AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

**WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI,
INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ**

KATEDRA INFORMATYKI STOSOWANEJ

Praca dyplomowa magisterska

Analiza i porównanie wybranych algorytmów dla gry karcianej
Analysis and comparison of selected algorithms for the card game

Autor: *Damian Malarczyk*
Kierunek studiów: *Informatyka*
Opiekun pracy: *dr inż. Edyta Kucharska*

Kraków, 2015

Uprzedzony o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.): „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystycznego wykonania albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie.", a także uprzedzony o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (t.j. Dz. U. z 2012 r. poz. 572, z późn. zm.): „Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godności studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwanym dalej «sądem koleżeńskim».", oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

Podziękowania

Spis treści

1. Wprowadzenie

1.1. Przedmiot pracy

Od kilku lat coraz większą popularnością cieszą się wszelkiego rodzaju gry planszowe i karciane. Przyciągają nie tylko coraz lepszą oprawą graficzną, lecz również ciekawą mechaniką pozwalającą na stosowanie różnych taktyk. Z tego powodu stanowią szerokie pole do testowania algorytmów optymalizujących dostępne ruchy tak, by zapewnić zwycięstwo.

Są również gry, w których kluczową rolę odgrywa tak zwana 'intuicja'. Obliczenie całego drzewa dostępnych ruchów jest zbyt skomplikowane i decyzja musi zostać podjęta na podstawie niepełnych danych. Przykładem takiej gry jest Łove Letter", gra niezbyt skomplikowana, jednak zawierająca dużo interakcji i możliwych ścieżek rozwoju sytuacji.

Inspirując się wyżej wymienioną grą, w poniższej pracy porównuję trzy algorytmy podejmowania decyzji w grze:

– probabilistyczny - który będzie podejmował decyzję w sposób losowy na podstawie prawdopodobieństwa wystąpień kart,

– zachłanny - który będzie wybierał zawsze najbardziej prawdopodobny scenariusz,

– Monte Carlo Tree Search - algorytm heurystyczny, który podejmowane decyzje opiera na symulacjach.

1.2. Cel pracy

1.3. Zawartość pracy

Rozdział 1 jest wprowadzeniem definiującym przedmiot pracy. W rozdziale 2 opisana została gra karciana wraz z problemem który przedstawia. Rozdział 3 przedstawia trzy proponowane algorytmy rozwiązujące problem. W rozdziale 4 zebrane są poszczególne etapy tworzenia aplikacji:

– analiza wymagań,

– koncepcja wykonania,

– wykorzystane technologie,

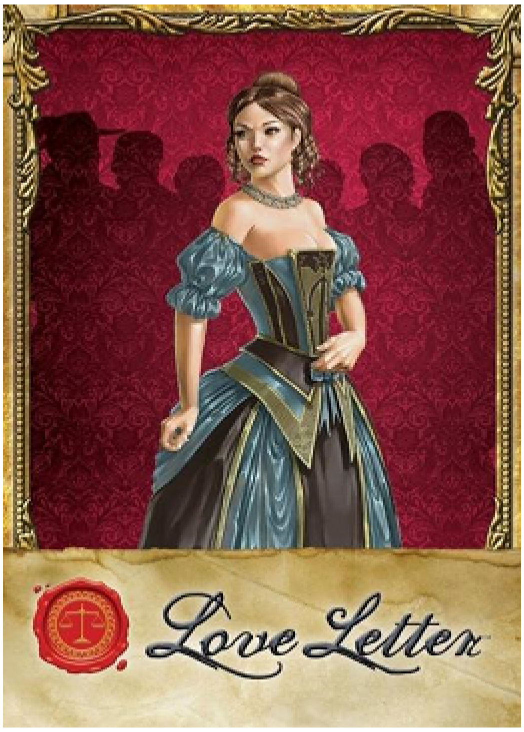
– diagramy,

– prezentacja systemu.

W rozdziale 5 zebrane są statystyki związane z implementacją algortytmów. Rozdział 6 stanowi podsumowanie pracy.

2. Gra karciana Love Letter

W tym rozdziale opisuję kontekst oraz zasady gry Love Letter. Opisuję działanie każdej karty oraz przedstawiam główny cel gry - wygranie określonej ilości rund. Następnie przedstawiam problem i analizuję jego złożoność. Wszystkie załączone zdjęcia oraz instrukcja zaczerpnięte są z [1] oraz [2].



Rysunek 2.1. Love Letter - okładka

2.1. Opis zasad gry

W trakcie gry wcielamy się w rolę jednego z adoratorów księżniczki starającego się o zdobycie jej serca. W tym celu przygotowaliśmy list miłosny, który chcemy jej dostarczyć. Niestety, księżniczka pogrążona jest obecnie w żałobie i nie przyjmuje do siebie nikogo obcego, w związku z czym musimy znaleźć inny sposób na przekazanie jej naszego listu. Oprócz księżniczki, na dworze znajdują się inne postacie, z których każda ma mniejszy lub większy dostęp do komnat naszej wybranki i może oddać jej list. Przekazujemy więc naszą przesyłkę swojemu tajnemu posłańcowi, a na koniec gry księżniczka jako pierwszy przeczyta ten list, który został przekazany przez najbardziej zaufaną postać. Serce wybranki zdobywa gracz, który jako pierwszy przekaże w ten sposób od 4 do 7 listów, w zależności od liczby graczy.

Cel i ustawienie początkowe rundy

Love Letter rozgrywa się jako serię rund. Grę wygrywa gracz o następującej ilości wygranych rund:

- 7 w grze na 2 graczy,
- 5 w grze na 3 graczy,
- 4 w grze na 4 graczy.

Każda runda dzieli się na tury, w których naprzemiennie jeden z graczy wykonuje ruch. Grę wygrywa ten z nich, który na końcu ostatniej tury posiada kartę o wyższym numerze.

Ustawienie początkowe każdej rundy wygląda następująco:

- przetasuj karty
- odrzuć 1 wierzchnią kartę nie odkrywając jej (nie bierze udziału w rundzie),
- jeśli gra tylko 2 graczy, odrzuć 3 wierzchnie karty, odkryte,
- rozdaj po 1 karcie wszystkim graczom,
- jeśli jest to pierwsza runda, turę zaczyna gracz, który jako ostatni był na randce, w przeciwnym wypadku zaczyna zwycięzca poprzedniej rundy.

Tura gracza i opis kart

Podczas swojej tury gracz dociąga jedną kartę ze stosu. Następnie wybiera jedną z dwóch kart, które posiada już w ręce, kładzie ją przed sobą tak, by była widoczna dla wszystkich i zastosowuje opisany na niej efekt - nawet jeśli jest negatywny. Zagrana karta pozostaje odkryta przez całą rundę, a druga pozostaje w ręce. Następnie tura przechodzi na osobę po lewej stronie aktywnego gracza.

W grze znajduje się 16 kart, w 8 typach. Każda z typów kart posiada wartość od 1 do 8. Są to kolejno: 4 karty Strażniczki, po 2 karty Kapłana, Barona, Pokojówki i Księcia, oraz po jednej karcie Króla, Hrabiny i Księżniczki. Ich szczegółowy opis wraz z wyglądem znajduje się poniżej:

Rysunek 2.2. Strażniczka

Na rysunku 2.2 przedstawiona jest karta typu Strażniczka. Zagrywając tę kartę należy wskazać jednego z pozostałych graczy i odgadnąć kartę którą posiada. Jeśli karta została prawidłowo odgadnięta, wskazany gracz odrzuca ją i przegrywa rundę.

Rysunek 2.3. Kapłan

Rysunek 2.3 przedstawia kartę typu Kapłan. Zagrywając tę kartę należy podglądnąć kartę wybranego gracza.

Rysunek 2.4. Baron

Na rysunku 2.4 przedstawiona jest karta typu Baron. Po zagraniu tej karty należy w ukryciu porównać drugą posiadaną kartą z wybranym graczem. Następnie ten gracz, który ma kartę o mniejszej wartości odrzuca swoją kartę i przegrywa rundę. W przypadku remisu nic się nie dzieje.

Rysunek 2.5. Pokojówka

Rysunek 2.5 przedstawia kartę typu Pokojówka. Zagranie tej karty sprawia, że gracz jest niewrażliwy na efekt pozostałych kart do czasu swojej następnej tury.

Rysunek 2.6. Książe

Na rysunku 2.6 przedstawiona jest karta typu Książę. Zagranie pozwala wybrać dowolnego gracza (w tym siebie), zmusić go do odrzucenia posiadanej karty i pociągnięcia następnej.

Rysunek 2.7. Król

Rysunek 2.7 przedstawia kartę typu Król. Po jej zagraniu należy wymienić się drugą kartą z innym graczem.

Rysunek 2.8. Hrabina

Rysunek 2.8 przedstawia kartę typu Hrabina. Ta karta ma działanie pasywne. Nie wywiera efektu po zagraniu, natomiast zmusza gracza do jej zagrania jeśli równocześnie posiada na ręce kartę typu Książę lub Król.

Rysunek 2.9. Księżniczka

Na rysunku 2.9 przedstawiona jest karta typu Księżniczka. Zagranie tej karty oznacza natychmiastową przegraną w rundzie. Ta zasada działa również, gdy gracz został zmuszony do zagrania tej karty, np. przez efekt karty Książe.

2.2. Definicja i analiza problemu

Z wyżej przedstawionych zasad wynika, że gra cechuje się wysokim stopniem losowości i jest niedeterministyczna, więc można ją przedstawić jako problem optymalizacyjny[4]. W związku z tym powstaje również pytanie, jaka jest najlepsza strategia mieszana[5] maksymalizująca prawdopodobieństwo wygrania gry. Dla dalszych rozważań zakładam, że w gra toczy się pomiędzy dwoma graczami.

Najprostszym sposobem na znalezienie takiej strategii, byłoby stworzenie drzew probabilistycznych dla wszystkich możliwych stanów początkowych gry a następnie opracowanie algorytmu podejmowania decyzji opartego na danych statystycznych. Jednakże ilość i rozmiar tych drzew może być zbyt duża by znaleźć rozwiązanie problemu w rozsądnym czasie. Z tego powodu postanowiłem najpierw oszacować ile jest wszystkich możliwych przebiegów gry. Nim przejdę do obliczeń, wprowadzę kilka definicji by ustandaryzować używane pojęcia:

– *Decyzja* - inaczej *Zagranie*, jest to typ karty wraz ze sposobem jej wykorzystania. Przykładowo: Zagranie karty typu Strażniczka z wyborem karty typu Król, lub zagranie karty typu Książę z wyborem na gracza przeciwnego.

– *Podjęcie decyzji* - to wybór zagrania według zastosowanego algorytmu, które zostanie użyte jako ruch w grze.

– *Strategia* - ciąg decyzji podejmowanych przez danych algorytm

– *Scenariusz* - inaczej przebieg gry, chronologiczny ciąg decyzji podjętych przez obu graczy od początku do końca rundy; Jedna ze ścieżek w drzewie probabilistycznym dla danego stanu początkowego rundy.

Oszacowanie ilości rozwiązań

Na przebieg każdej rundy wpływ mają następujące czynniki:

– Kolejność kart w talii na początku rundy

– Zagrywanie kart przez graczy

Zacząłem od oszacowania ilości możliwych stanów początkowych. W każdej rundzie bierze udział wszystkie 16 kart. Zakładając, że każda z nich jest unikalna, to liczba wszystkich możliwych kolejności kart to permutacja, którą obliczam wzorem podanym w [3]:

$$P_n = n!\text{ , gdzie } n \in N^+$$

Dla n = 16, $n! = 20922789888000$. Część kart się powtarza, więc tę liczbę należy jeszcze podzielić przez permutacje powtarzających się kart Strażniczki, Kapłana, Barona, Pokojówki oraz Księcia. Razem jest to $4! * 2! * 2! * 2! * 2! = 384$. Tak więc liczba unikalnych kolejności kart wynosi:

$$20922789888000/32 = 54486432000$$ - 54mld, 864mln i 432 tys.

Nie jest to jednak ostateczne oszacowanie możliwych rozwiązań. W każdej turze gracz ma do wyboru co najmniej dwa zagrania, ponieważ tyle ma dostępnych kart. Jednakże, w przypadku karty Strażniczki możliwości jest więcej, ponieważ można wytypować 7 typów kart, a w jednym przypadku może to pokonać przeciwnika i skończyć rundę. W związku z tym, by oszacować liczbę rozwiązań przy ustalonym stanie początkowym, zastosowałem następujące ograniczenia:

– zgodnie z zasadami gry dla dwóch graczy, odrzucam łącznie 4 pierwsze karty (1 zakryta, 3 odkryte).

– rozdaję po 1 karcie obu graczom. Pozostaje 10 kart w talii.

– zakładając, że żadna decyzja nie spowoduje zakończenia rundy, gracze łącznie 10 razy pociągną kartę, więc podejmą 10 decyzji.

– każdą decyzję można przedstawić jako 0 (zagranie posiadanej karty) lub 1 (zagranie pociągniętej karty) - w tym miejscu jeśli istnieje możliwość zagrania karty w wieloraki sposób, upraszczam to do jednego zagrania niekończącego rundę.

Na podstawie powyższego można oszacować, że możliwych rozwiązań dla danego stanu początkowego jest $2^{10} = 1024$. Łącząc to z ilością stanów początkowych, utrzymujemy przybliżoną liczbę rozwiązań:

$$54486432000 * 1024 = 55794106368000 \approx 5.6 * 10^{13}$$

Z uwagi na rząd wielkości, stworzenie optymalnej strategii na podstawie analizy statystycznej wszystkich dostępnych rozwiązań byłoby czasochłonne. Z tego powodu, zamiast odpowiadać na pytanie „Jaka jest najlepsza strategia podejmowania decyzji?”, dużo łatwiej będzie odpowiedzieć na pytanie „która z podanych strategii jest najlepsza?”. Kierując się tą zasadą, w następnym rozdziale opisałem wybrane strategie, których skuteczność sprawdzę implementując je w napisanej przeze mnie aplikacji. Pozostaje jeszcze sformalizować przedstawiony problem za pomocą modelu matematycznego.

2.3. Model matematyczny problemu

W celu ujednolicenia używanych pojęć i skrótów, przedstawię je następująco:

Tablica 2.1. Pojęcia i skróty

Nazwa karty	Skrót nazwy	Możliwe zagrania	Skrót zagrania
Strażniczka	Str	Strażniczka z wyborem na [Nazwa karty]	Str_Kap, Str_Bar, ... , Str_K-a
Kaplan	Kap	Zagranie kapłana	Kap_Z
Baron	Bar	Zagranie barona	Bar_Z
Pokojówka	Pok	Zagranie pokojówki	Pok_Z
Książe	K-e	Książe na siebie; Książe na przeciwnika	K-e_S, K-e_P
Król	Krl	Zagranie Króla	Krl_Z
Hrabina	Hra	Hrabina, gdy druga karta w ręce to Król lub Książe; Zagranie Hrabiny	Hra_K-e_Krl, Hra_Z
Księżniczka	K-a	Zagranie Księżniczki	K-a_Z

Inne:

– „ręka” - karty, które gracz aktualnie posiada.

– Z - zbiór zagrań (ruchów), oznaczanych jako z.

– DK - druga karta, którą gracz posiada w ręce.

– Karta, $typKarty$ - jeden z ośmiu rodzajów kart.

– Karty, $typyKart$ - zbiór typów kart.

– $W(Karta)$ - wartość karty od 1 do 8.

Patrząc z perspektywy jednego z graczy, cała gra składa się z szeregu etapów(tur), gdzie w każdym etapie gracz podejmuje decyzje, a stan początkowy następnego etapu jest wynikiem dwóch akcji: podjętej decyzji (której efekt możemy przewidzieć z pewnym prawdopodobieństwem) i reakcji gracza drugiego, której prawdopodobieństwo jest nieokreślone. Dodatkowo, rzeczywisty stan każdego etapu nie jest w pełni znany graczowi. Z tych cech wynika, że grę „Love Letter” możemy sklasyfikować jako **dynamiczny(wieloetapowy) proces podejmowania decyzji w warunkach niepewności**[6]. Jej zapis formalny można przedstawić jako grę w postaci ekstensywnej[7]:

$$T = (S, R), \text{ gdzie}$$

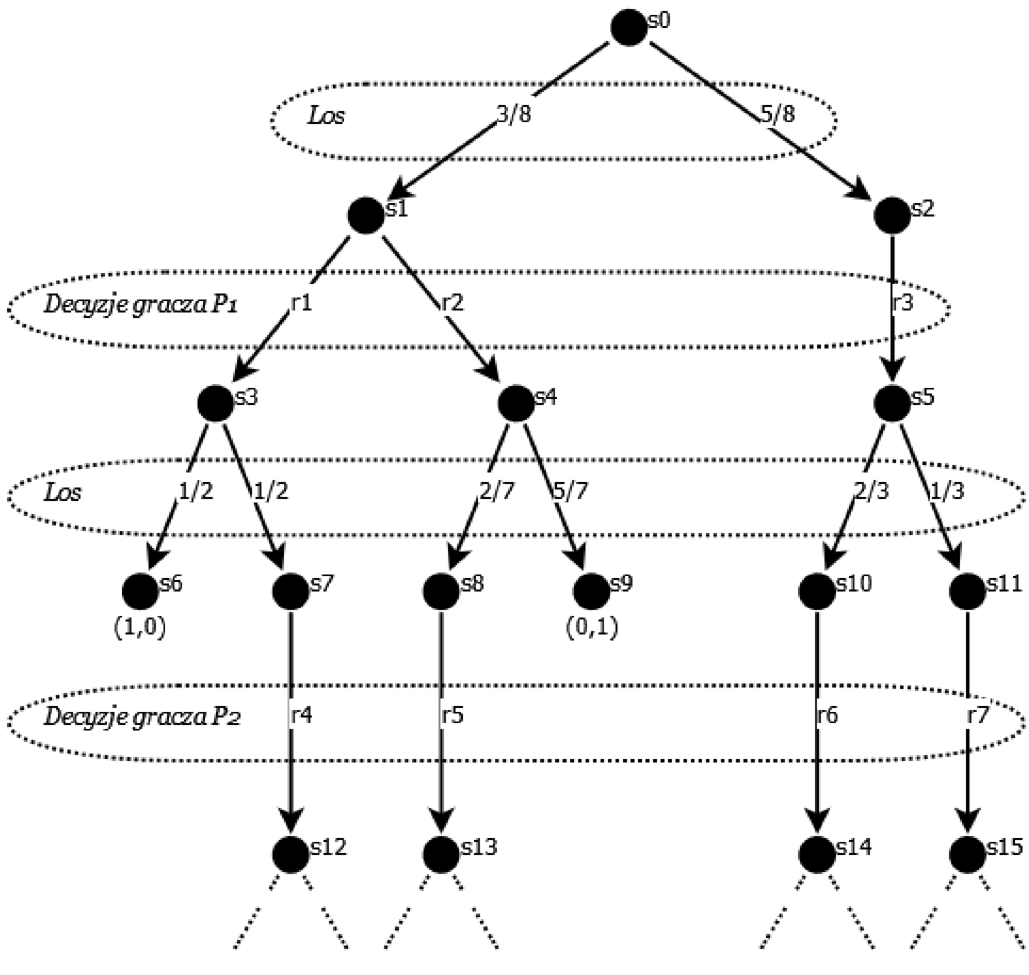
– T - graf skierowany

– S - zbiór wierzchołków (węzłów)

– R - relacja określona na parach wierzchołków (łuki)

Przypisując każdemu wierzchołkowi $s \in S$ jeden ze stanów rundy, a każdemu łukowi $r \in R$ ruch danego gracza ze zbioru Z lub udział losu, otrzymamy zapis pozwalający jednoznacznie określić możliwe przebiegi konkretnej gry, podjęte przez graczy decyzje, ich stan wiedzy na każdym etapie oraz wypłaty graczy (wynik gry).

Ponieważ w każdej rundzie bierze udział dwóch graczy, oznaczmy ich jako P_1 i P_2. Dodatkowo, część w której losowana jest karta oznaczmy jako Los. Zauważmy, że skoro korzeń s_0 to stan początkowy rundy przed losowaniem kart z talii, to wszystkie łuki wychodzące z s_0 są efektem losowania, a łuki z następnych poziomów należą kolejno do gracza P_1, $Losu$, gracza P_2 itd. Poglądowo zilustrowałem to w następujący sposób:



Rysunek 2.10. Pogląd gry w postaci ekstensywnej

Ważne wnioski:

– wszystkie łuki wychodzące z węzłów należących do $Losu$ oznaczone są prawdopodobieństwem sumującym się do 1.

– węzły s_6 i s_9 są liśćmi drzewa i zawierają wypłaty graczy (wynik gry) - 1 oznacza zwycięstwo, 0 oznacza przegraną. Przykładowo w węźle s_6 wygrał gracz P_1

– zdarza się, że gracze nie wiedzą w jakim węźle danego poziomu drzewa się znajdują. Załóżmy, że łuki r_4 i r_5 są takim samym ruchem dostępnym dla gracza P_2. Gracz nie wie czy gra znajduje się w stanie s_7 czy s_8, ponieważ oba stany udostępniają ten sam zbiór zagrań, co odzwierciedla niepełną wiedzę gracza $P2$. Grupa takich stanów, które udostępniają ten sam zbiór zagrań, tworzy **zbiór informacyjny**. W dalszej części pracy takie zbiory będziemy określać jako I_n, a zbiór udostępnianych zagrań jako X_n (zauważmy, że $X_n \subset Z$). Zbiór węzłów, które następują po węzłach z I_n nazywamy **loterią**.

Przedstawiony model jednoznacznie określa stan gry i graczy. Dla uwzględnienia decyzyjności, do takiego modelu należy dodać jeszcze funkcję wyboru (decyzji). Oznaczmy ją jako d:

$$d(s) \rightarrow r, \text{ gdzie } r \text{ jest dowolnym łukiem wychodzącym ze stanu } s$$

Jest to deklaracja ogólna. Zdecydowanie ważniejsze są funkcje wyboru należące do graczy P_1 i P_2, które uwzględniają zbiór informacyjny. Oznaczmy je odpowiednio jako d_1 i d_2. Ich postać będzie następująca:

$$d_{1,2}(I_n, X_n) \rightarrow z, \text{ gdzie } z \text{ jest dowolnym zagraniem należącym do } X_n$$

W takiej postaci widać, że gracze podejmą decyzję mając niepełną informację na temat stanu gry. W następnym rozdziale przedstawiam algorytmy, które będą dokładnie definiować funkcję wyboru gracza.

3. Przegląd wybranych algorytmów i innych rozwiązań

W tym rozdziale przedstawiam algorytmy definiujące funkcję wyboru. Są to kolejno: losowy, zachłanny, minimaksowy oraz Monte Carlo Tree Search. Dla każdego z nich opisuję jego użycie w odniesieniu do gry 'Love Letter'. Ponieważ wszystkie algorytmy będą implementowane w aplikacji komputerowej, przedstawiłem je również w postaci pseudokodu.

3.1. Algorytm losowy

3.1.1. Opis

Jest to najprostszy algorytm podejmowania decyzji. Na wejściu otrzymuje listę dostępnych zagrań, z których w całkowicie losowy sposób wybiera jedno. Każde z dostępnych zagrań ma takie samo prawdopodobieństwo wyboru przez ten algorytm.

3.1.2. Sposób wykorzystania

Z uwagi na prostotę algorytmu jest on wykorzystany głównie do przetestowania poprawności działania implementacji gry. Zastosowałem w nim jedną drobną modyfikację: nigdy nie podejmie decyzji o zagraniu Księżniczki - oznacza to natychmiastową przegraną niezależnie od momentu gry, co wypaczałoby wyniki porównań algorytmów.

3.1.3. Zapis pseudokodem

function $d_{losowa}(I_n, X_n)$
    for all $z_i \in X_n$ **do** $\triangleright\ i = |X_n|$
        if $z_i ==$ Księżniczka **then**
            $X_n \leftarrow X_n - z_i$
        end if
    end for
    return wylosujJeden(X_n)
end function

3.2. Algorytm zachłanny

3.2.1. Opis

Algorytm zachłanny polega na wyborze najlepszego możliwego zagrania dostępnego w danej chwili, nie analizując jego konsekwencji w przyszłości. Pomimo, że takie podejmowanie decyzji jest krótkowzroczne, to jest też łatwe w implementacji i daje atrakcyjne wyniki w niektórych problemach, np. przy szukaniu minimalnego drzewa rozpinającego[10].

3.2.2. Sposób wykorzystania

W kontekście gry 'Love Letter', implementacja algorytmu zachłannego wymaga pewnego doprecyzowania. Najważniejszą częścią jest funkcja kryterialna oceniająca wartość zagrania, która w niektórych przypadkach musi być oparta o prawdopodobieństwo wystąpienia kart u przeciwnika. Prawdopodobieństwo to uzależnione jest od stanu początkowego danej tury.

Rozważmy przypadek, w którym algorytm musi podjąć decyzję o zagraniu karty Strażniczki, lub karty Barona. Jest to pierwszy ruch gracza, a w widocznych kartach odrzuconych na starcie są odrzucone karty Króla, Księcia i Pokojówki. Oznacza to, że w talii pozostało 9 kart, a jedna z odrzuconych jest niewidoczna, niemniej jednak ją też trzeba brać pod uwagę. Wyliczenie prawdopodobieństwa *P[]* jaką kartę ma przeciwnik jest tym momencie proste, jednak trzeba jeszcze wziąć pod uwagę drobny szczegół - czy do liczenia *P[]* wliczać karty posiadane w ręce. Z jednej strony wydaje się to nielogiczne i może prowadzić do wybierania nieoptymalnych decyzji (co przeczyłoby idei algorytmu zachłannego), z drugiej strony można to potraktować jako element blefu, który jest nieodłączną częścią każdej gry towarzyskiej. W swojej implementacji założyłem absolutną zachłanność algorytmu i karty posiadane na ręce są pomijane w obliczeniach. Wobec powyższego, prawdopodobieństwa wystąpienia kart u przeciwnika wynikające z ustalonego stanu początkowego są następujące:

Tablica 3.1. Przykład rozkładu prawdopodobieństwa

Karta	$P[]$
Strażniczka	30%
Kapłan	20%
Baron	10%
Pokojówka	10%
Książę	10%
Hrabina	10%
Księżniczka	10%

Zagranie karty Baron oznacza porównanie drugiej karty z kartą przeciwnika. Łatwo policzyć, że w 70% przypadków skończyłoby to się porażką, a w 30% nie było by żadnego efektu. Drugim dostępnym ruchem jest zagranie karty Strażniczki, a w jej przypadku najlepszym wyborem jest wytypowanie Kapłana, co daje 20% szans na zwycięstwo i 80% szans, że nie nastąpi żaden efekt. Zauważmy, że gdybyśmy wliczali posiadane karty do obliczenia prawdopodobieństwa, wystąpienie Barona i Kapłana byłoby tak samo możliwe. W takich przypadkach algorytm powinien zawsze celować w kartę z wyższym numerem. By formalnie stwierdzić, jaka decyzja z powinna zostać podjęta, musimy obliczyć funkcję kryterialną dla dostępnych zagrań i wybrać to zagranie, dla której funkcja przyjmuje wyższą wartość. Przyjmijmy, że funkcja kryterialna wygląda następująco:

$$F(z) = 1 + prawdopodobienstwo\_wygranej - prawdopodobienstwo\_przegranej$$

Po podstawieniu otrzymujemy:

$$F(Str\_Kap) = 1.2 \text{ i } F(Bar) = 0.3$$
$$F(Str\_Kap) > F(Bar) => z = Str\_Kap$$

Najlepszą decyzją w tym wypadku jest zagranie karty Strażniczki z wyborem karty Kapłana. Jak jednak na podstawie powyższego wzoru ocenić zagranie karty Kapłana, Pokojówki lub Króla? Każda z nich wymaga indywidualnej oceny. Mając na uwadze, że w kontekście strategii zachłannej decyzja zawsze powinna być optymalna w ujęciu chwili, ustaliłem następujące wskazówki którymi się kierowałem przy tworzeniu funkcji kryterialnej dla zagrań każdej z kart:

– Strażniczka - ocena zagrania wzrasta gdy pozwala wyeliminować przeciwnika.

– Kapłan - efekt karty jest neutralny i ocena jego zagrania będzie zawsze stała.

– Baron - ocena zagrania wzrasta gdy mamy drugą kartę silniejszą niż może mieć przeciwnik i maleje gdy jest odwrotnie.

– Pokojówka - podobnie jak w przypadku kapłana, ocena zagrania karty będzie stała.

– Książę - ocena zagrania na przeciwnika wzrasta z prawdopodobieństwem Księżniczki u przeciwnika. Ocena zagrania na siebie jest stała, lecz 0 gdy druga karta to Księżniczka.

– Król - jak w przypadku kapłana i pokojówki, ocena zagrania jest stała.

– Hrabina - ocena stała, jednak musimy ją zagrać gdy druga posiadana karta to Król lub Książę.

– Księżniczka - nie może być nigdy wyrzucona.

– Dodatkowo, jeśli oba zagrania mają taką samą wartość, powinna być podjęta decyzja o zagraniu karty o niższej wartości W, w związku z tym ocena zagrania spada wraz z wyższą wartością karty.

Opierając się na powyższych wytycznych, zapisałem algorytm w formie pseudokodu.

3.2.3. Zapis pseudokodem

Wykorzystane zmienne pomocnicze:

– $P[]$ - tablica prawdopodobieństwa wystąpienia karty u przeciwnika

– $decyzja$ - zagranie, które ma zostać zwrócone

```
function d_zachlanna(I_n, X_n)
    P[] ← obliczPrawdopodobienstwo()
    decyzja ← NULL                                ▷ Szukanie zagrania o najwyższej wycenie
    for all z_i ∈ X_n do                          ▷ i = |X_n|
        if F(decyzja, P[]) < F(z_i, P[] then
            decyzja ← z_i
        end if
    end for
    return decyzja
end function
```

Gdzie funkcja kryterialna wygląda następująco:

```
function F(z, P[])
    switch z do
        case Str_Kap                              ▷ Zagranie Strażniczki na dany typ karty
            return 1 + P[Kaplan] + 0.08
        case Str_Bar
            return 1 + P[Baron] + 0.08
        ...
        case Str_K-a
            return 1 + P[Ksiezniczka] + 0.08
        case Kap_Z
            return 1 + 0.07
        case Bar_Z                                ▷ Kryterium zależy od wartości W(DK)
            szansePrzegranej ← 0
            szanseWygranej ← 0
            for all typKarty do
                if W(DK) < W(typKarty) then
                    szansePrzegranej ← szansePrzegranej + P[typKarty]
                else if W(DK) > W(typKarty) then
                    szanseWygranej ← szanseWygranej + P[typKarty]
                end if
            end for
```

```
            return 1 − szansePrzegranej + szanseWygranej + 0.06
        case Pok_Z
            return 1 + 0.05
        case K-e_S
            return 1 + P[K-a] + 0.04
        case K-e_P
            if DK == K-a then
                return 0
            else
                return 1 + 0.04
            end if
        case Krl_Z
            return 1 + P[K-a] + P[Hra] + 0.03
        case Hra_K-e_Kr
            return 10 + 0.02
        case Hra_Z
            return 1 + 0.02
        case K-a_Z
            return 0
end function
```

3.3. Algorytm min-max

3.3.1. Opis

Algorytm minimaksowy polega na „minimalizowaniu maksymalnych możliwych strat” bądź alternatywnie na „maksymalizacji minimalnego zysku (wypłaty)”[11]. Zgodnie z [13] często stosuje się go do gier o następujących zasadach:

- występuje dwóch graczy
- ruchy wykonywane są naprzemiennie
- w każdym stanie istnieje skończona liczba decyzji do podjęcia
- stan i podjęta decyzja jednoznacznie wyznaczają stan następny
- każdy stan może zakwalifikować do jednej z następujących kategorii:
  - wygrana pierwszego gracza
  - wygrana drugiego gracza

– remis
– sytuacja nierozstrzygnięta

Najczęstsze implementacje polegają na przeszukiwaniu drzewa dalszych przebiegów gry począwszy od zadanego stanu początkowego, tak jak na przykład w warcabach[12]. Istnieją także implementacje opierające się na liczbowej ocenie ruchu[13] i na tej idei opieram swoją implementacje algorytmu minimaksowego.

3.3.2. Sposób wykorzystania

W podanym powyżej opisie, gra Love Letter niezgodna jest w punkcie czwartym. Jak ustaliliśmy wcześniej, gracz nie wie w jakim dokładnie stanie znajduje się gra, zna natomiast zbiór informacyjny I_n. Z tego względu, z perspektywy gracza P_1 podjęcie decyzji $z_i \in X_n$ nie określa jednoznacznie następnego węzła, lecz loterię. Mimo to, jesteśmy w stanie statystycznie ocenić możliwości przeciwnika, a tym samym zminimalizować naszą maksymalną stratę. Ponieważ zysk należy rozumieć jako zwiększenie szansy na wygraną (tak jak w strategii zachłannej), to strata oznacza zwiększenie szansy na przegraną.

Rozważmy scenariusz, w którym gracz pierwszy posiada kartę Księcia oraz Pokojówki. Na stosie pozostało 2 karty, 1 karta jest zakryta zgodnie z zasadami rozpoczęcia rundy i przeciwnik również posiada 1 kartę. Wśród tych 4 nieznanych graczowi kart są karty Strażniczki, Barona, Księcia oraz Księżniczki. Prawdopodobieństwo wystąpienia każdej z nich u przeciwnika wynosi 25% i razem stanowią one tablicę prawdopodobieństwa $P[]$. Przyjmijmy, że wykorzystywana w poprzedniej strategii funkcja kryterialna $F(z, P[])$ to maksymalizacja zysku i nazwijmy ją $F_{max}(z, P[])$ Zgodnie z jej definicją $F_{max}(K\text{-}e, P[]) = 1.29$ i $F_{max}(Pokojowka, P[]) = 1.05$, więc zagranie Księcia maksymalizuje szansę na wygraną. Jeśli jednak weźmiemy pod uwagę, że w 75% przypadków nie wygrywamy, wówczas musimy rozważyć odpowiedź przeciwnika. W tym wypadku jest 6 par kart które może posiadać przeciwnik, więc dla każdej z nich, zgodnie ze strategią zachłanną, należy obliczyć funkcję kryterialną oraz szansę na przegraną. Dodatkowo należy zauważyć, że z perspektywy przeciwnika w każdym przypadku gracz pierwszy ma inny rozkład prawdopodobieństwa wystąpienia karty. Pełne obliczenia znajdują się w tabeli poniżej.

Gracz pierwszy po zagraniu Księcia posiada tylko kartę Pokojówki. $F(z_1)$ oznacza ocenę zagrania pierwszej karty, $F(z_2)$ oznacza ocenę zagrania drugiej karty.

Tablica 3.2. Scenariusze reakcji przeciwnika na decyzję

Karty 1 i 2	$P[]$ pierwszego gracza	$F(z_1)$	$F(z_2)$	Szanse przegranej
Str ; Bar	$P[\mathit{Pok}] = P[\mathit{K\text{-}e}] = P[\mathit{K\text{-}a}] = 33.(3)\%$	1.41	0.6	33.(3)%
Str ; K-e	$P[\mathit{Pok}] = P[\mathit{Bar}] = P[\mathit{K\text{-}a}] = 33.(3)\%$	1.41	1.37	33.(3)%
Str ; K-a	$P[\mathit{Str}] = P[\mathit{Bar}] = P[\mathit{K\text{-}e}] = 33.(3)\%$	1.41	0	33.(3)%
Bar ; K-e	$P[\mathit{Str}] = P[\mathit{Pok}] = P[\mathit{K\text{-}a}] = 33.(3)\%$	1.39	1.37	100%
Bar ; K-a	$P[\mathit{Str}] = P[\mathit{Pok}] = P[\mathit{K\text{-}e}] = 33.(3)\%$	2.06	0	100%
K-e ; K-a	$P[\mathit{Str}] = P[\mathit{Pok}] = P[\mathit{Bar}] = 33.(3)\%$	1.04	0	0%

Dodatkowo, wprowadźmy funkcję $K(z) = F_{max} - F_{min}$, która wyceni dane zagranie.

Każdy rozważany scenariusz ma taką samą szansę wystąpienia, czyli $\frac{1}{6}$. Statystycznie szansa na przegraną wynosi więc:

$$\frac{1}{6} * (\frac{1}{3} + \frac{1}{3} + \frac{1}{3} + 1 + 1 + 0) = \frac{1}{6} * 3 = 0.50$$

Warunkiem wystąpienia możliwości przegrania, jest brak wygranej po zagraniu Księcia na przeciwnika, czyli ostatecznie szanse na przegraną wynoszą:

$$F_{min}(\mathit{K\text{-}e\_P}, P[]) = 0.75 * 0.50 = 0.375$$

Modyfikując o tę wartość ocenę zagrania Księcia otrzymujemy finalnie:

$$K(\mathit{K\text{-}e\_P}) = F_{max}(\mathit{K\text{-}e\_P}, P[]) - F_{min}(\mathit{K\text{-}e\_P}, P[]) = 1.29 - 0.375 = 0.915$$

W przypadku zagrania Pokojówki szanse przegrania wynoszą 0%, ponieważ zgodnie z zasadami gry jesteśmy odporni na działanie kart przeciwnika:

$$K(Pok\_Z) = F_{max}(Pok\_Z, P[]) - F_{min}(Pok\_Z, P[]) = 1.05 - 0 = 1.05$$

Wynika z tego, że decyzją która minimalizuje maksymalną stratę jest zagranie $z = Pok\_Z$.

Na podstawie powyższych rozważań zapisałem implementację algorytmu zachłannego w postaci pseudokodu.

3.3.3. Zapis pseudokodem

Wykorzystane zmienne pomocnicze:

– $K[z]$ - oznacza wspomnianą wyżej funkcję wyceny zagrania.

– Y_n - oznacza zbiór zagrań dostępnych dla przeciwnika.

– $L[Y_n]$ - lista możliwych zbiorów decyzji przeciwnika.

– Funkcja $R(L[Y_n])$ oznacza szansę porażki po reakcji przeciwnika.

– RK - zbiór zagrań będących reakcją przeciwnika.

– $P_{porazki}[]$ - tablica zawierająca prawdopodobieństwo porażki po danym zagraniu

function $D_{minimaksowa}(I_n, X_n)$
    $P[] \leftarrow$ obliczPrawdopodobienstwo()
    for all $z_i \in X_n$ **do** ▷ $i = |X_n|$
        $K[z_i] \leftarrow F_{max}(z_i, P[]) - F_{min}(z_i, P[])$
    end for
    $decyzja \leftarrow z_0$ ▷ Szukanie zagrania o najwyższej wycenie
    for all $z_i \in X_n$ **do**
        if $K[decyzja] < K[z_i]$ **then**
            $decyzja \leftarrow z_i$
        end if
    end for
    return $decyzja$
end function

Funkcja $F_{max}(z, P[])$ jest identyczna do $F(z, P[])$ występującej w strategii zachłannej, w związku z czym zapiszę wyłącznie $F_{min}(z, P[])$.

function $F_{min}(z, P[])$
    $L[Y_n] \leftarrow$ utwórz listę możliwych zbiorów decyzji u przeciwnika
    switch z **do**
        case Str_Kap ▷ Szanse, że Strażniczka **nie** przyniesie zwycięstwa
            return $(1 - P[Kap]) * R(L[Y_n])$
        case Str_Bar
            return $(1 - P[Bar]) * R(L[Y_n])$
        ...
        case Str_K-a
            return $(1 - P[\text{K-a}]) * R(L[Y_n])$
        case Kap_Z
            return $R(L[Y_n])$
        case Bar_Z ▷ Szanse, że porównanie zakończy się remisem
            $szanseRemisu \leftarrow P[DK]$
            return $szanseRemisu * R(L[Y_n])$
        case Pok_Z

return 0
case K-e_P
return $(1 - P[K\text{-}a]) * R(L[Y_n])$
case K-e_S
if $DK == K\text{-}a$ **then**
return 0
else
return $R(L[Y_n])$
end if
case Krl_Z
return $R(L[Y_n])$
case Hra_K-e_Krl
return $R(L[Y_n])$
case Hra_Z
return $R(L[Y_n])$
case K-a
return 0
end function

Funkcja $R(L[Y_n])$ oznacza szansę porażki po reakcji przeciwnika.

function $R(L[Y_n])$
for all $Y_n \in L[Y_n]$ **do**
$P[] \leftarrow$ obliczPrawdopodobienstwo()
$RK \leftarrow RK \cup D_{zachlanna}(Y_n, P[])$ ▷ Lista potencjalnych reakcji
end for
for all $z_i \in RK$ **do** ▷ Sprawdzenie szansy porażki, $i = |RK|$
switch z_i **do**
case Str_DK
$P_{porazki}[z_i] \leftarrow F_{max}(z_i) - 1.08$ ▷ Wynika z oceny zagrania Strazniczki przez algorytm zachłanny
case Bar
if $W(DK) < W(DK_{przeciwnika})$ **then**
$P_{porazki}[z_i] \leftarrow 1$
else
$P_{porazki}[z_i] \leftarrow 0$
end if
case K-e
if $DK == K\text{-}a$ **then**

$P_{porazki}[z_i] \leftarrow 1$
else
$P_{porazki}[z_i] \leftarrow 0$
end if
end for
return $\sum_{i=0}^{|RK|} \frac{1}{|RK|} * P_{porazki}[i]$
end function

3.4. Algorytm Monte Carlo Tree Search

3.4.1. Opis

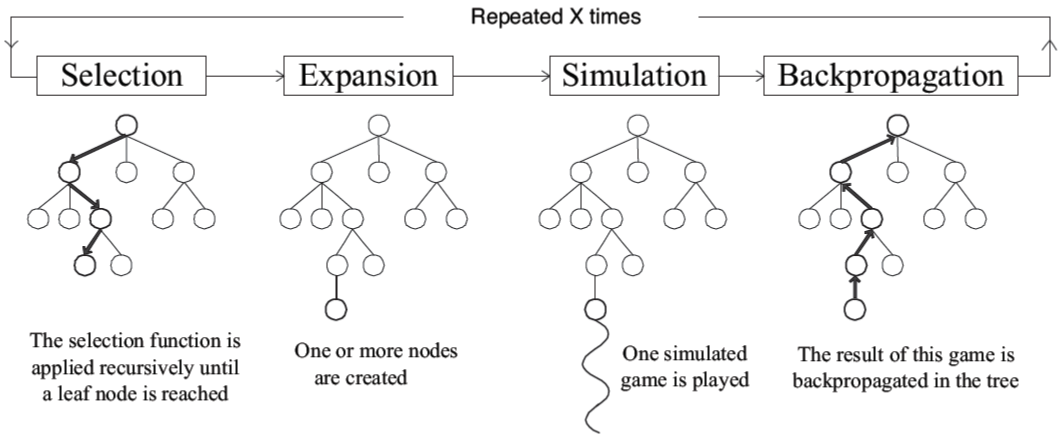
Drzewo Przeszukiwań Monte Carlo jest heurystyką podejmowania decyzji, często wykorzystywaną w grach typu Hex[14] czy Go[15]. W przeciwieństwie do deterministycznych algorytmów przeszukujących drzewo decyzyjne, MCTS buduje drzewo wariantów poprzez losowe próbkowanie najbardziej obiecujących ruchów. Dzięki temu jego dużą zaletą jest możliwość efektywnego wykorzystania w grach o wysokim rozgałęzieniu drzewa decyzyjnego[16]. Popularność wykorzystania tego algorytmu stale rośnie, czego przykładem jest zastosowanie w niedeterministycznej gry planszowej Osadnicy z Catanu[19], czy w strategii turowej Total War: Rome II[18]. Główne cechy algorytmu:

– przerywalność algorytmu w dowolnym momencie
– zależność wydajności od czasu - im dłużej działa algorytm, tym lepsze osiąga wyniki
– uniwersalność - algorytm do działania potrzebuje wyłącznie zbioru dostępnych ruchów oraz możliwości przeprowadzenia losowej rozgrywki od danego stanu
– niezależność od wiedzy eksperckiej występującej w grze

3.4.2. Sposób wykorzystania

Jak opisali autorzy artykułu *Monte-Carlo Tree Search: A New Framework for Game AI*[17], zasady działania algorytmu są następujące: otrzymując zbiór informacyjny I_n oraz zbiór dopuszczalnych ruchów X_n, tworzony jest korzeń T, oraz węzły s_0 .. s_i ($i = |X_n|$) odpowiadające stanom gry po wykonaniu dostępnych ruchów. Następnie przez zadany czas t powtarzane są następujące kroki:

1. **Selekcja** - zaczynając od korzenia T, wybieraj kolejne węzły balansując pomiędzy **eksploatacją** i **eksploracją**, aż dotrzesz do węzła-liścia L.
2. **Ekspansja** - jeśli wybór L nie kończy gry, utwórz węzły potomne i wśród nich wylosuj węzeł C
3. **Symulacja** - dla wybranego węzła przeprowadzać losową symulację, aż do osiągnięcia wyniku.



Rysunek 3.1. Schemat działania algorytmu MCTS[17]

4. **Propagacja wsteczna** - każdy węzeł od C do L jest aktualizowany o wartość wyniku.

Szczególnym elementem jest selekcja. Eksploatacja oznacza wybieranie ruchów o wysokiej częstości wygranych, a eksploracja to badanie ruchów o niskiej liczbie przeprowadzonych symulacji. Brak równowagi pomiędzy tymi dwoma strategiami próbkowania doprowadzi do zakłamań, na przykład poprzez pułapkę optimum lokalnego. W związku z tym w 2006 roku Kocsis i Szepervari opracowali wzór równoważący selekcję nazwany Upper Confidence Bound (górna granica ufności)[20]:

$$v_i + C * \sqrt{\frac{\ln N}{n_i}}$$

– v_i - estymowana wartość węzła

– C - empirycznie dobierany parametr eksploracji, zazwyczaj $\sqrt{2}$

– N - ilość odwiedzin węzła-rodzica

– n_i - ilość odwiedzin w danym węźle

Ważne jest, że każdy węzeł posiada informację o szacunkowej wartości opartej na wynikach symulacji, oraz o liczbie przeprowadzonych symulacji.

Ze względu na fakt, że algorytm do działania wymaga wyłącznie listy dostępnych ruchów oraz możliwości przeprowadzenia symulacji, jego implementacja do gry „Love Letter” nie wymaga specjalnych modyfikacji. Powyższa wiedza w pełni pozwala na wykorzystanie algorytmu w grze.

3.4.3. Zapis pseudokodem

Wykorzystane zmienne pomocnicze:

– $Nast(u, z)$ - funkcja wskazująca następnik węzła u po wyborze zagrania (łuku) z.

– X_L - zbiór ruchów dostępnych w węźle L.

– $wynik$ - wypłata gracza po symulacji.

```
function d_mcts(I_n, X_n)
    T ← utwórz korzeń T z obecnego zbioru informacyjnego I_n
    for all z_i ∈ X_n do                                    ▷ i = |X_n|
        T ← dodajDziecko(Nast(T, z_i))
    end for
    repeat
        L ← selekcja(T)
        if L nie jest końcem gry then
            for all z_i ∈ X_L do                            ▷ i = |X_L|,
                L ← dodajDziecko(Nast(L, z_i))
            end for
            C ← wylosujDziecko(L)
            wynik ← symuluj(C)                              ▷ Symulacja
            repeat                                          ▷ Propagacja wsteczna
                C ← aktualizuj(wynik)
                C ← rodzic(C)
            until C == T
        end if
    until koniec czasu
    decyzja ← najwięcejSymulacji(R)                         ▷ ruch z najbardziej obiecującego węzła
    return decyzja
end function
```

4. Implementacja

W tym rozdziale opisuję wymagania wobec projektu, który zawiera implementację gry „Love Letter” i opisanych algorytmów. Następnie przedstawiam koncepcję wykonania wraz z diagramami UML, a na końcu prezentuję system i opisuję problemy napotkane w trakcie realizacji.

4.1. Analiza wymagań

Ponieważ celem pracy jest porównanie działania algorytmów gry karcianej, pierwszym krokiem potrzebnym do jego realizacji jest zaimplementowanie samej gry. Jak zostało to napisane w [24], przed właściwą implementacją programu należy określić jego wymagania funkcjonalne i niefunkcjonalne. Większość wymagań jest już udokumentowana w postaci zasad gry opisanych w rozdziale 2, należy jednak wyszczególnić pewne dodatkowe wymogi:

– program musi udostępniać interfejs do którego można podłączyć różne algorytmy podejmujące decyzje,

– interfejs musi udostępniać aktualny stan gry oraz zbiór decyzji możliwych do podjęcia,

– interfejs musi zawierać funkcjonalność pozwalającą na utworzenie nowej instancji gry z zadanym stanem początkowym i podjętą decyzją. W nowej instancji wszystkie nieznane elementy gry są losowane na nowo (np. talia karta jest przetasowywana),

– program musi umożliwiać wybranie dwóch dowolnych algorytmów, które będą ze sobą grać, oraz ustawienie liczby gier które rozegrają,

– algorytm MCTS powinien mieć możliwość ustawienia dwóch parametrów: czasu wykonania, oraz algorytmu przeciwnika wykorzystywanego w symulacjach,

– po skończonej serii gier program powinien wyświetlać szczegółowe statystyki rozgrywek w formie wykresu i zapisywać dane do pliku.

Wymagania niefunkcjonalne:

– program powinien uruchamiać się bez względu na środowisko.

– program nie powinien wymagać od użytkownika znajomości szczegółów implementacji.

– program powinien udostępniać instrukcje pomocy.

Do zwracanych statystyk powinny należeć:

– procentowa liczba zwycięstw każdego algorytmu z wyszczególnieniem na tury w których została zakończona runda.

– rozkład procentowy zakończeń gry - ile rund skończyło się zagraniem karty Barona, Strażniczki, Księcia lub w ostatniej turze po wyczerpaniu talii.

– wyszczególnienie procentowej liczby zwycięstw w zależności od tego który algorytm wykonuje ruch jako pierwszy.

4.2. Koncepcja wykonania i wykorzystane technologie

By skupić się wyłączne na merytorycznym działaniu projektu, postanowiłem zaimplementować program w postaci aplikacji konsolowej, gdzie komunikacja odbywa się za pomocą wiersza poleceń[22]. Wprowadzenie warstwy graficznej jest zbędne wobec postawionych wymagań.

Pomimo prostoty i niezmienności wymagań funkcjonalnych, projekt zostanie wykonany zgodnie z metodyką zwinną (ang. agile methodology)[25], która wciąż zyskuje na popularności względem metodyk kaskadowych. Jej główną cechą jest porzucenie szczegółowego planowania projektu na rzecz obserwacji i reagowania. Całość oprogramowania jest wytwarzana w kolejnych iteracjach, w których implementowane są kolejne funkcjonalności, oraz testowane i poprawiane stare.

// Tu będzie obrazek ze schematem isteracji

Ze względu na osobiste umiejętności, do implementacji wybrałem język programowania Java SE w wersji 8[23]. Jest to język obiektowy, o ścisłym typowaniu, działający na maszynie wirtualnej Javy, co zapewnia kompatybilność ze wszystkimi systemami operacyjnymi. Całość projektu wykonałem w środowisku programistycznym IntelliJ IDEA w darmowej wersji Community[26].

4.3. Diagramy

Wrzucić use case'y, UML klas,

4.4. Prezentacja systemu

Parę screenów z wynikami.

4.5. Problemy napotkane w trakcie realizacji

Czas, czas, czas.

5. Rezultaty

5.1. Czas działania na 1000 partii

5.2. Statystyki zwycięstw

5.3. Zbieżność algorytmów podejmowania decyzji

5.4. Wnioski

6. Podsumowanie

Bibliografia

[1] Alderac Entertainment Group, *Love Letter*, 2012.

[2] Alderac Entertainment Group, *Love Letter*, http://www.alderac.com/tempest/love-letter, 2016-02-08.

[3] Alicja Cewe, Halina Nahorska, Irena Pancer, *Tablice matematyczne*, Wydawnictwo Podkowa, Gdańsk 2002, rozdział *Kombinatoryka*.

[4] *Wikipedia*, hasło *Problem optymalizacyjny*. https://pl.wikipedia.org/wiki/Problem_optymalizacyjny, 2016-02-21.

[5] *Wikipedia*, hasło *Strategia w teorii gier*. https://pl.wikipedia.org/wiki/Strategia_mieszana, 2016-04-10.

[6] Andrzej Z. Grzybowski *Matematyczne modele konfliktu. Wykłady z Teorii Gier i Decyzji*, Wydawnictwo Politechniki Częstochowskiej, Częstochowa 2012, rozdział *1.1 Klasyfikacja problemów decyzyjnych*.

[7] Andrzej Z. Grzybowski *Matematyczne modele konfliktu. Wykłady z Teorii Gier i Decyzji*, Wydawnictwo Politechniki Częstochowskiej, Częstochowa 2012, rozdział *2.1 Gry w postaci ekstensywnej*.

[8] *Wikipedia*, hasło *Drzewo (matematyka)*. https://pl.wikipedia.org/wiki/Drzewo_(matematyka), 2016-06-01.

[9] *Wikipedia*, hasło *Teoria decyzji*. https://pl.wikipedia.org/wiki/Teoria_decyzji, 2016-04-10.

[10] Sanjoy Dasgupta, Christos Papadimitriou, Umesh Vazirani, *Algorytmy*, Wydawnictwo Naukowe PWN, Warszawa 2012, rozdział *Algorytmy Zachłanne*, s. 133.

[11] *Wikipedia*, hasło *Algorytm min-max*. https://pl.wikipedia.org/wiki/Algorytm_min-max, 2016-04-24.

[12] *Funkcja oceniająca do algorytmu minimaksu w grze warcaby*, http://sequoia.ict.pwr.wroc.pl/witold/aiarr/2009_projekty/warcaby/, 2016-05-04.

[13] *Studia Informatyczne*, Sztuczna inteligencja/SI Moduł 8 - Gry dwuosobowe. http://wazniak.mimuw.edu.pl/index.php?title=Sztuczna_inteligencjaSI_Moduł_8_-_Gry_dwuosobowe, 2016-04-24.

[14] Broderick Arneson, Ryan Hayward, Philip Henderson, *MoHex Wins Hex Tournament*, ICGA Journal Vol. 32 No. 2 s. 114–116, Czerwiec 2009

[15] *Wikipedia*, hasło *Monte-Carlo Tree Search*. https://pl.wikipedia.org/wiki/Monte-Carlo_Tree_Search, 2016-05-04.

[16] *Introduction to Monte Carlo Tree Search*, https://jeffbradberry.com/posts/2015/09/intro-to-monte-carlo-tree-search/, 2016-05-04.

[17] Guillaume Chaslot, Sander Bakkes, Istvan Szita, Pieter Spronck, *Monte-Carlo Tree Search: A New Framework for Game AI*, Universiteit Maastricht / MICC. P.O. Box 616, NL-6200 MD Maastricht, The Netherlands

[18] *Monte-Carlo Tree Search in TOTAL WAR: ROME II's Campaign AI*, http://aigamedev.com/open/coverage/mcts-rome-ii/, 2016-05-04.

[19] Istvan Szita, Guillaume Chaslot, Pieter Spronck, *Monte-Carlo Tree Search in Settlers of Catan* Volume 6048 of the series Lecture Notes in Computer Science, s. 21-32, 2010.

[20] *Monte Carlo Tree Search*, sekcja *About*, http://www.cameronius.com/research/mcts/about/index.html, 2016-05-05.

[21] Krzysztof Sacha *Inżynieria oprogramowania* Wydawnictwo Naukowe PWN, Warszawa 2014, rozdział *Inżynieria wymagań*, s. 50.

[22] *Wikipedia*, hasło *Wiersz poleceń*. https://pl.wikipedia.org/wiki/Wiersz_poleceń, 2016-05-08.

[23] Oracle Corporation *Java* http://www.oracle.com/technetwork/java/javase/downloads/index.html, 2016-05-08.

[24] Krzysztof Sacha *Inżynieria oprogramowania* Wydawnictwo Naukowe PWN, Warszawa 2014, rozdział *Metodyka zwinna*, s. 334.

[25] *Wikipedia*, hasło *Model kaskadowy*. https://pl.wikipedia.org/wiki/Model_kaskadowy, 2016-05-08.

[26] JetBrains *IntelliJ IDEA* https://www.jetbrains.com/idea/, 2016-05-08.