

**UNIWERSYTET RZESZOWSKI**

**Kolegium Nauk Przyrodniczych**

**Albert Mazur**

Nr albumu:

117816

Kierunek

**Informatyka**

**Projekt i implementacja gry karcianej opartej o grę Uno z wykorzystaniem algorytmów Monte Carlo Tree Search (MCTS) i podejścia heurystycznego**

Praca inżynierska

Praca wykonana pod kierunkiem

dr inż. Piotr Lasek

Rzeszów, 2024

Spis treści

[1. Wstęp 3](#_Toc154878447)

[1.1. Wprowadzenie 3](#_Toc154878448)

[1.2. Motywacja i wybór tematu 3](#_Toc154878449)

[1.3. Cel i zakres prac 4](#_Toc154878450)

[2. Opis stosowanych algorytmów 5](#_Toc154878451)

[2.1. Dlaczego stosujemy algorytmy AI w grach 5](#_Toc154878452)

[2.2. Stosowane algorytmy 6](#_Toc154878453)

[2.2.1. Algorytm A\* 6](#_Toc154878454)

[2.2.2. Maszyny stanów skończonych 7](#_Toc154878455)

[2.2.3. Logika rozmyta 10](#_Toc154878456)

[3. Projektowanie 15](#_Toc154878457)

[3.1. Architektura 15](#_Toc154878458)

[3.2. Wykorzystane technologie 15](#_Toc154878459)

[3.3. Baza danych 16](#_Toc154878460)

[3.3.1. Schemat 16](#_Toc154878461)

[3.3.2. Opis 16](#_Toc154878462)

[4. Implementacja 18](#_Toc154878463)

[4.1. Interfejsu 19](#_Toc154878464)

[4.2. Podejście heurystyczne 21](#_Toc154878465)

[4.2.1. Opis algorytmu 21](#_Toc154878466)

[4.2.2. Opis implementacji 22](#_Toc154878467)

[4.3. MCTS (Monte Carlo Tree Search) 22](#_Toc154878468)

[4.3.1. Opis algorytmu 22](#_Toc154878469)

[4.3.2. Opis implementacji 22](#_Toc154878470)

[4.4. Porównanie algorytmów 22](#_Toc154878471)

[5. Zakończenie 23](#_Toc154878472)

[5.1. Dalsze możliwości rozwoju aplikacji 23](#_Toc154878473)

[5.2. Podsumowanie i wnioski 23](#_Toc154878474)

[6. Bibliografia 24](#_Toc154878475)

[7. Spis fotografii 25](#_Toc154878476)

# Wstęp

## Wprowadzenie

W dobie rosnącej popularności gier komputerowych oraz postępującego rozwoju technologii sztucznej inteligencji, projektowanie gier z wykorzystaniem zaawansowanych algorytmów stanowi fascynujące wyzwanie inżynieryjne. Niniejsza praca inżynierska skupia się na zaprojektowaniu i implementacji wirtualnej wersji gry karcianej, inspirowanej klasyczną grą Uno, wykorzystując w tym celu algorytmy Monte Carlo Tree Search (MCTS) i podejście heurystyczne opartą na własnych zasadach. Celem jest stworzenie gry, która nie tylko dostarcza rozrywki, ale również służy jako platforma do badania i demonstracji możliwości sztucznej inteligencji w symulowaniu i przewidywaniu ludzkich zachowań w grach. Praca ta ma na celu nie tylko zaprojektowanie i implementację samej gry, ale również zbadanie efektywności i skuteczności algorytmu MCTS w kontekście gry karcianej, oferując tym samym nowe perspektywy w dziedzinie sztucznej inteligencji i gier komputerowych.

## Motywacja i wybór tematu

Moja decyzja o napisaniu pracy inżynierskiej na temat podany wyżej temat wynika z głębokiego zainteresowania światem gier oraz fascynacji możliwościami, jakie oferuje sztuczna inteligencja w systemach decyzyjnych. Wybór gry Uno jako centralnego punktu mojego projektu był naturalny, ponieważ cenię ją za globalną popularność, prostotę reguł oraz elementy losowości, które stanowią idealną bazę do eksploracji technik AI.

Zastosowanie algorytmu Monte Carlo Tree Search w kontekście gry Uno otwiera dla mnie nowe horyzonty badawcze. MCTS, znanego z efektywności w grach takich jak Go czy szachy, jeszcze nie eksplorowano szeroko w kontekście gier karcianych, gdzie losowość i nieprzewidywalność mają kluczowe znaczenie. Integracja MCTS w grze Uno stanowi dla mnie wyzwanie inżynierskie i naukowe, mające na celu nie tylko stworzenie atrakcyjnej gry, ale także przyczynienie się do rozwoju algorytmów sztucznej inteligencji.

Moim celem jest również zbadanie, jak podejście heurystyczne może optymalizować decyzje w grze charakteryzującej się częściową losowością, jaką jest Uno. Jestem przekonany, że heurystyka, dostosowując strategie do specyfiki gry, może zaoferować unikalne spostrzeżenia w zarządzaniu ryzykiem i podejmowaniu decyzji w warunkach niepewności.

Ostatecznie, wybór tego tematu jest odzwierciedleniem moich osobistych zainteresowań oraz pragnienia połączenia pasji do gier i technologii. Ten projekt stanowi dla mnie okazję do zintegrowania teoretycznej wiedzy z praktycznymi umiejętnościami w zakresie programowania, projektowania systemów i analizy algorytmicznej, co ma znaczenie nie tylko akademickie, ale także praktyczne i komercyjne.

## Cel i zakres prac

Celem mojej pracy inżynierskiej jest projektowanie i implementacja cyfrowej wersji gry karcianej, inspirowanej popularną grą Uno, z zastosowaniem algorytmów Monte Carlo Tree Search (MCTS) oraz podejścia heurystycznego. Dążę do stworzenia gry, która nie tylko będzie atrakcyjna i angażująca dla graczy, ale także posłuży jako narzędzie do badania efektywności zastosowania MCTS w symulacji decyzji w grach karcianych.

# Opis stosowanych algorytmów

## Dlaczego stosujemy algorytmy AI w grach

Stosowanie algorytmów sztucznej inteligencji (AI) w grach ma swoje uzasadnienie w kilku kluczowych aspektach, które wspólnie przyczyniają się do zwiększenia wartości i atrakcyjności gier:

* **Poprawa Doświadczenia Gracza**

AI umożliwia tworzenie bardziej zaawansowanych i przekonujących przeciwników sterowanych przez komputer. Dzięki temu gracze mogą doświadczać bardziej dynamicznej i wymagającej rozgrywki, która lepiej odpowiada na ich działania i strategie.

* **Symulacja Realistycznych Zachowań**

W grach wymagających głębi strategicznej, takich jak szachy czy strategie wojenne, AI pozwala na symulowanie zachowań i strategii zbliżonych do ludzkich, co zwiększa realizm i wyzwanie.

* **Dostosowanie Poziomu Trudności**

Algorytmy AI mogą automatycznie dostosowywać poziom trudności gry w zależności od umiejętności i stylu gry użytkownika, co pozwala na zachowanie równowagi i zapewnienie ciągłego wyzwania dla graczy na różnych poziomach zaawansowania.

* **Generowanie Nowych Doświadczeń**

AI może tworzyć nieprzewidywalne i różnorodne scenariusze gry, co prowadzi do unikalnych i często zaskakujących doświadczeń dla graczy.

* **Rozwój Technologiczny i Badawczy**

Gry stanowią znakomite pole do eksperymentowania i doskonalenia algorytmów AI. Przetestowanie AI w kontrolowanych, ale złożonych środowiskach gier pozwala na zdobywanie cennej wiedzy i doświadczeń.

* **Automatyzacja i Tworzenie Zawartości**

AI może być wykorzystane do automatyzowania różnych aspektów produkcji gier, takich jak generowanie poziomów czy tworzenie fabuły, co pozwala twórcom skupić się na innych, kreatywnych aspektach procesu tworzenia gier.

Każdy z tych punktów przyczynia się do tworzenia gier, które są bardziej angażujące, wymagające i różnorodne, co przekłada się na lepsze doświadczenia dla użytkowników i rozwój całej branży gier.

## Stosowane algorytmy

### Algorytm A\*

Algorytm A\* (A-gwiazdka) jest uważany za najbardziej efektywny i powszechnie stosowany algorytm do wyszukiwania ścieżki między dwoma punktami, na przykład punktami A i B. Chociaż istnieje wiele innych metod, takich jak algorytm Dijkstry czy algorytm Monte-Carlo, używany na przykład w AlphaGo od Google, A\* wyróżnia się swoją prostotą implementacji oraz skutecznością działania.

Co sprawia, że algorytm A\* jest wyjątkowy, to jego zdolność do zawsze znajdywania ścieżki pomiędzy dwoma punktami, o ile istnieje możliwość dotarcia z punktu A do B. Co więcej, gwarantuje on znalezienie optymalnej ścieżki. Algorytm ten wykorzystuje heurystykę w taki sposób, że żaden inny algorytm, korzystający z tej samej heurystyki, nie będzie w stanie znaleźć bardziej optymalnej trasy sprawdzając mniejszą liczbę węzłów.

Zasadniczo, algorytm A\* rozpatruje planszę jako graf składający się z wierzchołków połączonych krawędziami, które mogą posiadać określony koszt. W każdym kroku algorytm wybiera wierzchołek, do którego koszt przejścia jest najniższy, kontynuując ten proces aż do osiągnięcia punktu docelowego. To podejście jest szczególnie skuteczne w przeszukiwaniu i wyznaczaniu optymalnej trasy na złożonych grafach. Idealnie obrazuje to obrazek poniżej

Obraz zawierający tekst, diagram, linia, krąg

Opis wygenerowany automatycznie

*Rysunek 1. Prezentacja przykładu algorytmu A\*.*

W początkowej fazie algorytmu A\* do rozpatrzenia są dwie krawędzie o wartościach 1,5 i 2. Do tych wartości algorytm dodaje heurystykę – h(a) dla każdego wierzchołka. W rezultacie, rzeczywiste opcje, które algorytm ma do rozważenia to: 5,5 i 6,5. W związku z tym, wybiera wierzchołek A. Następnie, stoi przed wyborem między dwoma ścieżkami: jedną o wartości 1,5 + 2 + 2 (suma dwóch krawędzi i heurystyki) oraz drugą o wartości 2 + 4,5. To daje łączny koszt 5,5 kontra 6,5, gdzie wybiera wierzchołek B.

Dalej, algorytm porównuje wierzchołek D, którego koszt pozostaje na poziomie 6,5, z wierzchołkiem C o koszcie 11,5 (6,5 za krawędzie plus 4 za heurystykę). W tej sytuacji wygrywa wierzchołek D. Kolejny etap to wybór między wierzchołkiem C o koszcie 11,5 a wierzchołkiem E o koszcie 7. Tym razem wybiera E. Ostatnim krokiem jest dotarcie do punktu docelowego, co kończy proces i osiąga cel.

### Maszyny stanów skończonych

W świecie gier komputerowych, najbardziej rozpowszechnioną metodą wykorzystywaną do tworzenia sztucznej inteligencji są Maszyny Stanów Skończonych (ang. Finite-State Machine – FSM), znane również jako automaty skończone. Te urządzenia działają na podstawie trzech prostych zasad:

* Automat musi posiadać określoną liczbę stanów, w których może się znajdować.
* Istnieją przejścia między tymi stanami, które są aktywowane, gdy spełnione są określone warunki.
* W danym momencie automat może znajdować się tylko w jednym stanie.

Dane wejściowe są podawane do automatu, na podstawie których określany jest stan wyjściowy. Choć może to brzmieć skomplikowanie, automat nie generuje żadnej wartości zwrotnej, a jedynie przemieszcza się z jednego stanu do drugiego. Stan wyjściowy to po prostu stan, w którym automat znajduje się po przetworzeniu danych wejściowych.

Przykłady z życia codziennego pomagają zrozumieć tę koncepcję. Weźmy na przykład żarówkę, która ma dwa stany: włączona i wyłączona. Innym przykładem mogą być drzwi, które mogą znajdować się w stanach: otwarte, zamknięte, zamknięte na klucz. Do manipulowania stanami drzwi, mogą być używane dwa sygnały wejściowe: "Użyj ręki" (naciśnij klamkę) oraz "Użyj klucza" (włóż i przekręć klucz w zamku).

W grach komputerowych, stany te zwykle odnoszą się do różnych akcji wykonywanych przez postacie, takich jak patrolowanie, atakowanie, ucieczka, leczenie się. Przejścia między stanami wymagają spełnienia określonych warunków. Na przykład, postać może przeładować broń tylko wtedy, gdy posiada zapasową amunicję i pusty magazynek, a atak na gracza może nastąpić tylko, gdy postać ma amunicję i widzi gracza.

Obraz zawierający tekst, diagram, zrzut ekranu, linia

Opis wygenerowany automatycznie

*Rysunek 2. Prezentacja przykładu algorytmu maszyny stanów skończonych.*

Na ilustracji została zaprezentowana podstawowa wersja maszyny stanów skończonych. Jest to najprostszy model, który można rozwijać, dodając więcej stanów i tworząc bardziej złożone warunki. Na przykład, w trakcie przejścia od podejścia do ataku, można wprowadzić warunek sprawdzający, czy gracz znajduje się w zasięgu ataku, bazując na zmianie odległości między graczem a agentem. Można też wziąć pod uwagę inne czynniki, jak na przykład dostępność amunicji.

Im bardziej skomplikowane są warunki i większa liczba stanów, tym bardziej rozbudowana i potencjalnie efektywniejsza staje się sztuczna inteligencja. Warunki te mogą być proste, na przykład reakcja na zauważenie gracza i przeprowadzenie ataku, lub mogą być bardziej złożone, oparte na funkcjach, które analizują szereg różnych warunków.

Podsumowując, maszyny stanów skończonych prezentują się w ten sposób i stanowią najbardziej podstawową metodę projektowania sztucznej inteligencji w grach.

### Logika rozmyta

Definiowanie logiki rozmytej staje się klarowne, gdy porównamy ją z logiką tradycyjną. W standardowej logice mamy do czynienia z dwoma stanami: Prawdą i Fałszem. Natomiast logika rozmyta wprowadza stan pośredni, umożliwiając operowanie na pojęciach, które nie są absolutne, lecz wymagają określenia "do jakiego stopnia" coś jest prawdziwe lub "jak bardzo" dane pojęcie się sprawdza. Przykładowo, używamy określeń takich jak "bardzo małe", "dość duże", "średnie" i podobnych.

Obraz zawierający tekst, zrzut ekranu, Czcionka, linia

Opis wygenerowany automatycznie

*Rysunek 3. Porównanie logiki klasycznej z logiką rozmytą.*

Ten scenariusz umożliwia tworzenie bardziej złożonych i realistycznych systemów sztucznej inteligencji, na przykład poprzez wprowadzenie emocji. W takim ujęciu, nasz wirtualny przeciwnik nie jest już jednoznacznie zdefiniowany jako zły lub spokojny. Może on przyjmować różnorodne stany emocjonalne, takie jak spokojny, podirytowany, zirytowany, bardzo zły, wściekły itd. Korzyści są podwójne. Po pierwsze, możliwe staje się operowanie na tych niedostrzegalnych stanach, a po drugie, w dialogu między programistą a projektantem gry, stwierdzenie takie jak "Przeciwnik atakuje, kiedy jest bardzo zły" nabiera większego znaczenia.

Kluczowe jest, że wartości w logice rozmytej są reprezentowane w sposób znormalizowany, czyli w skali od 0 do 1, na przykład 0, 0.23, 0.78.

Aby efektywnie zastosować logikę rozmytą, należy najpierw określić, jakie aspekty chcemy symulować. Załóżmy, że chcemy ocenić, czy nasz agent w grze powinien użyć zaklęcia leczenia. To będzie zależało od dwóch czynników: jego poziomu bezpieczeństwa (tutaj uproszczony do odległości od gracza) oraz aktualnego stanu zdrowia.

Następnie należy zamodelować sensowne zasady działania. Może to wyglądać następująco:

Jeśli jestem bardzo ranny i bezpieczny, rzucaj zaklęcie dużego leczenia.

Jeśli jestem trochę ranny, rzucaj zaklęcie małego leczenia, niezależnie od poziomu bezpieczeństwa.

Jeśli nie jestem ranny, nie używaj leczenia.

Nie, lecz się, jeśli jest niebezpiecznie.

Teraz trzeba określić zbiory rozmyte, które będą definiować te opisane słownie kategorie. Należy ustalić, kiedy kategoria "bardzo ranny" przechodzi w "ranny", a "bezpiecznie" w "niebezpiecznie". Do tego celu często używane są wykresy.

Obraz zawierający linia, Wykres, diagram, tekst

Opis wygenerowany automatycznie

*Rysunek 4. Wykres zbiorów życia agenta.*

Wyobraźmy sobie sytuację, w której agent w grze posiada 80% swojego maksymalnego poziomu zdrowia. W takim przypadku agent znajdowałby się:

Obraz zawierający linia, Wykres, diagram

Opis wygenerowany automatycznie

*Rysunek 5. Wykres zbiorów życia agenta z zaznaczonym punktem.*

Z tej analizy wynika, że nasz bohater jest lekko ranny z prawdopodobieństwem 0.5, a z prawdopodobieństwem 0.25 uznany jest za zdrowego. Teraz przeprowadzamy podobną ocenę dla zmiennej reprezentującej odległość. Zacznijmy od utworzenia odpowiedniego wykresu:Początek formularza

Obraz zawierający linia, Wykres, diagram, tekst

Opis wygenerowany automatycznie

*Rysunek 6. Wykres zbiorów odległości agenta od gracza.*

Następnie, na wykresie odnoszącym się do odległości, zaznaczamy aktualną pozycję. Załóżmy, że w tym przypadku gracz znajduje się w odległości 350 metrów od agenta.

Obraz zawierający linia, Wykres, diagram

Opis wygenerowany automatycznie

*Rysunek 7. Wykres zbiorów odległości agenta od gracza z zaznaczonym punktem.*

Z tej sytuacji wynika, że agent znajduje się w stanie, który można uznać za 25% bezpieczny i około 65% niebezpieczny. Rozważmy teraz, jakie zasady mogą być zastosowane w tej sytuacji. Gracz jest częściowo zdrowy i częściowo lekko ranny, co pozwala na zastosowanie trzech kluczowych zasad:

* Nie używam leczenia, gdy jestem zdrowy.
* Używam zaklęcia leczenia mniejszego, gdy jestem lekko ranny i sytuacja jest niebezpieczna.
* Używam zaklęcia leczenia mniejszego, gdy jestem lekko ranny i sytuacja jest bezpieczna.
* Dodając do tego ocenę poziomu bezpieczeństwa, mamy możliwość zastosowania dodatkowych zasad:
* Używam zaklęcia leczenia mniejszego, gdy jestem lekko ranny i sytuacja jest niebezpieczna (co jest powtórzeniem).
* Używam zaklęcia leczenia mniejszego, gdy jestem lekko ranny i sytuacja jest bezpieczna (również powtórzenie).
* Używam zaklęcia leczenia większego, gdy jestem bardzo ranny i sytuacja jest bezpieczna.

Następnie oceniamy prawdziwość każdej z tych zasad. Prawdziwość to najniższy procent z wszystkich warunków danej zasady. Na przykład, jeśli jestem lekko ranny w 50% i bezpiecznie jest w 25%, to maksymalny poziom prawdziwości tej zasady to 25%. Po dokonaniu takiej ewaluacji dla każdej zasady, otrzymujemy:

* Nie używam leczenia, gdy jestem zdrowy – 25% prawdziwości (Jestem zdrowy w 25%, odległość nie ma znaczenia).
* Używam zaklęcia leczenia mniejszego, gdy jestem lekko ranny i sytuacja jest bezpieczna – 25% prawdziwości.
* Używam zaklęcia leczenia mniejszego, gdy jestem lekko ranny i sytuacja jest niebezpieczna – 50% prawdziwości.
* Używam zaklęcia leczenia większego, gdy jestem bardzo ranny i sytuacja jest bezpieczna – 0% prawdziwości.

Ostatecznie wybieramy zasadę z najwyższym stopniem prawdziwości, co w tym przypadku oznacza użycie zaklęcia leczenia mniejszego, gdyż agent jest lekko ranny i sytuacja jest niebezpieczna. W ten sposób podejmowane są decyzje w różnych sytuacjach.

# Projektowanie

## Architektura

Wykonano wzorzec MVC (Główną koncepcją MVC jest podział aplikacji na 3 niezależne warstwy reprezentujące kolejno: (Model) Model danych - opis struktur danych i powiązań pomiędzy nimi (View) Interfejs, czyli to co widzi użytkownik (Controller) Logika działania - powiązania między zdarzeniami zachodzącymi w systemie):

* View – zostało napisane z wykorzystaniem Laraveli Blada.
* Controller – zostały napisane przy pomocy Laravel.
* Model – został wykorzystany ORM Eloquent który jest dostępny w Laravelu.

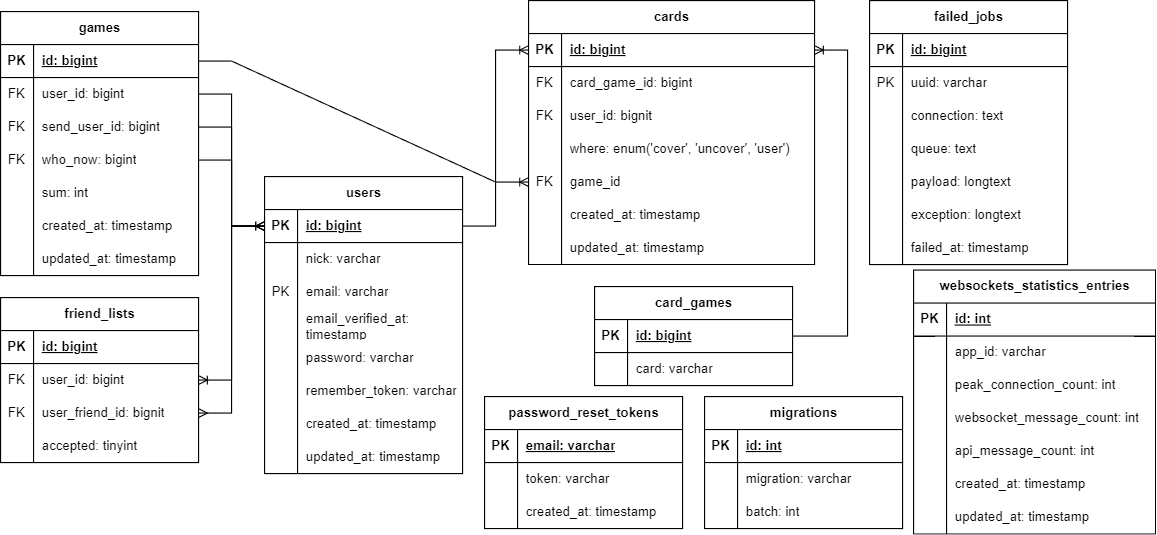
Dodatkowo do komunikacji w trybie multiplayer wykorzystano WebSockets, który jest technologia umożliwiająca otwarcie interaktywnego kanału komunikacyjnego pomiędzy przeglądarką użytkownika (klientem) a serwerem. Pozwala to na dwukierunkową komunikację w czasie rzeczywistym, co oznacza, że serwer i klient mogą wysyłać dane jednocześnie, bez konieczności odświeżania strony.

## Wykorzystane technologie

* PHP - język programowania, który służy do pisania skryptów po stronie servera. Adres do strony: <https://www.php.net/docs.php>
* Laravel - framework PHP dla aplikacji internetowych, zapewniający elegancką składnię i łatwość użycia. Używa wzorca MVC dla organizacji kodu, oferuje ORM Eloquent dla baz danych, posiada silnik szablonów Blade, narzędzie Artisan do automatyzacji zadań, a także zaawansowane funkcje routingu i bezpieczeństwa. Adres do strony: <https://laravel.com>
* MySQL – system zarządzania bazami danych (DBMS), który używa języka zapytań SQL (Structured Query Language) do zarządzania danymi.  
  Odnośnik do dokumentacji: <https://dev.mysql.com/doc/>
* Tailwind CSS - framework CSS oparty na klasach pomocniczych, który ułatwia szybkie tworzenie niestandardowych interfejsów użytkownika. Umożliwia stosowanie małych, konkretnych klas bezpośrednio do elementów HTML. Odnośnik do dokumentacji: <https://tailwindcss.com>

## Baza danych

## Schemat



*Rysunek 8. Schemat bazy danych.*

## Opis

websockets\_statistics\_entries - Zawiera statystyki dla połączeń websocket.

* + id: Unikalny identyfikator rekordu (autoinkrementowany).
  + app\_id: Identyfikator aplikacji korzystającej z websocket.
  + peak\_connection\_count: Maksymalna liczba połączeń websocket osiągnięta w danym okresie.
  + websocket\_message\_count: Liczba wiadomości przesłanych przez websocket.
  + api\_message\_count: Liczba wiadomości API przesłanych do serwera.
  + created\_at, updated\_at: Daty utworzenia i ostatniej aktualizacji rekordu.
* users - Przechowuje dane użytkowników.
  + id: Unikalny identyfikator użytkownika.
  + nick: Nazwa użytkownika/nick.
  + email: Adres email użytkownika.
  + email\_verified\_at: Data i czas zweryfikowania adresu email.
  + password: Zaszyfrowane hasło użytkownika.
  + rememberToken: Token do zapamiętywania sesji użytkownika.
  + created\_at, updated\_at: Daty utworzenia i ostatniej aktualizacji rekordu.
* password\_reset\_tokens - Tabela do zarządzania tokenami resetowania hasła.
  + email: Adres email użytkownika, dla którego jest resetowane hasło (klucz główny).
  + token: Token resetowania hasła.
  + created\_at: Data i czas utworzenia tokena.
* failed\_jobs - Zapisuje informacje o zadaniach, które nie zostały wykonane poprawnie.
  + id: Unikalny identyfikator zadania.
  + uuid: Unikalny identyfikator UUID dla zadania.
  + connection: Typ połączenia, w którym wystąpił błąd.
  + queue: Nazwa kolejki, w której wystąpił błąd.
  + payload: Szczegóły zadania.
  + exception: Informacje o wyjątku, który wystąpił.
  + failed\_at: Data i czas wystąpienia błędu.
* personal\_access\_tokens - Tabela dla tokenów dostępu, zawierająca informacje o tokenie.
  + id: Unikalny identyfikator tokena.
  + tokenable\_id, tokenable\_type: Polimorficzne powiązanie z użytkownikiem lub innym obiektem.
  + name: Nazwa tokena.
  + token: Token dostępu.
  + abilities: Możliwości/uprawnienia tokena.
  + last\_used\_at: Ostatni czas użycia tokena.
  + expires\_at: Data wygaśnięcia tokena.
  + created\_at, updated\_at: Daty utworzenia i ostatniej aktualizacji rekordu.
* friend\_lists - Zarządza listami znajomych użytkowników.
  + id: Unikalny identyfikator rekordu.
  + user\_id: Identyfikator użytkownika, którego dotyczy lista znajomych.
  + user\_friend\_id: Identyfikator znajomego na liście.
  + accepted: Czy zaproszenie do znajomych zostało zaakceptowane.
* games - Zawiera informacje o grach.
  + id: Unikalny identyfikator gry.
  + user\_id: Identyfikator pierwszego użytkownika.
  + send\_user\_id: Identyfikator drugiego użytkownika.
  + who\_now: Identyfikator użytkownika, który aktualnie może wykonać ruch.
  + sum: Suma kart do zabrania.
  + created\_at, updated\_at: Daty utworzenia i ostatniej aktualizacji rekordu.
* cards - Tabela do zarządzania kartami wszystkich gier
  + id: Unikalny identyfikator karty w grach.
  + card\_game\_id: Identyfikator nazwy karty.
  + user\_id: Identyfikator użytkownika posiadającego kartę (opcjonalnie).
  + where: Lokalizacja karty gdzie się znajduje (np. "cover", "uncover", "user").
  + game\_id: Identyfikator gry, w której używana jest karta.
  + created\_at, updated\_at: Daty utworzenia i ostatniej aktualizacji rekordu.
* card\_games - Przechowuje nazwa wszystkich kart
  + id: Unikalny identyfikator karty.
  + card: Nazwa karty w grze.

# Implementacja

## Interfejsu

Obraz zawierający tekst, Czcionka, logo, Grafika

Opis wygenerowany automatycznie

*Rysunek 9. Strona główna aplikacji*

Strona główna przestawania logo aplikacji, tekst „Witaj w grze Karcianka” i przycisk z zapisem „Dołącz do gry” gdzie po naciśnięciu przekieruje na stronę logowania.

Obraz zawierający tekst, zrzut ekranu, logo, Czcionka

Opis wygenerowany automatycznie

*Rysunek 10. Strona logowania*

Na obrazku znajduje się formularz logowania z dwoma polami „E-mail” i „Hasło”. Oraz trzy przyciski „ZALOGUJ SIĘ” który sprawdza czy dane są poprawne, „Zarejestruj się” gdzie przenosi użytkownika na stronę rejestracji oraz „Nie pamiętasz hasła?” która odnosi na stronę z formularzem do odnowienia hasła.

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

*Rysunek 11. Strona rejestracji użytkownika*

Do zarejestrowania się wymagane jest podanie nick, e-mail, hasło i po ponownym podaniu hasła.

Obraz zawierający tekst, zrzut ekranu, logo, Czcionka

Opis wygenerowany automatycznie

*Rysunek 12. Strona przypomnienia hasła*

Obraz zawierający zrzut ekranu, Prostokąt, zieleń

Opis wygenerowany automatycznie

*Rysunek 13. Strona planszy gry i historia zagranych kart*

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

*Rysunek 14. Wybór trybu gry i uruchomienie*

Obraz zawierający tekst, zrzut ekranu

Opis wygenerowany automatycznie

*Rysunek 15. Strona do zaproszenia i dołączenia do gry*

*Obraz zawierający tekst, Czcionka, zrzut ekranu

Opis wygenerowany automatycznie*

*Rysunek 16. Strona do zaproszenia i dołączenia do listy znajomych*

## Podejście heurystyczne

### Opis algorytmu

Heurystyka to sposób na rozwiązywanie problemów, który nie zawsze prowadzi do najlepszego lub nawet do poprawnego rozwiązania. Jest to rodzaj skrótu, który pomaga szybko znaleźć przybliżone rozwiązanie.

Opis przyjętych zasad do podejmowania decyzji:

* Agent stara się pozbywania się kart w swojej ręce.
* Używanie kart funkcyjnych w momencie kiedy mam ponieść karę.
* Agent unika używania kart funkcyjnych, kiedy ma możliwość zagrania kartą nie funkcyjną.
* Jeśli przeciwnikowi zostały miej niż 3 karty użyje karty funkcyjnie która zadaje karę.

### Opis implementacji

## MCTS (Monte Carlo Tree Search)

### Opis algorytmu

Metoda MCTS, czyli Monte-Carlo Tree Search, jest techniką stosowaną w dziedzinie sztucznej inteligencji, szczególnie przydatną w określaniu najlepszych posunięć w grach komputerowych. Ta strategia skoncentrowana jest na badaniu ruchów, które wydają się najbardziej obiecujące, rozwijając drzewo możliwości poprzez losowe wybieranie opcji z dostępnych ścieżek, co pozwala na efektywne przeszukiwanie rozległych przestrzeni decyzyjnych.

### Opis implementacji

## Porównanie algorytmów

# Zakończenie

## Dalsze możliwości rozwoju aplikacji

Dalszymi możliwościami rozwoju jest szereg ekscytujących funkcji, które znacznie wzbogacą doświadczenie użytkowników. Kluczowym elementem będzie możliwość dodawania szczegółowych statystyk swoich gier oraz gier znajomych. Ta funkcja pozwoli graczom na śledzenie postępów, analizę strategii i porównywanie swoich osiągnięć z osiągnięciami przyjaciół, co znacząco podniesie rywalizację i zaangażowanie.

Kolejnym ważnym krokiem będzie rozbudowa opcji multiplayer. Obecnie aplikacja obsługuje gry dla dwóch graczy, rozszerzając o możliwość dodania większej liczby graczy. Dzięki temu gracze będą mogli organizować większe i bardziej dynamiczne zawody.

Dodatkowo, wprowadzenie sztucznej inteligencji w postaci bota, który mógłby dołączyć do rozgrywki multiplayer. Boty będą stanowiły świetną opcję dla tych, którzy chcą trenować swoje umiejętności lub po prostu zagrać wtedy, gdy brak jest dostępnych ludzkich przeciwników. Wprowadzenie botów do gry multiplayer zwiększy elastyczność i dostępność rozgrywki, umożliwiając graczom cieszenie się grą w każdym momencie.

Te innowacje mają na celu nie tylko poprawę obecnych funkcji, ale także otwarcie nowych możliwości i ścieżek rozwoju dla aplikacji, co zapewni użytkownikom jeszcze bardziej satysfakcjonujące i angażujące doświadczenia.

## Podsumowanie i wnioski

# Bibliografia

* 1. Jak powstaje sztuczna inteligencja w grach komputerowych - <https://geek.justjoin.it/powstaje-sztuczna-inteligencja-grach-komputerowych/#Logika_Rozmyta>
  2. Monte-Carldo Tree Search - <https://pl.wikipedia.org/wiki/Monte-Carlo_Tree_Search#Wady_i_zalety>
  3. Laravel - <https://laravel.com/docs/10.x>

# Spis fotografii

* 1. Rys. 1 - [https://pl.wikipedia.org/wiki/Algorytm\_A\*#/media/Plik:AstarExample.gif](https://pl.wikipedia.org/wiki/Algorytm_A*#/media/Plik:AstarExample.gif)
  2. Rys. 2 - <https://geek.justjoin.it/wp-content/uploads/2018/10/grafika-patroluj.png>
  3. Rys. 3 - <https://geek.justjoin.it/wp-content/uploads/2018/10/Logika-Rozmyta.png>
  4. Rys. 4 - <https://geek.justjoin.it/wp-content/uploads/2018/10/ZdrowieAgenta.png>
  5. Rys. 5 - <https://geek.justjoin.it/wp-content/uploads/2018/10/ZdrowieAgenta_zaznaczenie.jpg>
  6. Rys. 6 - <https://geek.justjoin.it/wp-content/uploads/2018/10/Odległość.png>
  7. Rys. 7 – <https://geek.justjoin.it/wp-content/uploads/2018/10/Odległość_zaznacznie.jpg>