BAB 7

KESIMPULAN DAN SARAN

Bab ini akan membahas kesimpulan dari skripsi ini berdasarkan tujuan penelitian yang sudah ditetapkan. Lalu, bab ini akan diakhiri dengan saran-saran untuk penelitian lanjutan yang dapat dilakukan ke depannya.

7.1 Kesimpulan

Sesuai penjabaran pada Bagian 1.3, berikut adalah tujuan dan hasil pencapaian tujuan dari penelitian ini:

1. Membangun perangkat lunak permainan *2048*—menggunakan bahasa Java—yang dapat diintegrasikan dengan agen permainan:
   Mekanisme permainan *2048* telah dipelajari dengan memainkan permainan tersebut dan melakukan studi pustaka yang hasilnya terdapat pada Bagian 2.3. Selanjutnya, telah dilakukan analisis terhadap permainan tersebut pada Bagian 3.1 untuk menentukan pemodelan komputasional dari permainan 2048. Kemudian, telah dilakukan analisis dan perancangan pada perangkat lunak yang akan dibangun pada Bagian 3.6 dan Bab 4. Terakhir, perangkat lunak sudah diimplementasikan dengan bahasa Java dan diuji pada Bab 5 hingga menghasilkan perangkat lunak permainan *2048* yang sekaligus dapat dipakai untuk menguji performa agen permainan yang sudah berjalan dengan baik.
2. Merancang agen permainan *2048* yang menggunakan algoritma *Monte Carlo Tree Search* (MCTS) dan *Temporal Difference Tree Search* (TDTS), lalu mengimplementasikannya dengan bahasa Java:
   Mula-mula, telah dilakukan studi pustaka secara ekstensif mengenai algoritma MCTS dan TDTS yang menghasilkan landasan teori pada Bagian 2.4, Bagian 2.5, Bagian 2.6, dan Bagian 2.7. Selanjutnya, telah dilakukan analisis terhadap algoritma MCTS dan TDTS pada Bagian 3.2, Bagian 3.3, Bagian 3.4, dan Bagian 3.5 untuk menyesuaikan algoritmanya dengan permainan *2048*. Secara spesifik, varian algoritma MCTS yang dipakai adalah algoritma UCT (*upper confidence bounds applied to trees*), sedangkan varian algoritma TDTS yang dipakai adalah algoritma Sarsa-UCT(λ). Kemudian, telah dilakukan perancangan kelas untuk mengimplementasikan agen dengan kedua algoritma tersebut, yang hasilnya terlihat pada Subbagian 4.3.1. Bagian 5.3 menunjukkan hasil implementasi dari kedua algoritma tersebut dengan bahasa Java sedangkan Subbagian 5.4.1 dan Subbagian 5.4.2 menunjukkan proses pengujian yang sudah dilakukan untuk memastikan ketepatan implementasi algoritma.
3. Membandingkan performa agen permainan dengan algoritma TDTS dengan agen permainan yang menggunakan algoritma MCTS yang standar:
   Pada Bab 6, telah dilakukan berbagai eksperimen untuk membandingkan performa agen dengan algoritma UCT sebagai representasi dari MCTS dan algoritma Sarsa-UCT(λ) sebagai representasi dari TDTS. Pertama-tama, dilakukan eksperimen terhadap GPA *Random* yang selalu memilih aksi secara acak untuk mendapatkan tolak ukur dari performa agen. Lalu, telah dilakukan eksperimen-eksperimen untuk mencari nilai-nilai konfigurasi parameter algo-

ritma UCT dan Sarsa-UCT(λ) yang optimal untuk permainan *2048*. Berikut adalah daftar eksperimen yang dilakukan dengan kesimpulan yang dapat diambil dari setiap eksperimen.

- Eksperimen *best-child policy* pada Bagian 6.3.2 menunjukkan bahwa pada permainan *2048*, mekanisme *max child* menghasilkan performa yang lebih baik daripada mekanisme *robust child*, baik pada agen UCT maupun Sarsa-UCT(λ). Hal ini nampaknya disebabkan oleh penggunaan teknik normalisasi *space-local value normalization* yang melakukan normalisasi dengan batas-batas lokal. Akibatnya, bisa saja ada aksi yang memiliki estimasi utilitas relatif tinggi, namun nilai normalisasi yang relatif rendah sehingga memiliki jumlah kunjungan yang banyak selama proses pembelajaran.
- Eksperimen konstanta eksplorasi (C) pada Bagian 6.3.3 menunjukkan bahwa nilai C optimum memiliki korelasi positif dengan jumlah langkah waktu. Dengan jumlah langkah waktu yang sedikit, nilai C yang lebih kecil menghasilkan rata-rata skor yang lebih tinggi. Sebaliknya, dengan jumlah langkah waktu yang banyak, nilai C yang lebih besar menghasilkan rata-rata skor yang lebih tinggi. Diputuskan bahwa nilai C optimum untuk agen UCT dan Sarsa-UCT(λ) secara berturutan adalah $2{,}0 \times \sqrt{2} \approx 2{,}83$ dan $2{,}25 \times \sqrt{2} \approx 3{,}18$.
- Eksperimen *reward discount rate* (γ) pada Bagian 6.3.4 menunjukkan bahwa nilai γ optimum memiliki korelasi positif dengan jumlah langkah waktu. Dengan jumlah langkah waktu yang sedikit, nilai γ optimum menjadi lebih kecil dan sebaliknya dengan jumlah waktu yang banyak, nilai γ yang optimal cenderung membesar mendekati angka 1. Diputuskan bahwa nilai γ optimum untuk agen Sarsa-UCT(λ) adalah 0,99.
- Eksperimen *eligibility trace decay rate* (λ) pada Bagian 6.3.5 menunjukkan bahwa nilai λ optimum memiliki korelasi positif dengan jumlah langkah waktu. Dengan jumlah langkah waktu yang sedikit, nilai λ optimum menjadi lebih kecil yang lebih kecil dan sebaliknya dengan jumlah waktu yang banyak, nilai λ yang optimal cenderung membesar mendekati angka 1. Diputuskan bahwa nilai λ optimum untuk agen Sarsa-UCT(λ) adalah 1.

Setelah semua eksperimen dilakukan, dilakukan analisis terhadap performa agen UCT dan Sarsa-UCT(λ) dengan konfigurasi terbaik berdasarkan hasil eksperimen. Hasilnya, ditemukan bahwa algoritma Sarsa-UCT(λ) sebagai algoritma konkret dari metode TDTS memiliki performa yang setidaknya sama baiknya dengan algoritma UCT yang merupakan algoritma konkret dari metode MCTS. Pada jumlah langkah waktu yang sangat sedikit, agen Sarsa-UCT(λ) memiliki performa yang lebih baik daripada agen UCT. Namun seiring bertambahnya jumlah langkah waktu, performa agen UCT terus membaik hingga tidak berbeda secara signifikan dengan agen Sarsa-UCT(λ).

Agen UCT dan Sarsa-UCT(λ) sama-sama memiliki standar deviasi skor yang cukup besar. Hal ini disebabkan oleh sifat permainan *2048* itu sendiri, di mana untuk membentuk suatu *tile* dengan nilai besar, diperlukan sejumlah *tile* lain dengan nilai yang lebih kecil. Pada kisaran skor tertentu (misalnya 30000–40000), agen biasanya sedang mencoba membentuk *tile* dengan nilai yang besar (misalnya 4096) dan papan permainan cenderung terisi penuh. Akibatnya, resiko kekalahan menjadi lebih besar dan banyak permainan yang berakhir pada kisaran skor tersebut. Namun jika agen berhasil membentuk *tile* besar tersebut, maka jumlah *tile* dalam papan permainan akan berkurang secara drastis dan agen akan mampu mencapai nilai yang jauh lebih besar setelah melewati tahap tersebut. Secara umum, agen Sarsa-UCT(λ) memiliki standar deviasi yang lebih besar daripada agen UCT, namun dengan rata-rata skor yang tidak berbeda jauh. Artinya, agen Sarsa-UCT(λ) lebih sering mencapai skor permainan yang cukup tinggi, namun juga lebih sering mencapai skor yang relatif rendah daripada agen UCT.

7.2 Saran

Penelitian yang sudah dilakukan masih terbatas dan dapat terus dikembangkan. Berikut adalah saran-saran yang diusulkan untuk melakukan penelitian lanjutan pada topik ini:

1. Meneliti performa *game-playing agent* pada varian permainan *2048* lain, seperti *2048-3D*[1].
2. Mencoba algoritma permainan yang belum pernah diteliti pada permainan *2048*. Contohnya adalah algoritma Iterated Width yang turut diteliti oleh Soemers [9]. Contoh lainnya adalah algoritma-algoritma konkret lain dari metode MCTS, seperti yang disurvei oleh Browne dkk. [8].
3. Menggunakan algoritma MCTS dan TDTS yang bersifat paralel untuk memainkan *2048*.
4. Menggunakan pemodelan permainan yang lain untuk permainan *2048*. Pada skripsi ini, pemodelan yang digunakan adalah pohon. Contoh pemodelan lain yang bisa digunakan adalah *directed acyclic graph* (DAG). Pada DAG, setiap *state* hanya memiliki satu representasi simpul pada graf (dengan memanfaatkan transposisi) dan bisa dicapai melalui beberapa jalur dari simpul akar.
5. Menggunakan konsep *tree reuse* pada algoritma agen, seperti yang disebutkan pada Subbagian 2.4.1. Skripsi ini menggunakan konsep *forgetting* yang berlawanan dengan *tree reuse*.
6. Menambahkan fungsi heuristik dan pengetahuan domain untuk agen UCT atau Sarsa-UCT(λ) pada permainan *2048*. Pada algoritma Saras-UCT(λ), hal ini sama dengan mengimplementasikan fungsi nilai V_{init} dan V_{playout}.
7. Meneliti pengaruh pemakaian teknik normalisasi *space-local value normalization* pada algoritma UCT dan Sarsa-UCT(λ) dan perbandingannya dengan teknik-teknik normalisasi lainnya pada permainan *2048*.
8. Membangun perangkat lunak pengujian yang memiliki antarmuka pengujian yang berbentuk *command-line interface* dan *graphical user interface* yang dapat melakukan konfigurasi otomatisasi banyak eksperimen sekaligus. Sebagai fitur tambahan, perangkat lunak juga dapat menampilkan progres pengujian dan estimasi sisa waktu pengujian.
9. Membuat fitur visualisasi permainan pada perangkat lunak untuk menampilkan jalannya permainan yang dilakukan oleh GPA.
10. Membuat fitur pada perangkat lunak untuk menampilkan kembali riwayat permainan agen berdasarkan masukan berkas pencatatan pengujian.
11. Menguji performa agen UCT dan Sarsa-UCT(λ) pada lebih banyak variasi jumlah langkah waktu pada kisaran nilai yang sangat kecil atau sangat besar.

[1]Michal Opler, *2048-3D*, https://joppi.github.io/2048-3D/, diakses pada 04 Juni 2022 pukul 07:33 WIB.

DAFTAR REFERENSI

[1] Russell, S. J. dan Norvig, P. (2020) *Artificial Intelligence: A Modern Approach*, 4th edition. Pearson Education, Inc, Hoboken, New Jersey.

[2] Neller, T. W. (2015) Pedagogical Possibilities for the 2048 Puzzle Game. *Journal of Computing Sciences in Colleges*, **30**, 38–46.

[3] Yarasca, E. N. dan Nguyen, K. (2018) Comparison of Expectimax and Monte Carlo algorithms in solving the online 2048 game. *Pesquimat*, **21**, 1–10.

[4] Vodopivec, T. (2018) Monte Carlo Tree Search Strategies. Disertasi. University of Ljubljana, Slovenia.

[5] Sutton, R. dan Barto, A. (2018) *Reinforcement Learning: An Introduction*, 2nd edition. MIT press, Cambridge, Massachusetts.

[6] Jaśkowski, W. (2017) Mastering 2048 with Delayed Temporal Coherence Learning, Multistage Weight Promotion, Redundant Encoding, and Carousel Shaping. *IEEE Transactions on Games*, **10**, 3–14.

[7] Matsuzaki, K. (2021) Developing Value Networks for Game 2048 with Reinforcement Learning. *Journal of Information Processing*, **29**, 336–346.

[8] Browne, C., Powley, E., Whitehouse, D., Lucas, S., Cowling, P., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., dan Colton, S. (2012) A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games*, **4**, 1–43.

[9] Soemers, D. J. N. J. (2016) Enhancements for Real-Time Monte-Carlo Tree Search in General Video Game Playing. Thesis. Maastricht University, The Netherlands.

[10] Schuster, T. (2015) MCTS Based Agent for General Video Games. Thesis. Maastricht University, The Netherlands.