

**PROJEKT: SYMULACJA AGENTOWA BITWY POD  
ZBOROWEM (1649) Z UWZGLĘDNIENIEM WPŁYWU  
TERENU I WARUNKÓW ATMOSFERYCZNYCH**

**KOD KURSU: 120-ISI-1S-777**

**PRZEDMIOT: SYMULACJA SYSTEMÓW DYSKRETNÝCH  
(SSD)**

Autorzy projektu

Patrick Bajorski

Jan Banasik

Gabriel Filipowicz

Opiekun projektu

prof. dr hab. inż. Jarosław Wąs



**AGH**

EAIiIB / Katedra Informatyki Stosowanej  
Akademia Górniczo-Hutnicza im. Stanisława Staszica w  
Krakowie  
Kraków, Polska

09 stycznia 2026

# Spis treści

<b>Abstract</b>	<b>iv</b>
<b>1 CEL I ZAKRES PROJEKTU</b>	<b>1</b>
1.1 Cel projektu . . . . .	1
1.2 Oczekiwane rezultaty . . . . .	2
<b>2 CHARAKTERYSTYKA PROBLEMU</b>	<b>3</b>
2.1 Tło historyczne i teoretyczne . . . . .	3
2.2 Przegląd literatury . . . . .	3
<b>3 DANE PORÓWNAWCZE I WEJŚCIOWE</b>	<b>4</b>
3.1 Dane wejściowe symulacji . . . . .	4
3.1.1 Pełna charakterystyka jednostek . . . . .	4
3.1.2 Mapa i teren . . . . .	5
<b>4 MODEL FORMALNY</b>	<b>6</b>
4.1 Szczegółowy model zachowania agenta . . . . .	6
4.2 Algorytm ucieczki i regeneracji . . . . .	8
<b>5 REALIZACJA PRAKTYCZNA</b>	<b>9</b>
5.1 Założenia projektowe . . . . .	9
5.2 Komponenty sprzętowe . . . . .	9
5.3 Oprogramowanie . . . . .	10
5.3.1 Język programowania i środowisko . . . . .	10
5.3.2 Wykorzystane biblioteki (Backend) . . . . .	10
5.3.3 Technologie webowe (Frontend) . . . . .	11
5.4 Instalacja i uruchomienie . . . . .	11
5.5 Szczegóły implementacji . . . . .	12
5.5.1 Struktura projektu . . . . .	12
5.5.2 Kluczowe algorytmy i mechanizmy . . . . .	13
<b>6 REZULTATY SYMULACJI</b>	<b>14</b>

6.1	Wpływ warunków pogodowych (Scenariusz 2: Obrona Wałów) . . . . .	14
6.1.1	Pogoda słoneczna (Warunki bazowe) . . . . .	14
6.1.2	Ulewny deszcz . . . . .	15
6.1.3	Gęsta mgła . . . . .	15
6.2	Eksperymenty syntetyczne . . . . .	15
6.2.1	Eksperyment: Siła Ognia (Firepower) . . . . .	16
6.2.2	Eksperyment: Mobilność (Mobility) . . . . .	16
6.2.3	Eksperyment: Jakość vs Ilość (Quality vs Quantity) . . . . .	16
<b>7</b>	<b>DYSKUSJA WYNIKÓW</b>	<b>17</b>
7.1	Interpretacja wpływu pogody na wynik bitwy . . . . .	17
7.2	Analiza jednostek w świetle eksperymentów . . . . .	17
7.2.1	Fenomen Dragonii . . . . .	18
7.2.2	Mit Husarii potwierdzony . . . . .	18
7.2.3	Krytyczna masa piechoty . . . . .	18
7.3	Ograniczenia modelu i anomalie . . . . .	18
7.4	Podsumowanie . . . . .	19
<b>8</b>	<b>PODSUMOWANIE</b>	<b>20</b>
	<b>Bibliografia</b>	<b>21</b>

# Abstract

Edycja tego rozdziału w pliku: `pages/abstract.tex`

Abstrakt jest zamieszczony w charakterze informacyjnym. Można go zakomentować w pliku: `ju_cse_report.sty`. Odpowiednie miejsce jest zaznaczone komentarzem. Oczywiście jeśli zespół chce może tu wstawić własny abstrakt.

Kod źródłowy Latex dla raportu jest zorganizowany w sposób modułowy. Bazą dla tego szablonu jest wersja oryginalna dostępna pod adresem <https://www.overleaf.com/latex/templates/ju-cse-report-template/ctzvgmyrmbzq>.

**Przeczytaj poniższe instrukcje, aby zrozumieć strukturę dokumentu.**

`chapters` –: Zawiera rozdziały raportu.

`figures` –: Folder na rysunki i ilustracje.

`pages` –: Pliki definiujące zawartość dla streszczenia, bibliografii, itp.

`parameters` –: W tym folderze można dodać informacje o pracy dyplomowej, takie jak imię i nazwisko autora, ID, stopień naukowy, sesja, nazwisko opiekuna.

`ju_cse_report.sty` –: Można definiować ustawienia stylu dokumentu.

`ju_cse_report_Ref.bib` –: Lista referencji. Bibliografia w formacie Bib $\TeX$ .

`report.tex` –: Makro struktura raportu. Jeśli grupa koniecznie potrzebuje można tu dodać dodatkowe elementy np. rozdziały.

# 1. CEL I ZAKRES PROJEKTU

## 1.1 Cel projektu

Celem głównym projektu jest opracowanie i implementacja symulacji agentowej (ABM – Agent-Based Modeling) historycznej Bitwy pod Zborowem, która miała miejsce w 1649 roku. Projekt ma na celu zbadanie wpływu czynników stochastycznych (morale, panika) oraz środowiskowych (ukształtowanie terenu, warunki atmosferyczne) na wynik starcia pomiędzy Armią Koronną a wojskami Kozacko-Tatarskimi.

Problem naukowy, jaki projekt próbuje rozwiązać, dotyczy weryfikacji, w jakim stopniu zmiana parametrów początkowych (takich jak pogoda lub dyscyplina oddziałów) mogła wpłynąć na zmianę historycznego rezultatu bitwy, przy zachowaniu wierności historycznej charakterystyk jednostek.

Cele szczegółowe obejmują:

1. Opracowanie modelu formalnego zachowań jednostek wojskowych XVII wieku (husaria, piechota, artyleria) z uwzględnieniem ich parametrów bojowych.
2. Implementację algorytmów decyzyjnych agentów, w tym mechaniki morale, paniki oraz wyszukiwania ścieżki (pathfinding) w środowisku dyskretnym.
3. Stworzenie interaktywnego środowiska wizualizacyjnego pozwalającego na obserwację przebiegu bitwy w czasie rzeczywistym oraz analizę post-hoc (mapy ciepła).
4. Przeprowadzenie eksperymentów symulacyjnych dla różnych scenariuszy pogodowych (deszcz, mgła, pogoda słoneczna).

## 1.2 Oczekiwane rezultaty

Finalnym rezultatem projektu jest aplikacja symulacyjna działająca w architekturze klient-serwer, napisana w języku Python z wykorzystaniem biblioteki Mesa oraz frameworka Flask.

Oczekiwane cechy rozwiązania:

- **Ilościowe:** System obsłuży symulację z udziałem kilkudziesięciu niezależnych agentów reprezentujących oddziały, na mapie o wymiarach odwzorowujących pole bitwy, z krokiem czasowym (turą) odpowiadającym jednostce czasu rzeczywistego.
- **Jakościowe:** Model uwzględni nieliniowe efekty walki, takie jak załamanie morale prowadzące do łańcuchowej paniki oraz wpływ terenu (błoto, rzeka) na mobilność wojsk.
- **Funkcjonalne:** Użytkownik będzie miał możliwość wyboru scenariusza początkowego, podglądu statystyk w czasie rzeczywistym oraz analizy historycznej bitew.

## 2. CHARAKTERYSTYKA PROBLEMU

### 2.1 Tło historyczne i teoretyczne

Bitwa pod Zborowem była jednym z kluczowych starć powstania Chmielnickiego. Charakteryzowała się znaczną dysproporcją sił oraz kluczowym znaczeniem terenu (rzeka Strypa, przeprawy mostowe) i fortyfikacji polowych. Modelowanie takich zjawisk wymaga podejścia, które uwzględnia nie tylko liczebność wojsk, ale przede wszystkim ich interakcje lokalne i psychologię pola walki.

### 2.2 Przegląd literatury

W literaturze przedmiotu dotyczącej symulacji wojskowych dominuje podejście oparte na równaniach różniczkowych (modele Lancastera), które jednak słabo radzą sobie z niejednorodnością jednostek i terenem. Dlatego w niniejszym projekcie wybrano podejście symulacji agentowej (ABM).

Podstawę merytoryczną dla parametrów jednostek oraz mapy taktycznej stanowiły opracowania historyczne zgromadzone w dokumentacji projektu:

1. *Architectural Studies* – analiza topografii i fortyfikacji Zborowa, co pozwoliło na wierne odwzorowanie mapy kosztów ruchu.
2. *Bitwa pod Zborowem 15-16 sierpnia* – szczegółowy opis przebiegu starcia, wykorzystany do definicji scenariuszy i rozmieszczenia początkowego wojsk.
3. *Tatars, Cossacks and the Polish Army* – źródło danych na temat uzbrojenia, taktyki i morale poszczególnych formacji (np. różnice między jazdą tatarską a husarią).

Zastosowanie podejścia agentowego pozwala na emergencję zachowań globalnych (np. oskrzydlenie, ucieczka armii) z prostych reguł definiowanych na poziomie pojedynczego oddziału.

## 3. DANE PORÓWNAWCZE I WEJŚCIOWE

### 3.1 Dane wejściowe symulacji

Danymi wejściowymi dla modelu są parametryzowane charakterystyki jednostek oraz cyfrowa reprezentacja terenu. Dane te zostały pozyskane z literatury historycznej i przetworzone na formaty cyfrowe (pliki JSON, TMX).

#### 3.1.1 Pełna charakterystyka jednostek

W symulacji zaimplementowano 13 unikalnych typów jednostek, podzielonych na dwie frakcje: Armię Koronną oraz Wojska Kozacko-Tatarskie. Parametry te zostały zaszyte w pliku konfiguracyjnym modelu i determinują zachowanie agentów. Poniższa tabela prezentuje kompletny zestaw danych.

Tabela 3.1: Szczegółowe parametry wszystkich jednostek w symulacji.

Jednostka	HP	Morale	Dysc.	Atak (Wręcz/Dyst.)	Zasięg	Szybkość	Uwagi
<b>Armia Koronna</b>							
Husaria	150	140	95	100 / 0	1	6	Elitarna, bonus do szarży
Pancerni	120	110	85	70 / 0	1	7	Jazda uniwersalna
Rajtaria	110	100	90	40 / 30	3	6	Ciężka jazda ogniowa
Dragonía	100	95	85	30 / 25	4	5	Piechota konna
Piechota Niemiecka	110	100	95	25 / 35	5	3	Wysoka dyscyplina
Pospolite Ruszenie	90	50	20	20 / 10	2	6	Bardzo niska dyscyplina
Czeladź Obozowa	60	90	40	25 / 0	1	5	Obrona obozu
Artyleria Koronna	50	90	90	5 / 150	15	1	Bardzo duży zasięg
<b>Kozacy/Tatarzy</b>							
Jazda Tatarska	85	80	70	30 / 15	4	9	Najszybsza jednostka
Piechota Kozacka	115	110	80	35 / 35	5	4	Znakomici strzelcy
Jazda Kozacka	100	90	75	50 / 0	2	7	Jazda średnia
Czerń	70	60	40	20 / 0	1	5	Słabo uzbrojona masa
Artyleria Kozacka	40	80	80	5 / 130	14	1	Ostrzał obozu

Wartości parametrów (takie jak *rate\_of\_fire* czy *ammo*) wpływają bezpośrednio na logikę walki. Przykładowo, jednostki o niskiej dyscyplinie (Pospolite Ruszenie - 20, Czerń - 40) są znacznie bardziej podatne na panikę, co zostało opisane w algorytmach w Rozdziale 4.



### 3.1.2 Mapa i teren

Teren bitwy (plik `map.tmx`) został przekonwertowany na siatkę kosztów ruchu (`terrain_costs`). Koszty te wpływają na prawdopodobieństwo wykonania ruchu przez agenta w danej turze.

- Teren podstawowy: koszt 1.0.
- Teren trudny (lasy, wzgórza): koszt  $> 1.0$  (zmniejsza szansę na ruch).
- Błoto (podczas deszczu): mnożnik kosztu  $\times 2.5$ .
- Rzeka: wysoki koszt (ciężko przekraczalne).
- Przeszkody (np. ściany obozu): koszt nieskończony (nieprzekraczalne).

## 4. MODEL FORMALNY

### 4.1 Szczegółowy model zachowania agenta

Logika decyzyjna każdego agenta (*MilitaryAgent*) jest wykonywana w każdej turze symulacji. Proces ten jest deterministyczno-stochastyczny: reguły są stałe, ale ich wynik zależy od rzutów prawdopodobieństwa (np. test morale, szansa na trafienie).

Poniżej przedstawiono kompletny algorytm cyklu życia agenta (Algorytm 1) oraz dedykowaną procedurę obsługi ucieczki (Algorytm 2), która różnicuje zachowanie w zależności od przynależności frakcyjnej.

---

**Algorithm 1** Główna pętla decyzyjna agenta (Step)

---

**Input:** Agent  $A$ , Grid  $G$ , Weather  $W$ 

```

1: if  $A.HP \leq 0$  then
2:   return
3: end if
4: Zmniejsz  $A.fire\_cooldown$  jeśli  $> 0$ 
5:  $PanicThreshold \leftarrow 25 - (A.Discipline/5)$ 
6: if  $A.Morale < PanicThreshold$  and  $A.State \neq FLEEING$  then
7:   if  $Random(0, 100) > A.Discipline$  then
8:      $A.State \leftarrow FLEEING$ 
9:     Call  $MANAGEFLEEING(A)$ 
10:  end if
11: end if
12: if  $A.State == FLEEING$  then
13:   if brak ścieżki and  $Frakcja == "Armia Koronna"$  then
14:     Call  $MANAGEFLEEING(A)$  {Przelicz cel ucieczki}
15:   end if
16:   Call  $MOVE(A)$ 
17:   return
18: end if
19:  $Enemy \leftarrow FINDENEMY(A, radius=W.visibility)$ 
20: if  $Enemy$  istnieje then
21:    $Dist \leftarrow DISTANCE(A, Enemy)$ 
22:   if  $1 < Dist \leq A.Range$  and  $A.RangedDmg > 0$  then
23:      $Misfire \leftarrow 0.4$  jeśli  $W == Rain$  wpp. 0.0
24:     if  $A.Ammo > 0$  then
25:       if  $A.Cooldown \leq 0$  and  $Random() > Misfire$  then
26:          $A.State \leftarrow ATTACKING$ 
27:          $A.Ammo \leftarrow A.Ammo - 1$ 
28:          $DEALDAMAGE(Enemy, A.RangedDmg, TerrainModifier)$ 
29:       end if
30:     else
31:        $PATHTO(Enemy)$  {Brak amunicji, szarża}
32:        $MOVE(A)$ 
33:     end if
34:   else if  $Dist \leq 1.5$  then
35:      $A.State \leftarrow ATTACKING$ 
36:      $DEALDAMAGE(Enemy, A.MeleeDmg)$  {Walka wręcz}
37:   else
38:      $PATHTO(Enemy)$ 
39:      $MOVE(A)$ 
40:   end if
41: else
42:    $A.State \leftarrow MOVING\_TO\_STRATEGIC$ 
43:    $PATHTO(A.StrategicTarget)$ 
44:    $MOVE(A)$ 
45: end if

```

---

## 4.2 Algorytm ucieczki i regeneracji

Unikalnym elementem modelu jest zróżnicowana strategia ucieczki. Jednostki Kozackie zawsze uciekają do krawędzi mapy. Jednostki Koronne podejmują decyzję: jeśli Obóz (Centrum Leczenia) jest bliżej niż krawędź mapy i posiada wolne miejsca, agent spróbuje się tam schronić, aby zregenerować siły (HP i Morale).

---

**Algorithm 2** Procedura wyznaczania celu ucieczki (Manage Fleeing)

---

**Input:** Agent  $A$ , Grid  $G$

```

1:  $CurrentPos \leftarrow A.Position$ 
2:  $DistToEdge \leftarrow$  minimum dystansu do krawędzi (L, R, T, B)
3: if  $A.Faction == "Armia Koronna"$  then
4:    $TargetCenter \leftarrow$  Najbliższe niezapełnione Centrum Leczenia
5:    $ShouldFleeToEdge \leftarrow True$ 
6:   if  $TargetCenter$  istnieje then
7:      $DistToCenter \leftarrow DISTANCE(CurrentPos, TargetCenter)$ 
8:     if  $DistToCenter \leq 2 \times DistToEdge$  then
9:        $ShouldFleeToEdge \leftarrow False$ 
10:    end if
11:  end if
12:  if NOT  $ShouldFleeToEdge$  then
13:     $Entrance \leftarrow GETENTRANCE(TargetCenter)$ 
14:    if  $CurrentPos$  na  $Entrance$  then
15:      Wejdz głębiej w strefę leczenia
16:    else
17:       $CALCULATEPATH(Entrance)$ 
18:    end if
19:  else
20:     $CALCULATEPATH(Najbliższa Krawędź Mapy)$ 
21:  end if
22: else
23:    $CALCULATEPATH(Najbliższa Krawędź Mapy)$ 
24: end if

```

---

## 5. REALIZACJA PRAKTYCZNA

### 5.1 Założenia projektowe

W procesie realizacji systemu symulacji Bitwy pod Zborowem przyjęto następujące założenia projektowe, wynikające ze specyfiki modelowania agentowego oraz wymagań funkcjonalnych:

- **Architektura Klient-Serwer:** Rozdzielenie logiki symulacyjnej (Backend w języku Python) od warstwy prezentacji (Frontend w przeglądarce internetowej). Pozwala to na przeniesienie ciężaru obliczeń na serwer i łatwe zarządzanie stanem symulacji.
- **Dyskretna czasoprzestrzeń:** Model operuje na dyskretnej siatce (grid) odwzorowującej teren oraz w dyskretnych krokach czasowych (turach), gdzie jedna tura odpowiada ustalonej jednostce czasu rzeczywistego.
- **Autonomia agentów:** Każda jednostka wojskowa (agent) jest niezależnym bytem posiadającym własny stan (HP, morale, cel) i podejmującym decyzje na podstawie lokalnej percepcji środowiska (zasięg wzroku), bez centralnego sterowania.
- **Determinizm mapy:** Środowisko (teren, rzeki, przeszkody) jest statyczne i wczytywane z zewnętrznego pliku w formacie TMX, co gwarantuje powtarzalność warunków początkowych eksperymentów dla różnych scenariuszy.
- **Modułowość:** Struktura kodu (klasy `Model` i `Agent`) umożliwia łatwe dodawanie nowych typów jednostek poprzez konfigurację parametrów w słownikach, bez konieczności ingerencji w główną pętlę symulacyjną.

### 5.2 Komponenty sprzętowe

Symulacja nie wymaga specjalistycznego sprzętu serwerowego i może być uruchamiana na standardowych komputerach osobistych. Poniżej przedstawiono konfigurację

sprzętowa, na której weryfikowano działanie rozwiązania:

- **Procesor (CPU):** Zalecany procesor wielordzeniowy (np. Intel Core i5/i7 lub AMD Ryzen 5) o taktowaniu min. 2.5 GHz. Mimo że biblioteka Mesa działa w głównym wątku, dodatkowe rdzenie wspierają obsługę serwera Flask i procesy systemowe.
- **Pamięć RAM:** Minimum 8 GB. Ze względu na przechowywanie obiektowej reprezentacji każdego agenta oraz macierzy kosztów terenu dla algorytmów ścieżki (pathfinding), zalecane jest 16 GB RAM dla zapewnienia płynności przy dużej liczbie jednostek (>100).
- **Karta graficzna (GPU):** Dowolna karta graficzna wspierająca standardy webowe (WebGL/Canvas API). Renderowanie wizualizacji odbywa się po stronie klienta (przeglądarki internetowej).

## 5.3 Oprogramowanie

Projekt został zrealizowany w oparciu o otwarte oprogramowanie (Open Source) i biblioteki dostępne w ekosystemie języka Python.

### 5.3.1 Język programowania i środowisko

- **Język:** Python w wersji 3.10 lub nowszej.
- **System operacyjny:** Windows 10/11, Linux (Ubuntu 22.04 LTS) lub macOS.

### 5.3.2 Wykorzystane biblioteki (Backend)

Kluczowe zależności projektu zdefiniowane w pliku `pyproject.toml/requirements.txt`:

- **Mesa (2.1.2):** Podstawowy framework do modelowania agentowego (ABM). Dostarcza kluczowe abstrakcje takie jak `Model`, `Agent`, `MultiGrid` (siatka wielowarstwowa) oraz `RandomActivation` (harmonogramowanie).
- **Flask (3.0.0):** Mikro-framework webowy służący do udostępniania interfejsu użytkownika oraz API, przez które frontend komunikuje się z modelem symulacyjnym (przesyłanie stanu gry JSON).

- **NumPy (1.24.0):** Biblioteka do obliczeń numerycznych, wykorzystywana do operacji na macierzach reprezentujących mapę kosztów terenu oraz do generowania analitycznych map ciepła (heatmap).
- **Pathfinding (1.0.4):** Implementacja algorytmu A\* (A-Star), służąca do znajdowania optymalnych ścieżek ruchu dla agentów w złożonym terenie z uwzględnieniem kosztów ruchu.
- **PyTMX (3.31):** Biblioteka do parsowania i obsługi plików map stworzonych w programie Tiled Map Editor (.tmx).

### 5.3.3 Technologie webowe (Frontend)

- **HTML5 & CSS3:** Struktura i stylizacja panelu sterowania (Dashboard).
- **JavaScript (ES6+):** Logika klienta, obsługa asynchronicznych zapytań do serwera (Fetch API) oraz sterowanie renderowaniem.
- **Canvas API:** Wykorzystywane do rysowania mapy kafelkowej i sprite'ów agentów w wysokiej wydajności (60 FPS).

## 5.4 Instalacja i uruchomienie

Aplikacja symulacyjna jest oprogramowaniem wieloplatformowym i może być uruchamiana na systemach Windows, Linux oraz macOS. Do poprawnego działania wymaga ona obecności interpretera języka **Python w wersji 3.10** lub nowszej.

Proces instalacji i uruchomienia przebiega następująco:

1. **Instalacja zależności:** Wszystkie biblioteki zewnętrzne wymagane do działania symulacji (m.in. `mesa`, `flask`, `numpy`) zostały zdefiniowane w pliku konfiguracyjnym. Zalecane jest ich instalowanie w wyizolowanym środowisku wirtualnym (`virtualenv`) za pomocą menedżera pakietów `pip`:

```
pip install -r requirements.txt
```

2. **Uruchomienie serwera aplikacji:** Głównym punktem wejścia do programu jest skrypt `app.py`, który inicjalizuje serwer Flask. Należy go uruchomić z poziomu terminala:

```
python app.py
```

Po wykonaniu tego polecenia, w konsoli powinien pojawić się komunikat potwierdzający start serwera deweloperskiego (zazwyczaj: *Running on http://127.0.0.1:5000*).

3. **Obsługa symulacji:** Interfejs graficzny dostępny jest z poziomu przeglądarki internetowej. Należy wpisać w pasku adresu:

```
http://127.0.0.1:5000
```

Po załadowaniu strony użytkownik ma możliwość wyboru scenariusza początkowego oraz warunków pogodowych przed rozpoczęciem właściwej symulacji.

## 5.5 Szczegóły implementacji

### 5.5.1 Struktura projektu

Aplikacja została podzielona na moduły zgodnie z zasadą "Separation of Concerns" (rozdziału odpowiedzialności):

- `app.py`: Główny plik startowy aplikacji. Inicjalizuje serwer Flask i definiuje endpointy API (np. `/get_step`, `/reset_model`).
- `simulation/model.py`: Zawiera klasę `BattleOfZborowModel` dziedziczącą po `mesa.Model`. Odpowiada za inicjalizację świata, wczytanie warstw mapy z pliku TMX i zarządzanie globalnym stanem (np. pogodą).
- `simulation/agent.py`: Zawiera klasę `MilitaryAgent` dziedziczącą po `mesa.Agent`. Definiuje logikę zachowania pojedynczej jednostki, w tym maszyny stanów (`IDLE`, `MOVING`, `ATTACKING`, `FLEEING`).
- `static/`: Katalog z zasobami statycznymi: skrypty JS, style CSS, grafiki sprite'ów (.png) oraz pliki map.
- `templates/`: Pliki widoków HTML renderowane przez silnik Jinja2.



### 5.5.2 Kluczowe algorytmy i mechanizmy

#### 1. Inicjalizacja i obsługa mapy:

Model wykorzystuje bibliotekę PyTMX do wczytania pliku `assets/map/map.tmx`. Warstwa "Teren" jest iterowana, a właściwości kafelków (takie jak `movement_cost`) są konwertowane na macierz NumPy. Macierz ta stanowi podstawę dla siatki nawigacyjnej biblioteki `pathfinding`, gdzie wartości  $>1$  oznaczają teren trudny (las, wzgórze), a wartości bardzo wysokie – przeszkody nie do przebycia.

#### 2. System nawigacji i unikania kolizji:

Każdy agent, chcąc przemieścić się do celu (wroga lub punktu strategicznego), wywołuje metodę `calculate_path()`. Algorytm A\* wyznacza listę współrzędnych (węzłów). Aby uniknąć nakładania się jednostek, zaimplementowano mechanizm dynamicznego blokowania węzłów zajętych przez inne żywe jednostki na czas wyznaczania ścieżki. Jeśli agent napotka zator, aktywuje licznik `repath_timer`, czekając losową liczbę tur przed ponowną próbą znalezienia drogi, co redukuje obciążenie procesora.

#### 3. Pętla symulacyjna i wizualizacja:

Symulacja działa w trybie krokowym. Frontend wysyła żądanie HTTP GET na endpoint `/update`. W odpowiedzi serwer wykonuje metodę `model.step()`, która iteruje po wszystkich agentach, a następnie zwraca zserializowany obiekt JSON zawierający pozycje, typy i stan (HP, morale) wszystkich jednostek. Frontend czyści element `<canvas>` i przerysowuje scenę, co pozwala na płynną animację przebiegu bitwy.

## **6. REZULTATY SYMULACJI**

Przeprowadzono serię eksperymentów symulacyjnych mających na celu zbadanie wpływu warunków pogodowych na wynik starcia oraz weryfikację parametrów bojowych poszczególnych jednostek w izolowanych środowiskach.

Zgodnie z przyjętą metodologią, dla każdego scenariusza głównego oraz eksperymentu syntetycznego wykonano po 20 powtórzeń symulacji, co pozwoliło na zebranie statystycznie istotnych próbek danych.

### **6.1 Wpływ warunków pogodowych (Scenariusz 2: Obrona Wałów)**

Analiza wyników dla Scenariusza 2 wykazała drastyczny wpływ zmiennych środowiskowych na efektywność Armii Koronnej, która w dużej mierze polega na przewadze technologicznej (broń palna, artyleria).

#### **6.1.1 Pogoda słoneczna (Warunki bazowe)**

W warunkach idealnych (`weather="clear"`), Armia Koronna osiągnęła absolutną dominację. Na 20 przeprowadzonych symulacji, strona koronna zwyciężyła we wszystkich 20 przypadkach (100% skuteczności).

- Średnia liczba ocalałych agentów koronnych wynosiła ok. 3-4 jednostki.
- Siły kozackie były zazwyczaj eliminowane zanim doszło do walki w zwarciu na pełną skalę, co potwierdza kluczową rolę zasięgu widzenia i celności broni palnej.

### 6.1.2 Ulewny deszcz

Włączenie modyfikatora `weather="rain"` doprowadziło do wyrównania szans. W 20 symulacjach odnotowano idealny podział wyników:

- **10 zwycięstw Armii Koronnej (50%)**
- **10 zwycięstw Kozaków/Tatarów (50%)**

Deszcz, poprzez redukcję mobilności i karę do obrażeń dystansowych, pozwolił siłom kozackim na częstsze doprowadzanie do walki wręcz, niwelując przewagę ogniową obrońców.

### 6.1.3 Gęsta mgła

Scenariusz z mgłą (`weather="fog"`) okazał się najtrudniejszy dla strony koronnej. Ograniczenie zasięgu widzenia (Line of Sight) uniemożliwiło efektywne wykorzystanie artylerii i muszkieterów na dystans.

- **Armia Koronna zwyciężyła tylko 4 razy (20%).**
- **Kozacy/Tatarzy zwyciężyli aż 16 razy (80%).**

Jest to odwrócenie trendu z pogody słonecznej, wskazujące, że w modelu widoczność jest parametrem krytycznym dla armii opartej na sile ognia.

Tabela 6.1: Zestawienie wyników Scenariusza 2 w zależności od pogody (na 20 prób).

Warunki	Zwycięstwa Korony	Zwycięstwa Kozaków	Dominacja
Słonecznie	20 (100%)	0 (0%)	Całkowita (Korona)
Deszcz	10 (50%)	10 (50%)	Brak (Równowaga)
Mgła	4 (20%)	16 (80%)	Znaczna (Kozacy)

## 6.2 Eksperymenty syntetyczne

W celu głębszego zrozumienia mechaniki starć, przeprowadzono trzy izolowane eksperymenty.

### 6.2.1 Eksperyment: Siła Ognia (Firepower)

Symulacja starcia Piechoty Kozackiej (15 jednostek) przeciwko Piechocie Niemieckiej (10 jednostek).

- **Wynik:** 16 zwycięstw Kozaków (80%) vs 4 zwycięstwa Korony (20%).
- **Wniosek:** Mimo wyższej jakości wyszkolenia Piechoty Niemieckiej, przewaga liczebna (15 do 10) po stronie kozackiej okazała się decydująca w wymianie ogniowej.

### 6.2.2 Eksperyment: Mobilność (Mobility)

Symulacja szarży Jazdy Tatarskiej (20 jednostek) na pozycje Dragonii (10 jednostek).

- **Wynik:** 18 zwycięstw Dragonii (90%) vs 2 zwycięstwa Tatarów (10%).
- **Wniosek:** Dragonia, jako jednostka hybrydowa (walcząca wręcz i na dystans), wykazała się niezwykle skuteczną. Mimo dwukrotnej przewagi liczebnej wroga, Dragoni byli w stanie odeprzeć atak, prawdopodobnie dzięki wyższym statystykom defensywnym i sile ognia w bliskim kontakcie.

### 6.2.3 Eksperyment: Jakość vs Ilość (Quality vs Quantity)

Test legendy Husarii: 5 jednostek Husarii przeciwko 40 jednostkom Czerni (stosunek 1:8).

- **Wynik:** 19 zwycięstw Husarii (95%) vs 1 zwycięstwo Czerni (5%).
- **Wniosek:** Jest to najbardziej jednostronny wynik spośród eksperymentów syntetycznych. Potwierdza on, że w zaimplementowanym modelu jednostki elitarne są w stanie pokonać nawet ośmiokrotnie liczniejszego przeciwnika o niskim morale i słabym uzbrojeniu.

## 7. DYSKUSJA WYNIKÓW

### 7.1 Interpretacja wpływu pogody na wynik bitwy

Uzyskane wyniki jednoznacznie wskazują, że czynnik pogodowy w modelu nie jest jedynie "kosmetycznym" dodatkiem, ale zmienną fundamentalnie redefiniującą balans sił.

1. **Dylemat Mgła vs Deszcz:** Co ciekawe, wyniki pokazują, że mgła jest znacznie groźniejsza dla Armii Koronnej (80% porażek) niż deszcz (50% porażek).

- Deszcz nakłada kary do mobilności i obrażeń, co spowalnia grę, ale nadal pozwala na prowadzenie walki dystansowej, choć mniej efektywnej.
- Mgła drastycznie ogranicza *Field of View* (pole widzenia). Dla jednostek AI oznacza to, że artyleria i strzelcy nie "widzą" celu, dopóki ten nie znajdzie się w bezpośrednim sąsiedztwie. Pozwala to Kozakom na bezkarne podejście do wałów i narzucenie walki w zwarciu, w której ich przewaga liczebna jest decydująca.

2. **Stabilność w warunkach idealnych:** 100% wskaźnik zwycięstw w pogodzie słonecznej sugeruje, że przy dobrej widoczności parametry jednostek koronnych (zasięg, obrażenia) są wystarczające, by zniwelować przewagę liczebną wroga. Jest to zgodne z historyczną doktryną "ognia i żelaza", gdzie siła ognia miała łamać morale wroga przed zwarem.

### 7.2 Analiza jednostek w świetle eksperymentów

Eksperymenty syntetyczne rzuciły nowe światło na balans poszczególnych klas agentów:

### 7.2.1 Fenomen Dragonii

W eksperymencie mobilności Dragoni pokonali dwukrotnie liczniejszą Jazdę Tatarską w 90% przypadków. Wskazuje to, że w modelu Dragonia jest jedną z najbardziej opłacalnych jednostek (cost-effective). Ich zdolność do przetrwania zmasowanego ataku lekkiej kawalerii sugeruje wysokie parametry wytrzymałości (HP) lub pancerza w połączeniu z siłą ognia.

### 7.2.2 Mit Husarii potwierdzony

Wynik 95% zwycięstw w starciu 5 Husarzy na 40 Czerni pokazuje, że model poprawnie symuluje efekt "shock cavalry". Czern, jako jednostka o niskich statystykach, prawdopodobnie nie jest w stanie zadawać obrażeń przebijających pancerz Husarii, podczas gdy każde uderzenie Husarza jest śmiertelne. Oznacza to, że dla gracza (dowódcy) inwestycja w drogich, elitarnych agentów jest bardziej opłacalna niż rekrutacja masowa słabych jednostek.

### 7.2.3 Krytyczna masa piechoty

W starciu ogniowym (Piechota Niemiecka vs Kozacka) jakość przegrała z ilością (wynik 4:16). Oznacza to, że w czystej wymianie ognia w modelu, po przekroczeniu pewnego progu liczebnego, nawet lepsze wyszkolenie (Niemcy) nie rekompensuje mniejszej liczby łuf (Kozacy). Jest to istotna obserwacja taktyczna – piechota zaciężna wymaga wsparcia innych formacji, by być efektywną.

## 7.3 Ograniczenia modelu i anomalie

Mimo spójnych wyników, zauważono pewne ograniczenia symulacji:

- **Brak wpływu morale w eksperymencie jakość/ilość:** Mimo że Husaria wygrywała, walki trwały stosunkowo długo (średnio ok. 200-300 kroków). W rzeczywistości szarża Husarii na Czern często kończyła się natychmiastową paniką tej drugiej. Model kładzie większy nacisk na eliminację punktów życia (HP) niż na psychologię tłumu.
- **Skalowanie mgły:** 20% zwycięstw Korony we mgle może wydawać się historycznie zbyt niskie, biorąc pod uwagę, że obrońcy wałów znali przedpole. Model

traktuje mgłę jako "absolutną ślepotę" dla AI, nie uwzględniając strzelania w strefy (area denial) bez widocznego celu.

## **7.4 Podsumowanie**

Symulacja wykazała, że warunki pogodowe są kluczowym czynnikiem determinującym wynik Bitwy pod Zborowem. Model poprawnie odwzorował relacje sił: dominację technologiczną Korony w dobrych warunkach oraz przewagę masy Kozackiej w warunkach ograniczonej widoczności.

## 8. PODSUMOWANIE

W ramach projektu udało się zrealizować kompletną, działającą symulację bitwy historycznej. Osiągnięto wszystkie założone cele:

1. Stworzono wiarygodny model jednostek historycznych.
2. Zaimplementowano wpływ środowiska na przebieg walki (deszcz, teren).
3. Dostarczono narzędzia analityczne potwierdzające tezę, że warunki pogodowe mogły odwrócić losy bitwy pod Zborowem.

Projekt pokazał, że narzędzia symulacji systemów dyskretnych (Mesa) mogą być z powodzeniem stosowane w cyfrowej humanistyce i analizie historycznej.

Jako kierunek dalszego rozwoju wskazać można implementację algorytmów grupowych (boids) dla lepszego odwzorowania ruchu formacji oraz wprowadzenie trybu multiplayer, w którym użytkownicy mogliby wydawać rozkazy w czasie rzeczywistym, zmieniając cele strategiczne agentów.



# Bibliografia

- Mandzy, Adrian. "Tatars, Cossacks, and the Polish Army: The Battle of Zboriv." W *Fields of Conflict: Battlefield Archeology from the Roman Empire to the Korean War*, red. Douglas Scott, Lawrence Babits, and Charles Haecker, 193–207. Potomac Books, 2009.
- Mandzy, Adrian O. "THE 1649 BATTLEFIELD OF ZBORIV: IDENTIFICATION OF PLACE AND PLANNING RECONSTRUCTION." *Architectural Studies* 5, nr 2 (2019).
- Kucharski, Wojciech. "Bitwa pod Zborowem 15–16 Sierpnia 1649 r." *Studia z Dziejów Wojskowości* VI (2017): 70-111.