

**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 TreeSearch(MCTS) i podejścia heurystycznego**

Praca inżynierska

Praca wykonana pod kierunkiem

dr inż. Piotr Lasek

Rzeszów, 2024

Spis treści

[1. Wstęp 4](#_Toc155656790)

[1.1. Wprowadzenie 4](#_Toc155656791)

[1.2. Motywacja i wybór tematu 4](#_Toc155656792)

[1.3. Cel i zakres prac 5](#_Toc155656793)

[2. Sztuczna intiligencja w grach 6](#_Toc155656794)

[2.1. Dlaczego stosujemy algorytmy AI w grach 6](#_Toc155656795)

[2.2. Stosowane algorytmy 7](#_Toc155656796)

[2.2.1. Algorytm A\* 7](#_Toc155656797)

[2.2.2. Maszyny stanów skończonych 8](#_Toc155656798)

[2.2.3. Logika rozmyta 11](#_Toc155656799)

[3. Projekt 16](#_Toc155656800)

[3.1. Architektura 16](#_Toc155656801)

[3.2. Wykorzystane technologie 16](#_Toc155656802)

[3.3. Baza danych 17](#_Toc155656803)

[3.3.1. Schemat 17](#_Toc155656804)

[3.3.2. Opis 17](#_Toc155656805)

[4. Implementacja 20](#_Toc155656806)

[4.1. Opis interfejsu uzytkownika 20](#_Toc155656807)

[4.2. Podejście heurystyczne 27](#_Toc155656808)

[4.2.1. Opis algorytmu 27](#_Toc155656809)

[4.2.2. Schemat blokowy algorytmu i opis 27](#_Toc155656810)

[4.3. MCTS (Monte Carlo Tree Search) 29](#_Toc155656811)

[4.3.1. Opis algorytmu 29](#_Toc155656812)

[4.3.2. Opis implementacji 30](#_Toc155656813)

[4.4. Porównanie algorytmów 38](#_Toc155656814)

[5. Zakończenie 39](#_Toc155656815)

[5.1. Dalsze możliwości rozwoju aplikacji 39](#_Toc155656816)

[5.2. Podsumowanie i wnioski 39](#_Toc155656817)

[6. Bibliografia 40](#_Toc155656818)

[7. Spis fotografii 41](#_Toc155656819)

Praca inżynierska autorstwa dotyczy projektowania i implementacji gry karcianej opartej na grze Uno. Wykorzystano w niej algorytmy Monte Carlo Tree Search (MCTS) i podejście heurystyczne. Praca skupia się na zastosowaniu sztucznej inteligencji w grach, z naciskiem na algorytmy AI oraz ich wpływ na doświadczenie gracza i symulację zachowań. Opisuje także architekturę projektu, użyte technologie, bazę danych oraz implementację interfejsu użytkownika. Porównuje efektywność algorytmu MCTS z podejściem heurystycznym w kontekście gry karcianej, oferując perspektywy dla sztucznej inteligencji i gier komputerowych. Na koniec przedstawia wnioski i możliwości dalszego rozwoju aplikacji.

# 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 TreeSearch (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 fascynacją 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 TreeSearch 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 TreeSearch (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.

# Sztuczna inteligencja w grach komputerowych

## 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 przekonywujących 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, a także 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.

To 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.

# Projekt

## 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 technologią umożliwiającą 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 serwera. Adres do strony: <https://www.php.net/docs.php>
* Laravel - framework PHP dla aplikacji internetowych, zapewniający poprawną składnię i łatwość użycia. Używa on: 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 oficjalnej strony framework: <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 on stosowanie małych, konkretnych klas bezpośrednio do elementów HTML. Adres do oficjalnej strony framework: <https://tailwindcss.com>

## Baza danych

### Schemat

Obraz zawierający tekst, diagram, zrzut ekranu, Równolegle

Opis wygenerowany automatycznie

*Rysunek 8. Schemat bazy danych.*

### Opis

* websockets\_statistics\_entries - Zawiera statystyki dla połączeń websocket. Składa się z następujących kolumn id: app\_id, peak\_connection\_count, websocket\_message\_count, api\_message\_count, created\_at, updated\_at. Tabela jest tworzona przez narządzie WebSocket.
* users - Przechowuje dane użytkowników. Takie jak: nick, email, email\_verified\_at (Data i czas zweryfikowania adresu email), password (Hasło jest zaszyfrowane), rememberToken, created\_at, updated\_at. Tabela jest tworzona przez Laravela, gdzie zmieniono kolumnę name na nick, zmieniając tak, że kolumna ma być unikatowa i składająca się tylko z liter i cyfr.
* password\_reset\_tokens- tabela do zarządzania tokenami resetowania hasła. Składa się z kolumn email (klucz główny), token, created\_at, failed\_jobs. Ta tabela jest stworzona przez Laravela.
* failed\_jobs- Zapisuje informacje o zadaniach, które nie zostały wykonane poprawnie. Składa się z kolumn id, uuid, connection, queue, payload, exception, failed\_at. Tabela jest automatycznie stworzona przez Laravela.
* personal\_access\_tokens - tabela dla tokenów dostępu, zawierająca informacje o tokenie. Składa się z id, tokenable, name, token, abilities, last\_used\_at, expires\_at, created\_at, updated\_at. Tabela jest tworzona automatycznie przez Laravela.
* friend\_lists - Zarządza listami znajomych użytkowników. Tabelę napisałem samodzielnie.
  + 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. Umożliwiająca zarządzanie stanem danej gry. Tabelę napisałem samodzielnie.
  + 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 podczas gry.
  + created\_at, updated\_at: Data utworzenia i ostatniej aktualizacji rekordu.
* cards - Tabela do zarządzania kartami wszystkich gier. Tabel składa się z kart każdej gry w tabaligames
  + id: Unikalny identyfikator.
  + card\_game\_id: Identyfikator nazwy karty.
  + user\_id: Identyfikator użytkownika posiadającego kartę (opcjonalnie).
  + where: Lokalizacja gdzie karta 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 nazwy wszystkich kart. Ta tabela jest używana do wyświetlania odpowiednej karty dla użytkownika
  + id: Unikalny identyfikator karty.
  + card: Nazwa karty w grze.

# Implementacja

## Opis interfejs użytkownika

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, logo

Opis wygenerowany automatycznie

*Rysunek 11. Strona rejestracji użytkownika*

Formularz na zdjęciu to standardowy formularz rejestracyjny. Zawiera następujące pola do wypełnienia:

* „Nick": pole, w którym użytkownik wpisuje swoje swój wymyślony nick, który musi być unikalny
* „E-mail": pole do wpisania adresu e-mail użytkownika.
* „Hasło": pole przeznaczone do wpisania hasła.
* „Potwierdź Hasło": pole, w którym użytkownik ponownie wpisuje hasło w celu potwierdzenia.

Na dole znajduje się pytanie „Już zarejestrowany?" z przyciskiem „ZAREJESTRUJ SIĘ" po prawej stronie, co sugeruje, że jeśli ktoś już ma konto, może przejść do strony logowania.

Obraz zawierający tekst, zrzut ekranu, logo, Czcionka

Opis wygenerowany automatycznie

*Rysunek 12. Strona przypomnienia hasła*

Kolejną stroną jest formularz, w którym wpisuje się e-mail, na który przychodź link, na który przenosi na stronę, gdzie można utworzyć nowe hasło.

Obraz zawierający tekst, zrzut ekranu, Czcionka, logo

Opis wygenerowany automatycznie

*Rysunek 13. Strona z formularzem do utworzenia nowego hasła*

Na zdjęciu jest formularz, na który się dostajemy klikając w link w emailu, gdzie wprowadzamy nowe hasło i ponownie hasło email jest uzupełniany automatycznie.

Obraz zawierający tekst, Czcionka, linia, zrzut ekranu

Opis wygenerowany automatycznie

*Rysunek 14. Strona po zalogowani się*

Po zalogowaniu się przenosi do panelu użytkownika, gdzie na górze znajduje się menu z trzema zakładkami: „Panel", „Gra" i „Znajomi". W prawym górnym rogu jest rozwijane menu użytkownika z nikiem użytkownika, pod którym znajdują się opcje : „Profil" i „Wyloguj się", co wskazuje na możliwość zarządzania kontem użytkownika lub wylogowania. Na środku strony znajduje się komunikat: „Jesteś zalogowany/a!".

Obraz zawierający tekst, zrzut ekranu

Opis wygenerowany automatycznie

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

W zakładce „Gra” znajdują się dwie sekcje pierwsza sekcja umożliwia dołączenie do gry, gdzie zostaliśmy zaproszeni do gry. Druga sekcja jest do tworzenia gier na naciskając przycisk zagraj przy nicku znajomego do tworzenie gier można tylko z znajomymi, których mamy. Na końcu jest przycisk, gdzie możemy zagrać z wybranym algorytmem.

Obraz zawierający tekst, zrzut ekranu, Prostokąt, zieleń

Opis wygenerowany automatycznie

*Rysunek 16. Strona planszy gry i historia zagranych kart*

Na powyższym zdjęciu znajduje się strona z grą, która jest używana do trybu graj samemu i multiplayera. Po prawej stronie znajduje się plansza do gry w karty. Na środku planszy znajduje się zakryta karta, która możliwa po kliknięciu dobraniu katy/kart zgodnie z zasadami gry. Na samej górze jest checkbox, który umożliwia pokazanie kart przeciwnika. Ta opcja jest dostęp tylko w trybie graj samemu. Po niżej checboxa są katy przeciwnika i jego nick w trybie multiplyer jest nick naszego znajomego, a w graj samemu jest BOT. Na dole planszy są karty zalogowanego grasz a po wyżej jego nick. Po lewej stronie znajduj się historia zagranych kart, gdzie za samej górze znajduje się ostania karta, a na samej dole początkowa karta.

Obraz zawierający tekst, Czcionka, zrzut ekranu

Opis wygenerowany automatycznie

*Rysunek 17. Wybór trybu gry i zasady gry*

Na stronie „Graj samemu” pod stołem znajduje się formularz z dwiema opcjami, który umożliwia wybranie algorytmu, z którym chcemy grać do wyboru są heurystyczny i Monte Carlo TreeSearch (MCTS). Dodatkowo przy MCTS jest ilość symulacji jakie algorytm ma przetestować. Po naciśnięciu przycisku START uruchamia się gry z wybranym algorytmem. Na końcu strony jest lista zasad jakie są w grze.

*Obraz zawierający zrzut ekranu, tekst

Opis wygenerowany automatycznie*

*Rysunek 18. Strona do zaproszenia i dołączenia do listy znajomych*

Na stronie Znajomi znajduje się trzy sekcje po lewej list lista znajomych zalogowanego użytkownika z możliwością usunięci z znajomych. Po prawej jest formularz, który po podaniu nicku znajomego umożliwia wysłanie zaproszenia do znajomych. Na dole znajduje się sekcja z otrzymanymi zaproszeniami do znajomych, gdzie możemy przyjąć lub usunąć zaproszenia.

Ostania stroną jest możliwość zarzadzania swoim profilem, której poniższe zdjęcia prezentują sekcji z jakich się składa.

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

*Rysunek 19. Pierwsza sekcja strony do zarządzania profilem*

W pierwszej sekcja jest możliwość zmiany dany użytkownika takie jak: nick i email, które muszą być unikalne. Dodatkowe przy zmianie emaila jest wysyłany email w celu weryfikacji emaila.

Obraz zawierający tekst, zrzut ekranu

Opis wygenerowany automatycznie

*Rysunek 20. Druga sekcja strony do zarządzania profilem*

Druga sekcja umożliwia zmianę hasła, gdzie trzeba podać aktualne hasło i nowe hasło oraz powtórzyć nowe hasło.

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

*Rysunek 21. Trzecia sekcja strony do zarządzania profilem*

Ostatnią sekcją jest usuwanie konta, gdzie znajduje się przycisk, który po kliknięciu wyświetla alert.

Obraz zawierający tekst, Czcionka, zrzut ekranu

Opis wygenerowany automatycznie

*Rysunek 22. Alert po kliknięciu usuń konto*

Alert, po którym wpisze się hasło następuje usunięcie konta i przeniesienie na stronę główną aplikacji.

## 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.

### Schemat blokowy algorytmu i opis

|  |  |
| --- | --- |
| Obraz zawierający szkic, rysowanie, diagram, Grafika liniowa  Opis wygenerowany automatycznie  *Rysunek 22. Pierwsza cześć algorytmu* | *Rysunek 23. Druga cześć algorytm* |

1. Start: Początek działania algorytmu.
2. Definiuj tablice cardsSpecial i cardsNotSpecial: Tworzone są dwie puste listy, które będą przechowywać karty specjalne i niespecjalne.
3. Pobranie znaku i figury zagraną kartę: pobierana jest wartość atrybutu alt z odkrytej karty (uncoverMainCardImg), z której wyodrębniane są znak i figura.
4. Wyciągnięcie znaku i figury kart: Dla każdej karty w talii (cards) pobierany jest atrybut alt, z którego wyodrębniane są znak i figura.
5. Czy pasuje znak lub figura pod zagraną kartę?: Sprawdzane jest, czy znak lub figura obecnie analizowanej karty pasują do znaku lub figury karty uncoverMainCardImg.
   1. Tak: Jeśli pasują, przepływ jest kierowany do punktu 6.
   2. Nie: Jeśli nie pasują, przepływ jest kierowany z powrotem do punktu 4, aby przeanalizować kolejną kartę.
6. Czy to znak to: 02, 03, 0J, 0Q, 0K?: Jeśli karta pasuje, jest dodatkowo sprawdzane, czy znak karty jest jednym ze specjalnych znaków.
   1. Tak: Jeśli tak, karta jest dodawana do listy cardsSpecial.
   2. Nie: Jeśli nie, karta jest dodawana do listy cardsNotSpecial.
7. Czy liczba kart przeciwnika jest mniejsza niż 3?: Sprawdzenie, czy liczba kart w posiadaniu przeciwnika jest mniejsza niż 3.
   1. Tak: Jeśli tak, algorytm przechodź do punktu 8.
   2. Nie: Jeżeli nie, algorytm przewodzi do punktu 9.
8. Czy w taktach cardsSpecial jest karta z znakiem: 02, 03, 0K?: Jest tu sprawdzane czy jest tu karta 02, 03, 0K w tabeli cardsSpecial.
   1. Tak: Jeśli tak, algorytm zwraca pierwszą kartę, która jest z znaków: 02, 03, 0K i kończy działanie.
   2. Nie: Jeżeli nie, algorytm przewodzi do punktu 9.
9. Czy suma wynosi 0 i cardsNotSpecial zawiera karty?: Jest tu sprawdzane czy suma w grze jest równa 0, i tablica cardsNotSpecial zawiera karty
   1. Tak: Jeśli tak, algorytm zwraca pierwszą kartę z tablicycardsNotSpecial i kończy działanie.
   2. Nie: Jeżeli nie, przechodzi do punktu 10
10. Czy w cardsSpecial są jakieś karty?: Tu jest sprawdzanie czy w tabeli cardsSpecial jest jakaś karta.
    1. Tak: Jeśli lista cardsSpecial zawiera karty, algorytm zwraca pierwszą kartę z tej listy.
    2. Nie: Jeżeli lista jest pusta, algorytm kończy działanie i zwraca wartość null, oznaczającą brak możliwości zagrania kartą.

## MCTS (Monte Carlo Tree Search)

### Opis algorytmu

Metoda MCTS, czyli Monte-Carlo TreeSearch, 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. Metoda MCTS skupia się na analizie najbardziej obiecujących ruchów, opierając rozrost drzewa wariantów na losowym próbkowaniu przestrzeni przeszukiwań.

Sposób wyboru węzła odbywa się na podstawie wzoru wyliczający UCT (Upper ConfidenceBound 1 applied to trees), który wylicza czy bardziej opłaca się eksplotacja czyli wykonywanie kolejnych ruchów w danej gałęzi do wygrania czy eksploatację szykanie kolejnej gałęzi do wykonania mniej oczywistego ruchu.

gdzie:

* – ilość wygranych w tym węźle
* – ilość wykonanych iteracji danym węźle
* – parametr eksploracji zazwyczaj równa
* – jest to ilości symulacji rodzica

### Opis implementacji

import {shuffleCards, special\_card\_check,PLAYERS} from'./helper.js'

*Kod1. Importowanie metod i obiektów pomocniczych*

Importowanie metod i obiektów pomocniczych, do algorytmu takich jak shuffleCards do tasowania kart, special\_card\_check do sprawdzenia warunku kart specjalnych, PLAYERS Object przechowując stałe nazwy graczy.

exportfunctionmcts(botCards, youCards, coverMainCards, uncoverMainCards, PLAYERS, sum) {

    letinitialState=newGameState(botCards, youCards, coverMainCards, uncoverMainCards, PLAYERS.BOT, sum)

    letmctsIteration=document.getElementById("mctsIteration").value

    letbestMove=runMCTS(initialState, mctsIteration)

    if(bestMove=="add") returnnull

    elsereturnbestMove

}

*Kod2. Metoda mcts*

Metoda mcts jest eksportowana, którą można użyć w grze przyjmuje następujące wartości: karty przeciwnika, użytkownika, stos kart do pobrania i już zagranych kart. Karty przeciwnika i stos kart do pobrania w głównej rozgrywce są łączone i tasowanie następnie dzielone na odpowiednie do ich poprzednich ilości. To sprawia że algorytm nie wie jakie karty ma przeciwnik i takie są do pobrania w następnych ruchach w oryginalnej rozgrywce.

W pierwszej linii, kod jest tworzony stan gry na podstawie klasy GameState, która zostanie późnej opisana w kolejnej linii jest dobierany ilości iteracji jakie algorytm ma wykonać. Następnie jest wykonywania metoda runMCTS, która zostanie dokładnie opisana poniżej jej wynikiem jest karta lub wartość add, która mówi że ruchem jest dobranie kart/y.

Na końcu jest zwracana karta lub wartości null do głównej gry.

functionrunMCTS(rootState, iterations) {

    constrootNode=newMCTSNode(null, null, rootState)

    for (leti=0; i<iterations; i++) {

        letnode=rootNode

        letstate=rootState

        // Selection

        while (node.children.length&&node.state.getPossibleMoves().length) {

            node=node.selectChild()

            state=state.makeMove(node.move)

        }

        // Expansion

        if (node.state.getPossibleMoves().length) {

            constmoves=node.state.getPossibleMoves()

            constmove=moves[Math.floor(Math.random() \*moves.length)]

            state=state.makeMove(move)

            node=node.addChild(move, state)

        }

        // Simulation

        while (!state.isGameOver()) {

            constmoves=state.getPossibleMoves()

            constmove=moves[Math.floor(Math.random() \*moves.length)]

            state=state.makeMove(move)

        }

        // Backpropagation

        letresult=state.getWinner() ==PLAYERS.BOT?1:0

        while (node) {

            node.update(result)

            node=node.parent

        }

    }

    // Choose the best move at the root

    returnrootNode.selectChild().move

}

*Kod3. Metoda runMCTS*

Na początku metody runMCTS jest tworzony pierwszy węzeł na podstawie klasy MCTSNode, która będzie opisana poniżej. Następnie jest wykonywana pętla z ilości iteracji jakie podał użytkownik, w której jest robiona kopia korzenia i stanu gry. W pierwszym pętli w forche sprawdzany jest warunek w Chile czy węzeł posiada dzieci i czy są możliwe ruchy. Jeśli tak to wybierany jest nowy węzeł na podstawie wzoru podanego powyżej i robiony ruch który zwraca nowy jest gry.

Jeśli nie wychodzimy z drugiej pętli i sprawdzany jest czy są możliwe ruchy jeśli tak to jest losowany ruch i wykonywany, który zwraca nowy stan gry i tworzony nowy węzeł i dodawany jako dziecko wynikiem jest nowy węzeł, który jest zastąpiony poprzednim.

Następnie symulacja gry jest dalej wykonywana aż do końca gry. Na końcu jest sprawdzanie kto wygrał, jeśli bot do rezultat jest przypisywane 1, a jeśli nie - to 0. Następnie wynik jest przypisywany do węzła i wraca się do rodzica węzła.

Jako wynik jest zwracany ruch, najlepszego węzła wybranego z powyższego wzoru.

classGameState {

    constructor(botCards, humanCards, coverCards, uncoverCards, player=PLAYERS.HUMAN, suma=0) {

        this.botCards= [...botCards]

        this.humanCards= [...humanCards]

        this.coverCards= [...coverCards]

        this.uncoverCards= [...uncoverCards]

        this.player=player

        this.suma=suma

    }

*Kod 4. Deklaracja klasyGameSate z konstruktorem*

W konstruktorze są kopiowanie wartości do stanu gry takie jak: katy gracza, przeciwnika, do zabrania, już zagrane, kto teraz ma wykonywać ruch i suma kart do zabrania.

    getPossibleMoves() {

        letpossibleCard= []

        letunCard=this.uncoverCards.at(0)

        letuncoverCardSign=unCard.substring(0, 2)

        letuncoverCardFigure=unCard.substring(3, unCard.length)

        if(this.player==PLAYERS.BOT){

            this.botCards.forEach(card=> {

                letcardSign=card.substring(0,2)

                letcardFigure=card.substring(3, card.length)

                if(cardSign==uncoverCardSign||cardFigure==uncoverCardFigure){

                    if(this.suma==0) possibleCard.unshift(card)

                    elseif(special\_card\_check(cardSign)) possibleCard.unshift(card)

                }

            })

        }

        if(this.player==PLAYERS.HUMAN){

            this.humanCards.forEach(card=> {

                letcardSign=card.substring(0,2)

                letcardFigure=card.substring(3, card.length)

                if(cardSign==uncoverCardSign||cardFigure==uncoverCardFigure){

                    if(this.suma==0) possibleCard.unshift(card)

                    elseif(special\_card\_check(cardSign)) possibleCard.unshift(card)

                }

            })

        }

            if(this.suma<=this.coverCards.length&&this.coverCards.length>0 ){

                possibleCard.push("add")

            }

        returnpossibleCard

    }

*Kod 4. MetodagetPossibleMoves w klacieGameSate*

Metoda getPossibleMoves zwraca wszystkie ruch jaki są możliwe w danym stanie gry na początku jest wyciągana ostania zagrana karta i zapisana do osobnych zmiennych znak i figura karty. Następnie sprawdzane jest kto wykonuje ruch w zależności, czy jest ruch są sprawdzane katy tego gracza, czy karta pasuje od względem znaku lub figury.

Jeśli pasuje sprawdzane jest, czy suma jest równa. Jeśli tak, to karta jest dodawana do możliwych ruchów. Jeśli nie, to sprawdzane jest, czy jest to karta specjalna. Jeśli tak, dodawana jest do możliwych ruchów. Na końcu, jeśli suma jest mniejsza do możliwości pobrania karty, dodawany jest „add” w celu informacji, że możliwy jest ruch pobrania kart dodatkowych.

    makeMove(move) {

        letnewBotCards= [...this.botCards]

        letnewHumanCards= [...this.humanCards]

        letnewCoverCards= [...this.coverCards]

        letnewUncoverCards= [...this.uncoverCards]

        letnewSuma=this.suma

        if(this.player==PLAYERS.BOT){

            if(move=="add"){

                for(leti=1; i<newSuma; i++) newBotCards.unshift(newCoverCards.shift())

                newBotCards.unshift(newCoverCards.shift())

                newSuma=0

            }

            else{

                newUncoverCards.unshift(move)

                newBotCards.splice(newBotCards.indexOf(move), 1)

            }

        }

        if(this.player==PLAYERS.HUMAN){

            if(move=="add"){

                for(leti=1; i<newSuma; i++) newHumanCards.unshift(newCoverCards.shift())

                newHumanCards.unshift(newCoverCards.shift())

                newSuma=0

            }

            else{

                newUncoverCards.unshift(move)

                newHumanCards.splice(newHumanCards.indexOf(move), 1)

            }

        }

        if(move!="add"){

            letsing=move.substring(0, 2)

            switch(sing){

                case"02":

                    newSuma+=2

                    break

                case"03":

                    newSuma+=3

                    break

                case"0Q":

                    newSuma=0

                    break

                case"0J":

                    if(newSuma<=5)newSuma=0

                    elsenewSuma-=5

                    break

                case"0K":

                    newSuma+=5

                    break

            }

        }

        else{

            if(newCoverCards.length==0){

                letc=newUncoverCards.shift()

                newCoverCards=shuffleCards(newUncoverCards)

                newUncoverCards.splice(0, newUncoverCards.length)

                newUncoverCards.unshift(c)

            }

        }

        letnewPlayer=this.player===PLAYERS.HUMAN?PLAYERS.BOT:PLAYERS.HUMAN

        returnnewGameState(newBotCards, newHumanCards, newCoverCards, newUncoverCards, newPlayer, newSuma)

    }

*Kod 4. Metoda makeMove w kalsieGameState*

W tej metodzie na początku kopiowane są wartości stanu gry. Następnie jest sprawdzenie, czyj jest ruch. Po wybraniu gracza wykonuje się ruch gracza i jeśli jest to add, to dodawane są mu karty zgodnie z sumą. Jeśli jest to co innego, to dodawana jest karta do zagranych i graczowi usuwana. Po wykonaniu ruchu suma jest zerowania lub aktualizowana zgodnie z zasadami gry. Przed zwróceniem nowego stanu gry, sprawdzanie jest czy potrzeba jest zabranie zagranych kart i dodanie do możliwych zabrania i ich potasowanie. Na końcu jest aktualizowany, kto teraz ma wykonywać ruch i zwracany nowy stan gry.

    isGameOver() {

        if(this.humanCards.length==0||this.botCards.length==0|| ((this.coverCards.length==0&&this.uncoverCards.length==1) || (this.coverCards.length<this.suma))){

            returntrue

        }

        elsereturnfalse

    }

*Kod 4. Metoda isGameOver w kalsieGameState*

Metoda zwraca prawdę, jeśli któryś z graczy nie ma kart lub nie ma kart do zabrania lub suma do zabrania kart jest większa możliwych ich zabranie.

    getWinner(){

        letcountHumanCards=this.humanCards.length

        letcountBotCards=this.botCards.length

        if(countHumanCards==0||countHumanCards<countBotCards) returnPLAYERS.HUMAN

        if(countBotCards==0||countBotCards<countHumanCards) returnPLAYERS.BOT

    }

*Kod 4. MetodamakeMove w kalsie GameState*

W tej metodzie zwracana jest nazwa wygranego gracza, jeśli nie ma kart lub ma mniej kart niż przeciwnik.

classMCTSNode {

    constructor(parent=null, move=null, state) {

        this.parent=parent

        this.move=move

        this.state=state

        this.children= []

        this.wins=0

        this.visits=1

    }

*Kod 4.DeklaracjaklasyMCTSNode z konstruktorem*

Jest to klasa reprezentująca: węzeł, który jako wartości przechowuje swój rodzic, kartę, która wykonał jako ruch, stan gry, swoje dzieci, ilości wygranych, jakie przechowuje w swoich dziedzicach i ilość iteracji na jakie zostały wykonane na tym węźle.

    selectChild() {

        letselectedChild

        letbestValue=-Infinity

        this.children.forEach(child=> {

            letuctValue=

                (child.wins/child.visits) +

                (Math.sqrt(2) \*Math.sqrt(Math.log(this.visits) /child.visits))

            if (uctValue>bestValue) {

                selectedChild=child

                bestValue=uctValue

            }

        })

        returnselectedChild

    }

*Kod 4. Metoda selectChild z klasy MCTSNode*

Metoda wylicza najlepszy węzeł z swoich dzieci na podstawie wzoru podanego wyżej. Na końcu jest wykonywany return, który zwraca najlepszy węzeł.

    addChild(move, state) {

        constchild=newMCTSNode(this, move, state)

        this.children.push(child)

        returnchild

    }

*Kod 4. Metoda addChild z klasy MCTSNode*

Metoda dodaje nowy węzeł jako dziecko, na którym jest wykonywana, do którego jest przypisywana karta i stan gry jest metoda i zwraca nowo utworzony węzeł.

    update(result) {

        this.visits++

        this.wins+=result

    }

*Kod 4. Metoda update z klasy MCTSNode*

W ostatniej metodzie klasy MCTSNode aktualizowana jest ilość iteracji na którym odbyły się operację na wywołanym węźle i jest przypisywany czy udało się wygrać, czy nie.

## Porównanie algorytmów

Po rozegraniu 5 gier z każdym z algorytmów okazało się, że więcej wygranych ma MCTS, ponieważ na 5 gier wygrał wszystkie a podejście heurystyczne 4. To wskazywałoby, że jest nieznacznie lepszym algorytmem. Różnice między tymi algorytmami są znaczone.

Ponieważ algorytm z podejścia heurystycznego oparty na własnych zasadach jest prostszy, dzięki czemu jest szybki w implementacji i analizuje aktualny stan gry bez prowadzenia symulacji gry.

Natomiast trudniejsza w implementacjijest algorytmem szukający jaki ruch wykonać prowadzi symulację wybranych przez siebie gier na podstawę wzoru.

# 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.

Następną funkcjonalnością jest wprowadzenie dodatkowych zasad co umożliwi użytkownikom lepsze się wczucie w grę i podniesie atrakcyjność gry. Do funkcjonalności zasad gry oraz możliwością dalszego rozwoju, jest możliwość personalizacji zasad. Zwiększy również zaangażowanie użytkowników, dając im poczucie kontroli nad swoim doświadczeniem gry.

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 grymultiplayer 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

Podsumowując wykonaną przez siebie pracę ,takią jak: analiza tematu AI w grach komputerowych, zaprojektowanie i implementacja dwóch algorytmów podejścia heurystycznego, opartego o wymyślone przez siebie zasady zachowania przeciwnika i MCTS (Mont-Carlo TreeSearch) i implantacja muliplayer dla dwóch graczy.

# Bibliografia

* 1. Jak powstaje sztuczna inteligencja w grach komputerowych - <https://geek.justjoin.it/powstaje-sztuczna-inteligencja-grach-komputerowych/>
  2. Monte-Carldo Tree Search - <https://pl.wikipedia.org/wiki/Monte-Carlo_Tree_Search>
  3. Laravel -<https://laravel.com>
  4. Laravel Webssockets - <https://beyondco.de/docs/laravel-websockets/getting-started/introduction>
  5. Użyte karty w aplikacji - <https://www.freepik.com/free-vector/full-deck-poker-playing-cards_6086127.htm>

# 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>