Proceduralne generowanie zawartości dla klasy Simplified Boardgames

(Procedural content generation for Simplified Boardgames)

Łukasz Żarczyński

Praca magisterska

Promotor: dr Jakub Kowalski

Uniwersytet Wrocławski
Wydział Matematyki i Informatyki
Instytut Informatyki

18 czerwca 2018

Załącznik Nr 1
do Zasad

.....
(imiona i nazwisko)

....
(aktualny adres do korespondencji) (numer PESEL)

.....
(adres e-mail)

.....
(wydział)

.....
(kierunek studiów)

....
(poziom i forma studiów) (numer albumu)

OŚWIADCZENIE O PRAWACH AUTORSKICH I DANYCH OSOBOWYCH

Ja niżej podpisany/a ..... student/ka Wydziału ..... kierunek ..... oświadczam, że przedkładana praca dyplomowa na temat:.....
.....
.....

- jest mojego autorstwa i nie narusza autorskich praw w rozumieniu ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (tekst jednolity: Dz. U. z 2017 r. poz. 880, z późn. zm.) oraz dóbr osobistych chronionych prawem;
- nie zawiera danych i informacji uzyskanych w sposób niedozwolony;
- nie była wcześniej przedmiotem innej urzędowej procedury związanej z nadaniem dyplomu uczelni wyższej lub tytułu zawodowego;
- treść pracy dyplomowej załączona w wersji elektronicznej w APD, jest identyczna z jej wersją drukowaną.

Oświadczam, iż zostałem/am poinformowany/a o prawie dostępu do treści moich danych osobowych oraz ich poprawiania.

Wyrażam zgodę, na:

- udostępnienie mojej pracy dla celów naukowych i dydaktycznych;

☐ TAK ☐ NIE

- przetwarzanie moich danych osobowych w myśl ustawy z dnia 29 sierpnia 1997 r. o ochronie danych osobowych (tekst jednolity: Dz. U. z 2016 r., poz. 922.) w zakresie wynikającym z niniejszego oświadczenia i w celu jego realizacji.

Wrocław, ...

(rrrr – mm - dd) (czytelny podpis autora pracy)

Streszczenie

Praca ta jest kontynuacją badań nad grami szachopodobnymi należącymi do klasy Simplified Boardgames. Poprzednie prace w tej dziedzinie obejmują m.in. utworzenie zasad całej gry wykorzystując w tym celu metody ewolucyjne. W tej pracy poruszam problem miary złożoności danej gry dla człowieka. Przedstawiam metodę opisującą reguły dowolnej gry szachopodobnej w języku naturalnym, a dodatkowo wykorzystuję ją do zmierzenia tego, jak łatwo jest zrozumieć jej zasady. Eksperymenty uwzględniają popularne gry utworzone przez człowieka (m.in. szachy i Chess With Different Armies) oraz dwa zbiory utworzone proceduralnie.

Ponadto korzystam z właściwości wizualnych szachów jako inspiracji do wygenerowania nowych kształtów bierek dla innych gier szachopodobnych. Celem jest utworzenie kształtów spójnych dla danej gry, a jednocześnie pasujących do roli jaką dana bierka pełni w rozgrywce. Przedstawiona metoda wykorzystuje atrybuty szachów, znajdując podobieństwo ich reguł z nową grą i na tej podstawie odwzorowując cechy ich kształtów. Pomimo tego, że eksperymenty skupiają się na dwóch grach z klasy Simplified Boardgames, metoda ta może też zostać użyta w znacznie większej ilości gier, nawet tych pochodzących z innej domeny.

This thesis follows the study of chess-like games defined as a class of Simplified Boardgames. Previous work includes generating game rules using evolutionary approach. Here I present an algorithm generating natural language descriptions of piece movements that can be used as a tool not only for explaining them to the human player, but also for the task of procedural game generation. I test this method on some existing human-made games (including chess and Chess With Different Armies) and procedurally generated chess-like games.

Furthermore, I am using visual properties of chess pieces as an inspiration to generate new shapes for other chess-like games. The aim is to create shapes cohesive for a game, but also so that visual properties allude to the pieces' in-game function. The proposed method uses similarity measures in terms of pieces' strategic role and movement style in a game to identify the new pieces' closest representatives in chess. While experiments in this thesis focus on two chess-like games, the method can be used for broader generation of game visuals based on functional similarities of components to known, popular games.

Spis treści

Rozdział 1.

Wstęp

Dawniej uważano, że aby stworzyć system mający dorównać inteligencją człowiekowi, należy najpierw skupić się na implementacji umiejętności grania w zaawansowane gry strategiczne. Najpopularniejszymi przykładami takich gier są szachy, warcaby czy Go. Myślano, że system, który potrafi pokonać człowieka w tych grach, doprowadzi do utworzenia prawdziwej Sztucznej Inteligencji (SI). Jednak oczekiwaniom tym nie udało się sprostać, mimo tego, że potrafimy już stworzyć programy pokonujące w owe gry strategiczne nawet najlepszych graczy.

Wydawałoby się, że najłatwiejszym sposobem do stworzenia programu grającego w szachy będzie przekazanie mu całej wiedzy ekspertów. Metoda ta okazała się jednak nieefektywna i trudna w realizacji. Sukces osiągnął dopiero komputer IBM Deep-Blue, który w 1997 roku pokonał szachowego mistrza świata – Garry'ego Kasparova. Komputer ten posiadał wyspecjalizowane do gry w szachy procesory, ręcznie napisane przy pomocy ekspertów funkcje heurystyczne oraz ogromną bibliotekę otwarć oraz gier końcowych. Z jednej strony było to duże osiągnięcie, gdyż utworzony został zaawansowany komputer grający na mistrzowskim poziomie. Z drugiej jednak strony, na drodze do zbudowania systemu Sztucznej Inteligencji niewiele się zmieniło. W dzisiejszych czasach dysponujemy tak dużą mocą obliczeniową, że nawet mniej doświadczeni programiści i bez pomocy ekspertów potrafią napisać bardzo wysokiej jakości systemy do grania w szachy.

Argumentem do odejścia od pisania takich wyspecjalizowanych systemów jest to, że prawdziwie inteligentny program powinien sam zrozumieć zasady gry i potrafić w nią grać. Jest to zupełnie poza możliwościami komputerów takich jak IBM Deep-Blue, bo nie potrafią one wykonać nawet prostszych obliczeń matematycznych czy chociażby grać w warcaby. Powstała więc nowa dziedzina sztucznej inteligencji o nazwie General Game Playing, zajmująca się programami grającymi w gry bez wcześniejszej znajomości ich reguł.

Wzrost zainteresowania dziedziną General Game Playing, jak również pokrewną do niej General Video Game Playing, powoduje zwiększenie ilości badań w obszarze

proceduralnego generowania zawartości. Komputerowo tworzono reguły do gier takich jak system gier planszowych LUDI czy gry szachopodobne Pell'a. Inną klasą, do której powstał system generujący jej reguły, jest Simplified Boardgames. Powstało kilka typów generatorów, które wykorzystują metody ewolucyjne tworząc gry spełniające różne kryteria. Skupiają się one wyłącznie na właściwościach strategicznych tych gier. Jednak założenie, że wygenerowane gry będą wykorzystywane nie tylko przez konkursowe komputery, ale też przez zwykłych ludzi, wprowadza nowy wymiar ich oceny i daje nowe możliwości.

1.1. Cel i zawartość pracy

Celem pracy jest kontynuacja badań nad generowaniem zawartości dla gier należących do klasy Simplified Boardgames, dążąc do wykorzystania ich w środowisku, w którym głównymi graczami są ludzie.

Rozdział 2. stanowi wstęp do zagadnień proceduralnego generowania zawartości, problemu General Game Playing, klasy Simplified Boardgames oraz wprowadzenie wymaganej nomenklatury.

W rozdziale 3. przedstawiona jest metoda wyjaśniająca zasady poruszania się figur w grach szachopodobnych należących do klasy Simplified Boardgames. Zostaje ona wykorzystana do oceny danej gry, mierząc złożoność jej reguł czyli tego, czy gra jest zrozumiała i intuicyjna dla człowieka. Eksperymenty wykorzystują gry utworzone przez człowieka (takie jak szachy czy Chess With Different Armies) oraz dwa zbiory gier wygenerowanych proceduralnie w poprzednich pracach.

Rozdział 4. opisuje algorytm ewolucyjny tworzący nowe kształty bierek na podstawie ich wygenerowanych reguł poruszania się. Celem jest stworzenie elementów wizualnych gry szachopodobnej, tak, aby były one ze sobą spójne, a jednocześnie odzwierciedlały jej reguły. Bazą tej metody są właściwości wizualne tradycyjnych szachów. Eksperymenty uwzględniają utworzenie nowych kształtów dla proceduralnie wygenerowanych gier.

Rozdział 5. to podsumowanie całej pracy, w którym wyciągam wnioski oraz proponuję przyszłe działania.

Część tej pracy została już opublikowana w:

1. Kowalski, J., Żarczyński, Ł., and Kisielewicz, A. (2017). Evaluating Chess-like Games Using Generated Natural Language Descriptions. In *ACG 2017: Advances in Computer Games*, volume 10664 of *LNCS*, pages 127–139

2. Kowalski, J., Liapis, A., and Żarczyński, Ł. (2018). Mapping Chess Aesthetics onto Procedurally Generated Chess-Like Games. In *EvoApplications 2018: Applications of Evolutionary Computation*, volume 10784 of *LNCS*, pages 325–341

Rozdział 2.

Wprowadzenie

Rozdział ten ma na celu wprowadzenie wymaganych definicji i pojęć. Najpierw przedstawiony zostanie opis szachów, a na ich podstawie termin fairy chess razem z przykładami różnych wariantów i ich zastosowań. Następnie opisany zostanie problem gracza ogólnego i krótko przedstawiony stan badań w tej dziedzinie. W kolejnej części znajdą się informacje o klasie Simplified Boardgames wraz z sylwetką języka opisującego te gry. W ostatniej sekcji przedstawione zostaną aspekty proceduralnego generowania zawartości i jego wykorzystanie.

2.1. Szachy i fairy chess

2.1.1. Reguły szachów ortodoksyjnych

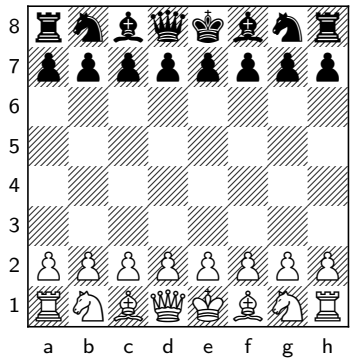
Szachy to bardzo popularna gra strategiczna rozgrywana pomiędzy dwoma graczami na szachownicy – kwadratowej planszy składającej się z 64 pól, której kolumny (pionowo) oznaczone są literami od *a* do *h*, a wiersze (poziomo) cyframi od 1 do 8. Każde pole na szachownicy jest więc oznaczone jedną literą oraz jedną cyfrą. Pole w lewym dolnym rogu nosi nazwę *a1*, a w prawym górnym *h8*.

Na szachownicy znajduje się 32 bierki – po 16 dla każdego gracza. Każda z nich różni się kształtem oraz sposobem poruszania się. Można wyróżnić 6 rodzajów bierek:

- ♔ Król (ang. King – K) – porusza się lub bije o jedno pole w każdą stronę,
- ♕ Hetman (ang. Queen – Q) – porusza się lub bije o dowolną liczbę wolnych pól w pionie, poziomie i po skosie,
- ♖ Wieża (ang. Rook – R) – porusza się i bije o dowolną liczbę wolnych pól w pionie i poziomie,
- ♗ Goniec (ang. Bishop – B) – porusza się i bije o dowolną liczbę wolnych pól po skosie,

♘ Skoczek (ang. Knight – N) – wykonuje skok najpierw o jedno pole w pionie lub poziomie, a następnie o jedno po skosie oddalając się od miejsca początkowego,

♙ Pion (ang. Pawn – P) – porusza się o jedno pole w przód (pierwszy ruch pionka można wykonać o dwa pola), zbija o jedno pole po skosie, ale tylko w przód.



Rysunek 2.1: Początkowy układ figur na szachownicy dla szachów.

Grę rozpoczyna gracz biały, a następnie on i przeciwnik wykonują ruchy na przemian. W jednym ruchu gracz musi poruszyć jedną ze swoich bierek zgodnie z jej regułami i zmienić jej pozycję. Jeżeli na nowej pozycji znajdowała się już bierka przeciwnika następuje tzw. *zbicie*, czyli usunięcie tej figury z gry.

Szach to groźba zbicia króla, przed którą przeciwny gracz musi uchronić się w kolejnym ruchu. Jeżeli nie jest w stanie tego zrobić następuje *mat*, a rozgrywka kończy się jego przegraną.

Roszada

Raz w trakcie trwania rozgrywki gracz może wykonać ruch zwany roszadą. Pozwala on na ruch królem oraz jedną z wież jednocześnie. Polega on na przemieszczeniu króla o 2 pola w stronę wieży oraz tejże wieży na polu za królem. Aby wykonanie roszady było możliwe muszą być spełnione następujące warunki:

- Ani król ani wieża nie wykonały ruchu od początku partii,
- Pomiędzy królem a wieżą nie ma innych bierek,
- Król nie jest szachowany,
- Król nie może przechodzić przez pole atakowane przez przeciwnika,
- Roszada nie może spowodować, że król będzie szachowany.

En passant

Możliwość bicia w przelocie następuje tylko wtedy, gdy pion gracza wykonał ruch o dwa pola, a pole, które minął, jest atakowane przez pion przeciwnika. Przeciwnik w kolejnym ruchu (tylko bezpośrednio po tym ruchu) może zbić tego pionka w przelocie, zajmując właśnie tę pominiętą pozycję.

Promocja

Promocja następuje wtedy, gdy pion gracza osiągnie ostatnią linię (czyli pierwszą linię przeciwnika). Wtedy w tym samym ruchu następuje zamiana tego pionka na dowolną inną figurę wybraną przez gracza o tym samym kolorze. Możliwym jest, aby poprzez promocje na planszy znajdowało się więcej figur danego typu niż w pozycji początkowej.

2.1.2. Fairy chess

Termin *fairy chess* został wprowadzony w 1914 roku przez Henryego Tate'a i do tej pory pozostał niezmieniony. Oznacza on pewien wariant szachów, w którym reguły gry lub poruszania się figur ulegają zmianie. Najczęstszym zjawiskiem jednak jest rozgrywka z całkiem nowymi bierkami. Przy użyciu takich figur powstało wiele zagadek, często publikowanych w czasopismach.

W książce [Dickins, 1971] Anthony Dickins przedstawia m.in. historię fairy chess. Opisuje on również wiele wariantów bierek, zmienionych rodzajów plansz oraz różnych warunków końcowych i problemów. Wiele figur jest opisanych również w [Wikipedia, 2017b] oraz w [Duniho, 2016].

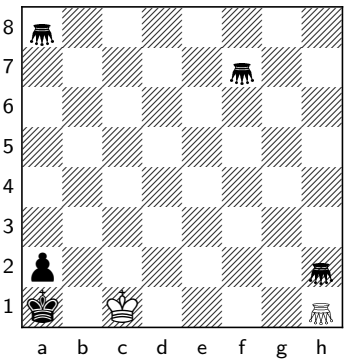
Przykłady figur

Jedną z najbardziej znanych figur fairy chess jest *grasshopper*, często oznaczana jako odwrócony hetman (⛛). Porusza się on podobnie do hetmana, z tą różnicą, że wymaga dokładnie jednej figury na swojej drodze, od której może się „odbić", a następnie zatrzymać na polu dokładnie za tą figurą – jeżeli stoi tam bierka przeciwnika, następuje zbicie. Grasshopper został wprowadzony przez T. R. Dawson'a pod koniec roku 1912. Pierwsza zagadka, w której wystąpił, została opublikowana w *Cheltenham Examiner* 3 lutego 1913 r.

Kolejną figurą jest *nightrider* (♘), który porusza się tak jak szachowy skoczek, jednak może on „skoczyć" w danym kierunku wiele razy (ogranicza go tylko rozmiar szachownicy).

Te dwie figury były jednymi z pierwszych powszechnie uznawanych przez problemistów. Ponadto stworzono tysiące innych nowych figur (m.in. *locust*, *lion*, *wazir*,

dabbaba, *alfil*, *camel*), jak również zagadek z ich wykorzystaniem.



Rysunek 2.2: Przykładowy problem z użyciem grasshopper'a. Biały gracz musi wykonać mat w 8 ruchach. [V. Onitiu, N. Petrović, T. R. Dawson, C. M. Fox, 1930]

Rodzaje plansz

Powstało wiele rodzajów plansz różniących się od klasycznej szachownicy 8×8. J. Boyer w swoich książkach rozważał m.in. plansze heksagonalne, okrągłe, w kształcie rombu i podłużne, jak również w nieregularnych kształtach. T. R. Dawson w [Dawson, 1938] używa takich rozmiarów dwuwymiarowych plansz jak 15×11, 9×15, 15×19, 15×13. Znajdują się tam również plansze wielowymiarowe.

Nieortodoksyjne warunki

Cel gry również może ulec zmianie i nie musi polegać tylko na zmatowaniu króla w n ruchach. Mogą to być np. *Selfmate* – gra białego gracza polega na doprowadzeniu do „zmatowania" siebie przez gracza czarnego w n ruchach, zaś czarnego na unikaniu tego; *Reflexmate* – gra obydwu graczy polega na doprowadzenia do „zmatowania" siebie oraz uniknięciu „zmatowania" przeciwnika przy dodatkowym warunku, że jeżeli istnieje ruch matujący to należy go wykonać; oraz wiele innych.

Znane są również gry z dodatkowymi warunkami, np. *Checkless chess* – żaden z graczy nie może „szachować", a jedynie wykonać od razu mat; *Must capture chess* – gracz musi zbić, jeżeli ma taką możliwość (tak jak w warcabach).

Szachy Los Alamos

Ważnym wariantem gier szachopodobnych są szachy Los Alamos, ponieważ są one pierwszym przykładem takiej gry, w którą potrafił grać program komputerowy. System został utworzony w 1956 r. w laboratorium Los Alamos. Z powodu ograniczonej mocy obliczeniowej komputerów w tym czasie rozmiar planszy został zmniejszony

do kwadratu o rozmiarze 6×6 oraz zrezygnowano z gońców. Reguły także zostały nieco uproszczone – brakowało np. pierwszego podwójnego ruchu pionka, ruchu en passant czy roszady.

Chess with different armies

Na uwagę z pewnością zasługuje też gra *Chess with different armies* [Wikipedia, 2017a] wprowadzona przez Ralph'a Betza. Jest to gra, w której dwójka graczy używa innych zbiorów figur fairy chess. Gracz przed rozpoczęciem (bądź też czasami w inny sposób) wybiera armię, którą będzie prowadził rozgrywkę. Ralph Betza opisał cztery armie, które przetestował i uznał, że są o podobnej sile. Są to: *Fabulous FIDE*, *Colorbound Clobberers*, *Nutty Knights* i *Remarkable Rookies*.

2.2. Problem General Game Playing

Gry strategiczne, takie jak szachy, wymagają kreatywności i umiejętności podejmowania decyzji. Element rozgrywki w takich grach pozwala w pewnym stopniu na porównanie swoich umiejętności z innymi. W podobny sposób można porównać dwa komputery grające ze sobą i dzięki temu ocenić ich umiejętności. Jest to motywacja do tworzenia środowisk testowych dla systemów Sztucznej Inteligencji (SI). Mogą to być np. konkursy, w których programy, które wygrywają więcej rozgrywek, będą uważane za „mądrzejsze".

Jednakże, na drodze do zbudowania systemu mającego na celu naśladowanie inteligencję człowieka, wyspecjalizowane systemy przynoszą niewielką wartość. Na przykład komputer IBM Deep Blue pokonał aktualnego wówczas mistrza świata w szachy, ale nie potrafi on grać w inne gry, takie jak np. warcaby. Problemem jest to, że takie programy wykonują tylko część wymaganej pracy. Wstępna analiza danego problemu jest zazwyczaj wykonywana wcześniej przez programistę, często przy pomocy ekspertów i zawodowych graczy, jak miało to miejsce w przypadku systemów szachowych IBM Deep Blue i Alpha Go.

Aby gry komputerowe były użyteczne w ocenie systemów SI, wymagane jest przeniesienie na nie większej odpowiedzialności i wymuszenie na nich wykonywania analizy problemów, którą inaczej wykonywałby programista. Obszar, który zajmuje się tym problemem to General Game Playing (GGP).

Celem GGP jest stworzenie agenta potrafiącego grać w różne gry bez ingerencji człowieka oraz wcześniejszej wiedzy o ich regułach. Prowadzi to do tworzenia uniwersalnych algorytmów rozwiązujących różne problemy w rozmaitych środowiskach, wykorzystując gry jako podłoże testowe.

Należy jednak unikać przekazywania takim programom ludzkiej wiedzy o problemie, aby nie mógł on bezmyślnie jej wykorzystywać. Gry potrafią różnić się prawie

w każdym aspekcie, więc nie ma takich informacji, które można przekazać agentowi. Mogą one być jednoosobowe, wieloosobowe, z przeciwnikiem bądź w trybie kooperacji, czasu rzeczywistego lub turowe itd. System taki powinien potrafić grać w gry prostsze, takie jak kółko i krzyżyk, jak również o znacznie wyższym stopniu skomplikowania, jak np. szachy.

Nauka na temat GGP to złożony problem, który wymaga innowacyjnych rozwiązań, przysparza wiele wyzwań oraz dotyka wielu działów SI. Domeny, które odgrywają ważną rolę w GGP, to m.in. reprezentacja wiedzy, przeszukiwanie drzewa gry, planowanie (modelowanie przeciwnika, heurystyka) oraz uczenie maszynowe.

Głównym aspektem GGP jest formalizm opisujący klasy gier. Język ten powinien być wystarczająco skomplikowany, by opisać szeroką gamę gier, łatwy do przetworzenia przez maszyny, a także czytelny dla człowieka, aby mógł on w prosty sposób tworzyć w nim gry.

Podczas gdy GGP cieszy się coraz większym zainteresowaniem, technologie leżące u jego podstaw mają dużo więcej zastosowań niż tylko gry. Przykładami mogą być zarządzanie procesami biznesowymi, handel elektroniczny czy inne konteksty badawcze.

Od pierwszego podejścia w [Pitrat, 1968] powstało wiele różnych standardów GGP. Każdy z nich posiadał inną semantykę, protokół komunikacji i opisywał inne gry.

2.2.1. Uogólnienie gier szachopodobnych

Początkowo GGP skupiało się na uogólnieniu gier szachopodobnych i do dzisiaj jest to najczęściej wykorzystywana przez nią dziedzina. Pierwsza praca w tym obszarze powstała w 1968 r. w [Pitrat, 1968]. Autor owej pracy zapoczątkował też ideę wprowadzenia języków opisujących dowolne reguły szachopodobnych gier planszowych w [Pitrat, 1971].

Kolejne systemy grające w wiele gier powstały we wczesnych latach 90. Był to m.in. system SAL (Search And Learning) przedstawiony w [Gherrity, 1993], który potrafił grać w gry dwuosobowe o sumie zerowej, deterministyczne i z pełną wiedzą. Podczas eksperymentów zaprezentowane zostały wyniki gier m.in. w kółko i krzyżyk, szachy i Connect Four.

Stworzony w 1992 r. przez Barney Pell'a system *Metagame* ([Pell, 1992a, Pell, 1992b, Pell, 1993]) jest uważany za poprzednika nowoczesnego GGP. Zawierał on m.in. język opisujący klasę gier, gracza ogólnego i generator gier. System obsługiwał wiele cech gier fairy chess, takich jak promocje, obowiązkowe zbijanie figur, zmiana właściciela bierek czy plansze cylindryczne.

2.2.2. Projekt Uniwersytetu Stanford

Chociaż termin General Game Playing odnosi się do każdej platformy pozwalającej grać w wiele gier, to jednak najczęściej kojarzony jest z projektem o tej samej nazwie grupy Stanford Logic Group Uniwersytetu Stanford w Kalifornii ([Genesereth et al., 2005]). Jest on na tyle popularny, że stał się standardem GGP w dziedzinie sztucznej inteligencji. Reguły gier w tym projekcie przedstawione są w języku Game Description Language (GDL) oraz w rozszerzeniu tego języka w 2010 o gry z niepełną wiedzą: Game Description Language for Incomplete Information games (GDL-II). Celem było tworzenie programów potrafiących bez ingerencji człowieka grać w gry opisane przez jeden z tych języków. Gracze komunikowali się z serwerem hostingowym, który monitorował rozgrywki i informował ich o zmianach stanów gry.

Od 2005 roku odbywa się również coroczny konkurs GGP o nazwie The International General Game Playing Competition (IGGPC), który początkowo związany był z konferencją AAAI. Tam porównywane są umiejętności systemów uczestników na szerokiej gamie gier komputerowych. Najpierw oceniana jest ich zdolność w wykonywaniu poprawnych ruchów, zdobywaniu przewagi i szybkości w kończeniu zadań. W kolejnych rundach programy zmagają się między sobą w bardziej wymagających grach.

Projekt ten jest idealnym środowiskiem do doskonalenia swoich umiejętności w dziedzinie SI, ponieważ wymaga on użycia wielu popularnych technik, takich jak reprezentacja wiedzy, przeszukiwanie, planowanie i uczenie się. Dodatkową pomocą są darmowe kursy organizowane przez Uniwersytet Stanford dostępne w internecie na platformie Coursera [Genesereth, 2014].

2.2.3. General Video Game Playing

Kolejną szybko rozwijającą się i zyskującą popularność dziedziną GGP są gry wideo, a w szczególności The General Video Game AI (GVGAI). Jest to konkurs, który zapewnia również platformę programistyczną, do której wszyscy uczestnicy muszą się dostosować. Ogranicza się on do gier podobnych do tych na platformę Atari (2-wymiarowych dla jednego gracza), a od niedawna również do gier dla dwóch graczy. Ponadto, na żadnym etapie nie są dostarczane zasady gry, a jedynym sposobem, w jaki można je poznać, jest dostęp do kilku funkcji platformy. Użytkownicy mogą dowiedzieć się m.in. jaki jest status gry (zwycięzca, zasoby) i gdzie znajdują się dane obiekty. Dzięki temu programy powinny potrafić grać w daną grę bez potrzeby odtwarzania jej silnika. Pozwala to wykorzystać te systemy w testowaniu nowych, proceduralnie wygenerowanych gier w poszukiwaniu błędów lub ocenianiu ich jakości. Dodatkowo, czas jednego ruchu jest ograniczony do 40 ms.

2.3. Simplified Boardgames

Klasa gier Simplified Boardgames została wprowadzona przez Yngviego Björnssona w pracy [Björnsson, 2012]. Jest ona pewnym podzbiorem gier planszowych, ograniczającym się do gier turowych o sumie zerowej dla dwóch graczy rozgrywanych na prostokątnej planszy. Definicja została następnie nieco rozszerzona w pracy [Kowalski and Kisielewicz, 2015], a w [Kowalski et al., 2016] został sformalizowany język opisujący te gry. Formalizm ten polegał m.in. na reprezentowaniu reguł ruchów figur jako wyrażeń regularnych.

Język pozwala na opisanie wielu figur różnych wariantów fairy chess w zwięzły sposób, a także umożliwia przedstawienie gier z asymetrią oraz ruchami zależnymi od pozycji (takimi jak pierwszy podwójny ruch pionka w szachach). Jednak wprowadza on również ograniczenia, nie pozwalając np. na roszadę, ruch en passant lub promocję.

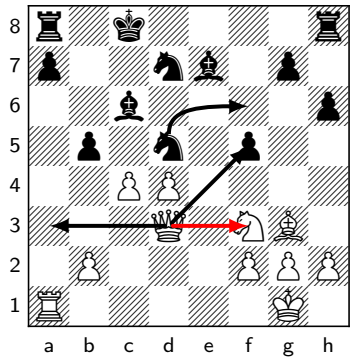
2.3.1. Składnia i semantyka

W tym rozdziale, bazując na formalizmach znajdujących się w [Kowalski et al., 2016], przedstawię skrócony opis składni oraz semantyki tej klasy.

Gra jest rozgrywana pomiędzy dwoma graczami, czarnym i białym, na prostokątnej planszy. Biały gracz rozpoczyna rozgrywkę. Rozmiar planszy jest podany jako dwie liczby w sekcji `<BOARD>`, które reprezentują jej wysokość i szerokość. Następnie podana jest początkowa pozycja figur: puste pola reprezentowane są przez kropki, białe figury jako wielkie litery, a czarne jako małe litery. Rozkład figur na planszy może być asymetryczny, a początkowa pozycja jest ukazana z perspektywy białego gracza, czyli ruch w przód oznacza „w górę” dla białego gracza, oraz „w dół” dla czarnego.

Podczas tury gracz musi wykonać ruch jedną ze swoich bierek. Może tego dokonać poprzez wybranie figury oraz zmianę jej pozycji zgodnie z jej regułami poruszania się. W danym czasie co najwyżej jedna bierka może stać na danym polu, więc zakończenie ruchu na kwadracie okupowanym przez inną figurę (niezależnie od właściciela) kończy się jej usunięciem (zbiciem). W sytuacji, gdy punkt startowy jest również punktem końcowym, stan planszy pozostaje niezmieniony. Nie jest możliwe dodawanie figur. Po wykonaniu ruchu gracz oddaje kontrolę przeciwnikowi. Reguły ruchów bierek są opisane w sekcji `<PIECES>`. Każda z nich może mieć co najwyżej jedną regułę składającą się z litery oznaczającej nazwę bierki oraz wyrażenia regularnego. Dozwolony jest brak reguł dla danej bierki, lecz wtedy pozostaje ona nieruchoma przez całą rozgrywkę. Dla danej bierki zbiór możliwych ruchów jest zdefiniowany jako zbiór słów opisanych przez wyrażenie regularne nad alfabetem Σ zawierającym krotki $(\Delta x, \Delta y, on)$, gdzie Δx oraz Δy oznaczają relatywną odległość wszerz/wzdłuż?, a $on \in \{e, p, w\}$ oznaczają zawartość pola docelowego: e oznacza

pole puste, p pole zajęte przez przeciwnika, a w pole zajęte przez własną figurę. Dodatnia wartość Δy oznacza ruch w przód dla danego gracza.



Rysunek 2.3: Przykład szachów. Dwa możliwe ruchy dla hetmana na $d3$ są przedstawione na rysunku. Zbicie figury na $f5$ jest kodowane jako $(1,1,e)(1,1,p)$, a ruch na $a3$ jako $(-1,0,e)(-1,0,e)(-1,0,e)$. Ruch na $f3$ jest nielegalny, gdyż w zbiorze ruchów dla tej figury nie ma takiego, który kończy się na własnej bierce. Ruch $d5-f6$ skoczka jest bezpośrednim skokiem i jest kodowany jako $(2,1,e)$.

Rozważmy figurę F oraz słowo $w \in \Sigma^*$ będące opisem reguł poruszania się tej figury zapisanym jako suma wszystkich możliwych ruchów $w = m_i + m_2 + \ldots + m_k$. Niech $a = m_j$ będzie dowolnym ruchem j figury F. Wtedy $a = a_1 a_2 \ldots a_k$, gdzie każde $a_i = (\Delta x_i, \Delta y_i, on_i)$. Załóżmy, że F znajduje się na polu $\langle x, y \rangle$. Wówczas a opisuje ruch, który jest legalny wtedy i tylko wtedy, gdy dla każdego l takiego, że $1 \leq l \leq k$, warunek on_l jest spełniony na polu $\langle x + \sum_{j=1}^{l} \Delta x_j, y + \sum_{j=1}^{l} \Delta y_j \rangle$. Ruch opisany przez a zmienia pozycję figury F z $\langle x, y \rangle$ na $\langle x + \sum_{i=1}^{k} \Delta x_i, y + \sum_{i=1}^{k} \Delta y_i \rangle$.

Gra może zakończyć się remisem, gdy zostanie osiągnięty limit ruchów. Gracz może wygrać poprzez ustawienie pewnej figury na polu sprecyzowanym w sekcji `<GOALS>` (wygranie pozycyjne) lub poprzez przechwycenie pewnej liczby figur przeciwnika sprecyzowanych w tej samej sekcji (wygranie poprzez przechwycenie), bądź też poprzez doprowadzenia przeciwnika do momentu, w którym nie ma on żadnych legalnych zamiast ruchów. Warunki zwycięstwa mogą być asymetryczne.

2.4. Proceduralne generowanie zawartości

Głównym zadaniem proceduralnego generowania zawartości (ang. Procedural Content Generation – PCG) jest stworzenie cyfrowej zawartości używając metod algorytmicznych. Z punktu widzenia GGP na szczególną uwagę zasługuje generowanie reguł gry. Dla danego języka opisującego pewne gry chcielibyśmy stworzyć reguły, które nie tylko będą poprawne, ale również posiadające odpowiednie właściwości zachęcające ludzkiego gracza.

PCG może być wykorzystane prawie na każdym etapie tworzenia gry, od tekstur, różnych poziomów, poprzez muzykę oraz przeciwników. Główną przyczyną automatyzacji generowania tych elementów jest ich koszt. Projektanci i artyści są zbyt powolni i kosztowni. Ponadto wiele zadań podczas tworzenia gry nie wymaga w pełni ich kreatywności. Nowoczesne gry wideo rozgrywają się zazwyczaj w wielkich lokalizacjach, wręcz niemożliwych do stworzenia całkowicie ręcznie. Ważnym jest, aby uwaga gracza skupiona była na głównym i starannie przygotowanym wątku, a otoczenie grało poboczną rolę mającą na celu wypełnienie świata gry i sprawienie, żeby nie wydawał się on pusty i bez życia. Dlatego też częstym zabiegiem jest mieszanie elementów generowanych automatycznie z tymi stworzonymi ręcznie przez twórców gry. Narzędzia PCG nie muszą być używane do tworzenia końcowej wersji gry, lecz mogą być jedynie pomocą, np. przy tworzeniu wstępnej wersji, która jest następnie modyfikowana według potrzeb. Pozwala to na tworzenie gier lepiej dostosowanych do gracza oraz zwiększenie przyjemności z samej gry. W przypadku tworzenia map, takie techniki pozwalają też na lepsze dostosowywanie gry już w trakcie rozgrywki, np. poprzez odpowiednie dopasowanie ilości nadchodzących przeciwników. Innym aspektem jest przedłużenie rozgrywki. Dobra mapa lub misja potrafi zainteresować gracza nawet po zakończeniu głównego wątku, w dalszym ciągu dostarczając nowych wyzwań.

PCG jest także szeroko używany w grach niezależnych (ang. independent – indie), gdy programiści nie mają czasu i środków na to, by tworzyć każdy element ręcznie. Jedną z takich gier jest Minecraft, który używa PCG do generowania całego świata. Warto też wspomnieć o wykorzystaniu PCG w komercyjnych grach „AAA". Przykładem takiej gry jest Diablo, która wykorzystuje te techniki nie tylko do tworzenia nowych poziomów, ale również ilości, typu, a czasem nawet nazw potworów i przedmiotów. To sprawiło, że gra odniosła duży sukces w trybie wieloosobowym, w którym grupa graczy mogła grać na danym poziomie wiele razy, za każdym razem w innych warunkach.

2.5. Generatory gier

W pracach [Kowalski and Szykuła, 2015, Kowalski and Szykuła, 2016] został poruszony problem generowania całych gier z klasy Simplified Boardgames. W każdej z nich została użyta inna metoda tworzenia gier, a ich wyniki zostały wykorzystane w dalszych częściach mojej pracy.

2.5.1. Generator oparty na symulacjach

Generator z pracy [Kowalski and Szykuła, 2015] wykorzystuje adaptacyjny algorytm ewolucyjny, w którym funkcja fitness opiera się na symulacjach rozgrywek.

Struktura chromosomu składa się tam z trzech elementów: planszy, zasad ruchu figur oraz warunków końcowych. Plansza to kwadratowa tablica, zawierająca informacje o wszystkich polach i znajdujących się na nich bierkach. Figury reprezentowane są jako mapa, której kluczem jest ich symbol, a wartością – reguły poruszania się. Warunki końcowe są przechowywane jako lista krotek 3-elementowych, zawierających typ figury, jej właściciela oraz typ warunku końcowego (zbicie lub ruch).

Do wykreowania populacji początkowej generowane są najpierw figury, tworząc zbiory ruchów, z których następnie losowane będą reguły pasujące do danej klasy figury. Plansza wypełniana jest bierkami również poprzez użycie funkcji losowej, z odpowiednio dobranym prawdopodobieństwem balansującym klasy figur. Warunki końcowe, podobnie jak poprzednie elementy gry, są również tworzone losowo.

Funkcja fitnessu ocenia daną grę symulując rozgrywkę. Pomimo tego, że duża ilość remisów jest dozwolona, to jednak w grach strategicznych lepszy gracz powinien wygrywać więcej gier. Dlatego też użyte są dwa rodzaje symulacji: pierwszy wykorzystuje dwóch agentów min-max (a konkretnie agentów RBgPlayer z [Kowalski and Kisielewicz, 2015]), drugi używa jednego agenta losowego i drugiego min-max. Ewolucja przebiega standardowo, dodatkowo dostosowując swoje parametry.

Lepsze wyniki uzyskane zostały dla plansz mniejszych rozmiarów oraz dla liczniejszych populacji użytych w algorytmie. Najlepsza – dla plansz o rozmiarze 6×6 oraz populacji 50 osobników – posiadała wartość fitness na poziomie 0.98. Podczas symulacji agent min-max wygrał wszystkie rozgrywki z agentem losowym, a w przypadku dwóch agentów min-max każdy z nich wygrał ok. 20% z nich, co pozwala stwierdzić, że gra jest odpowiednio zrównoważona. Posiada również ciekawą asymetrię i zawiera 6 typów figur – po 2 dla każdej z klas. Nie posiada ona warunków zwycięstwa, co jest też popularne w grach wygenerowanych tą metodą.

2.5.2. Generator RAPP

RAPP to skrót od Relative Algorithm Performance Profiles. Reprezentuje on ideę używania agentów nie do porównywania ich umiejętności, ale do oceniania gier. Pozwala to skupić się na zachowaniu gry, a nie na jej budowie. Dzięki temu podejściu RAPP stało się popularne w dziedzinie GGP, ponieważ języki opisujące te gry są często bardzo skomplikowane.

W pracy [Nielsen et al., 2015a] zastosowana została ta idea w dziedzinie GVG-AI (z wykorzystaniem języka VGDL - The Video Game Definition Language). Autorzy pokazują tam, że różnice pomiędzy wynikami algorytmów zastosowanych do gier stworzonych przez człowieka są znacznie większe, niż gdy zastosuje się je do gier wygenerowanych przez komputer. Pokazuje to, że przy generowaniu gier należy dążyć do zwiększenia tych różnic.

Generator gier klasy Simplified Boardgames z pracy [Kowalski and Szykuła,

2016] również wykorzystuję to podejście. Użyta tam metoda testuje algorytmy na zbiorze znanych gier stworzonych przez człowieka. Są to m.in. *Gardner*, *Action Man's Chess*, *Half Chess*, *Los Alamos Chess*, *Shatranj* oraz tradycyjne szachy. Jako zbiór agentów użyty został algorytm min-max w 16 wariantach, różniących się funkcją heurystyczną, czyli w efekcie strategią gry.

Zbiór wszystkich rozważanych tam gier został ograniczony do sześciu (w tym tradycyjne szachy). Na ich podstawie można sprawdzić, jak te gry zachowują się przy wykorzystaniu różnych algorytmów. Niestety, liczba szesnastu wariantów agentów prowadzi do zbyt dużej złożoności obliczeniowej, dlatego wyodrębniono tylko kilka z nich jako najlepszych ich przedstawicieli. W efekcie zostały wybrane dwa zbiory algorytmów – jeden 3-, a drugi 4-elementowy. Zbiory te zostały następnie wykorzystane do utworzenia funkcji oceny. Została ona wykorzystana w algorytmie ewolucyjnym do generowania gier. Populacja początkowa w tym algorytmie jest tworzona losowo. Następują mutacja i krzyżowanie, które operują na zbiorze bierek na planszy oraz ich regułach poruszania się.

Eksperymenty uwzględniały wygenerowanie 200 gier oraz ocenę ich za pomocą zbiorów 3 lub 4 algorytmów. Wyniki pokazały, że użycie większych zbiorów algorytmów prowadzi do zwiększenia jakości gier. Zauważono również, że wysoko oceniane gry mają ograniczoną użyteczność pionków lub podobnych do nich rolą figur (np. nie mogą ruszać się wprzód, a jedynie na boki). Wydaje się, że ruchy pionków są tak ważne i tak bardzo wpływają na rozgrywkę, że lepiej jest ograniczyć ich rolę. Oprócz tego, w wygenerowanych grach często reguły są atrakcyjne pod względem strategicznym, jednak zbyt skomplikowane, aby mógł w nie grać człowiek. Odnosi się to do problemu różnicy między grami dobrymi do testowania systemów SI a grami dobrymi dla ludzi.

Rozdział 3.

Ocena opisu figur w języku naturalnym

Algorytm tworzenia zawartości do gry wymaga ograniczenia zbioru gier do pewnej dobrze zdefiniowanej dziedziny. Naturalnym staje się więc zastosowanie PCG w obszarze GGP, ponieważ posiada on bardzo ściśle określoną domenę.

Początkiem wykorzystania proceduralnego generowania zawartości w problemie GGP jest system *Metagame* opisany przez Barney'a Pell'a w [Pell, 1992a], w którym opisał on tzw. Symmetric Chess-like Games. Ocena jakości wygenerowanych gier była wtedy w całości w rękach człowieka.

Jeden z najbardziej obiecujących przykładów PCG to system *Ludi* Cameron'a Browne'a zaprezentowany w [Browne and Maire, 2010]. Wykorzystał on programowanie genetyczne, łącząc je z funkcją ewaluacji opartej na symulacjach. Bazą systemu były miary estetyczne, które zostały użyte do wygenerowania kombinatorycznych gier planszowych. Wynikiem tego były pierwsze stworzone w pełni przez komputer gry, które zostały wydane później w formie papierowej.

Ocena jakości gry to bardzo trudne zadanie z wielu powodów. Po pierwsze, nawet człowiek może mieć problemy, by określić, czy gra jest „dobra" i jest to jedynie jego subiektywna ocena. Gry można oceniać przecież pod względem różnych kryteriów. Po drugie, możliwych kombinacji zasad gry jest bardzo dużo, a czasami najmniejsza ich zmiana potrafi znacząco wpłynąć na cały przebieg rozgrywki. Dlatego też łatwo jest pominąć atrakcyjne gry wśród zbioru wszystkich rozwiązań.

Ponieważ ciężko jest zmierzyć wprost jakość danej gry, wykorzystuje się do tego jej właściwości strategiczne. Dobra gra powinna być sprawiedliwa i wystarczająco skomplikowana, aby stworzyć pewne wyzwanie dla jej graczy. Jeżeli brać pod uwagę gry dla systemu SI, np. w zawodach GGP, to te wymagania zazwyczaj wystarczają. Jednakże, jeżeli weźmiemy pod uwagę to, że gra jest tworzona również dla człowieka, wtedy należy uwzględnić również złożoność jej zasad. Jeszcze trudniejszym zadaniem

jest sprawienie, aby zasady gry, pomimo swego poziomu skomplikowania, były dla człowieka intuicyjne i łatwo przyswajalne. W tym rozdziale poruszam problem miary złożoności opisu gry szacho-podobnej dla gracza ludzkiego.

Opisy niestandardowych figur podobnych do szachów bazują zazwyczaj na analogiach do znanych ruchów [Wikipedia, 2017b]. Użyję podobnego podejścia, najpierw rozłożę ruchy danej figury na części bazując na klasyfikacji ruchów szachowych [Dickins, 1971], a następnie stworzę opis figury w języku naturalnym i zmierzę jego złożoność. Opis figur będzie generowany w języku angielskim.

Metody w tym rozdziale zostały wykorzystane do opisu i oceny figur fairy chess przy użyciu wyrażeń regularnych w formacie Simplified Boardgames. Poza tym mogą zostać one uogólnione do znacznie większej klasy gier, posiadających cechy innych wariantów szachów.

Algorytm został ponadto wykorzystany do oceny całych gier – zarówno tych stworzonych przez człowieka, jak i tych wygenerowanych proceduralnie w pracach [Kowalski and Szykuła, 2015, Kowalski and Szykuła, 2016]. Dodatkowo użyto go do wyznaczenia tych wygenerowanych gier, których reguły są łatwe do przyswojenia, a przy tym rozgrywka wydaje się być ciekawa.

3.1. Teoria ruchów fairy chess

Termin fairy chess służy do opisania nieortodoksyjnych wariantów szachów popularnych w problemach szachowych lub uogólnienia gier szacho-podobnych. T. R. Dawson's w *Theory of Movements* stworzył podstawę do opisu typów ruchów figur w fairy chess. Wyodrębnił on trzy rodzaje: *leap* (skok), *ride* (przejazd) oraz *hop* (przeskok) [Dickins, 1971].

Skok przenosi dana figurę bezpośrednio na inne pole (zgodnie z wektorem ruchu), nie biorąc pod uwagę innych figur znajdujących się pomiędzy. Dla przykładu zasady ruchów skoczka szachowego mogą być opisane jako $\langle 1, 2\rangle$-leap, $\langle 2, 1\rangle$-leap, $\langle -1, 2\rangle$-leap itd. Mówimy, że figura to $\langle x, y\rangle$-leaper, jeżeli skacze we wszystkie strony po wektorach $\langle \pm x, \pm y\rangle$ oraz $\langle \pm y, \pm x\rangle$. Wobec tego szachowy skoczek to $\langle 2, 1\rangle$-leaper, a król to $\langle 1, 0\rangle$ i $\langle 1, 1\rangle$-leaper. Oprócz tego, skok na pole zajęte przez przeciwnika oznacza zbicie figury.

Następnie, przejazd oznacza ruch o dowolną ilość pól we wskazanym kierunku, pod warunkiem, że nie ma po drodze innych figur. Ruch może zakończyć się na pustym polu bądź na takim, na którym znajduje się przeciwnik – następuje wtedy zbicie. Kierunek ruchu opisywany jest przez wektor $\langle x, y\rangle$ i podobnie jak w przypadku leaper'a oznacza on poruszaniesię po wektorach zarówno $\langle \pm x, \pm y\rangle$ jak i $\langle \pm y, \pm x\rangle$. Bierkę można wtedy nazwać $\langle x, y\rangle$-rider. Przykładem takiej figury jest wieża, którą można opisać jako $\langle 1, 0\rangle$-rider lub goniec jako $\langle 1, 1\rangle$-rider.

Figura wykonująca przeskok opisywana jest jako *hopper* i charakteryzuje się tym, że potrafi przeskoczyć przez inną figurę. Zazwyczaj oznacza to skok dokładnie raz przez bierkę dowolnego koloru. Jest wiele podtypów hopper'ów, więc jest też wiele ich formalnych definicji. W szachach taka figura nie występuje, za to w chińskich szachach jest to *cannon* (armata), która porusza się jak wieża, jednak aby zbić inną figurę, potrzebuje dokładnie jednej bierki dowolnego koloru na swojej drodze.

3.2. Funkcja ewaluacji

Dysponując grą opisaną pod względem jej strategicznych właściwości, chcielibyśmy ocenić jak trudno jest nauczyć się jej reguł, czyli tego jak dana gra działa. Ponieważ w klasie Simplified Boardgames gry mają wystarczająco jasno opisane warunki końcowe, skupię się na oszacowaniu złożoności ruchów figur. Skorzystam z teorii ruchów używanych w fairy chess, aby stworzyć opis reguł bierek w języku naturalnym, nawiązując przy tym do dobrze znanych figur oraz innych, łatwych do wytłumaczenia zasad. Poza użyciem opisu tych zasad jako miary ich złożoności, mogą one zostać wykorzystane przy tworzeniu instrukcji do gry.

3.2.1. Generowanie opisów

Dla zadanej figury F algorytm służy do opisania jej reguł, czyli słowa w_F będącego wyrażeniem regularnym opisanym w rozdziale 2.3.1. Przedstawia on je jako złożenie kilku prostszych typów ruchów. Najpierw wyrażenie regularne opisujące reguły figury jest rozwijane na sumę pojedynczych ruchów.

$$w_F = \ m_1 + \ m_2 + \ ... + \ m_n \quad m_i \in \Sigma^*, \quad n \ = 0, 1, 2, ... \tag{3.1}$$

W przypadku gdy $n = \ 0$ figura nie posiada reguł, więc jest nieruchoma.

Następnie algorytm wykorzystuje zachłanne podejście, by opisać całą figurę używając najbardziej obiecujące częściowe opisy.

Bazą algorytmu są klasy oraz operatory. Można wyróżnić dwie klasy: *leaper* oraz *rider* (ten późniejszy zawiera w sobie klasę *hopper*). Każda klasa jest parametryzowana wektorem $\langle x, y \rangle$. Na przykład klasa *rider* w połączeniu z wektorem $\langle 1, 1 \rangle$ reprezentuje gońca szachowego. Mówimy wtedy, że ta figura to $\langle 1, 1 \rangle$-rider. Operatory są używane do ograniczania zbioru ruchów. Jeden opis składa się zatem z klasy oraz zbioru operatorów (być może pustego). Na przykład goniec może zostać ograniczony w ten sposób tylko do ruchów w przód poprzez użycie operatora $forwards$.

Wstępna faza procedury dzieli dane reguły ruchów na części spójne, czyli takie, które zawierają ten sam wektor przesunięcia – np. $(0, 1, e)^*(0, 1, p)(0, 2, e)$ składa się

Algorithm 1 Generowanie opisu ruchu m figury F

```
1: function findDescription(m : Σ*, w_F : [Σ*])
2:     moves ← Ruchy z w_F z tą samą ilością części spójnych co m
3:     movesByPrefix ← {(ϵ, moves)}        ▷ Mapa, której wartościami jest zbiór ruchów, a kluczami ich wspólny prefiks
4:     return findDescription(m, movesByPrefix, 1)
5: end function
6:
7: function findDescription(m : Σ*, movesByPrefix : Σ* → [Σ*], i : ℕ)
8:     for operators in SortedOperatorSets do
9:         for class in PieceClasses do
10:            (moves, description) ← getMoves(class, operators)
11:            H ← {}                                   ▷ Tworzy pustą mapę
12:            for (prefix, M) in movesByPrefix do
13:                if moves ⊆ getParts(M, i) then
14:                    G ←Ruchy z M których i-ta część spójna jest w moves
15:                    H.updateWith(G)
16:                else                                 ▷ Zły wybór operators i class
17:                    break and continue in line 8
18:                end if
19:            end for
20:            if m nie znajduje się w żadnym zbiorze będącym wartością H then
21:                continue ▷ Opis nie pasuje do aktualnie przetwarzanego ruchu m
22:            end if
23:            if i < numberOfParts(m) then
24:                r ← findDescription(m, H, i+1) ▷ Rekurencyjnie dla kolejnych części spójnych
25:                if r is Fail then continue
26:                else return description "and then" r        ▷ Łączenie opisów kolejnych części spójnych
27:                end if
28:            end if
29:            return description
30:        end for
31:    end for
32:    return Fail
33: end function
34:
35: function getParts(M : [Σ*], i : ℕ )
36:    return Rzut elementów z M na i-te części spójne
37: end function
38:
39: function numberOfParts(m : Σ*)
40:    return Liczba części spójnych ruchu m
41: end function
```

z dwóch części spójnych.

$$m_i = S^1_{m_i} S^2_{m_i} \dots S^k_{m_i}, \quad k = 1, 2, \dots \tag{3.2}$$

Każda część spójna $S^j_{m_i} = b_1 b_2 \dots$ to słowo opisujące część jednego ruchu (w przypadku, gdy $k = 1$ opisuje ono cały ruch), a każde $b_i = (\Delta x_{S^j_{m_i}}, \Delta y_{S^j_{m_i}}, on_{b_i})$.

Jednym opisem pełnych ruchów staje się następnie połączenie opisów kolejnych części spójnych $S^j_{m_i}$ wyrażeniem "and then". Niektóre operatory (takie jak *outwards*) zależą od poprzedników, dlatego nie mogą zostać użyte w pierwszej części opisu. Jeden opis może dotyczyć więcej niż jednego ruchu, dlatego rozważając w danym momencie jeden ruch m_i należy brać również pod uwagę wszystkie inne ruchy tej figury. Dodatkowo, każdy z nich może dotyczyć jedynie ruchów z taką samą liczbą części spójnych.

Dla przykładu, figura z regułami $(1, 1, e)^*(0, 2, p) + (-1, 1, e)^*(0, 2, p)$ zostanie opisana jako "rides diagonally forward and then captures outwards 2 vertically", gdzie pierwsza część to połączenie klasy $\langle 1, 1\rangle$-rider z operatorem $forwards$, a druga część to połączenie klasy $\langle 0, 2\rangle$-leaper z operatorami *only capture* oraz *outwards*.

W ogólnym przypadku dla każdego nieopisanego jeszcze ruchu m_i Algorytm 1 próbuje znaleźć dla niego opis. Często zdarza się, że jeden opis odwzorowuje więcej niż jeden ruch figury, dlatego algorytm stara się opisywać ruchy zachłannie od najtrudniejszego (w tym wypadku o najdłuższym słowie). Z tego samego powodu argumentami funkcji są aktualnie przetwarzany ruch m_i oraz lista wszystkich ruchów figury w_F.

W przypadku gdy dla danego ruchu metoda się nie powiedzie (czyli zostanie zwrócone $Fail$), stosujemy procedurę generyczną. Przedstawia ona ruch jako wektor będący miejscem docelowym oraz listę wektorów będących warunkami, jakie muszą być spełnione, żeby go wykonać.

Gdy wszystkie reguły danej bierki zostaną już opisane, następuje przebieg naprawczy, który łączy odpowiednie opisy zgodnie z zasadami gramatyki. Sprawia to, że tekst staje się krótszy, lepiej zrozumiały oraz bardziej naturalny.

Główna część procedury

W pierwszy kroku algorytmu (linia 2) dla aktualnie przetwarzanego ruchu m wybierane są takie ruchy z w_F, które posiadają tę samą liczbę części spójnych co m. W linii 3 tworzona jest mapa, która jest wykorzystywana w dalszej części procedury. W każdym momencie działania przechowuje ona w sobie podzbiór ruchów z w_F zawierających również słowo m. Następnie wywoływana jest główna metoda algorytmu opisująca kolejne części spójne ruchu (linia 4).

Iteracja (linie 8–9) następuje dla każdej możliwej pary: klasa, lista operatorów. Zbiór *SortedOperatorsSet* zawiera listy co najwyżej 4 operatorów. Zbiór ten posortowany jest według wartości heurystycznej, przez co cała procedura działa zachłannie. Z tego powodu zwracane wyniki niekoniecznie są optymalne, jednakże wzrost prędkości działania jest znaczący w porównaniu do np. algorytmu pokrycia zbiorów (ang. set-cover). Ze względów wydajnościowych, listy operatorów są również filtrowane, aby wyeliminować sprzeczne elementy (np. połączenia operatora *forward* z operatorem *backward*).

W kolejnym kroku (linia 10) dla wybranej pary klasy i operatorów obliczane są wszystkie ruchy (*moves*) odpowiednie dla tej pary oraz ich opis w języku naturalnym (*description*). Następnie (linie 11–19) algorytm próbuje utworzyć nową mapę prefiksów H na podstawie tych ruchów, rozszerzając dotychczas utworzoną mapę *movesByPrefix* o i-tą część spójną. Nastąpi to tylko wtedy, gdy dla każdego prefiksu obliczonego dotychczas (*movesByPrefix*) i jego odpowiedniej listy ruchów, zawiera się w niej cały zbiór *moves*. Później należy sprawdzić, czy dany opis rzeczywiście odpowiada aktualnie przetwarzanemu słowu m (linie 20–22).

W przypadku gdy nie była to ostatnia część spójna (linia 23), następuje wywołanie rekurencyjne procedury dla kolejnej (linia 24). Po opisaniu ostatniej części następuje złączenie wyników (linia 28).

Jeżeli procedura nie powiedzie się, zwracana jest wartość *Fail* (linia 34), która jest również obsługiwana w przypadku wywołań rekurencyjnych w celu dalszego poszukiwania odpowiedniej pary klasy i operatorów (linie 25–27).

3.2.2. Ocena opisów

Funkcja f_{NLD} ocenia figurę korzystając z jej rozkładu na sumę klas i operatorów. Zdefiniujmy funkcję f_v do oceny jednego wektora przesunięcia $\langle x, y\rangle$. Jego intuicyjność jest mierzona jako suma wartości bezwzględnych obydwu współrzędnych oraz odległość do najbliższej linii prostokątnej lub przekątnej. Stąd wektor $\langle 2, 0\rangle$ jest łatwiejszy do przyswojenia niż wektor $\langle 2, 1\rangle$ lub $\langle 3, 0\rangle$.

$$f_v(\langle x, y\rangle) = \ 1 + \max(|x|, |y|) + \ \min(dist_+(x, y), dist_\times(x, y)). \tag{3.3}$$

Następnie zdefiniujmy f_o dla każdego możliwego operatora o. Ich wartości zostały dobrane ręcznie na podstawie eksperymentów. Niektóre z nich zostały przedstawione w Tablicy 3.1.

Każda część figury jest opisywana przez połączenie klasy z listą operatorów. Koszt takiej części zależy od wektora $\langle x, y\rangle$ powiązanego z klasą oraz kosztu opera-

Tablica 3.1: Koszt opisu dla niektórych operatorów

$f_o(o)$	o
0	none
1	backwards, forwards
2	sideways, only capture, without capture, outwards max times, min times, not horizontal
3	exactly times, over own piece instead
5	only odd, only even

torów.

$$f_p(\langle x, y\rangle, [o]) = \ f_v(\langle x, y\rangle) \cdot (1 + \sum_i f_o(o_i)) \tag{3.4}$$

Jeżeli jakiś ruch składa się z k części spójnych, jego wartością jest ich iloczyn:

$$f_p^k(\langle x, y\rangle_1, [o]_1, \ldots \langle x, y\rangle_k, [o]_k) = \ \prod_{i=1}^{k} f_p(\langle x, y\rangle_i, [o]_i) \tag{3.5}$$

W przypadku braku dopasowania opisu bazującego na klasach i operatorach do danej reguły ruchu, istnieje ogólna procedura opisująca je jako końcowy wektor przesunięcia, wraz z wektorami reprezentującymi warunki jakie muszą być spełnione, by dany ruch wykonać. Przykładem jest figura, która skacze (bez bicia) o 3 pola do przodu pod warunkiem, że jedno pole za nią jest puste a dwa pola przed nią zajmuje bierka przeciwnika. Wtedy zostanie ona opisana tak: Moves to $(0, 3, e)$ with conditions: $(0, -1, e)$,$(0, 2, p)$. Wartość takiego ruchu (wyznaczana wzorem 3.6) to iloczyn wartości wszystkich jej wektorów obliczonych za pomocą wzoru 3.3.

$$f_g([\langle x_i, y_i\rangle]) = \ \prod f_v(\langle x_i, y_i\rangle). \tag{3.6}$$

Gdy już wszystkie reguły ruchów figury zostały opisane oraz ocenione, wartością całej figury jest wtedy suma wszystkich jej komponentów z jakich się składa.

Wreszcie, aby ocenić całą grę, potrzebna jest lista figur $[p]$ oraz funkcja $\#$ zliczająca ilość wystąpień danej figury na planszy. Niech k będzie liczbą różnych typów figur. Wtedy wartość gry f_{NLD} jest liczona jako:

$$f_{NLD}([p]) = \ \frac{\sum_{i=1}^{k} \#p_i \cdot f_p(p_i)}{\sum_{i=1}^{k} \#p_i} \cdot (10 + k). \tag{3.7}$$

Stałe wartości we wzorach powyżej mają na celu wygładzenie różnic dla niektórych przypadków. Dla przykładu, gra posiadająca 4 figury powinna być generalnie lepiej oceniona od tej posiadającej 5 figur.

3.3. Eksperymenty i wyniki

3.3.1. Opisy popularnych figur

System został przetestowany na znanych figurach szachowych oraz popularnych przykładach fairy chess wziętych z *Piececlopedia*.

Zgodnie z terminologią, *leap* oznacza ruch na puste pole lub zbicie figury przeciwnika. Podobnie *ride* oznacza przejazd po pustych polach i może zakończyć się zbiciem lub pozostaniem na niezajętym polu.

Poniżej znajduje się lista popularnych figur wraz z ich oceną (w nawiasach).

- queen *(6)*: Rides in every direction
- short rook *(12)*: Rides max 4 times horizontally or vertically
- lance *(4)*: Rides vertically forward
- centaur *(14)*: Leaps 1 in every direction or over vector (1,2)/(2,1)
- griffon *(78)*: Moves 1 diagonally or moves 1 diagonally and then rides outwards vertically or horizontally
- hippogriff *(120)*: Moves 1 diagonally and then rides minimum 2 times outwards horizontally or vertically
- duke *(60)*: Rides without capturing diagonally and then leaps outwards 1 vertically or horizontally
- moa *(36)*: Moves 1 diagonally and then leaps outwards 1 vertically or horizontally
- ferz then wazir *(12)*: Moves 1 diagonally and then leaps 1 vertically or horizontally
- buffalo *(28)*: Leaps over vector (3,2)/(2,3) or (3,1)/(1,3) or (1,2)/(2,1)
- pionek (uproszczony) *(16)*: Captures 1 forward diagonally and moves 1 forward vertically

Warto zaznaczyć, że proceduralnie wygenerowane figury nie mają tak jasno zdefiniowanych reguł. Mogą one zawierać skomplikowane zasady ruchu, zbicia figur

przeciwnika lub swoich oraz reguły sztucznych warunków. Przykładem jest figura opisana przez system jako: "captures own forward over vector (5,2)/(2,5) or rides capturing but riding only over own pieces horizontally or vertically" (oceniana na 136). Takie figury są zazwyczaj dobrze opisane przez nasz algorytm, jednakże nie radzi on sobie w przypadku figur hopper'ów oraz tzw. *double-bent riders*, czyli figur składających się z przynajmniej 3 części spójnych (są one rzadkie, nawet w proceduralnie wygenerowanych grach). Pomimo tego, że opisy są poprawne, ciężko powiedzieć o nich, że są intuicyjne.

3.3.2. Ocena gier

Głównym zadaniem funkcji jest umiejętność opisywania oraz oceniania całych gier, w szczególności tych wygenerowanych proceduralnie. Przetestowałem tę funkcję f_{NLD} na zbiorze gier utworzonych przez ewolucję RAPP z [Kowalski and Szykuła, 2016] oraz ewolucję bazującą na symulacjach (SIMB) z [Kowalski and Szykuła, 2015]. Dodatkowo funkcja została przetestowana na niektórych grach utworzonych przez człowieka.

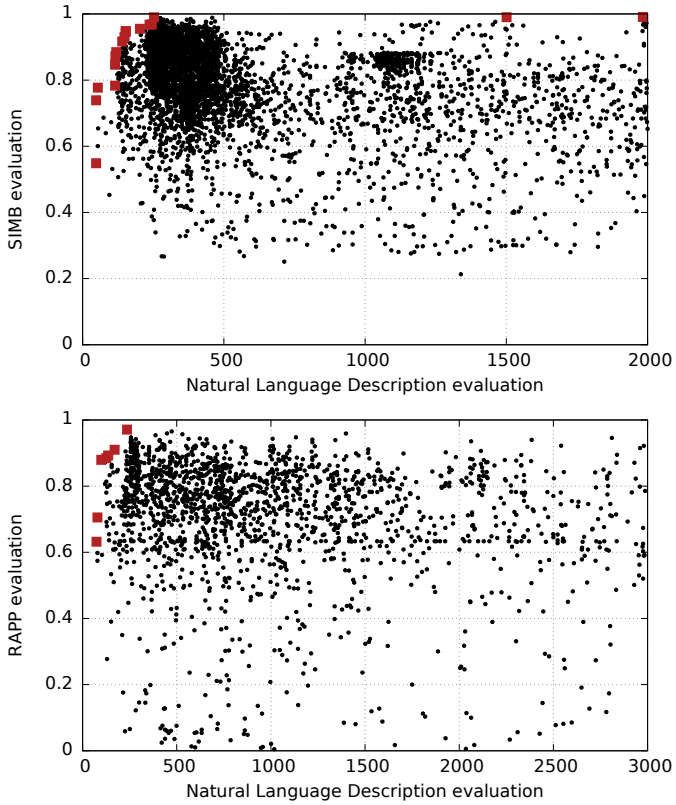
Najpierw przedstawię zależność pomiędzy złożonością reguł, a ich właściwościami strategicznymi w wygenerowanych grach, biorąc pod uwagę użyty generator. Rysunek 3.1 przedstawia te zależności oraz wyodrębnia gry należące do frontu Pareto. Wykres przedstawiający zależności RAPP zawiera gry, których wartość f_{NLD} nie przekracza 3000, a jest to 75% z wszystkich 2581 gier. Zbiór Pareto składa się z 8 gier, przy czym gra, której wartość RAPP to 0.978, a wartość F_{NLD} to 78557, nie jest pokazana na wykresie. Wartości w wykresie zawierającym gry utworzone przez generator SIMB są ograniczone do 2000 i jest to 54% ze wszystkich 8002 gier. Front Pareto w tym wypadku zawiera 17 gier, gdzie najgorszy wynik f_{NLD} to 1983.

To porównanie doskonale przedstawia różnice pomiędzy dwoma podejściami. Ocena SIMB jest mniej restrykcyjna i mniej wiarygodna, ponieważ więcej gier posiada wyższą jakość strategiczną. Jednak, co ważniejsze, gry te są jednocześnie bardziej czytelne dla ludzi. 35% z nich posiada wartość f_{NLD} mniejszą niż 500 w porównaniu do 20% dla generatora RAPP. Wpływ na to mogą mieć dwa czynniki: większa liczba elementów opartych na szablonach oraz złożoność użytej funkcji fitness.

Legacy of Ibis

Algorytm został wykorzystany nie tylko do oceny gier, ale również do opisania ich reguł. Pierwszą z takich gier jest znana z [Kowalski and Szykuła, 2016] Legacy of Ibis. Jej opis można zobaczyć na rysunku 3.2.

Reguły tej gry są dość skomplikowane w porównaniu do innych, ale pomimo tego nasz algorytm opisał je w jasny sposób. W przypadku figur, które posiadają



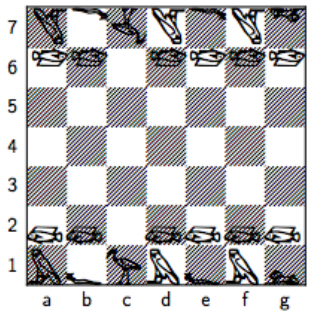
Rysunek 3.1: Zależność pomiędzy wartością f_{NLD} a właściwościami strategicznymi: generator oparty na symulacjach [Kowalski and Szykuła, 2015] na górze, generator oparty o RAPP [Kowalski and Szykuła, 2016] na dole. Czerwone wierzchołki to elementy należące do frontu Pareto. (Celem jest zmaksymalizowanie wartości strategicznych oraz zminimalizowaniu oceny opisu f_{NLD})

kilka typów ruchów (jak np. ), ciężko znaleźć opis, który byłby krótszy lub bardziej zrozumiały. Wyjątkiem jest tutaj najbardziej złożona bierka  posiadająca kilka ruchów rozpoczynających się od opisu „Rides 2 times", który mógłby zostać uproszczony w przebiegu naprawczym np. połączenie ich spójnikiem „or".

Przykładowa gra

Inny przykład wygenerowanej gry opisanej przez ten algorytm znajduje się na rysunku 3.3. Posiada ona drugą najwyższą wartość f_{NLD} z frontu Pareto ze zbioru gier RAPP. Dodatkowo, jest to trzecia najlepiej oceniana gra z tego zbioru przez ów generator.

Z gry tej można wywnioskować, że ocena właściwości strategicznych gier wspiera te, które posiadają wiele nieruchomych figur (w szczególności pionków), a tylko kilka mobilnych z dużymi możliwościami ataku, potwierdzając przy tym również wnioski z [Kowalski and Szykuła, 2016]. Sprawia to, że gry stają się bardzo statyczne oraz,



Leaps 1 forward vertically or diagonally
(złożoność: 8)

Moves forward over vector (1,2)
(złożoność: 16)

Captures 1 forward diagonally
Rides 2 times diagonally forward
Leaps forward over vector (2,1)
(złożoność: 26)

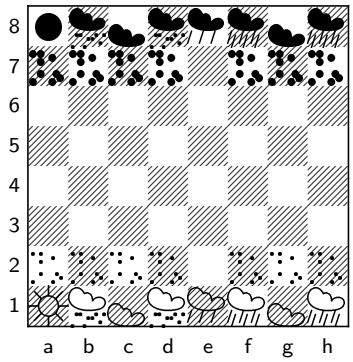
Leaps 1 forward vertically
Moves forward over vector (3,1)
Captures 1 horizontally
(złożoność: 30)

Captures 1 forward diagonally
Rides 2 times only over opponent pieces capturing or staying on empty diagonally forward
Rides 2 times without capturing horizontally
Rides 2 times without capturing diagonally forward
(złożoność: 56)

Rysunek 3.2: Układ początkowy figur oraz ich reguły dla gry Legacy of Ibis z pracy [Kowalski and Szykuła, 2016] z wartością f_{NLD} równą 387.

jako że większość ruchów nie może zostać cofnięta, wymagają dużo planowania naprzód i uważnych posunięć.

W przypadku tej gry, ruchy wstecz okazują się bardzo użyteczne. Większość mobilnych bierek jest ograniczona do ruchów w przód o długości 2. Dzięki temu mogą one dosyć łatwo przejść przez inne mobilne figury przeciwnika. Można to wykonać odpowiednio zarządzając figurą , która może atakować i wracać. Dodatkowo są one w stanie łatwo atakować linię pionków. Bronić się przed tym można poprzez cofanie pionków i obronę 7 linii.



Moves 1 backwards vertically
Captures 1 forward vertically
(złożoność: 16)

Leaps 2 forward diagonally
Captures 1 vertically or horizontally
(złożoność: 18)

Leaps 1 backwards diagonally
Leaps forward over vector (1,2)
(złożoność: 12)

Leaps 2 forward vertically
Leaps 1 vertically
(złożoność: 8)

Leaps 1 backwards diagonally
Captures 1 horizontally or diagonally
(złożoność: 16)

Leaps 1 backwards diagonally
Captures 1 diagonally
(złożoność: 10)

Rysunek 3.3: Układ początkowy figur oraz ich reguły dla gry wygenerowanej proceduralnie z wartością f_{NLD} równą 236 oraz wartością RAPP równą 0.971. Warunki zwycięstwa to zbicie króla (☼), dotarcie do ostatniego wiersza pionkiem (⁖) lub sprawienie, że przeciwnik nie ma żadnego legalnego ruchu. Limit ruchów to 209. Gra ta nosi nazwę **Weather Chess**.

Przez ograniczone możliwości ruchu figur, przeciwnik może zostać zmuszony do poddania gry, jeżeli nie będzie posiadał żadnego możliwego ruchu. Wymaga to zbicia lub zablokowania jego wszystkich mobilnych bierek. Alternatywną strategią jest zbicie króla. Wymaga to pozbycia się obrony w postaci pionków poprzez zbicie bądź

wymuszenie na nich, by się cofnęły. W każdym przypadku kluczowa wydaje się być kontrola nad środkiem planszy i wymuszanie na przeciwniku ruchów w niekorzystne dla niego pozycje, w których figury stają się bezużyteczne bądź łatwe do zbicia.

Porównanie gier

W celu porównania poprzednio opisanych dwóch gier z innymi, znanymi przykładami, w tabeli 3.2 przedstawione zostały wartości funkcji f_{NLD} dla różnych popularnych gier, w tym szachów oraz innych, wygenerowanych proceduralnie. W niektórych przypadkach reguły musiały zostać lekko uproszczone, aby zawarły się one w klasie Simplified Boardgames.

Wariant szachów Chess With Different Armies (CWDA) to zbiór alternatywnych figur do szachów ortodoksyjnych [Wikipedia, 2017a]. Szachy Tamerlane to duża i złożona gra z wieloma nieortodoksyjnymi figurami [Duniho, 2016]. Legacy of Ibis to gra wygenerowana przez RAPP przedstawiona w [Kowalski and Szykuła, 2016], a SIMB 30-P4 to gra wygenerowana przez ewolucję SIMB i przedstawiona w [Kowalski and Szykuła, 2015].

Tablica 3.2: Wartości funkcji f_{NLD} dla wybranych gier.

Gra	f_{NLD}
Shatranj	166
Szachy	168
CWDA Colorbound Clobberers	196
CWDA Remarkable Rookies	222
Weather chess (Rys. 3.3)	236
CWDA Nutty Knights	248
SIMB 30-P4	253
Szachy Tamerlane	376
Legacy of Ibis (Rys. 3.2)	387

Rozdział 4.

Generowanie figur

Gry cyfrowe są przedmiotem badań w dziedzinie sztucznej inteligencji już od ponad dekady. Większość tych badań traktuje je jako *systemy*, skupiając się na ich właściwościach funkcjonalnych. Na przykład kreując agentów do grania, ich rola koncentruje się na jakości wykonywanych zadań, a wynik z gry jest miarą ich sukcesu ([Perez et al., 2015]). Co więcej, badania nad PCG często ograniczają się do aspektów funkcjonalnych, takich jak m.in. zasady i poziomy gier i oceniają je pod względem tego, czy da się je rozwiązać. Przykładem tego mogą być systemy stworzone w domenie GGP jako agenci ogólnego użytku m.in. w [Genesereth et al., 2005, Genesereth and Björnsson, 2013]. Ostatnie badania w dziedzinie General Video Game AI poszerzają te ograniczenia m.in. w [Nielsen et al., 2015b, Khalifa et al., 2016] o generowanie poziomów gier oraz w [Khalifa et al., 2017] o generowanie reguł.

Jednakże, gry przykuwają uwagę gracza nie tylko swoimi atrybutami funkcjonalnymi (jak ich reguły i kombinacja poziomów gier), ale też estetyką. Gry wideo swoją warstwą wizualną i dźwiękową potrafią wywołać w graczach odpowiednie emocje oraz zachęcić go do odkrywania świata, w którym się znajduje. Elementy gry mogą być w pewnym stopniu generowane algorytmicznie, jednak wciąż nie jest jasnym, jak ich własności powinny wpływać na ich estetykę. W [Lopes et al., 2015] system *Sonancia* potrafi generować poziomy gier, a następnie wykorzystywać elementy znajdujące się w danej lokacji, by utworzyć podkład dźwiękowy. Kieruje się on przy tym założeniami, że miejsca zawierające większą liczbę przeciwników powinny przekładać się na bardziej dramatyczną muzykę. Na przykładzie [Karavolos et al., 2017] można wnioskować, że systemy mogą nauczyć się znajdować takie zależności pomiędzy elementami rozgrywki a ich walorami estetycznymi.

Gry planszowe pomimo tego, że mają mniejsze możliwości pokazania swoich fizycznych aspektów, używają warstwy wizualnej do uwydatnienia ważnych elementów rozgrywki. Przykładami są symbole w grach karcianych takich jak *Uno* (Mattel, 1992) czy różne kształty, rozmiary i kolory figurek domków w grze *Monopoly* (Parker Bros., 1935). W ten sam sposób figury szachowe swoim kształtem i rozmiarem określają swoją funkcję w grze. Duże rozmiary hetmana i króla, w porównaniu do

mniejszych i prostszych figur np. pionków, pokazują, że są to figury ważne, posiadające kluczową rolę w rozgrywce.

Gry, których reguły zostały wygenerowane proceduralnie, często wymagają utworzenia elementów wizualnych ręcznie bądź też użycia standardowych kształtów, tak jak miało to miejsce w systemie METAGAME lub w [Kowalski and Szykuła, 2015]. Wszystko po to, by odzwierciedlały one w pewnym stopniu zasady rozgrywki, były ze sobą spójne oraz aby możliwe było rozróżnienie ich wśród innych gier. W przypadku gier szachopodobnych powstało już wiele kształtów figur. Mogą one zostać wykorzystane jako inspiracje do wygenerowania nowych oraz stanowić bazę metody znajdującej zależności pomiędzy właściwościami funkcjonalnymi i wizualnymi.

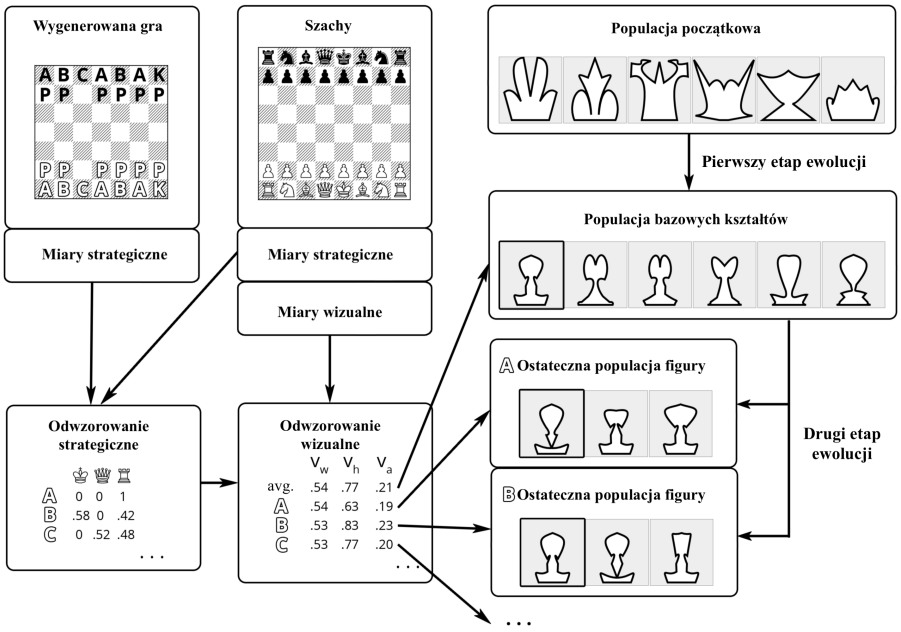
W tym rozdziale zostanie przedstawiony algorytm ewolucyjny generujący kształty figur dla dowolnej gry należącej do klasy Simplified Boardgames. Jednym z głównych celów metody jest wygenerowanie zbioru figur, które byłyby do siebie podobne na tyle, że możliwe stałoby się stwierdzenie, iż należą do tej samej gry. Następnie chcielibyśmy, aby kształt figury odzwierciedlał jej rolę w grze. Bazą algorytmu będą figury szachowe, a w szczególności zależności pomiędzy ich kształtem a regułami poruszania się i rolą w rozgrywce. Eksperymenty będą opierać się na znalezieniu nowych kształtów dla proceduralnie wygenerowanych gier z Rysunku 3.2 oraz Rysunku 3.3

4.1. Metoda

Główną funkcją systemu jest wczytanie zasad dowolnej gry szachopodobnej z klasy Simplified Boardgames i wyprodukowanie kształtu dla każdej z jej figur. System będzie bazować na szachach, ponieważ reguły i kształty bierek są powszechnie znane.

W tym rozdziale przedstawiona zostanie metoda użyta do generowania wszystkich kształtów figur jednej gry. Jej podsumowanie znajduje się na Rysunku 4.1. W pierwszej kolejności obliczane są właściwości strategiczne i wizualne szachów, a następnie miary strategiczne gry docelowej. Na podstawie miar strategicznych tworzone jest *odwzorowanie strategiczne*, które określa podobieństwo pomiędzy bierkami szachowymi a tymi z nowej gry. Na podstawie tego odwzorowania oraz miar wizualnych szachów tworzone jest *odwzorowanie wizualne* dla każdej z nowych figur oraz dodatkowo ich uśredniona wartość. Każde z odwzorowań wizualnych jest następnie użyte w algorytmie ewolucyjnym jako funkcja celu. Algorytm ewolucyjny jest podzielony na dwa etapy. Pierwszy z nich tworzy, na podstawie pewnej początkowej populacji, bazowy kształt figur. Funkcją celu w tym wypadku jest uśredniona wartość miar wizualnych (z odwzorowania) wszystkich nowych figur. W kolejnym etapie każda figura z osobna przechodzi ewolucję, startując z tego samego kształtu, a jej miary wizualne stają się funkcją celu tej ewolucji.

W rozdziale 4.1.1. przedstawione są miary strategiczne, sposób ich obliczania



Rysunek 4.1: Schemat działania metody.

oraz to, w jaki sposób zachowanie figury ma na nie wpływ. Podobnie w rozdziale 4.1.2. opisane są miary wizualne figur. Rozdział 4.1.3. opowiada, w jaki sposób szachy porównywane są z nowymi, wygenerowanymi grami. W rozdziale 4.1.4. przedstawiony jest sposób reprezentacji kształtów, a schemat algorytmu ewolucyjnego oraz różne warianty wyborów ostatecznych kształtów znajdują się w rozdziale 4.1.5..

4.1.1. Miary strategiczne

Biorąc pod uwagę rolę figury w rozgrywce można wyróżnić kilka miar strategicznych. Mają one opisać atrybuty bierki, takie jak: znaczenie, ruchliwość (zwinność bądź masywność), użyteczność w ataku i obronie itd.

s_{po} Ułamek wystąpień figury w ustawieniu początkowym gry.

s_{ec} Ułamek ruchów figury, które kończą się zbiciem.

s_{pw} Ta wartość wynosi 1, jeżeli figura jest używana do zwycięstwa przez zajęcie pozycji, 0 w przeciwnym przypadku.

s_{cw} Ta wartość wynosi 1, jeżeli figura jest używana do zwycięstwa przez zbicie, 0 w przeciwnym przypadku.

s_{ba} Stosunek liczby pól, które są osiągalne przez figurę do liczby wszystkich pól na planszy.

s_{mr} Liczba ruchów wymagana do osiągnięcia najbardziej odległego pola na planszy.

Tablica 4.1: Wartości metryk strategicznych dla każdej z rozważanych gier.

	s_{po}	s_{em}	s_{ec}	s_{pw}	s_{cw}	s_{ba}	s_{mr}	s_{lm}	s_{sd}	s_{dd}
Szachy										
	0.5	0.33	0.66	1	0	0.55	6	2.48	1.28	1.28
	0.13	0.5	0.5	0	0	1	5	5.25	2.24	2.24
	0.13	0.5	0.5	0	0	1	2	8.75	1.41	5.66
	0.13	0.5	0.5	0	0	1	2	14	1	4
	0.06	0.5	0.5	0	0	1	2	22.7	1.21	4.8
	0.06	0.5	0.5	0	1	1	7	6.56	1.21	1.21
Legacy of Ibis										
	0.46	1	0	1	0	0.1	2	1.16	2.36	2.36
	0.23	0.75	0.25	0	0	0.5	4	3.92	1.31	2.27
	0.15	0.5	0.5	0	0	0.7	6	2.49	1.28	1.28
	0.08	0.4	0.6	0	0	0.82	4	3.95	1.74	2.31
	0.08	0.5	0.5	0	1	1	7	3.55	1.72	1.72
Weather Chess										
	0.46	0.5	0.5	1	0	0.13	6	1.75	1	1
	0.13	0.25	0.75	0	0	1	5.5	4.63	1.91	1.91
	0.13	0.5	0.5	0	0	1	10	2.84	1.83	1.83
	0.07	0.5	0.5	0	0	0.13	4	2.5	1.33	1.33
	0.13	0.25	0.75	0	0	1	8	4.81	1.31	1.31
	0.07	0.33	0.66	0	1	0.5	7	3.06	1.41	1.41

s_{lm} Średnia liczba możliwych do wykonania ruchów z każdego osiągalnego pola na planszy.

s_{sd} Średnia wartość przesunięcia figury, tzn. $\sqrt{\Delta x^2 + \Delta y^2}$.

s_{dd} Średnia wartość przemieszczenia figury, tzn. $\sqrt{(\sum \Delta x)^2 + (\sum \Delta y)^2}$.

Każda z tych miar odzwierciedla pewną cechę bierki: s_{po} ocenia unikalność figury na planszy; s_{ec} szacuje jej agresywność; s_{pw} i s_{cw} oceniają wagę figury pod względem warunków zwycięstwa; s_{ba}, s_{mr} oraz s_{lm} opisują jej mobilność; a s_{sd} i s_{dd} określają styl, w jakim się poruszają. Rozważane były również inne metryki, lecz zostały one uznane za nadmiarowe.

Tablica 4.1 przedstawia listę wartości metryk strategicznych dla wszystkich rozważanych gier. Szachy na te potrzeby zostały lekko zmodyfikowane: nie ma początkowego podwójnego ruchu pionka oraz, aby zrekompensować brak promocji, gracz może wygrać poprzez osiągnięcie drugiego końca planszy pionkiem.

4.1.2. Miary wizualne

Aby uchwycić najważniejsze cechy wizualne figury, można wyróżnić cztery aspekty, które powinny być odzwierciedlone przez metryki wizualne. Są to: rozmiar kształtu, styl zajętego przez nią pola, styl linii oraz kątów pomiędzy nimi.

Bazując na pracach [Liapis, 2016] oraz [Liapis et al., 2012] użyte zostały następujące miary:

v_w Stosunek szerokości figury do szerokości całego obszaru.

v_h Stosunek wysokości figury do wysokości całego obszaru.

v_a Stosunek pola figury do pola całego obszaru.

v_{ta} Stosunek pola górnej $\frac{1}{3}$ części figury do pola całej figury.

v_{ma} Stosunek pola środkowej $\frac{1}{3}$ części figury (względem osi x) do pola całej figury.

v_s Stosunek iloczynu do sumy pomiędzy lewą połową oraz odwróconą prawą połową pola figury. Symetryczne bierki otrzymują tutaj wartość 1.

v_{my} Stosunek pola środkowej połowy figury względem osi y do pola całej figury.

v_{tr} Pole iloczynu figury i trójkąta skierowanego w górę o wysokości tej figury.

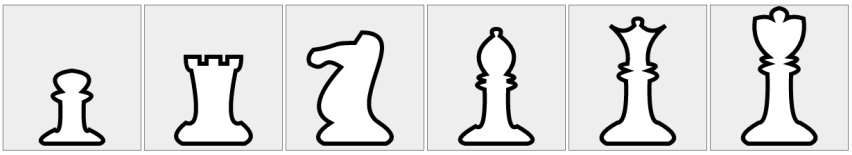
v_p Stosunek obwodu figury do podwojonego obwodu ramy ograniczającej.

v_{sl} Stosunek długości linii prostych do obwodu figury.

v_{sa} Stosunek liczby ostrych kątów (0°–60°) do liczby wszystkich kątów.

v_{ga} Stosunek liczby łagodnych kątów (120°–180°) do liczby wszystkich kątów.

Wartości każdej z metryk są ograniczone do przedziału $[0, 1]$. Tabela 4.2 przedstawia wizualne metryki dla gier tutaj rozważanych. Wartości dla szachów zostały wzięte ze zbioru kształtów z Rys. 4.2, zaś wartości dla innych gier zostały obliczone używając metod opisanych w rozdziale 4.1.3.



Rysunek 4.2: Zbiór figur szachów użyty jako baza do dalszych obliczeń.

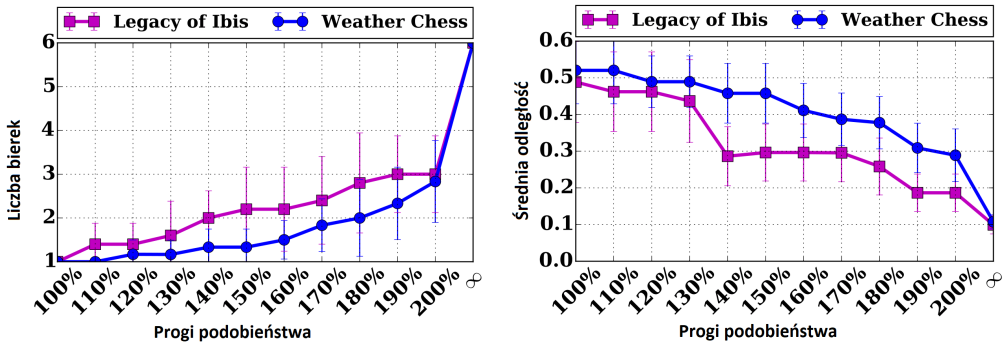
Tablica 4.2: Wartości metryk wizualnych dla każdej figury z rozważanych gier. Wartości dla szachów zostały obliczone na podstawie kształtów z Rysunku 4.2, natomiast dla pozostałych gier na podstawie kształtów uzyskanych przez algorytm w rozdziale 4.1.3..

	v_w	v_h	v_a	v_{ta}	v_{ma}	v_s	v_{my}	v_{tr}	v_p	v_{sl}	v_{sa}	v_{ga}
Szachy												
♙	0.45	0.52	0.12	0.29	0.28	1	0.84	0.73	0.41	0	0.22	0.33
♘	0.59	0.8	0.29	0.31	0.31	0.77	0.73	0.58	0.43	0.04	0	0.6
♗	0.52	0.82	0.18	0.26	0.29	1	0.88	0.79	0.43	0	0.15	0.38
♖	0.54	0.63	0.19	0.46	0.18	1	0.73	0.5	0.45	0.29	0	0.29
♕	0.52	0.9	0.2	0.24	0.29	1	0.85	0.81	0.49	0	0.27	0.33
♔	0.52	0.98	0.26	0.35	0.25	1	0.83	0.69	0.47	0	0.4	0.33
Legacy of Ibis												
	0.55	0.81	0.23	0.28	0.3	0.91	0.82	0.7	0.44	0.02	0.09	0.5
	0.54	0.63	0.19	0.46	0.18	1	0.73	0.5	0.45	0.29	0	0.29
	0.53	0.83	0.23	0.4	0.22	1	0.79	0.61	0.46	0.12	0.23	0.32
	0.53	0.77	0.2	0.34	0.24	1	0.79	0.66	0.47	0.14	0.14	0.31
	0.55	0.83	0.24	0.34	0.26	0.94	0.78	0.64	0.46	0.08	0.16	0.42
Weather Chess												
	0.54	0.63	0.19	0.46	0.18	1	0.73	0.5	0.45	0.29	0	0.29
	0.53	0.76	0.2	0.35	0.23	1	0.79	0.65	0.47	0.15	0.13	0.31
	0.53	0.75	0.19	0.34	0.25	1	0.82	0.68	0.44	0.11	0.09	0.35
	0.53	0.78	0.2	0.33	0.24	1	0.8	0.68	0.47	0.12	0.15	0.32
	0.52	0.98	0.26	0.35	0.25	1	0.83	0.69	0.47	0	0.4	0.33
	0.45	0.53	0.12	0.29	0.29	1	0.85	0.73	0.41	0	0.22	0.33

4.1.3. Powiązanie ogólnej gry z szachami

Bierki szachowe posiadają pewne cechy wizualne, które pozwalają określić ich rolę w grze. Jednak nowe gry mogą być ocenione jedynie poprzez ich właściwości strategiczne, a określenie kształtu ich figur w tym przypadku nie jest oczywiste. Próbując określić wygląd bierki, można porównać jej wartości strategiczne z wartościami figur szachowych i na tej podstawie przybliżyć kształt nowej figury do najlepiej pasującej bierki szachowej. Mimo tego nawet przy tak bazowych założeniach należy odpowiedzieć sobie na pytania, jak mierzone są podobieństwa strategiczne pomiędzy figurami z różnych gier oraz czy wygenerowana figura może być podobna do więcej niż jednej figury szachowej. Warto też zastanowić się, jak wpływa to na jej wygląd?

W rozdziale 4.1.1. przedstawionych zostało wiele właściwości figur i intuicji,



Rysunek 4.3: Wykres wrażliwości figur szachowych dla różnych progów. Wartość progów jest to najwyższa wartość stosunku odległości figury od jej najbliższej figury szachowej. Wyniki to uśredniona liczba użytych figur szachowych oraz uśredniona odległość wszystkich par figur.

jakie za nimi stoją. Najprostszym sposobem zmierzenia podobieństwa dwóch bierek jest użycie odległości Euklidesowej, w której 9 miar strategicznych traktowanych będzie jako jeden 9-wymiarowy wektor. Dzięki temu, możliwe jest porównywanie figur pochodzących z różnych gier, przy jednoczesnym pominięciu faktu, że zakres wartości figur w grach się różni. Weźmy przykład takiej gry, w której większość figur porusza się o jedno lub dwa sąsiadujące pola. Wtedy figury będą przyrównywane do pionka lub króla, pomimo tego, że te przemieszczające się o dwa pola są dużo bardziej dynamiczne od tych poruszających się o jedno. Z uwagi na to, z praktycznego punktu widzenia nie wszystkie miary strategiczne są tej samej wagi (np. s_{mr} i s_{sd}), jak również ich wartości mogą pojawiać się dla każdej figury z innych powodów.

W związku z powyższym, bierki z dwóch gier są na początku normalizowane za pomocą Standaryzacji Z, która przedstawia wartość wektora x jako $z = (x - \bar{x})/\sigma_x$, gdzie $\bar{x}$ jest średnią wartością wszystkich figur w jednej grze, a σ_x standardowym odchyleniem tych wartości. Standaryzacja Z zapewnia, że wszystkie metryki znajdują się w tym samym zakresie. Pozwala ona również na wychwycenie odstających osobników. W ten sposób, figury jednej gry będą zbliżone do figur odgrywających podobną rolę w innych grach (porównując np. średnią mobilność każdej z gier).

Gdy już podobieństwa pomiędzy bierkami nowej gry a szachowymi zostaną wyznaczone używając do tego dziewięciu znormalizowanych metryk strategicznych, należy znaleźć odwzorowanie metryk wizualnych dla każdej nowej figury. Teoretycznie, każda nowa bierka mogłaby być odwzorowana przez najlepiej pasującą bierką szachową, obierając ją w algorytmie ewolucyjnym jako funkcję celu. Jednak wynikiem tego podejścia będą figury mocno przypominające te szachowe. Może to prowadzić również do tego, że do wielu figur nowej gry będzie przyporządkowana ta sama bierka szachowa. Wpływa to znacząco na rozróżnialność nowych elementów. Innym sposobem mogłoby być traktowanie każdej figury szachowej jako w pewnym stopniu podobnej do tej nowej, a reprezentacja jej metryk wizualnych przedstawiona byłaby jako suma ważona tych podobieństw. To rozwiązanie wydaje się dobrym pomysłem,

ale w praktyce wagi poszczególnych figur stają się na tyle do siebie podobne, że nowe figury również niewiele się od siebie różnią.

Zamiast tego opracowane zostało rozwiązanie pośrednie, które uwzględnia jedynie te figury szachowe, które są względnie blisko nowej figury. Dzieje się to poprzez podzielenie odległości pomiędzy bierkami szachowymi a nową figurą i odległością do najbliższej figury szachowej. Wynikiem jest metryka W_i, której wartości znajdują się w przedziale $(0, 1]$. Wartości bliżej 1 określają figurę szachową, która jest bardzo podobna do nowej figury. Wartości, od której figura szachowa jest brana pod uwagę (jest wystarczająco podobna), została wybrana arbitralnie, a jej wybór może wpłynąć na wyniki. Bazując na wstępnej analizie wrażliwości (patrz Rys. 4.3), wybrane zostały tylko te figury, których odległość w rozumieniu strategicznym wynosi co najwyżej 160% do najbliższej figury szachowej, czyli $W_i \geq 0.625$.

W ostatnim kroku wszystkie wartości podobieństwa powyżej danego przedziału są normalizowane w taki sposób, aby ich suma była równa 1. Te znormalizowane wartości dla dwóch rozważanych gier znajdują się w Tabeli 4.3.

Wartością metryk wizualnych każdej nowej figury jest zatem ważona suma tych samych metryk bierek szachowych, gdzie wagą każdej z nich jest znormalizowana wartość miary W_i.

Tablica 4.3: Znormalizowany stopień podobieństwa pod względem metryk strategicznych figur z dwóch rozważanych gier i figur szachowych.

	♙	♘	♗	♖	♕	♔
Legacy of Ibis						
		0.42	0.58			
				1.00		
				0.42		0.58
				0.48	0.52	
		0.29		0.24	0.20	0.28
Weather Chess						
				1.00		
				0.52	0.48	
			0.61	0.39		
				0.42	0.58	
						1.00
	1.00					

4.1.4. Reprezentacja

Wygenerowane kształty są przedstawione w formacie grafiki wektorowej Scalable Vector Graphics (SVG), który w naturalny sposób pozwala na użycie ich w prawdziwej aplikacji. Obszar rysowania figury został ustawiony na 200×200 jednostek, gdzie punkt znajdujący się w lewym górnym rogu ma współrzędne $(0, 0)$, a w prawym dolnym rogu $(200, 200)$. Każda figura reprezentowana jest jako lista kilku elementów, w których każdy z nich może być linią prostą lub krzywą Béziera drugiego bądź trzeciego stopnia. Punktem początkowym każdego elementu jest koniec jego poprzednika, a więc w reprezentacji jest on pomijany. Wyjątek stanowi pierwszy element, którego początkiem jest punkt znajdujący się na najniżej położonej linii poziomej obszaru rysowania (wartość współrzędnej y wynosi wtedy 200). Punkt końcowy ostatniego elementu, znajdujący się również na tej linii, jest automatycznie połączony z punktem początkowym pierwszego elementu, zamykając cały kształt figury i tworząc jej podstawę.

Reprezentacja figury różni się w zależności od tego, czy figura jest symetryczna czy nie. W przypadku figur symetrycznych w tablicy znajduje się tylko prawa połowa figury, podczas gdy druga (lewa) strona jest jej lustrzanym odbiciem. Figury te mają oś symetrii dokładnie względem prostej o współrzędnej x równej 100. Figury asymetryczne reprezentowane są jako lista wszystkich ich elementów.

4.1.5. Ewolucja

Do ewolucji kształtów został użyty standardowy schemat algorytmu ewolucyjnego.

Krzyżowanie

Do krzyżowania (ang. crossover) n elementowej populacji wybieranych jest $\frac{n}{2}$ par figur za pomocą wyboru turniejowego (ang. tournament selection). Efektem jednego krzyżowania jest dwójka potomków, która dodatkowo może przejść mutacje.

Krzyżowanie polega na przecięciu figur a i b w połowie, dzieląc każdą na dwie części $a_L a_R$ oraz $b_L b_R$ odpowiednio, a następnie używając po jednej części z każdego rodzica do utworzeniu dwójki dzieci: $a_L b_R$ oraz $b_L a_R$. Jeżeli tylko jeden z rodziców jest figurą asymetryczną, wtedy tymczasowo drugi z nich zmienia swoją reprezentację również na taką.

Mutacja

Jednym z typów mutacji figury symetrycznej jest zamiana jej na asymetryczną. W takim wypadku figura nie przechodzi więcej mutacji.

Inną możliwością jest wybranie losowo jednego z elementów figury oraz przekształcenie go na jeden z dwóch sposobów. Pierwszym jest zamiana typu elementu na inny: prostą linię w dwie linie (poprzez przecięcie w połowie) lub w łuk (krzywą Béziera drugiego stopnia), łuk w podwójny łuk (krzywą Béziera trzeciego stopnia) a podwójny łuk z powrotem w linię. Drugim sposobem jest przesunięcie punktów elementu o pewien losowy wektor. Może to być punkt końcowy (linii lub łuków) bądź punkty kontrolne krzywych Béziera.

Selekcja

Aby wspierać różnorodność (ang. novelty) podczas selekcji, kształty zbyt podobne do tych z wcześniejszej populacji są usuwane. Stopień podobieństwa jest obliczany jako stosunek przecięcia obszarów dwóch figur do sumy ich obszarów.

Niepoprawne elementy zostają usunięte z populacji. Kolejna generacja jest utworzona przez wybranie n najlepszych kształtów ze zbiorów wcześniejszej populacji i nowo utworzonych oraz zmutowanych osobników.

4.1.6. Schemat algorytmu

Wariant dwuetapowy

Algorytm można podzielić na dwa etapy. W pierwszym populacja początkowa przechodzi ewolucję dążąc do uśrednionych wartości wszystkich docelowych figur. Robi to bazując na 12-wymiarowych wektorach określających metryki figury z Sekcji 4.1.2., uśredniając je.

W drugi etapie ewolucja kontynuuje dla każdej figury docelowej z osobna. Celem każdego z tych przebiegów ewolucji jest uzyskanie podobieństwa z figurą docelową za pomocą wspomnianych wcześniej metryk wizualnych. Populacją początkową każdej z ewolucji w drugim etapie jest kopia najlepszej jednostki uzyskanej w pierwszym etapie algorytmu. Dla wszystkich figur następuje ewolucja trwająca kilka generacji, a następnie najsilniejszy element populacji jest wybrany jako ostateczny kształt bierki. Dodatkowo wybrane kształty są dobierane tak, aby przynajmniej w pewnym stopniu się od siebie różniły w celu zapewnienia różnorodności.

Wariant jednoetapowy

W tym wariancie drugi etap poprzedniej wersji algorytmu jest znacznie uproszczony. Wybierane są w nim wszystkie figury z pierwszego etapu algorytmu, a następnie dla każdej z bierek przypisywana jest najbardziej pasująca do jej cech figura. Należy jednak uważać, bo mogą one być do siebie zbyt podobne, dlatego wymagane

jest dodatkowe sprawdzenie ich podobieństwa i dobranie takiego zestawu, który wystarczająco się od siebie różni. Ta metoda skraca cały proces, jednak zmniejsza prawdopodobieństwo, że wybrana figura będzie bardzo podobna do modelu który chcieliśmy uzyskać.

Wariant indywidualny

Ostatnim wariantem algorytmu jest taki, w którym każda figura od samego początku ewoluuje niezależnie, używając do tego podobieństwa z jej figurą docelową i obierając te wartości jako funkcje celu (na podstawie Tabeli 4.2). Ta metoda jest najwolniejsza i ma dużą szansę na utworzenie niespójnych (w ramach jednej gry) kształtów.

4.2. Wyniki

W celu przetestowania metody użyję jej aby wygenerować kształty do szachów, Legacy of Ibis z Rysunku 3.2 oraz Weather Chess z Rysunku 3.3. W przypadku szachów celem jest uzyskanie podobnych figur, zaś dla kolejnych dwóch utworzenie nowych kształtów na podstawie metody z rozdziału 4.1.3..

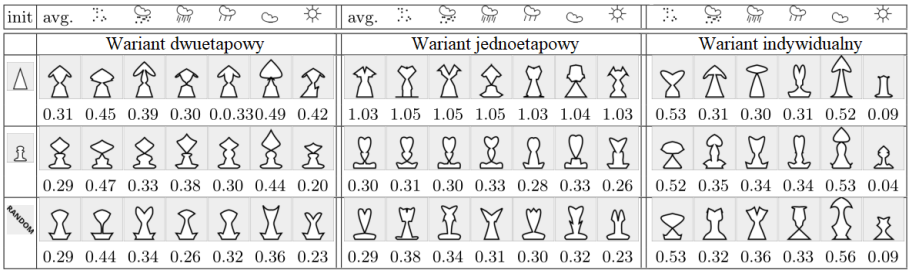
Pierwsze testy

W początkowym etapie testowania metody użyte zostały losowe parametry i obserwowane były różne typy generowanych kształtów. Pierwszy krok ewolucji działał w dwóch równomiernie rozłożonych układach parametrów: głębokim (od 200 do 400 generacji, rozmiar populacji od 40 do 100) oraz szerokim (od 50 do 200 generacji, rozmiar populacji od 200 do 500). W przypadku ewolucji dwuetapowej, drugi etap zawierał od 1 do 10 generacji. Populacja początkowa była wybierana na trzy sposoby: używając wielu kopii figury w kształcie trójkąta, figur pionka z szachów i używając losowych kształtów.

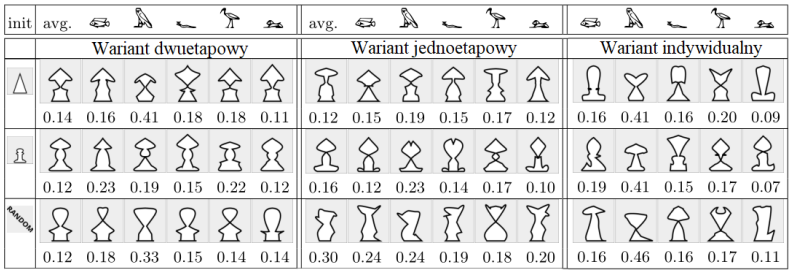
Najlepsi przedstawiciele z każdego przebiegu dla tych wariantów zostali wybrani ręcznie. Część z nich znajduje się na Rysunku 4.4 i pokrywają większość kombinacji i wariantów kształtów początkowych dla wszystkich rozważanych tutaj gier.

Porównanie statystyczne

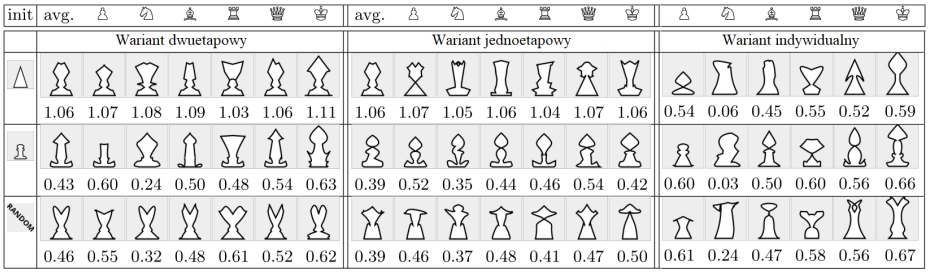
Dzięki wynikom pierwszych testów algorytmu możliwe było ustalenie odpowiednich jego parametrów. Aby wykonać porównanie statystyczne pomiędzy różnymi wariantami metod, gier i kształtów początkowej populacji, wykonano 20 niezależnych przebiegów tych samych kombinacji. Dla każdej z nich wyniki zostały następnie



Wyniki badań dla szachów



Wyniki badań dla Legacy of Ibis



Wyniki badań dla Weather Chess

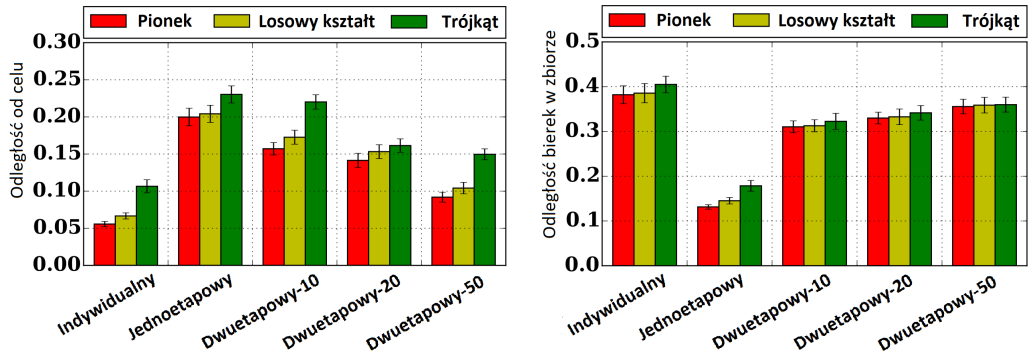
Rysunek 4.4: Przykładowe zbiory kształtów utworzonych metodą ewolucyjną dla każdej z rozważanych gier, wszystkich kształtów początkowych (trójkąt, pionek, losowy kształt) i wariantów algorytmu. Oceny kształtów znajdują się pod obrazkami.

uśrednione. Każdy przebieg ewolucji dla jednej kombinacji parametrów posiada populację wielkości 100 i działa przez 100 generacji. W przypadku ewolucji dwuetapowej, 100 generacji zostało podzielone pomiędzy pierwszy i drugi etap w stosunkach: 90:10, 80:20 oraz 50:50. Na rysunku 4.5 przedstawione zostały uśrednione wartości wyników z przedziałem ufności na poziomie 95%.

Testowane były dwie ważne własności:

- cel ewolucji, czyli odległość Euklidesowa metryk wizualnych wygenerowanych kształtów od metryk docelowych dla każdej z figur,
- spójność figur w ramach jednej gry, liczona jako średnia odległość Euklidesowa metryk wizualnych pomiędzy wszystkimi parami jej figur.

Dla 5 figur z gry Legacy of Ibis, 6 figur z Weather Chess oraz 6 figur szachów, każda kombinacja parametrów w 20 przebiegach utworzyła 340 nowych kształtów z tego samego kształtu początkowego.



(a) Odległość od docelowego kształtu (b) Odległość figur w ramach jednej gry

Rysunek 4.5: Wykres przedstawia wyniki z 20 przebiegów dla każdej kombinacji gry, początkowego kształtu i wariantu algorytmu. Przedział ufności to 95%.

Z Rysunku 4.5 można jasno wywnioskować, że różne warianty metody zachowują się zgodnie z założeniami. Wariant indywidualny najlepiej dopasowuje pojedyncze kształty do figur, jednak w ramach jednej gry różnią się one od siebie o wiele bardziej niż w innych wariantach. Dokładnym przeciwieństwem tego jest wariant jednoetapowy, w którym kształty są dobierane z populacji, której celem były wartości uśrednione. Metoda dwuetapowa natomiast znajduje pewien balans pomiędzy nimi dwoma, przy czym figury lepiej pasujące do cech bierek można zaobserwować przy większej liczbie generacji w drugim etapie ewolucji. Jednakże, nawet kilka generacji w drugim etapie algorytmu powoduje, że figury tracą spójność i odległość jest nawet dwa razy większa niż w wariancie jednoetapowym.

Na podstawie wyników można również zaobserwować, że pionek jest najlepszą figurą użytą jako populacja początkowa. Jest on bardziej złożony od trójkąta oraz bliższy wizualnie różnym wariantom bierek niż losowe kształty.

Zaskakującym wynikiem w przypadku konkretnych gier jest to, że trudniej jest uzyskać kształt figur szachowych niż pozostałych gier (porównując odległość metryk wizualnych), pomimo tego, że celem tych ewolucji były wartości pochodzące dokładnie z tej gry.

Rozdział 5.

Podsumowanie

Pracę tę można podzielić na dwie części. W pierwszej przedstawiona została metoda generowania opisu ruchu bierek w języku naturalnym dla dowolnych figur fairy chess należących do klasy Simplified Boardgames. Opisy te bazują na figurach utworzonych przez ekspertów, m.in. w [Wikipedia, 2017b]. Są one intuicyjne, opierają się na wiedzy jaką posiada człowiek, a ponadto są tworzone na podstawie teorii ruchów fairy chess, co pozwala szybko zrozumieć reguły gry. Celem tych opisów jest wyjaśnienie zasad całej, proceduralnie wygenerowanej gry szachopodobnej w sposób zrozumiały dla człowieka. Omawiana metoda może być również rozszerzona o inne cechy wariantów szachów, co mogłoby stanowić przedmiot przyszłych badań. Poza tym metoda ta może zostać wykorzystana jako ocena gry na etapie generowania. Pozwala to na tworzenie wysokiej jakości gier, które są atrakcyjne nie tylko ze względu na ciekawą rozgrywkę, ale także łatwe do wytłumaczenia zasady.

Eksperymenty uwzględniały opisanie zarówno niektórych znanych figur i gier stworzonych przez człowieka, jak i gier wygenerowanych proceduralnie. Były to m.in. szachy ortodoksyjne, szachy Tamerlane, Chess with Different Armies oraz dwa zbiory utworzonych komputerowo gier – jeden utworzony przy użyciu generatora bazującego na symulacjach, a drugi wykorzystujący metodę RAPP. Przedstawione zostało również powiązanie między właściwościami strategicznymi a oceną opisu danych gier. Ponadto, z tych zestawów gier wybrana i opisana została jedna przykładowa gra, której reguły są proste w zrozumieniu.

Jednym z możliwych udoskonaleń metody z rozdziału 3. jest obsługa trudniejszych przypadków, takich jak na przykład tzw. doubly-bent riders, czyli figur, których ruchy składają się przynajmniej z 3 części spójnych oraz pozostałych nieintuicyjnych bierek. Kolejnym usprawnieniem mogłoby być wykorzystanie innych, bardziej zaawansowanych od podejścia zachłannego, algorytmów do znajdowania najlepszych opisów figur, jak np. algorytm pokrycia zbiorów. To znacząco wpłynęłoby na czas działania algorytmu, ale prowadziłoby do lepszych wyników. Oprócz tego możliwym byłoby wprowadzenie poprawek gramatycznych wygenerowanych opisów, aby były one bardziej zwięzłe.

W drugiej części pracy opisana została metoda generowania kształtów figur szachopodobnych. Podstawą tego algorytmu są tradycyjne szachy i powiązanie pomiędzy funkcją bierek a ich kształtem. To pozwala oszacować docelowe kształty nowych figur, opierając się na ich roli w danej grze. Dobranie odpowiednich miar strategicznych i wizualnych pozwala na porównanie ze sobą figur z różnych gier. Dzięki temu możliwym jest wyznaczenie figur, które np. poruszają się wolniej od niektórych bierek w szachach, jednak porównując je do innych bierek z ich gry mogą okazać się najbardziej mobilne, przez co pełnią w nich całkowicie inną funkcję. Utworzone kształty zapisywane są w formacie skalowalnej grafiki wektorowej SVG, co pozwala na wykorzystanie tych figur w prawdziwych grach.

Celem metody z rozdziału 4. jest wyprodukowanie kształtów całej gry na tyle podobnych, aby możliwym było powiedzenie, że należą one do tej samej gry. Z drugiej strony powinny one różnić się od siebie w taki sposób, żeby każdy kształt odzwierciedlał rolę danej bierki w grze.

Osiągnięcie pierwszego celu miało zapewniać użycie różnych wariantów algorytmu ewolucyjnego. Warianty jedno- i dwuetapowe prowadzą do kształtów, które stają się bardzo podobne do siebie. Na każdym etapie z populacji usuwane są figury, których pola w znacznym stopniu się pokrywają, jednak mimo tego znaleźć można wiele przypadków, w których dla człowieka te kształty są niemal identyczne ze względu na pewne cechy wizualne. Trzeci wariant to ten, w którym każda figura przechodzi ewolucję z osobna. Prowadzi on do kształtów najlepiej pasujących do roli danej figury w rozgrywce, ale po zebraniu ich wszystkich razem nie wyglądają one na należące do tej samej gry i mogą być w łatwy sposób zamienione na kształty z innych gier. Utrzymanie równowagi pomiędzy zbyt dużym podobieństwem i zbyt dużą różnorodnością jest subiektywne i jedynie dwuetapowy wariant algorytmu przy odpowiednio dobranych parametrach potrafi zachować ten balans.

Trudno natomiast określić, czy drugi cel którym było odwzorowanie roli figury za pomocą jej kształtu został osiągnięty. Na podstawie samych reguł ciężko określić jakiego typu kształt powinna mieć dana bierka. Wykorzystanie szachów jako bazy pozwala na wychwycenie cech specyficznych dla tej gry. Na przykład figury, które potrzebne są do zwycięstwa często są większe, co jest właściwością króla w szachach. Jeżeli więc nowa bierka swoim zachowaniem bardzo przypomina jedną figurę szachową, łatwo jest uchwycić jej kształt. Jednak proces ewolucji uniejednoznacznia wygląd takich figur, które posiadają cechy kilku bierek szachowych. Dalsze eksperymenty nad innymi przedziałami podobieństwa (na podstawie rysunku 4.3) mogą pozytywnie wpłynąć na ocenę stopnia w jakim figury szachowe są podobne do nowej bierki oraz które z nich należy uwzględnić w dalszych obliczeniach, aby nie rozmywać specyficznych dla nich cech.

Jest to dopiero początek badań nad tematem generowania kształtów figur dla gier utworzonych proceduralnie. Pomimo tego, że nie wszystkie zestawy kształtów są na tyle atrakcyjne, by mogły być używane od razu przez graczy, to możliwe jest

już wybranie przez człowieka kilku dobrze wyglądających figur. Kolejne etapy pracy w tej dziedzinie obejmują dogłębniejszą analizę metryk wizualnych i strategicznych dla lepszego odwzorowania zależności pomiędzy różnymi grami. Oprócz tego można wykorzystać podobną metodę jak w pracy [Liapis, 2016], to znaczy tzw. *novelty search* szerzej opisaną w pracy [Lehman and Stanley, 2011]. Zmienia to podejście do problemu, skupiając się raczej na poszukiwaniu nowości i oryginalności, w przeciwieństwie do użycia standardowej ewolucji, starającej się zachować różnorodność figur. Następny element, który można usprawnić, to identyfikowanie spójnych ze sobą figur jednej gry tak, aby zachować ich różnorodność. Można wykorzystać do tego algorytm „głębokiego uczenia się" (ang. deep learning) podobnie jak w [Liapis et al., 2013]. Mógłby się on okazać skuteczny w ocenie podobieństwa dwóch figur, odchodząc od porównywania kształtów piksel po pikselu, jak dzieje się to teraz.

W połączeniu z poprzednimi osiągnięciami w dziedzinie generowania reguł gier szachopodobnych w pracy [Kowalski and Szykuła, 2016], moja praca może prowadzić do utworzenia systemu potrafiącego wygenerować na żądanie całkiem nową grę. Może też stworzyć do niej instrukcję w języku naturalnym i wykreować wszystkie obecne w niej elementy wizualne. Granie w takie gry mogłoby stać się pewnym odpowiednikiem problemu General Game Playing dla ludzi, pozwalając im mierzyć się w rozgrywkach wcześniej dla nich nieznanych, a tym samym otwierając przed nimi zupełnie nowe horyzonty.

Bibliografia

[Björnsson, 2012] Björnsson, Y. (2012). Learning Rules of Simplified Boardgames by Observing. In *European Conference on Artificial Intelligence*, volume 242 of *FAIA*, pages 175–180.

[Browne and Maire, 2010] Browne, C. and Maire, F. (2010). Evolutionary game design. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(1):1–16.

[Dawson, 1938] Dawson, T. R. (1938). *Caissa's wild roses in Clusters*. Thornton Heath.

[Dickins, 1971] Dickins, A. (1971). *A Guide to Fairy Chess*. Dover.

[Duniho, 2016] Duniho, F. (2016). The Chess Variant Pages. http://www.chessvariants.org/.

[Genesereth, 2014] Genesereth, M. (2014). Stanford University General Game Playing Coursera online course. `https://www.coursera.org/course/ggp`.

[Genesereth and Björnsson, 2013] Genesereth, M. and Björnsson, Y. (2013). The International General Game Playing Competition. *AI Magazine*, 34(2):107–111.

[Genesereth et al., 2005] Genesereth, M., Love, N., and Pell, B. (2005). General Game Playing: Overview of the AAAI Competition. *AI Magazine*, 26:62–72.

[Gherrity, 1993] Gherrity, M. (1993). *A Game-Learning Machine*. PhD thesis, University of California, San Diego.

[Karavolos et al., 2017] Karavolos, D., Liapis, A., and Yannakakis, G. N. (2017). Learning the patterns of balance in a multi-player shooter game. In *FDG workshop on Procedural Content Generation in Games*.

[Khalifa et al., 2017] Khalifa, A., Green, M., Perez, D., and Togelius, J. (2017). General Video Game Rule Generation. In *IEEE Conference on Computational Intelligence and Games*.

[Khalifa et al., 2016] Khalifa, A., Perez, D., Lucas, S., and Togelius, J. (2016). General Video Game Level Generation. In *Genetic and Evolutionary Computation Conference*.

[Kowalski and Kisielewicz, 2015] Kowalski, J. and Kisielewicz, A. (2015). Testing General Game Players Against a Simplified Boardgames Player Using Temporal-difference Learning. In *IEEE Congress on Evolutionary Computation*, pages 1466–1473.

[Kowalski et al., 2018] Kowalski, J., Liapis, A., and Żarczyński, Ł. (2018). Mapping Chess Aesthetics onto Procedurally Generated Chess-Like Games. In *EvoApplications 2018: Applications of Evolutionary Computation*, volume 10784 of *LNCS*, pages 325–341.

[Kowalski et al., 2016] Kowalski, J., Sutowicz, J., and Szykuła, M. (2016). Simplified Boardgames. arXiv:1606.02645 [cs.AI].

[Kowalski and Szykuła, 2015] Kowalski, J. and Szykuła, M. (2015). Procedural Content Generation for GDL Descriptions of Simplified Boardgames. arXiv:1108.1494 [cs.AI].

[Kowalski and Szykuła, 2016] Kowalski, J. and Szykuła, M. (2016). Evolving Chesslike Games Using Relative Algorithm Performance Profiles. In *Applications of Evolutionary Computation*, volume 9597 of *LNCS*, pages 574–589.

[Kowalski et al., 2017] Kowalski, J., Żarczyński, Ł., and Kisielewicz, A. (2017). Evaluating Chess-like Games Using Generated Natural Language Descriptions. In *ACG 2017: Advances in Computer Games*, volume 10664 of *LNCS*, pages 127–139.

[Lehman and Stanley, 2011] Lehman, J. and Stanley, K. O. (2011). Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, 19(2).

[Liapis, 2016] Liapis, A. (2016). Exploring the visual styles of arcade game assets. In *Evolutionary and Biologically Inspired Music, Sound, Art and Design (EvoMusArt)*. Springer.

[Liapis et al., 2013] Liapis, A., Martínez, H. P., Togelius, J., and Yannakakis, G. N. (2013). Transforming exploratory creativity with DeLeNoX. In *International Conference on Computational Creativity*.

[Liapis et al., 2012] Liapis, A., Yannakakis, G. N., and Togelius, J. (2012). Adapting models of visual aesthetics for personalized content creation. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(3):213–228.

[Lopes et al., 2015] Lopes, P., Liapis, A., and Yannakakis, G. N. (2015). Targeting horror via level and soundscape generation. In *AAAI Artificial Intelligence for Interactive Digital Entertainment Conference*.

[Nielsen et al., 2015a] Nielsen, T. S., Barros, G. A. B., Togelius, J., and Nelson, M. J. (2015a). General Video Game Evaluation Using Relative Algorithm Performance

Profiles. In *Applications of Evolutionary Computation*, volume 9028 of *LNCS*, pages 369–380.

[Nielsen et al., 2015b] Nielsen, T. S., Barros, G. A. B., Togelius, J., and Nelson, M. J. (2015b). Towards generating arcade game rules with VGDL. In *IEEE Conference on Computational Intelligence and Games*, pages 185–192.

[Pell, 1992a] Pell, B. (1992a). METAGAME: A New Challenge for Games and Learning. In *Heuristic Programming in Artificial Intelligence: The Third Computer Olympiad.*

[Pell, 1992b] Pell, B. (1992b). METAGAME in Symmetric Chess-Like Games. In *Heuristic Programming in Artificial Intelligence: The Third Computer Olympiad.*

[Pell, 1993] Pell, B. (1993). *Strategy Generation and Evaluation for Meta-Game Playing.* PhD thesis, Computer Laboratory, University of Cambridge.

[Perez et al., 2015] Perez, D., Samothrakis, S., Togelius, J., Schaul, T., Lucas, S., Couëtoux, A., Lee, J., Lim, C., and Thompson, T. (2015). The 2014 General Video Game Playing Competition. *IEEE Transactions on Computational Intelligence and AI in Games.*

[Pitrat, 1968] Pitrat, J. (1968). Realization of a general game-playing program. In *IFIP Congress*, pages 1570–1574.

[Pitrat, 1971] Pitrat, J. (1971). A general game playing program. In *Artificial Intelligence and Heuristic Programming*, pages 125–155.

[Wikipedia, 2017a] Wikipedia (2017a). Chess with different armies — Wikipedia, The Free Encyclopedia. `https://en.wikipedia.org/wiki/Chess_with_different_armies`.

[Wikipedia, 2017b] Wikipedia (2017b). Fairy chess piece — Wikipedia, The Free Encyclopedia. `https://en.wikipedia.org/wiki/Fairy_chess_piece`.