMating_1_stats hat 53 Generationen gebraucht
Exp1_NonMating_2_stats hat 43 Generationen gebraucht
Exp1_NonMating_3_stats hat 38 Generationen gebraucht
Exp1_NonMating_4_stats hat 94 Generationen gebraucht
Exp1_NonMating_5_stats hat 117 Generationen gebraucht
Exp1_NoGrowth_1_stats hat 86 Generationen gebraucht
Exp1_NoGrowth_2_stats hat 156 Generationen gebraucht
Exp1_NoGrowth_3_stats hat 217 Generationen gebraucht
Exp1_NoGrowth_4_stats hat 112 Generationen gebraucht
Exp1_NoGrowth_5_stats hat 250 Generationen gebraucht
Exp1_NoSpecie_1_stats hat 44 Generationen gebraucht
Exp1_NoSpecie_2_stats hat 30 Generationen gebraucht
Exp1_NoSpecie_3_stats hat 47 Generationen gebraucht
Exp1_NoSpecie_4_stats hat 52 Generationen gebraucht
Exp1_NoSpecie_5_stats hat 71 Generationen gebraucht
Normal hat im Durchschnitt 51.80 Generationen gebraucht
InitRandom hat im Durchschnitt 161.00 Generationen gebraucht
NonMating hat im Durchschnitt 69.00 Generationen gebraucht
NoGrowth hat im Durchschnitt 164.20 Generationen gebraucht
NoSpecie hat im Durchschnitt 48.80 Generationen gebraucht
```

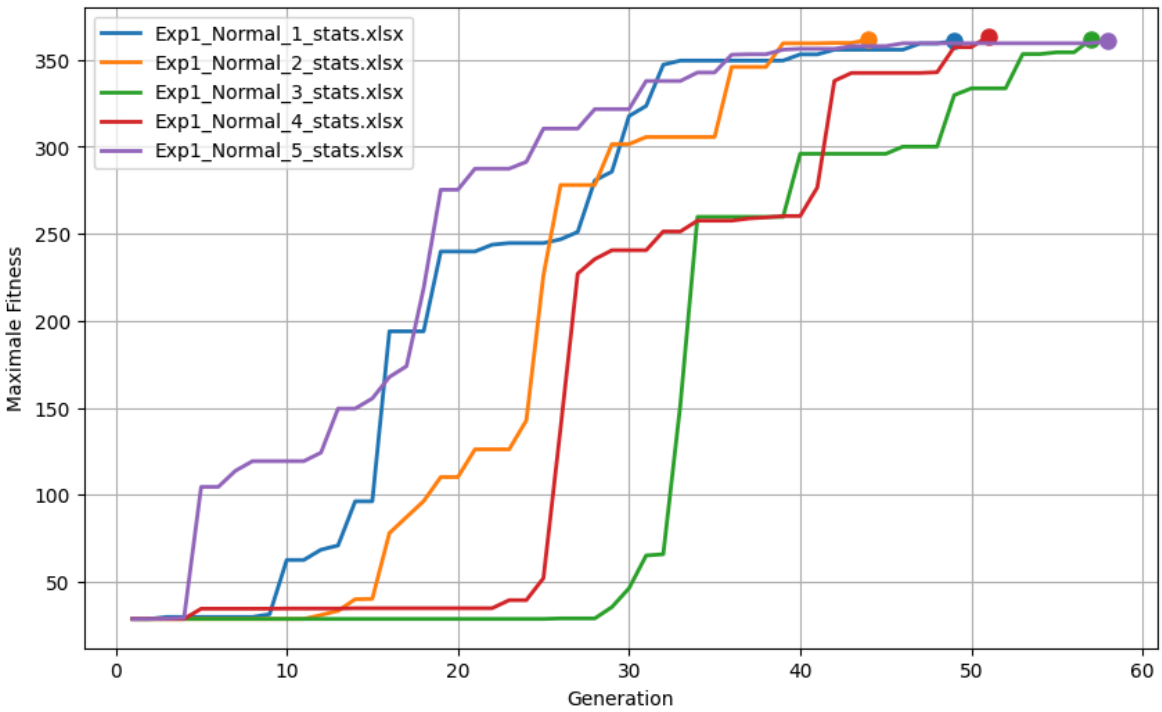
In [8]:

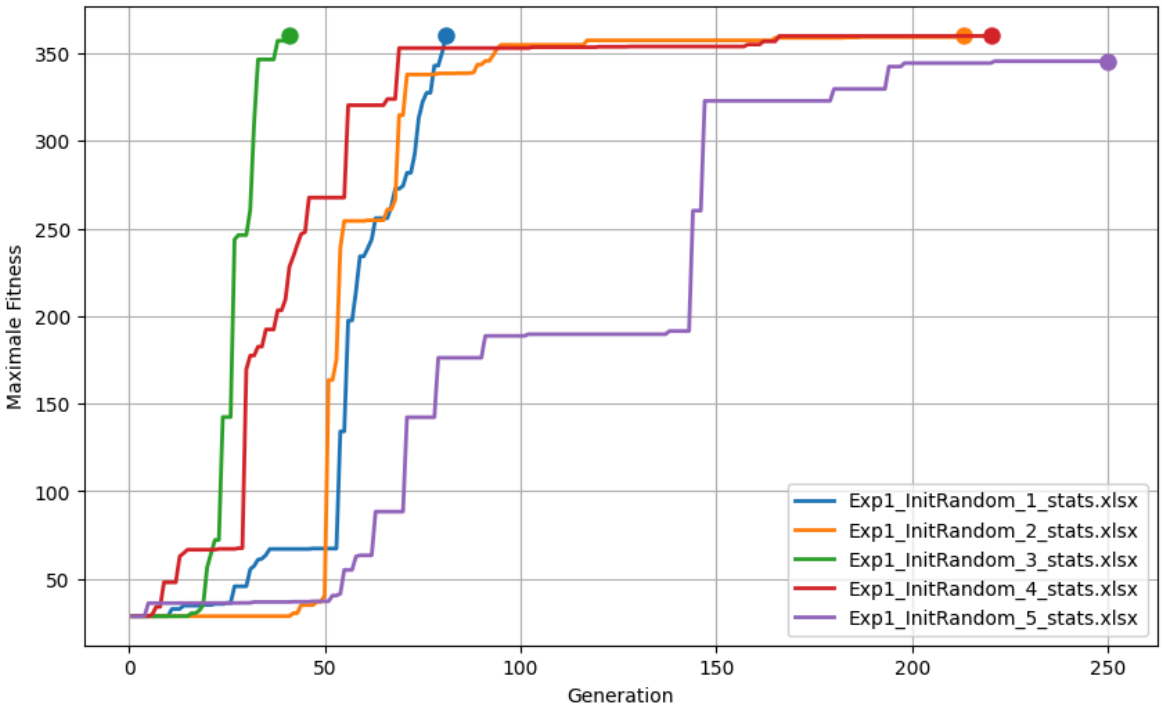
```
# Definieren der Zieldateien für die Durchschnittswerte
file_name_averages_normal = "Exp1_Normal_STD.xlsx"
file_name_averages_no_specie = "Exp1_NoSpecie_STD.xlsx"

# Berechnen und Speichern der Durchschnittswerte für jede Version
StatsVis.create_average_excel(base_path, filesNormal, file_name_averages_normal,
StatsVis.create_average_excel(base_path, filesNoSpecie, file_name_averages_no_sp

# Definieren der Dateien und Spaltenindizes für den Plot
filesSTD = [
    "Exp1_Normal_STD.xlsx",
    "Exp1_NoSpecie_STD.xlsx"
]

column_indices = [0, 0]

# Plotten der Standardabweichungen
StatsVis.plot_from_excel(base_path, filesSTD, column_indices, "Standardabweichun
```

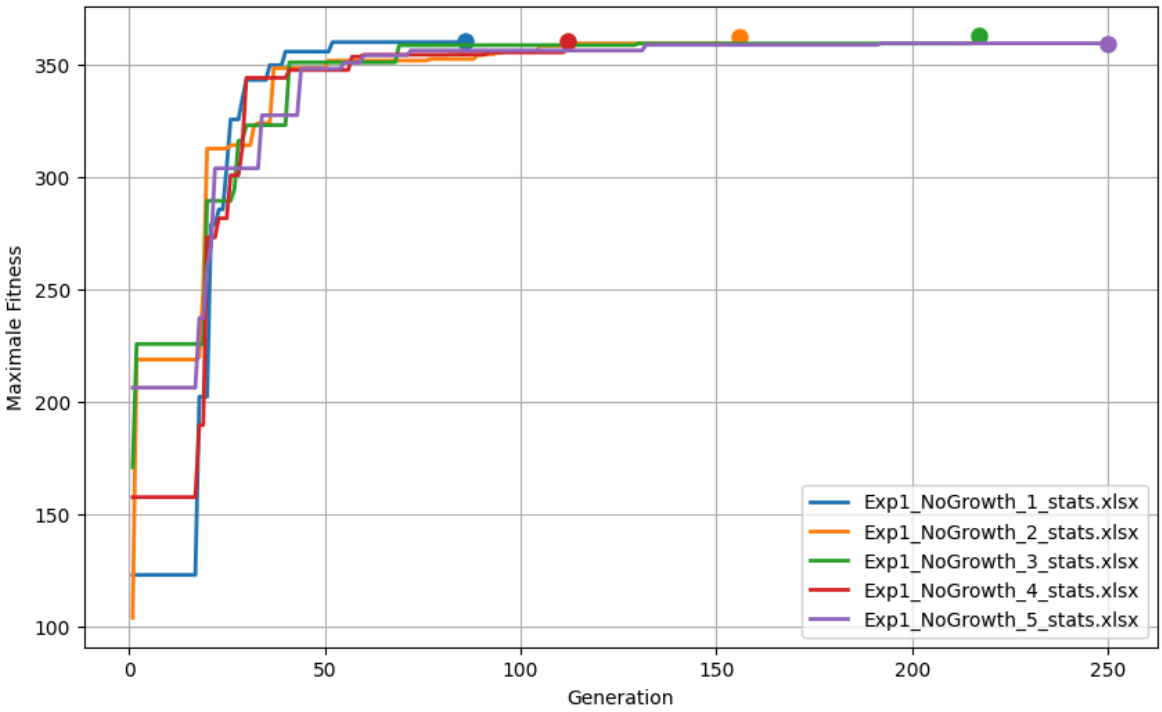
```
Die Durchschnittswerte wurden in d:\Python Workspace\BA Neat\Excel Results\Exp1_N
ormal_STD.xlsx gespeichert.
Die Durchschnittswerte wurden in d:\Python Workspace\BA Neat\Excel Results\Exp1_N
oSpecie_STD.xlsx gespeichert.
```



In [9]:
```
column_indices = [4]*5
StatsVis.plot_from_excel_global_max(base_path, filesNormal, column_indices, "Max
StatsVis.plot_from_excel_global_max(base_path, filesInitRandom, column_indices,
#StatsVis.plot_from_excel_global_max(base_path, filesNonMating, column_indices,
StatsVis.plot_from_excel_global_max(base_path, filesNoGrowth, column_indices, "M
#StatsVis.plot_from_excel_global_max(base_path, filesNoSpecie, column_indices, "
```





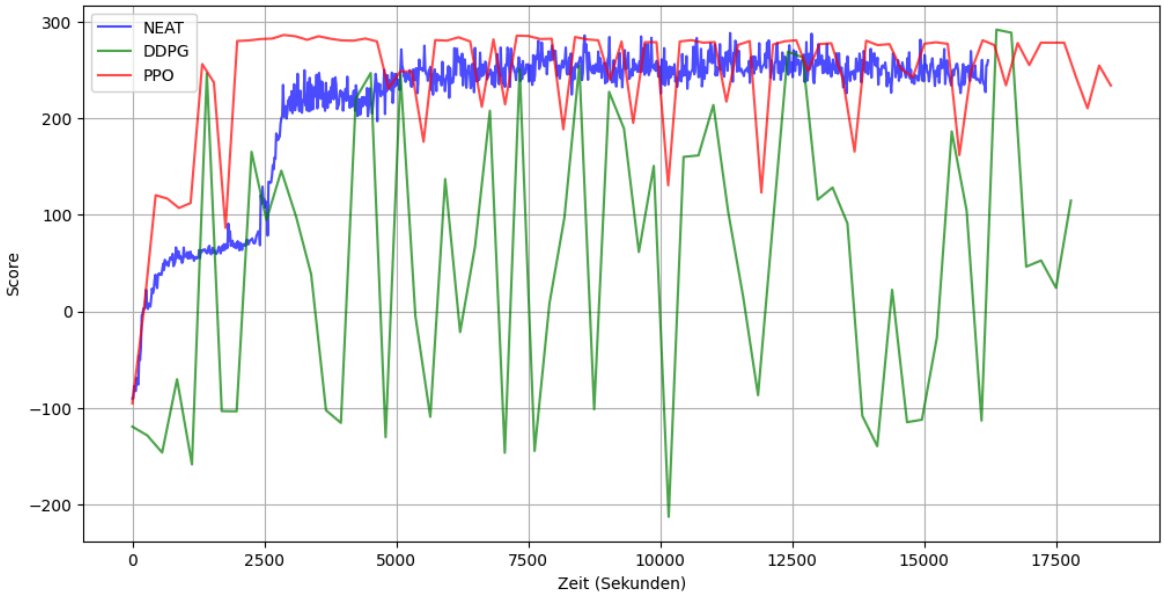


Experiment 2: Ab hier werden die die Daten aus den 2 Experiment aufbereitet. Es werden alle Algorithmen anhand von Generationen vergliechen falls der Algorithmus keine Generationen hat werden die Schritte zeitlich ca. auf Generationen skaliert (DDPG: 6.400 Schritte ca. pro Generation)

In [11]:
```
import matplotlib.pyplot as plt
import numpy as np

# Zeiten und Daten
time_neat = 16208.28
time_ddpg = 17778.81
time_ppo = 18534.61
```

```
scores_DDPG = [
    -119.5159594829634, -128.61174747674428, -146.35191924293542, -70.3384885235
    246.60797118102784, -103.38276662786018, -103.62408372890961, 165.3651284791
    145.8177538887523, 97.76832914801378, 38.61056149754819, -102.54909396802759
    220.68992405761304, 247.0662675543298, -130.38023889984837, 245.412384760267
    -109.30629008478559, 137.14871412607687, -21.443773522977118, 67.01861013104
    -146.57793318606372, 251.1553075462882, -144.73655387719145, 8.4994806444587
    254.75127295177532, -101.5250743051208, 227.61617688222887, 189.446841340644
    150.80449872782597, -213.0149073216847, 160.1706231984476, 161.6802766774246
    102.75406838282677, 15.091489807598014, -86.86320216941682, 91.7026679388885
    263.2991738667665, 115.75430388748627, 128.34407898270268, 91.2244012218865,
    -139.77335685184215, 22.30191830459005, -114.9197858654264, -112.33588326750
    186.52572917507908, 104.56332832612847, -113.23659556273681, 292.06232731940
    46.344225264969154, 52.75512561935027, 24.34496237894877, 114.82573785473964
]

scores_PPO = [
    -95.38498874095698, 2.6285446312608793, 120.40950165131123, 116.918142756556
    107.14913780393915, 112.26974076427477, 256.3612170451032, 237.2765286513655
    86.60066800331586, 280.2838603312337, 280.8440511682186, 282.30812519304317,
    282.772538611699, 286.53251546254046, 285.1972824926032, 281.50600343735266,
    285.2440910866718, 282.6450860700701, 280.8598381090261, 280.615747256517,
    282.862328281928, 279.8422673260378, 231.1227824003285, 248.8169498872875,
    248.6251209953034, 175.8769105132239, 281.21932727242637, 280.62779271921926
    284.1095541667453, 279.8570725929268, 212.10596294236788, 282.04104341011947
    214.54821787337588, 285.77767407120217, 285.475389636491, 282.3781447027076,
    282.62062853406167, 188.74694634740658, 284.3888317339061, 282.0457656768735
    280.9527015644499, 240.04365097957188, 279.7480268997213, 195.4056271860979,
    279.0214503319736, 279.049635245982, 130.7042071226899, 279.76272792162916,
    281.1619813193581, 278.5145150906577, 279.1088796700777, 217.41856065105375,
    276.2439128659953, 280.2089292895818, 123.20521267886784, 276.62217997208626
    279.97800071298195, 281.02023161340463, 249.23366261037822, 277.061770882258
    277.9510566970715, 240.5151028089659, 165.62874131698263, 280.58901427256416
    275.9875878354045, 277.0950306339843, 251.23140936522887, 242.6757564383669,
    277.48739626574053, 278.93127041159596, 277.36093474278744, 162.038138908611
    248.7284550091215, 281.05318235130665, 276.0382669196599, 234.3702326593662,
    278.11982337274884, 255.34120541775255, 278.55289393135297, 278.542776148848
    278.4569986775814, 243.26323821864716, 210.50903692303388, 254.8265015015059
    234.15480744522392
]


# NEAT-Scores aus Excel-Datei laden
excel_file = "D:/Python Workspace/BA NEAT/Excel Results/Exp2_NeatRun1_stats.xlsx
df = pd.read_excel(excel_file, engine='openpyxl')
neat_scores = df.iloc[:, 4].values  # 5. Spalte

# Zeiten für die Algorithmen (in Sekunden)
time_neat = 16208.28
time_ddpg = 12865.62 + 4913.19
time_ppo = 11935.53 + 8999.08 - (40 * 60)  # 40 Minuten abgezogen

# Zeit pro Eintrag berechnen
x_neat = np.linspace(0, time_neat, len(neat_scores))  # Zeitachse für NEAT
x_ddpg = np.linspace(0, time_ddpg, len(scores_DDPG))  # Zeitachse für DDPG
x_ppo = np.linspace(0, time_ppo, len(scores_PPO))    # Zeitachse für PPO

# Plotten
plt.figure(figsize=(12, 6))
```

```
plt.plot(x_neat, neat_scores, label="NEAT", color='blue', alpha=0.7)
plt.plot(x_ddpg, scores_DDPG, label="DDPG", color='green', alpha=0.7)
plt.plot(x_ppo, scores_PPO, label="PPO", color='red', alpha=0.7)

# Details für den Plot
plt.xlabel("Zeit (Sekunden)")
plt.ylabel("Score")
plt.legend()
plt.grid()
plt.show()
```



In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np

# Zeiten und Daten
time_neat = 16208.28
time_ddpg = 17778.81
time_ppo = 18534.61

scores_DDPG = [
    -119.5159594829634, -128.61174747674428, -146.35191924293542, -70.3384885235
    246.60797118102784, -103.38276662786018, -103.62408372890961, 165.3651284791
    145.8177538887523, 97.76832914801378, 38.61056149754819, -102.54909396802759
    220.68992405761304, 247.0662675543298, -130.38023889984837, 245.412384760267
    -109.30629008478559, 137.14871412607687, -21.443773522977118, 67.01861013104
    -146.57793318606372, 251.1553075462882, -144.73655387719145, 8.4994806444587
    254.75127295177532, -101.5250743051208, 227.61617688222887, 189.446841340644
    150.80449872782597, -213.0149073216847, 160.1706231984476, 161.6802766774246
    102.75406838282677, 15.091489807598014, -86.86320216941682, 91.7026679388885
    263.2991738667665, 115.75430388748627, 128.34407898270268, 91.2244012218865,
    -139.77335685184215, 22.30191830459005, -114.9197858654264, -112.33588326750
    186.52572917507908, 104.56332832612847, -113.23659556273681, 292.06232731940
    46.344225264969154, 52.75512561935027, 24.34496237894877, 114.82573785473964
]

scores_PPO = [
    -95.38498874095698, 2.6285446312608793, 120.40950165131123, 116.918142756556
    107.14913780393915, 112.26974076427477, 256.3612170451032, 237.2765286513655
    86.60066800331586, 280.2838603312337, 280.8440511682186, 282.30812519304317,
    282.772538611699, 286.53251546254046, 285.1972824926032, 281.50600343735266,
    285.2440910866718, 282.6450860700701, 280.8598381090261, 280.615747256517,
```

```
    282.862328281928, 279.8422673260378, 231.1227824003285, 248.8169498872875,
    248.6251209953034, 175.8769105132239, 281.21932727242637, 280.62779271921926
    284.1095541667453, 279.8570725929268, 212.10596294236788, 282.04104341011947
    214.54821787337588, 285.77767407120217, 285.475389636491, 282.3781447027076,
    282.62062853406167, 188.74694634740658, 284.3888317339061, 282.0457656768735
    280.9527015644499, 240.04365097957188, 279.7480268997213, 195.4056271860979,
    279.0214503319736, 279.049635245982, 130.7042071226899, 279.76272792162916,
    281.1619813193581, 278.5145150906577, 279.1088796700777, 217.41856065105375,
    276.2439128659953, 280.2089292895818, 123.20521267886784, 276.62217997208626
    279.97800071298195, 281.02023161340463, 249.23366261037822, 277.061770882258
    277.9510566970715, 240.5151028089659, 165.62874131698263, 280.58901427256416
    275.9875878354045, 277.0950306339843, 251.23140936522887, 242.6757564383669,
    277.48739626574053, 278.93127041159596, 277.36093474278744, 162.038138908611
    248.7284550091215, 281.05318235130665, 276.0382669196599, 234.3702326593662,
    278.11982337274884, 255.34120541775255, 278.55289393135297, 278.542776148848
    278.4569986775814, 243.26323821864716, 210.50903692303388, 254.8265015015059
    234.15480744522392
]


# NEAT-Scores aus Excel-Datei laden
excel_file = "D:/Python Workspace/BA NEAT/Excel Results/Exp2_NeatRun1_stats.xlsx
df = pd.read_excel(excel_file, engine='openpyxl')
neat_scores = df.iloc[:, 4].values  # 5. Spalte

# Funktion zum Glätten der Scores
def smooth_scores(scores):
    smoothed = []
    current_max = float('-inf')
    for score in scores:
        current_max = max(current_max, score)
        smoothed.append(current_max)
    return smoothed

# Glätten der Scores für alle Algorithmen
neat_scores_smoothed = smooth_scores(neat_scores)
scores_DDPG_smoothed = smooth_scores(scores_DDPG)
scores_PPO_smoothed = smooth_scores(scores_PPO)


# Zeiten für die Algorithmen (in Sekunden)
time_neat = 16208.28
time_ddpg = 12865.62 + 4913.19
time_ppo = 11935.53 + 8999.08 - (40 * 60)  # 40 Minuten abgezogen

# Zeit pro Eintrag berechnen
x_neat = np.linspace(0, time_neat, len(neat_scores))  # Zeitachse für NEAT
x_ddpg = np.linspace(0, time_ddpg, len(scores_DDPG))  # Zeitachse für DDPG
x_ppo = np.linspace(0, time_ppo, len(scores_PPO))    # Zeitachse für PPO

# Plotten
plt.figure(figsize=(12, 6))

plt.plot(x_neat, neat_scores_smoothed, label="NEAT", color='blue', alpha=0.7)
plt.plot(x_ddpg, scores_DDPG_smoothed, label="DDPG", color='green', alpha=0.7)
plt.plot(x_ppo, scores_PPO_smoothed, label="PPO", color='red', alpha=0.7)

# Details für den Plot
plt.xlabel("Zeit (Sekunden)")
plt.ylabel("Score")
```

```
plt.legend()
plt.grid()
plt.show()
```