



**AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE**  
**WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI, INFORMATYKI I INŻYNIERII**  
**BIOMEDYCZNEJ**

Automatyka i Robotyka

Informatyka w sterowaniu i zarządzaniu

Modelowanie i analiza procesów biznesowych

*Przygotowanie i uruchomienie prostego symulatora  
procesów biznesowych oraz opracowanie przykładu wraz  
tutorialem użycia na bazie istniejącej implementacji*

---

<i>L.p.</i>	<b>Członek</b>	<b>Numer albumu</b>	<b>Adres e-mail</b>
1	Michał Motyl	401943	michamotyl@student.agh.edu.pl

# Spis treści

<b>1. Wstęp</b>	<b>3</b>
<b>2. Camunda Platform 7</b>	<b>3</b>
2.1. Instrukcja instalacji	3
2.2. Utworzenie diagramów BPMN	3
2.3. Ekstrakcja logów z procesów.	5
2.4. Wyniki	6
<b>3. Wykorzystanie biblioteki PM4PY</b>	<b>7</b>
3.1. Implementacja	7
3.2. Wyniki	10
<b>4. Podsumowanie</b>	<b>10</b>

# 1.Wstęp

Założeniem projektu było utworzenie symulatora zwracającego dane sumaryczne na podstawie utworzonych wcześniej diagramów BPMN w celu ich analizy i tworzenia statystyk, oraz opracowanie przykładu wraz tutorialiem użycia. W tym celu wykorzystano dwa sposoby ekstrakcji logów z wykonywanych procesów by na ich podstawie umożliwić dalszą obróbkę danych oraz utworzenie aplikacji.

Wykorzystane narzędzia i biblioteki:

- Camunda Platform 7
- Biblioteka pm4py

## 2.Camunda Platform 7

W celu realizacji projektu wykorzystano oprogramowanie Camunda Platform 7 będące kompleksowym narzędziem do zarządzania procesami biznesowymi, wspierającym w modelowaniu, automatyzacji i optymalizacji procesów. Platforma zawiera moduły BPMN Workflow Engine, the DMN Decision Engine, Tasklist i Cockpit Basic

Dodatkowo Camunda udostępnia interfejs API REST, umożliwiający programowalną interakcje z platformą, a także przeprowadzenie operacji uruchamiania i zamykania instancji procesów, oraz pobieranie ich danych procesów i zarządzanie nimi.

### 2.1. Instrukcja instalacji

Aby umożliwić instalację aplikacji na urządzeniu wymagane jest pobranie oraz zainstalowanie:

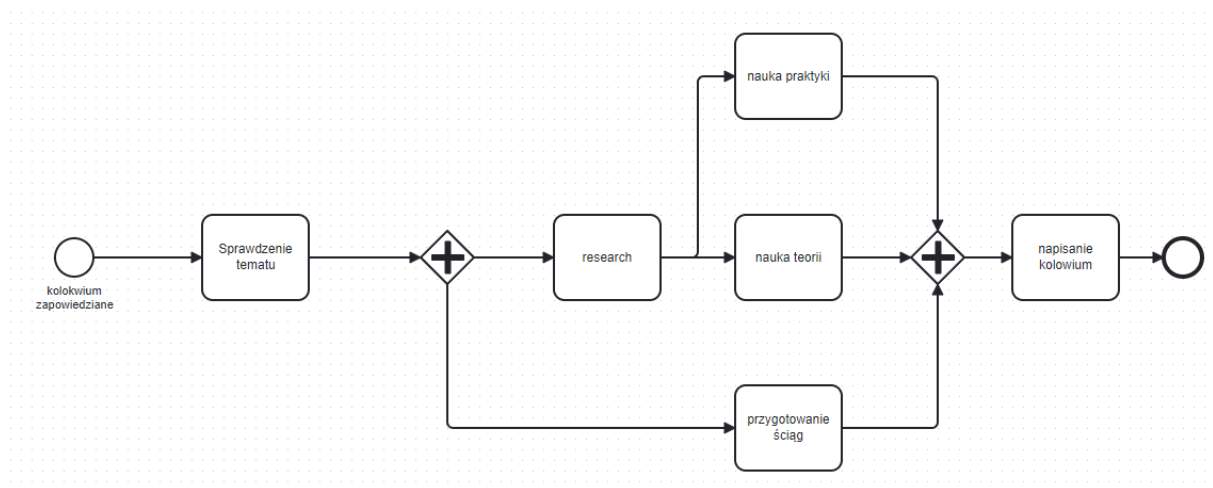
- Pakietu Java SE Development Kit,
- Camunda Open Source Desktop Modeler
- Camunda Platform 7 Open Source Community Edition

Następnie należy uruchomić plik start.bat znajdujący się w folderze Camunda Platform. Uruchomienie pliku inicjuje platformę Camunda i umożliwia jej użycie, co pozwala na interakcję z aplikacjami internetowymi, a także dostęp do uruchomionych procesów.

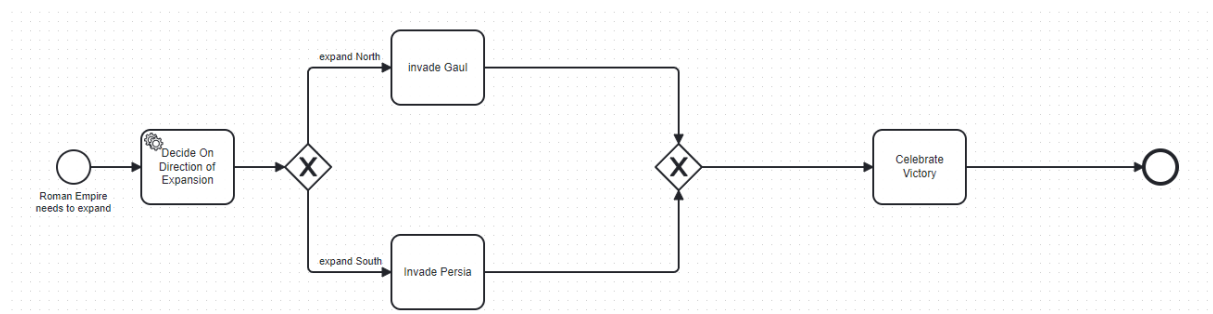
Po pomyślnym zainicjowaniu, platforma Camunda jest ustawiona na localhost:8080. Aby rozpocząć pracę, należy zalogować się do Tasklist Camunda przy użyciu danych uwierzytelniających demo/demo. Po wykonaniu tych kroków można przejść do tworzenia diagramów BPMN.

### 2.2. Utworzenie diagramów BPMN

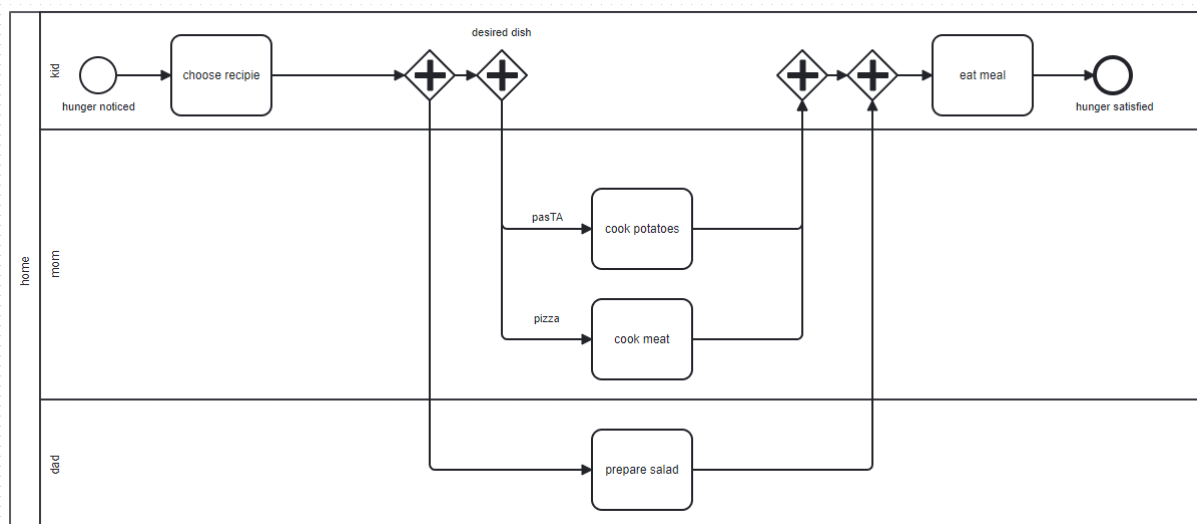
Tworzenie diagramów BPMN w oprogramowaniu Camunda jest operacją bardzo szybką i intuicyjną, jednak aby umożliwić ich poprawne uruchomienie i dalszą analizę wymagane jest utworzenie oprogramowania odpowiedzialnego za realizację warunków logicznych poszczególnych zadań. Na ilustracjach Rys. 2.1, Rys. 2.2, Rys. 2.3 przedstawiono proste przykładowe diagramy w celu realizacji dalszych założeń projektu.



Rys. 2.1. Diagram kolokwium.bpmn



Rys. 2.2. Diagram rome.bpmn

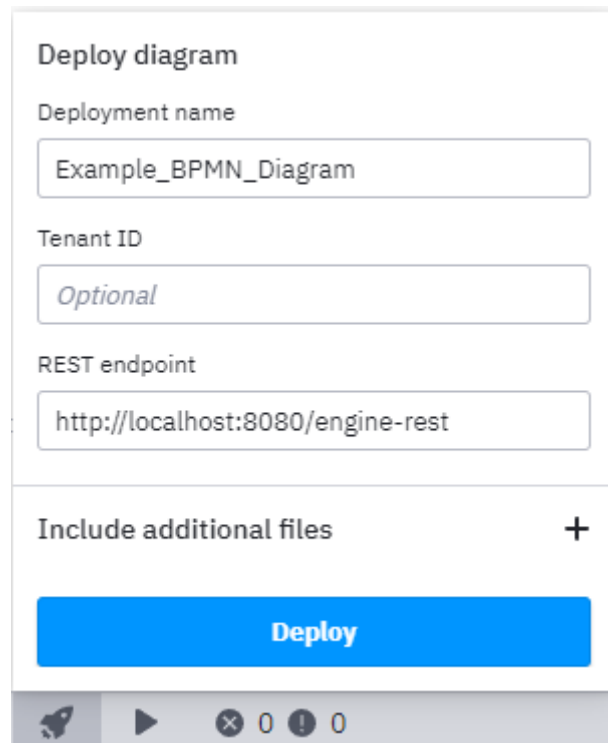


Rys. 2.3. diagram obiad.bpmn

### 2.3. Ekstrakcja logów z procesów.

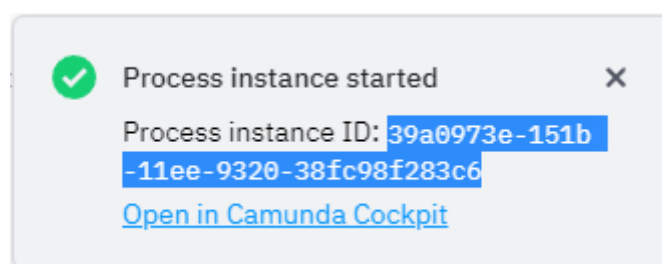
Po utworzeniu diagramów BPMN należy je wdrożyć do uruchomionego środowiska poprzez przycisk deploy

Następnie uruchom Camunda Modeler, stwórz pierwszy diagram BPMN i wdroż go co zostało przedstawione na rysunku Rys. 2.4



Rys. 2.4. Deployment chart

Następnie należy uruchomić proces i skopiować ID instancji procesu które się pojawi następujący komunikat (Rys. 2.5).



Rys. 2.5. ID instancji procesu

Skopiowany kod należy wkleić do uruchomionego programu, wysyłającego żądanie do silnika Camundy i zapisującego historię procesu do pliku CSV. Kod został przedstawiony na ilustracji Rys. 2.6.

**Zapisane logi są gotowe do preparacji symulatora, lecz ze względów opisanych w podrozdziale 2.4 rozwiązanie okazało się nieoptymalne i wymagającym było zmienienie podejścia do problemu projektowego.**

```

logtaker.py > ...
1 import requests
2 import csv
3
4 # REST API endpoint URL
5 api_url = "http://localhost:8080/engine-rest/engine/default/history/activity-instance"
6
7 # Prompt the user to enter the Process Instance ID
8 process_instance_id = input("Enter the Process Instance ID: ")
9
10 # Send GET request to the API endpoint with the process instance ID as a query parameter
11 response = requests.get(api_url, params={"processInstanceId": process_instance_id})
12
13 # Check if the request was successful
14 if response.status_code == 200:
15     # Extract JSON data from the response
16     json_data = response.json()
17
18     # Extract relevant information from JSON data
19     data = []
20     for item in json_data:
21         data.append({
22             'Activity ID': item['activityId'],
23             'Activity Name': item['activityName'],
24             'Activity Type': item['activityType'],
25             'Assignee': item['assignee'],
26             'Called Process Instance ID': item['calledProcessInstanceId'],
27             'Called Case Instance ID': item['calledCaseInstanceId'],
28             'Canceled': item['canceled'],
29             'Complete Scope': item['completeScope'],
30             'Duration In Millis': item['durationInMillis'],
31             'End Time': item['endTime'],
32             'Execution ID': item['executionId'],
33             'ID': item['id'],
34             'Parent Activity Instance ID': item['parentActivityInstanceId'],
35             'Process Definition ID': item['processDefinitionId'],
36             'Process Instance ID': item['processInstanceId'],
37             'Start Time': item['startTime'],
38             'Task ID': item['taskId'],
39             'Tenant ID': item['tenantId'],
40             'Removal Time': item['removalTime'],
41             'Root Process Instance ID': item['rootProcessInstanceId'],
42         })
43
44     # Save the data to a CSV file with a specified delimiter
45     csv_filename = input("Give csv file name: ")
46     csv_filename = "logs_folder/" + csv_filename + ".csv"
47     delimiter = ';' # Specify the desired delimiter here
48     with open(csv_filename, 'w', newline='') as csvfile:
49         writer = csv.DictWriter(csvfile, fieldnames=data[0].keys(), delimiter=delimiter)
50         writer.writeheader()
51         writer.writerows(data)
52
53     print(f"Activity instance logs saved to {csv_filename}")
54 else:
55     print(f"Error accessing the API: {response.status_code} - {response.text}")
56

```

Rys. 2.6. Program wyciągający logi z Camundy

## 2.4. Wyniki

Podjęcie z wykorzystaniem platformy Camunda nie zadziałało, ze względu stopnia skomplikowania uruchamiania procesów z uwagi na konieczność pisania skryptów do każdego diagramu BPMN. Dodatkowo ekstrakcja danych przy takiej konfiguracji nie posiada symulowania tokenów, które wdrażają element losowości i zmiennych potrzebnych do dalszych działań.

Docelowo byłoby to rozwiązanie, które przy większej dostępności zasobów przyniosłoby zadowalające efekty umożliwiające dogłębną i intuicyjną analizę procesów biznesowych.

## 3. Wykorzystanie biblioteki PM4PY

Ze względu na znaczące trudności prowadzenia projektu przy wykorzystaniu Camunda Platform zdecydowano na zmianę podejścia do generacji symulacji i implementacji przy wykorzystaniu skryptu napisanego w języku Python z wykorzystaniem dedykowanej do niego biblioteki PM4PY. Została ona stworzona do obsługi i analizy procesów biznesowych. Oferuje wiele funkcji do wczytywania, przetwarzania, analizy i wizualizacji modeli procesów oraz dzienników zdarzeń

### 3.1. Implementacja

Przygotowany skrypt wykorzystuje wcześniej przygotowane diagramy BPMN. W celu ekstrakcji logów biblioteka wymaga importu diagramu oraz jego reprezentacji w postaci modelu Petri net. Poniżej prezentowane jest rozwiązanie krok po kroku.

```
~ Import data

Specify the names of bpmn diagram (should be stored in bpmn_diagrams), log file (as a csv file, should be stored in logs_folder) and bpmn specifics (should be stored in bpmn_specifics folder as csv files)

# EDIT BELOW #
#####
bpmn_diagram = 'kolokwium.bpmn'
bpmn_specifics = 'kolokwium.csv'
csv_to_save_results = 'kolokwium_results.csv'
#####
# END EDIT #

path_to_bpmn_diagram = 'bpmn_diagrams/' + bpmn_diagram
path_to_specifics = 'bpmn_specifics/' + bpmn_specifics
csv_destination = csv_to_save_results

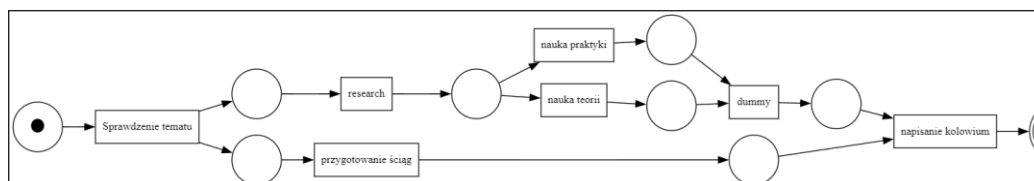
with open(csv_destination, 'w') as creating_new_csv_file:
    | pass
    print("Empty File Created Successfully")

# loading bpmn into python through pm4py
bpmn = pm4py.read_bpmn(path_to_bpmn_diagram)

# transforming it into petri net
# sometimes loses the last task for some reason?
net, im, fm = pm4py.convert_to_petri_net(bpmn)
pm4py.visualization.petri_net.variants.alignments.apply(net, im, fm)
```

Rys. 3.1. Kod odpowiedzialny za import danych

Wykorzystana funkcja **pm4py.convert\_to\_petri\_net(bpmn)** jest obciążona pewnym błędem, skutującym utratą części danych w przypadku bardziej skomplikowanych diagramów. W celu weryfikacji interpretacji skrypt został wyposażony w funkcję wizualizującą przekonwertowany diagram co zostało przedstawione na Rys. 3.2



Rys. 3.2. Reprezentacja procesu w reprezentacji Petri Net

Kolejna część kodu odpowiada za znalezienie wszystkich możliwych permutacji wykonania procesów, a następnie ekstrakcję logów do struktury Data Rys. 3.3

```

# simulating logs with pm4py
from pm4py.algo.simulation.playout.petri_net.variants import extensive
# log = pm4py.play_out(net, im, fm)
# log = pm4py.play_out(process_tree_test)
from pm4py.algo.simulation.playout.petri_net import algorithm as simulator
log = simulator.apply(net, im, variant=simulator.Variants.EXTENSIVE, parameters={simulator.Variants.EXTENSIVE.value.Parameters.MAX_TRACE_LENGTH: 20})

from pm4py.objects.conversion.log_variants import to_data_frame
log_df = to_data_frame.apply(log)

log_df.head(10)

```

	conceptname	time:timestamp	case:conceptname
0	Sprawdzenie tematu	1970-04-26 18:46:41+00:00	0
1	przygotowanie ściąg	1970-04-26 18:46:42+00:00	0
2	research	1970-04-26 18:46:43+00:00	0
3	nauka praktyki	1970-04-26 18:46:44+00:00	0
4	Sprawdzenie tematu	1970-04-26 18:46:45+00:00	1
5	przygotowanie ściąg	1970-04-26 18:46:46+00:00	1
6	research	1970-04-26 18:46:47+00:00	1
7	nauka teorii	1970-04-26 18:46:48+00:00	1
8	Sprawdzenie tematu	1970-04-26 18:46:49+00:00	2
9	research	1970-04-26 18:46:50+00:00	2

Rys. 3.3. Kod odpowiedzialny za ekstrakcję logów

Następnie następuje ich pogrupowanie zgodnie z wartością ID case:concept:name i utworzenie tabeli definiującej wartości czasu wykonania, kosztów i zysków poszczególnych procesów, które mogą być modyfikowane przez użytkownika Rys. 3.4

```

# having the logs we split the dataframe into individual dataframes according to case:concept:name
unique_ids = log_df['case:concept:name'].unique()
log_dfs_by_id = []
for id in unique_ids:
    log_df_id = log_df[log_df['case:concept:name'] == id]
    log_dfs_by_id.append(log_df_id)

# assuming equal probability of each found path from the graph --> customisation of this part to be added later
length = len(unique_ids)
unique_ids_times_of_occurrence = np.random.normal(100, 3, length).round()

# most important element - generating data based off of specifics
# first create a df to store data
bpmn_specifics_df = pd.read_csv(path_to_specifics, na_values=[0, '-'])
bpmn_specifics_df.head()

```

	task_name	time_dist	time_unit	costs_dist	costs_unit	profit_dists	profit_unit
0	Sprawdzenie tematu	NaN	NaN	NaN	NaN	NaN	NaN
1	research	linear(1.4)	hour	linear(1.4)	morale	linear(1.2)	knowledge
2	nauka praktyki	linear(1.3)	hour	linear(1.3)	mental_units	linear(1.2)	street_smarts
3	przygotowanie ściąg	linear(1.10)	hour	linear(1.10)	mental_units	linear(1.2)	knowledge
4	nauka teorii	linear(1.2)	hour	linear(1.2)	mental_units	linear(1.2)	knowledge

Rys. 3.4. Grupowanie logów

Kolejną częścią jest ta odpowiedzialna za utworzenie słowników mających służyć do sumowania wartości poszczególnych zadań podczas wykonywania symulacji. Algorytm przechodzi po każdym Data Frame i oblicza wartości czasu, kosztów i zysku na podstawie wczytanych danych dotyczących rozkładów czasu, kosztów i zysku dla poszczególnych zadań. Obliczone wartości są sumowane dla odpowiednich jednostek i przechowywane w słownikach *time\_dict*, *costs\_dict* i *profit\_dict*. Przedstawiony kod został na rysunkach Rys. 3.5 - Rys. 3.8

```

specifics_copy = bpmn_specifics_df.dropna()
time_units = specifics_copy['time_unit'].unique()
costs_units = specifics_copy['costs_unit'].unique()
profit_units = specifics_copy['profit_unit'].unique()

profit_dict = dict.fromkeys(profit_units, 0)
time_dict = dict.fromkeys(time_units, 0)
costs_dict = dict.fromkeys(costs_units, 0)

for i, path_log in enumerate(log_dfs_by_id):
    num_of_occurrence = unique_ids_times_of_occurrence[i]
    for j in range(int(num_of_occurrence)):
        for k, event in enumerate(path_log.values):
            task_name = event[0]
            task_specifics = bpmn_specifics_df[bpmn_specifics_df['task_name'] == task_name_]

            time_dist_string = task_specifics['time_dist'].values[0]
            time_unit = task_specifics['time_unit'].values[0]

            cost_dist_string = task_specifics['costs_dist'].values[0]
            cost_unit = task_specifics['costs_unit'].values[0]

            profit_dist_string = task_specifics['profit_dists'].values[0]
            profit_unit = task_specifics['profit_unit'].values[0]

```

Rys. 3.5. Tworzenie słowników



```

# calculating time
if time_dist_string is not np.nan:
    name, data = time_dist_string.split(',')
    data = data.replace(' ', '')
    data = np.array(data.split(','), dtype=int)

    if name == 'linear':
        value_generated = np.random.uniform(data[0], data[1])
        time_dict[time_unit] = time_dict[time_unit] + value_generated

    elif name == 'normal':
        value_generated = np.random.normal(data[0], data[1])
        time_dict[time_unit] = time_dict[time_unit] + value_generated

    elif name == 'lognormal':
        value_generated = np.random.lognormal(data[0], data[1])
        time_dict[time_unit] = time_dict[time_unit] + value_generated

    else:
        raise ValueError('wrong input from specifics')

```

Rys. 3.6. Kalkulacja czasu

```

# calculating costs:
if cost_dist_string is not np.nan:
    name, data = cost_dist_string.split(',')
    data = data.replace(' ', '')
    data = np.array(data.split(','), dtype=int)

    if name == 'linear':
        value_generated = np.random.uniform(data[0], data[1])
        costs_dict[cost_unit] = costs_dict[cost_unit] + value_generated

    elif name == 'normal':
        value_generated = np.random.normal(data[0], data[1])
        costs_dict[cost_unit] = costs_dict[cost_unit] + value_generated

    elif name == 'lognormal':
        value_generated = np.random.lognormal(data[0], data[1])
        costs_dict[cost_unit] = costs_dict[cost_unit] + value_generated

    else:
        raise ValueError('wrong input from specifics')

```

Rys. 3.7. Kalkulacja kosztów

```

#calculating profits
if profit_dist_string is not np.nan:
    name, data = profit_dist_string.split(',')
    data = data.replace(' ', '')
    data = np.array(data.split(','), dtype=int)

    if name == 'linear':
        value_generated = np.random.uniform(data[0], data[1])
        profit_dict[profit_unit] = profit_dict[profit_unit] + value_generated

    elif name == 'normal':
        value_generated = np.random.normal(data[0], data[1])
        profit_dict[profit_unit] = profit_dict[profit_unit] + value_generated

    elif name == 'lognormal':
        value_generated = np.random.lognormal(data[0], data[1])
        profit_dict[profit_unit] = profit_dict[profit_unit] + value_generated

    else:
        raise ValueError('wrong input from specifics')

```

Rys. 3.8. Kalkulacja zysków

Ostatnim krokiem symulacji jest kalkulacja bilansów i zapis ich do pliku csv Rys. 3.9

```

>>> profits_and_costs = profit_units.tolist() + costs_units.tolist()
>>> balance = {}
>>> for unit in profits_and_costs:
>>>     if unit in profit_units and unit in costs_units:
>>>         balance[unit] = profit_dict[unit] - costs_dict[unit]
>>>     elif unit in profit_units:
>>>         balance[unit] = profit_dict[unit]
>>>     else:
>>>         balance[unit] = -costs_dict[unit]
>>>
>>> print(balance)
{'knowledge': 2278.476401920102, 'street_smarts': 444.22696204806385, 'morale': -1585.2992532786525, 'mental_units': -4340.09406199708}

>>> print('Final calculated balance after simulation:')
>>> print('\n')
>>> for key, value in balance.items():
>>>     print(key, ': ', value)

>>> print('\n')
>>> print('Saving to csv, destination: ', csv_destination)
>>> pd.DataFrame(balance, index=[0]).to_csv(csv_destination, mode='w')
>>> print('Saved')
Final calculated balance after simulation:

knowledge : 2278.476401920102
street_smarts : 444.22696204806385
morale : -1585.2992532786525
mental_units : -4340.09406199708

Saving to csv, destination: kolokwium_results.csv
Saved

```

Rys. 3.9. Kalkulacja bilansów i zapis do pliku

Ostatecznie prezentowany wyżej notebook został w prosty sposób przerobiony na postać skryptową, dzięki czemu finalny użytkownik może wywołać i uruchomić symulację bez potrzeby zagłębiania się w plik notebookowy, lecz tylko wywołanie skryptu i wykonywanie instrukcji w nim zawartych. Dopiero jeżeli w wynikach zobaczymy dane których użytkownik mógł się nie spodziewać, można wejść do notatnika i przeglądać etapami algorytm: sprawdzić czy diagramy są poprawnie wczytywane, czy poprawnie są przerabiane na sieci petriego, oraz czy znalezione ścieżki są kompletne.

### 3.2. Wyniki

Wykorzystana implementacja symulatora w postaci skryptu zwraca interesujące nas dane na temat wykonujących się procesów, nie jest jednak rozwiązaniem w pełni automatycznym, wymaga ona interakcji z użytkownikiem. Program faktycznie zwraca interesujące nas dane, lecz bazuje on przede wszystkim na naiwnym założeniu równego prawdopodobieństwa wystąpienia każdej z wykrytych ścieżek w diagramie. Dodatkowo, wykorzystywane przez nas rozwiązanie (a dokładniej metoda z biblioteki pm4py) wymusza reprezentację procesu w postaci sieci petriego, co potencjalnie może powodować problemy. Po pierwsze, wygenerowana sieć petriego na podstawie diagramu bpmn może być zwyczajnie błędna. Metody automatycznej generacji sieci petriego na podstawie diagramu bpmn dostępne w bibliotece pm4py nie nadają się do automatycznej generacji sieci dla dużych i skomplikowanych diagramów. Stąd ograniczenie na złożoność plików bpmn. Dodatkowo dla niektórych plików bpmn po poprawnym wygenerowaniu sieci petriego, algorytm poszukujący wszystkich możliwych ścieżek przez sieć czasami pomija ostatnie zadanie w sieci. Nie udało nam się znaleźć żadnej prawidłowości w tym zachowaniu. Zależnie od diagramu takie zadanie może być szybko dodane do ścieżek w sposób ręczny, lecz jako że celem było stworzenie automatycznego symulatora takie rozwiązanie nie jest pożądane.

## 4. Podsumowanie

Projekt miał na celu stworzenie symulatora procesów biznesowych i opracowanie przykładu z tutorialiem. Mimo trudności na początku, zmiana podejścia i wykorzystanie biblioteki PM4PY pozwoliły na uzyskanie interesujących wyników analizy procesów biznesowych.

Docelowo wykorzystanie oprogramowania Camunda Platform jest rozwiązaniem, do którego należałoby dążyć gdyż jest w stanie umożliwić pełną automatyzację działania symulacji. Aby to osiągnąć, projekt wymagałby jednak znacznego nakładu pracy przekraczającego wstępne założenia.

Jako jedną ze ścieżek, którą można by przyjąć za krok ku rozwojowi tego problemu byłoby utworzenie osobnego symulatora generującego samodzielnie działające diagramy BPMN, jednak ilość pracy potrzebna do wykonania tego założenia wymagałaby wielokrotnie większego nakładu pracy niż zakładaliśmy podczas realizacji obecnego projektu.