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

KOD KURSU: 120-ISI-1S-777
PRZEDMIOT: SYMULACJA SYSTEMÓW DYSKRETNYCH (SSD)

Autorzy projektu

Patrick Bajorski

Jan Banasik

Gabriel Filipowicz

Opiekun projektu

prof. dr hab. inż. Jarosław Wąs

EAIiIB / Katedra Informatyki Stosowanej

Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie

Kraków, Polska

09 stycznia 2026

Spis treści

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 spośób modułowy. Bazą dla tego szablonu jest wersja oryginalna dostepna 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 definujące zawartośc 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 BibTEX.

`report.tex` –: Makro struktura raportu. Jesli 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. Poniższe tabele prezentują kompletny zestaw danych.

Tabela 3.1: Podstawowe atrybuty jednostek (HP, Morale, Dyscyplina, Obrona, Szybkość).

Jednostka	**HP**	**Morale**	**Dysc.**	**Obrona**	**Szybkość**
Armia Koronna					
Husaria	150	140	95	8	6
Pancerni	120	110	85	5	7
Rajtaria	110	100	90	6	6
Dragonia	100	95	85	4	5
Piechota Niemiecka	110	100	95	6	3
Pospolite Ruszenie	90	50	20	2	6
Czeladź Obozowa	60	90	40	0	5
Artyleria Koronna	50	90	90	0	1
Kozacy/Tatarzy					
Jazda Tatarska	85	80	70	1	9
Piechota Kozacka	115	110	80	3	4
Jazda Kozacka	100	90	75	3	7
Czerń	70	60	40	0	5
Artyleria Kozacka	40	80	80	0	1

Tabela 3.2: Zdolności bojowe jednostek (Atak, Zasięg, Amunicja, Szybkostrzelność).

Jednostka	Atak Wręcz	Atak Dyst.	Zasięg	Ammo	RoF	Uwagi
Armia Koronna						
Husaria	100	0	1	0	1.0	Elitarna ciężka jazda przełamująca
Pancerni	70	0	1	0	1.0	Jazda średniozbrojna, uniwersalna
Rajtaria	40	30	3	12	0.8	Ciężka jazda z bronią palną
Dragonia	30	25	4	15	1.2	Mobilna piechota konna
Piechota Niemiecka	25	35	5	20	1.3	Wysoka dyscyplina, silny ogień
Pospolite Ruszenie	20	10	2	5	0.8	Niska dyscyplina, podatność na panikę
Czeladź Obozowa	25	0	1	0	1.0	Słabo uzbrojona, zdeterminowana
Artyleria Koronna	5	150	15	30	0.25	Potężna siła ognia, bardzo wolna
Kozacy/Tatarzy						
Jazda Tatarska	30	15	4	40	1.8	Szybcy łucznicy
Piechota Kozacka	35	35	5	25	1.4	Znakomici strzelcy
Jazda Kozacka	50	0	2	0	1.0	Jazda średnia
Czerń	20	0	1	0	1.0	Liczni, słabo uzbrojeni
Artyleria Kozacka	5	130	14	25	0.25	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 x2.5.
- 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

if $A.HP \leqslant 0$ **then**
  return
end if
Zmniejsz $A.fire\_cooldown$ jeśli > 0
$PanicThreshold \leftarrow 25 - (A.Discipline/5)$
if $A.Morale < PanicThreshold$ **and** $A.State \neq FLEEING$ **then**
  if $Random(0, 100) > A.Discipline$ **then**
    $A.State \leftarrow FLEEING$
    Call MANAGEFLEEING(A)
  end if
end if
if $A.State == FLEEING$ **then**
  if brak ścieżki **and** Frakcja == "Armia Koronna" **then**
    Call MANAGEFLEEING(A) {Przelicz cel ucieczki}
  end if
  Call MOVE(A)
  return
end if
$Enemy \leftarrow$ FINDENEMY(A, radius=$W.visibility$)
if $Enemy$ istnieje **then**
  $Dist \leftarrow$ DISTANCE(A, $Enemy$)
  if $1 < Dist \leqslant A.Range$ **and** $A.RangedDmg > 0$ **then**
    $Misfire \leftarrow 0.4$ jeśli $W == Rain$ wpp. 0.0
    if $A.Ammo > 0$ **then**
      if $A.Cooldown \leqslant 0$ **and** $Random() > Misfire$ **then**
        $A.State \leftarrow ATTACKING$
        $A.Ammo \leftarrow A.Ammo - 1$
        DEALDAMAGE($Enemy$, $A.RangedDmg$, TerrainModifier)
      end if
    else
      PATHTO($Enemy$) {Brak amunicji, szarża}
      MOVE(A)
    end if
  else if $Dist \leqslant 1.5$ **then**
    $A.State \leftarrow ATTACKING$
    DEALDAMAGE($Enemy$, $A.MeleeDmg$) {Walka wręcz}
  else
    PATHTO($Enemy$)
    MOVE(A)
  end if
else
  $A.State \leftarrow MOVING\_TO\_STRATEGIC$
  PATHTO($A.StrategicTarget$)
  MOVE(A)
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

$CurrentPos \leftarrow A.Position$
$DistToEdge \leftarrow$ minimum dystansu do krawędzi (L, R, T, B)
if $A.Faction ==$ "Armia Koronna" **then**
  $TargetCenter \leftarrow$ Najbliższe **niezapełnione** Centrum Leczenia
  $ShouldFleeToEdge \leftarrow True$
  if $TargetCenter$ istnieje **then**
    $DistToCenter \leftarrow$ DISTANCE($CurrentPos$, $TargetCenter$)
    if $DistToCenter \leqslant 2 \times DistToEdge$ **then**
      $ShouldFleeToEdge \leftarrow False$
    end if
  end if
  if NOT $ShouldFleeToEdge$ **then**
    $Entrance \leftarrow$ GETENTRANCE($TargetCenter$)
    if $CurrentPos$ na $Entrance$ **then**
      Wejdź głębiej w strefę leczenia
    else
      CALCULATEPATH($Entrance$)
    end if
  else
    CALCULATEPATH(Najbliższa Krawędź Mapy)
  end if
else
  CALCULATEPATH(Najbliższa Krawędź Mapy)
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

sprzętową, 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 wymagana jest obecność 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 maszynę 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, wzgórza), 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:** Z wyłączeniem wyższej dyscypliny Piechoty Niemieckiej (95 vs 80), parametry bojowe obu jednostek są zbliżone (np. Niemcy: HP 110, Speed 3, Rate of Fire 1.3; Kozacy: HP 115, Speed 4, Rate of Fire 1.4). Pomimo wyższej dyscypliny Piechoty Niemieckiej, przewaga liczebna (15 do 10) po stronie kozackiej, w połączeniu z minimalnie wyższą szybkostrzelnością oraz szybkością, okazała się decydująca w otwartej 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, wykazała się niezwykłą skutecznością. Kluczowa okazała się różnica w wytrzymałości i pancerzu: Dragonia posiada **100 HP i Obronę 4**, podczas gdy Tatarzy mają tylko **85 HP i Obronę 1**. Pozwoliło to Dragonom przetrwać ostrzał i wygrać starcie mimo dwukrotnej przewagi liczebnej wroga.

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. Wynika on z dysproporcji statystyk: Husaria zadaje **100 pkt obrażeń** (natychmiastowa eliminacja Czerni mającej 70 HP), podczas gdy Czerń zadaje tylko 20 pkt obrażeń, dodatkowo redukowanych przez wysoką Obronę Husarii (8). Potwierdza to, że jednostki elitarne są w stanie pokonać nawet ośmiokrotnie liczniejszego, ale słabego przeciwnika.

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 (Speed -3 dla kawalerii) i obrażeń dystansowych (mnożnik 0.3), co spowalnia grę, ale nadal pozwala na prowadzenie walki dystansowej, choć mniej efektywnej.
    - Mgła drastycznie ogranicza *Field of View* (pole widzenia) z 20 do 6 pól. 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 zwarciem.

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 wynika z wysokich parametrów defensywnych (HP 100, Obrona 4) w porównaniu do kruchej Jazdy Tatarskiej (HP 85, Obrona 1).

7.2.2 Mit Husarii potwierdzony

Wynik 95% zwycięstw w starciu 5 Husarzy na 40 Czerni pokazuje, że model poprawnie symuluje efekt "shock cavalry". Czerń, jako jednostka o niskich statystykach (Melee Damage 20), zadaje obrażenia, które są skutecznie redukowane przez pancerz Husarii (Obrona 8). Z kolei Husaria zadaje 100 pkt obrażeń, co przy 70 HP Czerni oznacza eliminację przeciwnika jednym uderzeniem (one-shot). 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 wyższa dyscyplina (Niemcy: 95) nie rekompensuje mniejszej liczby luf (Kozacy). Co więcej, Piechota Kozacka w modelu posiada minimalnie wyższy współczynnik szybkostrzelności (1.4 vs 1.3), co przy przewadze liczebnej przesądziło o wyniku. 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 Czerń 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, a przeprowadzone eksperymenty dostarczyły istotnych danych analitycznych:

1. **Wiarygodny model jednostek:** Zaimplementowano 13 typów jednostek o unikalnych parametrach. Eksperymenty syntetyczne potwierdziły poprawność balansu, wykazując m.in. dominację elitarnej Husarii nad liczniejszą Czernią (95% zwycięstw) oraz wysoką efektywność Dragonii w starciu z lekką jazdą.
2. **Wpływ środowiska:** Potwierdzono hipotezę o kluczowym wpływie pogody na losy bitwy. Symulacje wykazały, że w warunkach idealnych Armia Koronna wygrywa w 100% przypadków, podczas gdy gęsta mgła odwraca te proporcje, dając 80% zwycięstw stronie kozackiej poprzez neutralizację przewagi ogniowej.
3. **Narzędzia analityczne:** Stworzono system zbierania danych i wizualizacji (dashboard, heatmap), który pozwolił na ilościową ocenę starć, w tym analizę wpływu "mgły wojny" i terenu na efektywność poszczególnych formacji.

Projekt pokazał, że narzędzia symulacji systemów dyskretnych (Mesa) mogą być z powodzeniem stosowane w cyfrowej humanistyce do weryfikacji hipotez historycznych. Wyniki eksperymentów "Jakość vs Ilość" oraz "Siła Ognia" dostarczyły ciekawych obserwacji na temat taktyki XVII-wiecznej, wskazując na granice efektywności dyscypliny w starciu z przewagą liczebną.

Jako kierunek dalszego rozwoju wskazać można implementację bardziej zaawansowanych 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. Wartościowe byłoby również pogłębienie modelu psychologicznego o mechanikę paniki łańcuchowej.

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.