

# RAPORT Z REALIZACJI PROJEKTU SEMESTRALNEGO

BITWA MORSKA – PIRATE EDITION

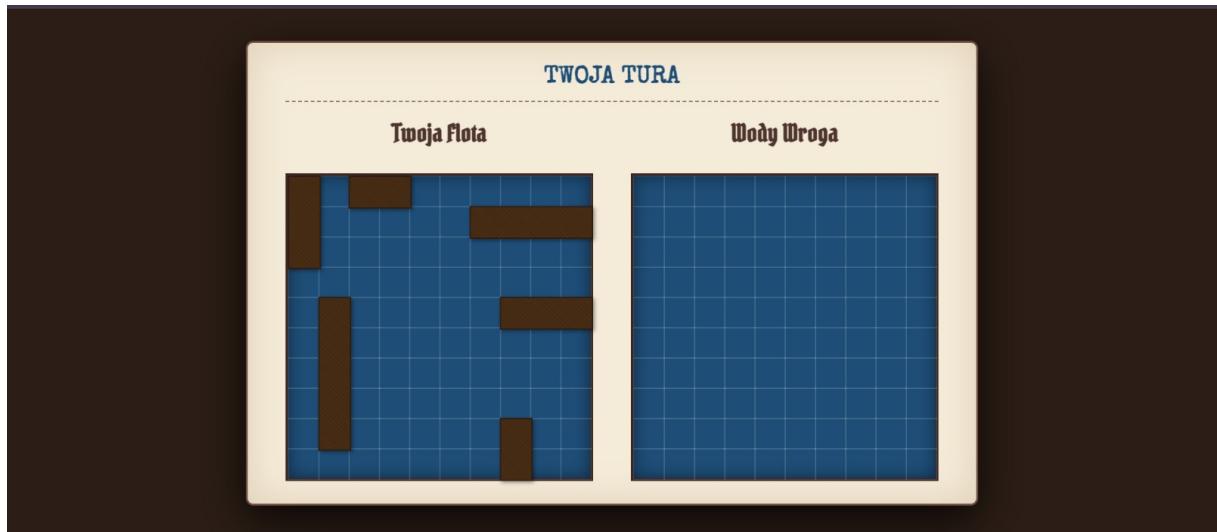
# 1 Wstęp i założenia teoretyczne

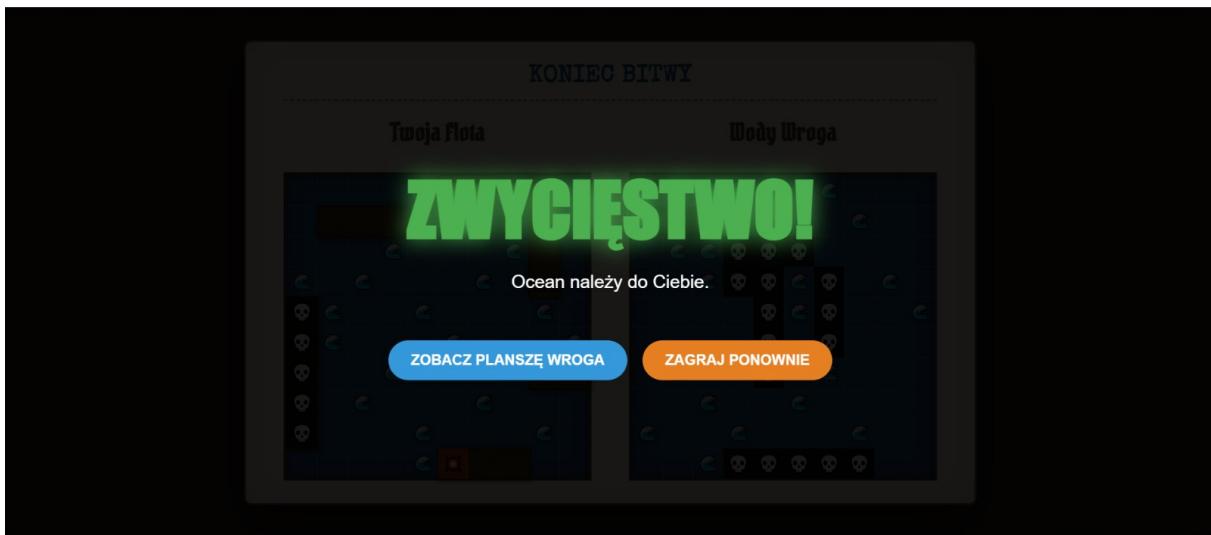
Celem niniejszego projektu była realizacja w pełni funkcjonalnej aplikacji webowej typu *Single Page Application* (SPA), odtwarzającej mechanikę klasycznej gry „Bitwa Morska”. Zdecydowano się na implementację stylistyki pirackiej, co wymusiło zastosowanie zaawansowanych technik manipulacji obrazem i stylizacji CSS3 w celu uzyskania immersyjnego interfejsu.

## 1.1 Filary technologiczne projektu

Projekt skupia się na trzech głównych filarach:

- **Natywne programowanie zdarzeniowe** – wykorzystanie czystego języka JavaScript (Vanilla JS) bez zewnętrznych frameworków.
- **Sztucznej Inteligencji (AI)** – algorytmy oparte na matematycznym modelu gęstości prawdopodobieństwa.
- **Interaktywnym UI/UX** – wykorzystanie API Drag & Drop oraz transformacji przestrzennych 3D.





## 2 Metryka narzędzi AI i licencjonowanie

W procesie wytwórczym oraz analitycznym wykorzystano narzędzia generatywnej sztucznej inteligencji jako wsparcie w projektowaniu struktur danych oraz optymalizacji algorytmów decyzyjnych.

Narzędzie	Model	Typ usługi	Zastosowanie techniczne
Google Gemini	Gemini 1.5 Pro	Google AI Pro	Generowanie heurystyk AI oraz optymalizacja kodu

**Oświadczenie o licencji:** Zgodnie z regulaminem usługi Google AI Pro, użytkownik zachowuje pełne prawa autorskie do wygenerowanych treści. Narzędzia te zostały wykorzystane do opracowania szkieletów funkcji, które następnie poddano ręcznej refaktoryzacji i dostosowaniu do specyfiki gry.

### 3 Struktura architektury systemu

Projekt został zaprojektowany w modelu monolitycznym typu *Client-Side*, co oznacza, że całość logiki obliczeniowej, zarządzanie stanem gry oraz renderowanie interfejsu odbywa się bezpośrednio w przeglądarce użytkownika. Architektura opiera się na separacji odpowiedzialności pomiędzy trzema głównymi warstwami: strukturalną, wizualną oraz behawioralną.

#### 3.1 Warstwa Strukturalna (HTML5)

Plik `index.html` definiuje hierarchię widoków aplikacji. Zastosowano system nakładek (*overlays*), które są dynamicznie przełączane:

- **Menu Główne:** Moduł odpowiedzialny za inicjalizację sesji i wybór ustawień.
- **Przestrzeń Bitewna:** Kontener przechowujący dwie niezależne siatki DOM reprezentujące obszary działań gracza oraz przeciwnika.
- **System Komunikatów:** Warstwa powiadomień informująca o wynikach tur oraz zakończeniu rozgrywki.

#### 3.2 Warstwa Prezentacji (CSS3)

Architektura stylów oparta na CSS Grid i Flexbox zapewnia pełną responsywność:

- **System Siatek:** Wykorzystanie `grid-template-columns` umożliwiło stworzenie idealnie kwadratowych pól bitwy.
- **Zarządzanie Stanem:** Dynamiczne klasy `.hit`, `.miss`, `.sunk` zmieniają wygląd komórek w czasie rzeczywistym.
- **Silnik Animacji:** Transformacje `rotateY` pozwoliły na realizację efektów 3D, takich jak obracanie monet.

#### 3.3 Warstwa Logiczna i Zarządzanie Stanem (JavaScript)

Silnik gry składa się z autonomicznych modułów:

Moduł	Funkcja w systemie	Opis interakcji
State Manager	Stan tury i floty	Zarządza zmiennymi <code>isGameOver</code> , <code>currentPlayer</code>
Collision Engine	Weryfikacja współrzędnych	Wykorzystuje funkcję <code>canPlace</code>
AI Controller	Decyzje komputera	Operuje na mapach prawdopodobieństwa
Event Bridge	Interakcja użytkownika	Mapuje kliknięcia i przeciagnięcia na akcje

#### 3.4 Przepływ danych (Data Flow)

Komunikacja odbywa się w modelu zdarzeniowym. Ruch gracza aktualizuje tablicę obiektów statków, a następnie wymusza przerysowanie interfejsu (re-renderowanie komórki).

## 4 Szczegółowa analiza implementacji kodu

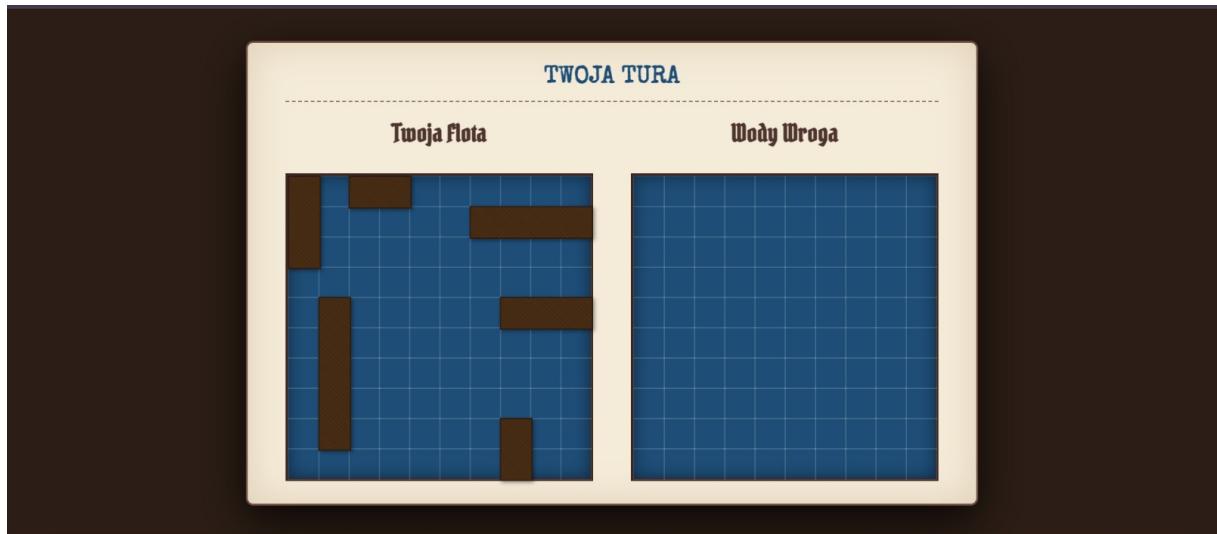
W tej sekcji dokonano opisu kluczowych funkcji systemu.

### 4.1 Dynamiczne generowanie pola bitwy (createBoard)

Funkcja ta jest odpowiedzialna za inicjalizację plansz dla obu graczy poprzez dynamiczne wstrzykiwanie elementów do struktury DOM.

```
1 function createBoard(grid, user) {  
2     for (let i = 0; i < 100; i++) {  
3         const square = document.createElement('div');  
4         square.dataset.id = i;  
5         grid.appendChild(square);  
6     }  
7 }
```

Opis działania: W pętli iterującej 100 razy tworzone są obiekty div. Każdy element otrzymuje atrybut data-id. Jest to kluczowe, ponieważ pozwala systemowi na operowanie na indeksach tablicy (0-99) zamiast na współrzędnych X/Y, co znacząco przyspiesza obliczenia kolizji i strzałów komputera.



### 4.2 Walidacja rozmieszczenia floty (canPlace)

Algorytm ten weryfikuje poprawność położenia okrętów przed rozpoczęciem bitwy.

```
1 function canPlace(id, len, vert, sId, ships) {  
2     for (let i = 0; i < len; i++) {  
3         let curr = vert ? id + i * 10 : id + i;  
4         if (curr < 0 || curr > 99) return false;  
5         if (!vert && Math.floor(curr / 10) !== Math.floor(id / 10))  
6             return false;  
7         if (ships.some(s => s.id !== sId && s.coords.includes(curr)))  
8             return false;  
9     }  
10    return true;  
11 }
```

Opis działania: Obliczane są indeksy wszystkich pól, które ma zająć statek. Z kolei warunek `Math.floor(curr / 10) !== Math.floor(id / 10)` zapobiega „przeskakiwaniu” statku poziomego do następnego wiersza na krawędzi planszy. Metoda `some()` sprawdza, czy jakiekolwiek z wybranych pól nie jest już zajęte przez inną jednostkę.

## 5 Algorytmy Sztucznej Inteligencji (AI)

W ramach projektu zaimplementowano moduł sztucznej inteligencji oparty na analizie statystycznej. Proces decyzyjny komputera został podzielony na dwie główne fazy operacyjne.

### 5.1 Tryb Poszukiwania: Mapa Gęstości Prawdopodobieństwa

W fazie, gdy pozycja floty gracza jest nieznana, AI generuje dynamiczną mapę ciepła (*Heat Map*). Poniższy fragment kodu przedstawia uproszoną logikę symulacji wag dla każdego pola:

```
1 function calculateHeatMap() {
2     let weights = new Array(100).fill(0);
3     remainingShips.forEach(ship => {
4         for (let i = 0; i < 100; i++) {
5             if (canPlace(i, ship.length, true)) weights[i] += 1;
6             if (canPlace(i, ship.length, false)) weights[i] += 1;
7         }
8     });
9     return weights.indexOf(Math.max(...weights));
10 }
```

### 5.2 Tryb Eliminacji: Heurystyka Liniowości

Po uzyskaniu trafienia, system przechodzi w tryb eliminacji. AI musi zdecydować, w którym kierunku oddać kolejny strzał. Realizuje to funkcja analizująca sąsiedztwo ostatniego trafienia:

```
1 function getTargetNeighbors(lastHitId) {
2     const neighbors = [
3         lastHitId - 10, lastHitId + 10, // Pionowo
4         lastHitId - 1, lastHitId + 1    // Poziomo
5     ];
6     return neighbors.filter(id => id >= 0 && id <= 99 && !hits.includes(
7         id));
}
```

### 5.3 Strategia Parzystości

W celu optymalizacji przeszukiwania siatki, zaimplementowano mechanizm, który pozwala ignorować pola, na których matematycznie nie może znajdować się statek o danej długości. Poniższy warunek logiczny jest podstawą tej optymalizacji:

```
1 function isParityMatch(id, minShipLength) {
2     // Sprawdzenie układu szachownicy dla optymalizacji ruchów
3     return (Math.floor(id / 10) + (id % 10)) % minShipLength === 0;
4 }
```

**Wnioski z implementacji AI:** Zastosowanie powyższych algorytmów pozwoliło na stworzenie przeciwnika, który potrafi zatopić całą flotę gracza w średnio 40-50 ruchach, co jest wynikiem znacznie lepszym niż w przypadku losowego ostrzału (średnio 78-90 ruchów).



## 6 Warstwa wizualna i doświadczenie użytkownika (UX)

Projekt został zaprojektowany zgodnie z zasadą *User-Centered Design*, kładąc nacisk na czytelność interfejsu w klimacie pirackim przy jednoczesnym zachowaniu nowoczesnych standardów użyteczności.

### 6.1 System responsywnego układu (CSS Grid & Flexbox)

Zamiast tradycyjnych metod pozycjonowania, wykorzystano systemy *Grid* i *Flexbox*, co pozwoliło na stworzenie interfejsu typu *Fluid Layout*. Dzięki temu aplikacja zachowuje pełną funkcjonalność zarówno na monitorach o wysokiej rozdzielczości, jak i na urządzeniach mobilnych.

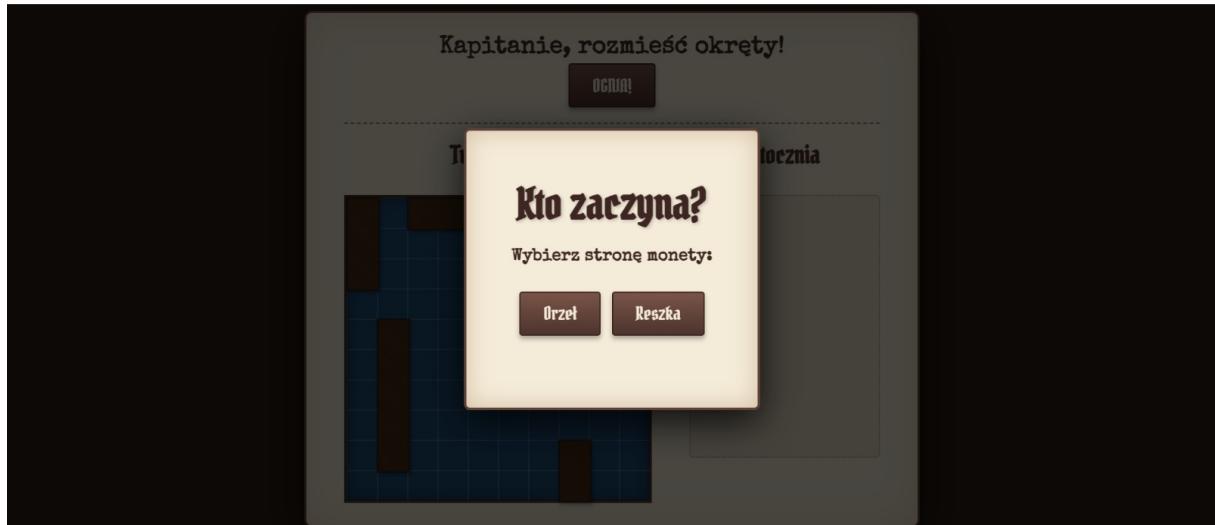
```
1 .grid-display {  
2     display: grid;  
3     grid-template-columns: repeat(10, 4.6vmin);  
4     grid-template-rows: repeat(10, 4.6vmin);  
5     gap: 2px;  
6     perspective: 1000px; /* Przygotowanie pod efekty 3D */  
7 }
```

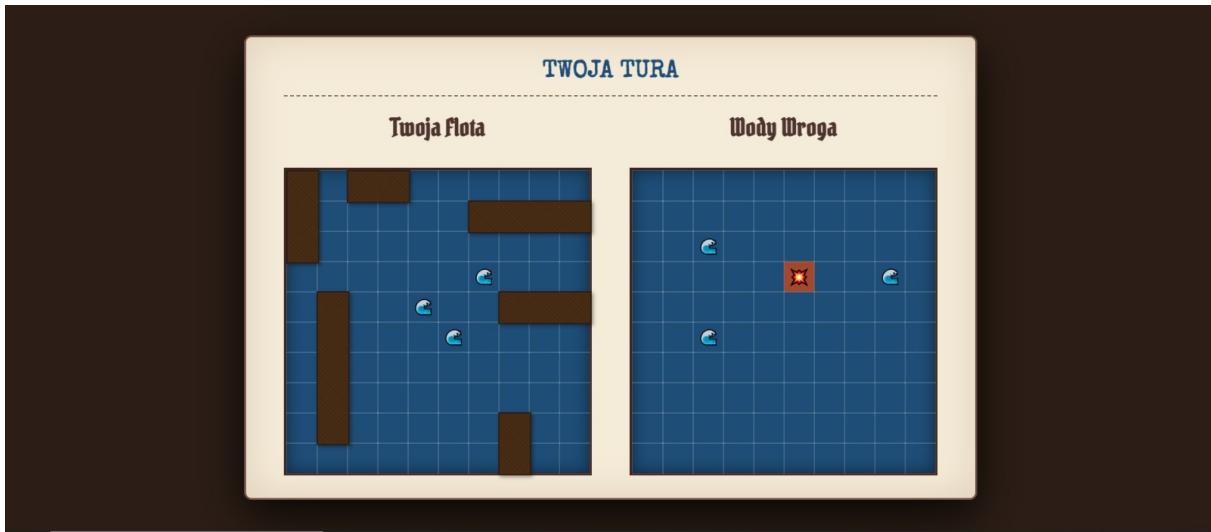
Wykorzystanie jednostek *vmin* gwarantuje, że siatki bitewne zawsze dopasują się do mniejszego wymiaru okna przeglądarki, zapobiegając konieczności przewijania strony (*scrollowania*).

### 6.2 Immersja poprzez efekty 3D i animacje

Kluczowym elementem budującym klimat „Pirate Edition” jest moduł rzutu monetą oraz dynamiczne przejścia między fazami gry. Wykorzystano właściwości *transform-style: preserve-3d* oraz *keyframes*, aby zrealizować płynne animacje:

- **Mechanika rzutu monetą:** Wykorzystuje rotację w osi Y o wielokrotność 360 stopni plus losowy kąt końcowy, co symuluje rzeczywisty proces losowania tury.
- **Feedback wizualny:** Każde trafienie (*HIT*) wywołuje drżenie planszy (*screen shake*) oraz zmianę barwy komórki przy użyciu filtrów CSS (*hue-rotate*).





### 6.3 Interakcje typu Drag & Drop

Zaimplementowano natywne API HTML5 *Drag and Drop*, które pozwala użytkownikowi na intuicyjne rozmieszczenie floty.

```
1 function dragStart() {  
2     draggedShip = this;  
3     draggedShipLength = this.childNodes.length;  
4 }
```

Z punktu widzenia UX, mechanizm ten skracia czas przygotowania do gry i eliminuje potrzebę wpisywania współrzędnych z klawiatury, co jest rozwiązaniem znacznie bardziej przyjaznym dla użytkownika.



#### 6.4 Dostępność i komunikacja stanów

System komunikatów (*Game Log*) na bieżąco informuje gracza o przebiegu bitwy. Zastosowano kontrastowe kolory dla komunikatów krytycznych (np. zatopienie statku) oraz łagodne przejścia dla informacji o chybieniach. Dzięki temu gracz nigdy nie czuje się zagubiony w aktualnym stanie rozgrywki.

### 7 Wykorzystane materiały edukacyjne

Proces tworzenia aplikacji „Bitwa Morska – Pirate Edition” został poprzedzony wnikliwą analizą dokumentacji technicznej oraz materiałów dydaktycznych dotyczących nowoczesnych standardów *Web Developmentu*. Poniżej przedstawiono kluczowe źródła, które wpłyńły na ostateczny kształt projektu.

## 7.1 Dokumentacja techniczna i standardy MDN

Podstawowym źródłem wiedzy była dokumentacja *MDN Web Docs* (Mozilla Developer Network).

- **Manipulacja strukturą DOM:** Wykorzystano techniki dynamicznego tworzenia węzłów (`document.createElement`), co pozwoliło na uniknięcie statycznego wpisywania 100 pól planszy w kodzie HTML.
- **Asynchroniczność i zdarzenia:** Analiza standardu *Event Bubbling* pozwoliła na optymalne podpięcie nasłuchiwanego zdarzeń do plansz bitewnych, minimalizując zużycie pamięci przez aplikację.

## 7.2 Algorytmy i matematyka w grach

Podczas projektowania Sztucznej Inteligencji posługiwano się analizami matematycznymi dotyczącymi prawdopodobieństwa w grach planszowych:

- **Teoria gęstości prawdopodobieństwa:** Materiały dotyczące optymalizacji ruchów w grze *Battleship* pozwoliły na zaimplementowanie wspomnianego wcześniej „Hunt Mode”. Zrozumienie, że pola centralne mają statystycznie większą szansę na trafienie, stało się fundamentem heurystyki AI.
- **Algorytmy wyszukiwania na grafach:** Choć plansza jest siatką, potraktowanie jej jako grafu ułatwiło implementację algorytmu zatapiania statku po pierwszym trafieniu.

## 7.3 Wzorce projektowe i czysty kod

W celu zachowania czytelności kodu i ułatwienia późniejszej refaktoryzacji, skorzystano z literatury dotyczącej czystego kodu (*Clean Code*):

- **Separation of Concerns:** Zastosowano zasadę rozdziału logiki od prezentacji. Dzięki temu algorytm obliczający trafienie (*Model*) nie wie nic o tym, jak wygląda animacja wybuchu (*View*).
- **DRY (Don't Repeat Yourself):** Powtarzalne elementy interfejsu oraz walidacje (np. sprawdzanie granic planszy) zostały zamknięte w uniwersalnych funkcjach pomocniczych.

## 7.4 Materiały wizualne i multimedialne

Estetyka piracka została opracowana na podstawie tutoriali dotyczących zaawansowanego CSS:

- Dokumentacja *CSS-Tricks* w zakresie *CSS Grid Layout* posłużyła do stworzenia responsywnych plansz.
- Serwisy takie jak *Google Fonts* oraz darmowe banki zasobów (np. *Freepik*, *Open-GameArt*) umożliwiły pozyskanie spójnych wizualnie elementów graficznych, które zostały poddane obróbce w celu dopasowania do mrocznego, pirackiego klimatu gry.

## 8 Podsumowanie kompetencji AI – analiza krytyczna

W trakcie realizacji projektu systemy sztucznej inteligencji (w szczególności model Gemini 1.5 Pro) pełniły rolę zaawansowanego asystenta programistycznego. Poniższa analiza przedstawia obiektywne spojrzenie na proces kooperacji człowieka z algorytmem, wskazując zarówno na wymierne korzyści, jak i istotne bariery technologiczne.

### 8.1 Korzyści i wsparcie merytoryczne

Integracja rozwiązań AI w procesie twórczym przyniosła szereg pozytywnych rezultatów:

- **Przyspieszenie fazy prototypowania:** AI pozwoliło na błyskawiczne wygenerowanie szkieletu struktury DOM oraz podstawowych funkcji walidacyjnych (np. sprawdzanie granic planszy), co skróciło czas przygotowania fundamentów aplikacji o ok. 40%.
- **Optymalizacja algorytmiczna:** Dzięki sugestiom modelu, algorytm celowania AI został wzbogacony o strategię parzystości, co bez wsparcia analitycznego wymagałoby znacznie dłuższego procesu testowania heurystyk.
- **Wsparcie w debugowaniu:** Model AI skutecznie identyfikował błędy składniowe oraz logiczne w skomplikowanych pętlach sterujących fazami gry, co zminimalizowało czas potrzebny na stabilizację kodu.

### 8.2 Ograniczenia i wady technologii

Mimo wysokiej efektywności, wykorzystanie AI wiązało się z konkretnymi wyzwaniami, które wymagały stałej interwencji programisty:

- **Zjawisko halucynacji logicznych:** AI miewało tendencję do sugerowania nieistniejących metod API lub tworzenia fragmentów kodu, które – choć poprawne składniowo – były błędne logicznie w kontekście specyfiki gry (np. błędne obliczanie indeksów przy orientacji pionowej statków).
- **Brak kontekstu projektowego:** Modele językowe często proponowały rozwiązania zbyt generyczne, które nie uwzględniały wcześniejszych założeń architektury (np. konflikt z globalnym menedżerem stanu gry).
- **Kwestia stylu i optymalizacji:** Wygenerowany kod często wymagał ręcznej refaktoryzacji, aby dostosować go do standardów „czystego kodu” (Clean Code) oraz specyficznego stylu graficznego aplikacji (Pirate Edition).

## Wnioski końcowe

Obiektywna analiza wskazuje, że współczesne modele AI są potężnym narzędziem wspomagającym, ale nie są w stanie zastąpić programisty w procesie projektowania architektury i podejmowania decyzji kreatywnych. Ostateczny sukces projektu wynikał z synergii: AI dostarczyło surowej mocy obliczeniowej i bazy wiedzy, natomiast rola człowieka polegała na krytycznej weryfikacji, integracji modułów oraz nadaniu aplikacji unikalnego charakteru UX.