Politechnika Warszawska

WYDZIAŁ MATEMATYKI
I NAUK INFORMACYJNYCH

Praca dyplomowa magisterska

na kierunku Informatyka

w specjalności Metody sztucznej inteligencji

Zestawienie technik uczenia maszynowego i heurystyk zastosowanych do proceduralnego generowania poziomów w grze Sokoban

Patryk Fijałkowski

Numer albumu 286350

promotor

dr inż. Maciej Bartoszuk

WARSZAWA 2021

....

podpis promotora   podpis autora

Streszczenie

Zestawienie technik uczenia maszynowego i heurystyk zastosowanych do proceduralnego generowania poziomów w grze Sokoban

Celem pracy jest zaproponowanie i wszechstronne porównanie czterech technik, które zostaną zastosowane do proceduralnego generowania poziomów w grze Sokoban.

W pracy zostanie sprawdzone, jak konfigurowalne są poszczególne metody pod kątem trudności i rozmiaru poziomu. Zostaną zbadane czasy uczenia w przypadku uczenia głębokiego i algorytmów genetycznych oraz czasy generowania poziomów przez każdą z testowanych metod. Przy użyciu odpowiednich metryk zostanie również określone, jak skomplikowane poziomy mogą zostać wygenerowane przez prezentowane techniki. Sokoban jest prostą grą łamigłówkową, w której gracz ma za zadanie przemieścić przedmioty w wyznaczone miejsca na planszy złożonej z układu kwadratów. Trudność gry jest zależna od rozmieszczenia przedmiotów i przeszkód na planszy (poziomie). Zastosowanie proceduralnego generowania poziomów ma na celu odciążyć twórcę gry z wymyślania ich. Ponadto, generowanie proceduralne może potencjalnie tworzyć poziomy znacznie bardziej zawiłe niż człowiek.

Większość opisywanych w literaturze technik wymaga zebrania pewnej liczby gotowych plansz, opracowania odpowiedniej metody przechowywania najważniejszych elementów takich plansz w formie cech, a następnie tak spreparowane dane zasilają techniki uczenia maszynowego, które następnie są w stanie nauczyć się, jakimi właściwościami powinna się cechować dobrze zaprojektowana plansza i generować je samemu.

Poza technikami z działu uczenia maszynowego zostanie również zaprezentowana heurystyka bazująca na wiedzy eksperckiej, co pozwoli na zestawienie podejścia heurystycznego z uczeniem maszynowym. W ramach pracy zostanie przygotwana implementacja gry w silniku do gier Unity, co pozwoli na praktyczne testowanie generowanych poziomów.

Słowa kluczowe: Generowanie proceduralne, sztuczna inteligencja, heurystyka, Sokoban

Abstract

Comparison of machine learning techniques and heuristics used for procedural level generation in Sokoban game

TODO

Keywords: Procedural content generation, artificial intelligence, heuristics, Sokoban

Warszawa, dnia .....

Oświadczenie

Oświadczam, że pracę magisterską pod tytułem „Zestawienie technik uczenia maszynowego i heurystyk zastosowanych do proceduralnego generowania poziomów w grze Sokoban", której promotorem jest mgr inż. Maciej Bartoszuk, wykonałem samodzielnie, co poświadczam własnoręcznym podpisem.

.....

Spis treści

Wykaz najważniejszych oznaczeń i skrótów

- **MCTS** – Monte Carlo Tree Search
- **UCT** – Upper Confidence Bound Applied to Trees

1. Wstęp

Przemysł komputerowych gier wideo jest stale rozwijającym się sektorem gospodarki.

1.1. Proceduralne generowanie zawartości w grach

Generowanie proceduralne to tworzenie zawartości przy wykorzystaniu algorytmów. Zamiast tworzyć zawartość, można to zadanie zlecić wyspecjalizowanym metodom, które są w stanie stworzyć zawartość wyższej jakości niż ludzie.

Mówi się o zastosowaniu generowania proceduralnego w grach różnych gatunków. Jest to technika stosowana powszechnie, mająca swoje początki w roku 1978, wraz z wydaniem *Beneath Apple Manor*. Podczas tworzenia gier, zatrudniani są ludzie odpowiedzialni za tworzenie grafik, animacji czy projektowania poziomów. Zamiast tego, można posiłkować się odpowiednimi technikami do wygenerowania potrzebnego rodzaju zawartości.

Koncept proceduralnego generowania zawartości jest odciążeniem twórców gier. Coraz powszechniejsze staje się wykorzystywanie technik uczenia maszynowego w tym celu, co wykazano w rozdziale 1.1.1. Istnieją jednak pewne obostrzenia z tym związane, na przykład czasowe. Większość dzisiejszych gier zapewnia graczom wysokie odświeżanie ekranu. Za standard uznaje się gry generujące 120 klatek na sekundę. To oznacza, że na obliczenia związane z każdą klatką, można przeznaczyć co najwyżej 8 milisekund. Większość prezentowanych metod, korzystających z uczenia maszynowego, nie jest w stanie w tak krótkim czasie generować dobrej jakości zawartości.

1.1.1. Przykłady

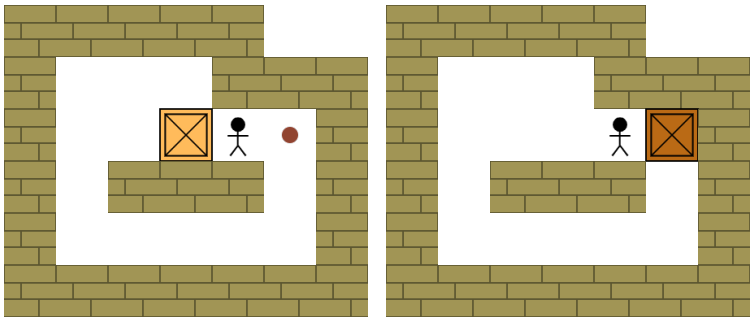
TODO - samotworząca się gra [12]

TODO - spelunky – wysokiej jakości heurystyka [15]

TODO - mario GAN [16]

1.2. Sokoban

Sokoban jest grą logiczną, rozgrywaną na planszy złożonej z kwadratów. Celem gracza jest przesunięcie wszystkich pudeł w wyznaczone kwadraty docelowe, tak jak na rysunku 1.1(b). Zaczynając z planszą jak na rysunku 1.1(a), gracz musi wykonać co najmniej 8 ruchów, w tym 4 pchnięcia pudeł. Jednak istnieje też rozwiązanie tej planszy, wykorzystujące 12 ruchów, w tym 2 pchnięcia. Różne implementacje gry uznają liczbę ruchów albo liczbę pchnięć jako ocenę rozwiązania.



(a) Plansza początkowa (b) Plansza końcowa

Rysunek 1.1: Przykładowa plansza Sokoban

Oryginalnie Sokoban jest grą wideo autorstwa Hiroyuki Imabayashi, wydaną w 1982 roku na platformę PC-88. Od tamtego czasu koncepcja gry jest rozwijana i implementowana w innych grach, takich jak Pokémon Emerald czy Grand Theft Auto: San Andreas. Ponadto, użytkownicy tworzą coraz bardziej skomplikowane plansze, czego dowodem są zbiory wymienione w tabeli 1.1.

Nazwa zbioru	**Liczba poziomów**
Sven	1911
Sasquatch I-VII	350
Sokoban Perfect	306
Sokoban Revenge	306
Aymeric	282
SokHard	163

Tabela 1.1: Zbiory plansz Sokoban, źródło: http://sokobano.de

1.2.1. Rozwiązanie

Zgodnie z [13], problem podania rozwiązania czy choćby ustalenia, czy zadana plansza Sokoban jest rozwiązywalna, jest problemem PSPACE-zupełnym. Drzewo decyzyjne odpowiadające rozgrywce, mimo swojego niskiego współczynnika rozgałęzienia (ang. *braching factor*), jest wysokie. W związku z tym, powstają heurystyczne solvery, które potrafią rozwiązywać plansze Sokoban w czasie wielomianowym. Mimo to, odnosząc się do raportu prezentowanego na http://sokobano.de, nie powstał jeszcze solver, który rozwiązywałby poprawnie wszystkie plansze z badanego zbioru.

2. Prezentacja metod

W niniejszej pracy zdecydowano się zestawić cztery metody generowania poziomów Sokoban, pogrupowanych w tabeli 2.1. Dogłębna analiza metod proponowanych w literaturze dla problemu generowania poziomów Sokoban pozwoliła na dobór reprezentatywnych metod. Wybrano dwie będące heurystykami, które nie korzystają z technik uczenia maszynowego, oraz dwie – korzystające. Ponadto, dobrano dwie, które bazują na symulacji rozgrywki oraz dwie, które działają bez symulowania działań gracza.

	Heurystyka	**Uczenie maszynowe**
Symulacja rozgrywki	SYM	MCTS
Brak symulacji	PDB	QDCA (?)

Tabela 2.1: Analizowane metody

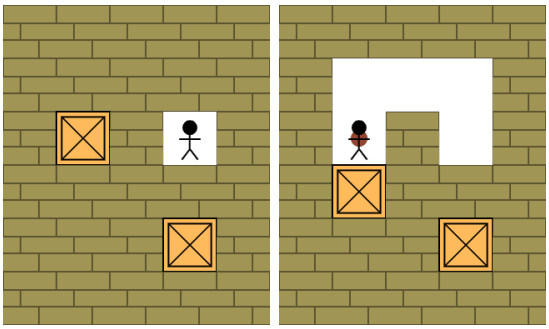
2.1. Metoda SYM

Metoda SYM, opisana w [1], opiera swoje działanie na symulowaniu zmodyfikowanej rozgrywki Sokoban.

2.1.1. Opis metody

Metoda SYM wykonuje ustaloną liczbę razy dwa rodzaje rozgrywek: najpierw rozgrywkę typu forward, a następnie – backward, by potem uporządkować generowaną planszę.

W rozgrywce forward, gracz wybiera w określony sposób pudło i kierunek jego przepychania, by dokonać przepchnięcia. W tej rozgrywce gracz może poruszać się po polach z przeszkodami, usuwając je. Przykładowo, jeśli dla planszy na rys. 2.1(a) gracz wybierze wyższe pudło i dół jako kierunek przepychania, to jego akcja spowoduje że plansza przybierze układ jak na rys. 2.1(b).



(a) Plansza przed wykonaniem akcji

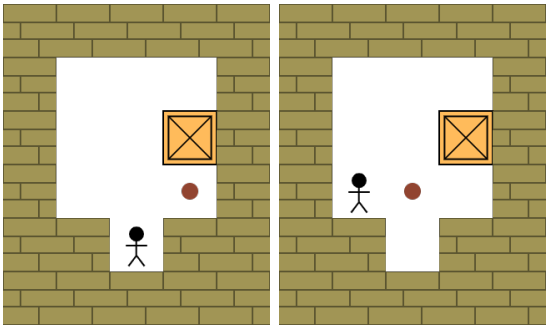
(b) Plansza po wykonaniu akcji

Rysunek 2.1: Działanie akcji w rozgrywce forward

W rozgrywce backward, gracz wybiera w określony sposób pole docelowe i kierunek jego przeciągania, by dokonać przeciągnięcia. W tej rozgrywce gracz przeciąga pola docelowe zamiast przepychać pudeł. Przykładowo, jeśli dla planszy na rys. 2.2(a) gracz wybierze jedyne dostępne pole docelowe i lewo jako kierunek przepychania, to jego akcja spowoduje że plansza przybierze układ jak na rys. 2.1(b).

2.1.2. Strategie

TODO



(a) Plansza przed wykonaniem akcji

(b) Plansza po wykonaniu akcji

Rysunek 2.2: Działanie akcji w rozgrywce backward

2.2. Metoda PDB

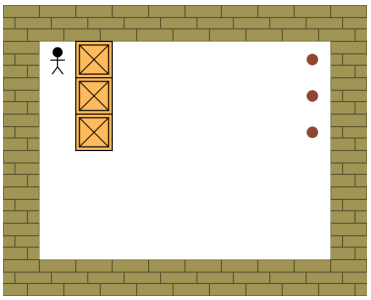
Autorzy [2] wprowadzają metodę PDB, bazując na teorii problemu przeszukiwań przestrzeni stanu (ang. *state-space search problem*) oraz baz danych wzorców (ang. *pattern databases*).

2.2.1. Problem przeszukiwań przestrzeni

Problemem przeszukiwań przestrzeni stanu nazywa się krotkę opisaną w równaniu 2.1. Składowe problemu to kolejno zbiór stanów S, stan początkowy s_0, zbiór stanów docelowych S^* oraz zbiór akcji A. Stan definiuje się jako pełne przypisanie wartości wszystkim zmiennym $v \in V$, przy zachowaniu dziedzin D_v. Rozwiązaniem problemu przeszukiwań przestrzeni jest ścieżka rozwiązująca (ang. *solution path*) – skończony ciąg akcji, po wykonaniu którego ze stanu s_0 otrzyma się stan ze zbioru S^*.

$$\mathcal{P} =< S, s_0, S^*, A > \tag{2.1}$$

Opisywany problem różni się od innych problemów przeszukiwania tym, że przestrzeń stanów jest niejawna (ang. *implicit*). W prezentowanym rozwiązaniu, graf przestrzeni stanów jest zbyt duży, aby można go było wygenerować i przechowywać w pamięci. Zamiast tego, jedynie interesujące węzły są generowane i robione jest to na bieżąco – w trakcie ich eksploracji. Stanami w metodzie PDB są przypisania lokalizacji k pudeł i gracza, a więc składają się z $k+1$ zmiennych. Dziedziną każdej zmiennej są puste pola. W związku z tym, liczba możliwych stanów opisana jest wzorem $(n-k)\binom{n-k-1}{k}$, gdzie n to liczba wolnych pól. Już dla mało skomplikowanej planszy przedstawionej na rys. 2.3 istnieje więc 595 980 możliwych stanów (wierzchołków grafu).



Rysunek 2.3: Przykładowa plansza o rozmiarze 8 x 10

Autorzy [2] definiują ponadto problem wygenerowania początkowego stanu (ang. *initial state generation task*) $\mathcal{P}_{-s_0}$, opisany krotką w równaniu 2.2. Jest on równoważny problemowi przeszukiwań przestrzeni ze stanem początkowym s_0, jednak celem nie jest wyznaczenie ścieżki, a

wskazanie stanu $s \in S$, dla którego problem $\mathcal{P}_{-s_0}$ z s jako stanem początkowym, jest rozwiązywalny.

$$\mathcal{P}_{-s_0} =< S, S^*, A > \tag{2.2}$$

2.2.2. Bazy danych wzorców

W celu rozwiązania problemu opisanego w 2.2.1, stosuje się heurystykę bazy danych wzorców, wprowadzoną przez Josepha Culbersona oraz Jonathana Schaeffera w [9]. Idea tej heurystyki oparta jest na zignorowaniu części problemu (podzbioru zmiennych V) i wyznaczenie dolnego ograniczenia długości ścieżki optymalnej dla każdego stanu uproszczonego problemu.

W celu zobrazowania działania heurystyki bazy danych wzorców, zaadaptowano przykład opisany w [10], który ukazano w układzie 2.3. W tym przykładzie stanem początkowym jest przyporządkowanie $(0, 0, 0)$, a docelowym – $(3, 3, 3)$. Zdefiniowane są dwa rodzaje akcji - *inc* oraz *jump*. Pierwsza z nich inkrementuje wybraną zmienną, a druga ustawia wartość wybranej zmiennej na 3, pod warunkiem że wszystkie pozostałe mają wartość 4. Zgodnie z rozwiązaniem przedstawionym w [10], długość ścieżki optymalnej to 9. Istnieje dokładnie $9! = 362\,880$ ścieżek optymalnych i każda składa się z dziewięciu akcji *inc*. Co więcej, nie istnieje ścieżka rozwiązująca zawierająca akcji *jump*, ponieważ wykonanie jej związane byłoby ze stanem w którym dwie zmienne mają wartość 4, a z takiego stanu nie da się uzyskać s_*.

Niech dla analizowanego przykładu będą dane dwa wzorce: $\{v_1\}$ oraz $\{v_2, v_3\}$. Wtedy wartość bazy danych dla wzorca $\{v_1\}$ i stanu s_0 wynosi 1, ponieważ dla uproszczonego problemu wystarczy wykonać jedną akcję *jump*, by uzyskać stan docelowy. Z kolei dla drugiego wzorca wartość bazy danych w stanie początkowym to 6, gdyż dla uproszczonego problemu należy wykonać sześć akcji *inc*, by uzyskać stan docelowy. Jako że przykładowe wzorce są addytywne, zgodnie z [11], można przyjąć $1 + 6 = 7$ jako dolne ograniczenie długości ścieżki optymalnej badanego przykładu.

$$\begin{aligned}
\mathcal{V} &= \{v_1, v_2, v_3\}, \\
\forall_{v_i \in \mathcal{V}}\ D_{v_i} &= \{0, 1, 2, 3, 4\}, \\
s_0 &= \{v_1 = 0, v_2 = 0, v_3 = 0\}, \\
s_* &= \{v_1 = 3, v_2 = 3, v_3 = 3\}, \\
A &= \{inc_x^v \mid v \in \mathcal{V}, x \in \{0, 1, 2\}\} \cup \{jump^v \mid v \in \mathcal{V}\}, \\
& inc_x^{v_i} = \langle v_i = x, v_i = x + 1\rangle, \\
& jump_{v_i} = \langle \mathcal{V} - v_i = 4 \wedge v_i = 0, v_i = 3\rangle
\end{aligned} \tag{2.3}$$

2.2.3. Opis metody

Metoda PDB, nazywana przez autorów [2] algorytmem β, rozwiązuje problem opisany wzorem 2.2 w oparciu o podejście zachłanne. W tym celu wykorzystywane jest przeszukiwanie grafu wstecz, które priorytetyzuje węzły o najbardziej obiecującej wartości heurystyki (ang. *best-first search*). Graf stanów jest przeszukiwany wstecz, od wierzchołków odpowiadającym stanom docelowym, co gwarantuje istnienie ścieżki docelowej. Autorzy [2] podkreślają, że przeszukiwanie grafu od wierzchołka początkowego s_p wymagałoby zapewnienia o istnieniu ścieżki docelowej rozpoczynającej się w s_p, co jest wymagające obliczeniowo.

Podczas eksploracji, algorytm korzysta z funkcji nowości (ang. *novelty function*), opisanej w [14].

$$novelty(s) \tag{2.4}$$

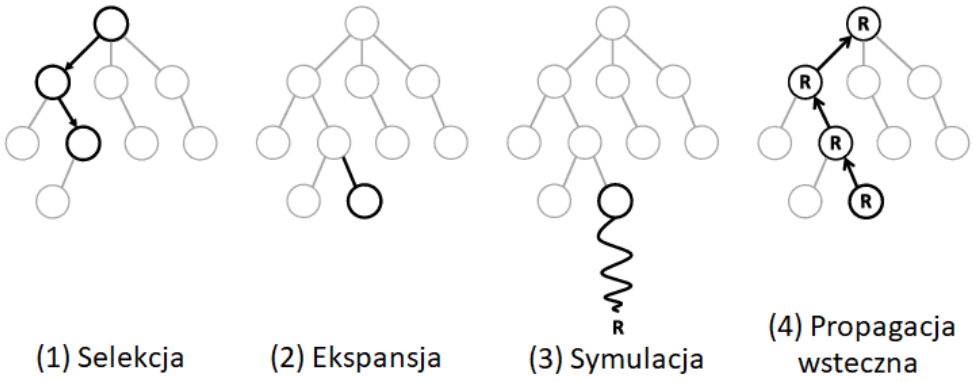
TODO: Funkcja nowości

2.3. Metoda MCTS

Autorzy [3] wprowadzają metodę MCTS jako autorską adaptację heurystyki *Monte-Carlo Tree Search*.

2.3.1. Wprowadzenie

Monte-Carlo Tree Search jest heurystyką, której celem jest podejmowanie decyzji w pewnych zadaniach sztucznej inteligencji. Metoda opiera swoje działanie na przeszukiwaniu możliwych stanów zapisanych w wierzchołkach drzewa i losowym symulowaniu rozgrywek. Algorytmy z grupy MCTS opierają się na iteracyjnym rozbudowywaniu drzewa stanów przez sekwencyjne wykonanie czterech faz – selekcji, ekspansji, symulacji i propagacji wstecznej. Poszczególne fazy zostały zobrazowane na rys. 2.4.



Rysunek 2.4: Fazy MCTS, źródło: [5]

Faza selekcji w heurystyce *Monte-Carlo Tree Search* pozostawia dowolność w wyborze liścia, który będzie eksploatowany w kolejnych fazach. Oczywiście sposób wybierania liści w kolejnych iteracjach jest krytyczny z punktu widzenia eksploracji drzewa i działania metody. W literaturze opisywane są różne podejścia, przykładowo UCB_Minimal w [6], czy UCB-V w [7]. Najbardziej powszechny i użyty w metodzie MCTS jest jednak wariant UCT, opisany w [8]. Ten wariant stara się zachować równowagę między eksploatacją bardziej obiecujących ruchów a eksploracją tych rzadko odwiedzonych. Formuła, która odpowiada za wyznaczenie najbardziej obiecującego wierzchołka w fazie wyboru MCTS jest przedstawiona jako wyrażenie (2.5). Indeks i odnosi się do liczby wykonanych przez algorytm iteracji, czyli czterech faz MCTS. W pierwszym składniku sumy wyrażenia (2.5), licznik w_i oznacza sumę wszystkich wypłat w danym węźle, a mianownik n_i oznacza liczbę rozegranych symulacji. Parametr eksploracji c, może być dostosowany do badanego problemu. Odwołując się do [8], dla problemu z wypłatami w przedziale $[-1, 1]$, optymalnie jest przyjąć $c = \sqrt{2}$.

$$\frac{w_i}{n_i} + c\sqrt{\frac{\ln\left(\sum_i n_i\right)}{n_i}} \tag{2.5}$$

2.3.2. Opis metody

Metoda MCTS opiera swoje działanie na symulowaniu zmodyfikowanej rozgrywki Sokoban, która składa się z dwóch faz. W pierwszej gracz ma do wyboru trzy akcje: może usuwać przeszkody graniczące z pustymi polami, stawiać pudła lub zamrozić planszę. Algorytm zaczyna pracę z planszą w pełni wypełnioną przeszkodami, więc zadaniem fazy pierwszej jest przygotowanie pustych pól. W wyniku akcji zamrożenia, rozgrywka przechodzi do fazy drugiej, w której gracz prowadzi normalną rozgrywkę, poruszając się i przesuwając pudła, do momentu aż zdecyduje się ją zewaluować. Ewaluacja jest akcją, która kończy rozgrywkę i dokonuje dodatkowych procesów czyszczenia w celu uzyskania maksymalnie dobrej i poprawnej planszy.

Kluczowe dla metody MCTS jest dobranie funkcji oceniającej jakość generowanych poziomów. Jako że *Monte-Carlo Tree Search* opiera swoje działanie na wielokrotnym symulowaniu rozgrywek, wyznaczanie wypłaty dla danego poziomu musi być mało kosztowne obliczeniowo. Autorzy [3] zdecydowali się na funkcję opisaną wzorem 2.6, która jest sumą ważoną trzech metryk, podzieloną przez stałą z, dla znormalizowania wyniku. Wyrazy w_b, w_c i w_n są wagami dla odpowiednich metryk, opisanych poniżej.

1. Metryka pudeł 3 x 3 P_b - liczba pól, która nie znajduje się w bloku 3 x 3 przeszkód lub pustych pól. Zadaniem tej metryki jest nagrodzenie plansz, które nie zawierają bloków 3 x 3 tych samych pól, gdyż tego typu czynią plansze mniej skomplikowanymi.

2. Metryka zagęszczenia P_c - funkcja nagradzająca plansze o długich ścieżkach między pudłami a ich docelowymi punktami.

3. Metryka pudeł P_n - liczba pudeł na planszy.

$$\frac{w_b P_b + w_c P_c + w_n P_n}{z} \tag{2.6}$$

2.4. Metoda QDCA(?)

TODO - [4]

2.5. Hiperparametry

TODO - opisać

Metoda	Parametry problemu	Parametry czasu	Parametry pracy
SYM	size, max box	f-it, b-it	boxs, pullds, pushds, routegen
PDB	instance	time, runs	heur
MCTS	size, max box	it	heur
QDCA	?	?	?

Tabela 2.2: Hiperparametry analizowanych metod

3. Zastosowane metryki

TODO

4. Wyniki eksperymentów

TODO

Bibliografia

[1] Wu Yueyang, *An Efficient Approach of Sokoban Level Generation*, Graduate School of Computer and Information Science, Hosei University, Japan, 2020.

[2] Damaris S. Bento, Andre G. Pereira and Levi H. S. Lelis, *Procedural Generation of Initial States of Sokoban*, Federal University of Rio Grande do Sul, Federal University of Viçosa, Brazil, 2019.

[3] Bilal Kartal, Nick Sohre, and Stephen J. Guy, *Data-Driven Sokoban Puzzle Generation with Monte Carlo Tree Search*, University of Minnesota, United States, 2016.

[4] Sam Earle , Justin Snider, Matthew C. Fontaine, Stefanos Nikolaidis, Julian Togelius, *Illuminating Diverse Neural Cellular Automata for Level Generation*, New York University, United States, 2021.

[5] Steven James, George Konidaris, Benjamin Rosman, *An Analysis of Monte Carlo Tree Search*, University of the Witwatersrand, Johannesburg, South Africa.

[6] Francis Maes, Louis Wehenkel, Damien Ernst, *Automatic Discovery of Ranking Formulas for Playing with Multi-armed Bandits*, European Workshop on Reinforcement Learning, Athens, Greece, September 9–11, 2011.

[7] Jean-Yves Audibert, Remi Munos, Csaba Szepesvári, *Tuning Bandit Algorithms in Stochastic Environments*, Algorithmic Learning Theory 18th International Conference, Sendai, Japan, October 1–4, 2007.

[8] Levente Kocsis, Csaba Szepesvári, *Bandit based Monte-Carlo Planning*, European Conference on Machine Learning, Berlin, Germany, September 18–22, 2006.

[9] Joseph C. Culberson, Jonathan Schaeffer, *Searching with pattern databases*, Canadian Conference on Artificial Intelligence, pages 402—416, 1996.

[10] Florian Pommerening, Gabriele Roger, Malte Helmert, *Getting the most out of pattern databases for classical planning*, International Joint Conference on Artificial Intelligence, 2013.

[11] Ariel Felner, Richard E. Korf, Sarit Hanan, *Additive pattern database heuristics*, Journal of Artificial Intelligence Research, 22:279-–318, 2004.

[12] Erin J. Hastings, Ratan K. Guha, Kenneth O. Stanley, *Automatic Content Generation in the Galactic Arms Race Video Game*, IEEE Transactions on Computational Intelligence and AI in Games 1(4):245–263, 2010.

[13] Joseph C. Culberson, *Sokoban is PSPACE-complete*, 1997.

[14] Nir Lipovetzky, Hector Geffner, *Best-first width search: Exploration and exploitation in classical planning*, AAAI Conference on Artificial Intelligence, 3590—3596, 2017.

[15] Walaa Baghdadi, Fawzya Shams Eddin, Rawan Al-Omari, Zeina Alhalawani, Mohammad Shaker, Noor Shaker *A Procedural Method for Automatic Generation of Spelunky Levels*, European Conference on the Applications of Evolutionary Computation, 2015.

[16] Vanessa Volz, Jacob Schrum, Jialin Liu, Simon M. Lucas, Adam Smith, Sebastian Risi, *Evolving Mario Levels in the Latent Space of a Deep Convolutional Generative Adversarial Network*, 2018.

Spis rysunków

Spis tabel

Spis załączników

1. Płyta CD zawierająca:
    - dokument z treścią pracy dyplomowej,
    - streszczenie w języku polskim,
    - streszczenie w języku angielskim,
    - kod programów.