Menyelesaikan Game 2048 Menggunakan Algoritma Minimax dan Expectimax

Rayza Al Khensha Kacita Ananda
Fakultas Sains dan Teknologi
Universitas Al-Azhar Indonesia
Jakarta, Indonesia
khensha@gmail.com

***Abstract*—Strategi permainan 2048 mempunyai popularitas yang besar. Meski mudah dimainkan, orang tidak bisa memenangkan permainan dengan mudah, karena mereka biasanya tidak mempertimbangkan kemungkinan di masa yang akan datang. Dalam tulisan ini, terdapat beberapa strategi pohon pencarian (Minimax dan Expectimax) dalam menyelesaikan permainan 2048.**

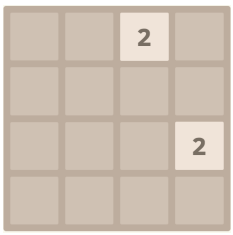
***Index Terms*—game, 2048, minimax, expectimax**

I. Pendahuluan

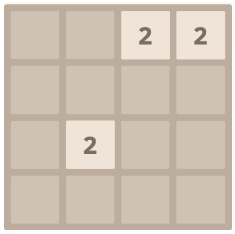
Seorang programmer asal Italia, Gabriele Cirulli merilis sebuah game video *single-player* yaitu 2048. Game ini dimainkan di papan *grid* 4x4, dengan setiap ubin berisi angka genap. Pemain dapat mengarahkan ubin di papan tersebut dengan empat arah: atas, bawah, kiri dan kanan. Jika dua ubin dengan nomor yang sama digeser, mereka akan bergabung menjadi ubin baru yang memenuhi penjumlahannya. Setelah pindah, ubin baru dengan nomor 2 atau 4 akan muncul secara acak disalah satu ubin kosong, dan kemudian, pemain membuat langkah baru.

Bila tidak ada tempat kosong dan tidak ada gerakan yang bisa memenuhi, permainan berakhir. Skor pengguna dimulai pada nol, dan bertambah setiap dua kali gabungan, dengan nilai ubin baru.

Tujuan penulisan ini adalah untuk membangun agen AI yang bisa memainkan game secara otomatis dan memaksimalkan skor permainan dalam waktu yang cepat. Dengan melibatkan pencarian pohon permainan dengan fungsi evaluasi heuristik yang dirancang secara manual, yang berdasarkan pada analisis manusia tentang *game property*.



(a) contoh awal permainan pada state s0.



(b) state s1 setelah aksi gerakan ke atas di s0. dan muncul ubin angka random 2

II. Dasar Teori

Game 2048 merupakan game *two-player* atau game dengan dua pemain (game manusia melawan komputer). Giliran manusia adalah memindahkan papan ke salah satu dari empat arah (kanan, kiri, atas, bawah) sementara giliran komputer mengeluarkan ubin 2 atau 4 di salah satu sel kosong.

Game dengan dua pemain dapat diterminologikan, [1] mempunyai lingkungan deterministik dan sepenuhnya dapat diamati dimana dua agen bertindak secara bergantian dan di mana nilai utilitas pada akhir permainan selalu sama dan berlawanan. Dalam hal ini, algoritma minimax dan expectimax cocok digunakan [2].

Minimax dan *expectimax* adalah algoritma untuk menentukan gerakan mana yang terbaik dalam permainan dua pemain. Karena itu, keduanya biasanya disebut aturan keputusan.

A. Algoritma Minimax

Algoritma *minimax* adalah algoritma dengan aturan keputusan yang meminimalkan kemungkinan kerugian akibat skenario terburuk, atau dengan kata lain untuk kemungkinan kerugian maksimal. Itulah sebabnya disebut *minimax*. Aturan awalnya digunakan untuk permainan *zero-sum* dua pemain (permainan yang memiliki keuntungan pemainnya sama dengan kerugiannya pemain lain, dan sebaliknya).

Teorema minimax menyatakan bahwa [3]:
Untuk setiap permainan dua orang, *zero-sum* dengan banyak strategi, ada nilai V dan strategi campuran untuk setiap pemain, sehingga:

1) Dengan strategi pemain 2, hasil terbaik mungkin pemain 1 adalah V, dan
2) 2. Dengan strategi pemain 1, hasil terbaik mungkin pemain 2 adalah -V.

Pada game 2048, lawan pemain yaitu komputer, disederhanakan menjadi sebuah *fixed policy*, yang menempatkan ubin baru 2 atau 4 dengan rasio probabilitas 8:2. Dengan notasi berikut:

1) $player$: *agen, lawan*
2) s: papan grid yang menandakan keadaan saat ini.
3) a: aksi yang diambil.
4) $actions(s)$: aksi yang mungkin terjadi di state s.

5) $succ(s, a)$: keadaan yang dihasilkan jika aksi a dipilih pada state s.
6) $isEnd(s)$: bila s adalah state akhir (permainan berakhir).
7) $eval(s)$: evaluasi pada state s.
8) $player(s) \in players$: pemain yang mengendalikan state s.
9) $totalScore(s)$: nilai skor pada akhir permainan.
10) d: kedalaman saat ini pada pohon pencarian.

Notasi ini untuk mewakili rumus matematika Minimax [4] adalah sebagai berikut:

$$V_{max,min}(s,d)=\begin{Bmatrix} TotalScore(s), isEnd(s) \\ eval(s), d=0 \\ max_{a \in action(s)} \\ V_{max,min}(succ(s,a),d) \\ (players(a)=opponent) \end{Bmatrix}$$

Selama permainan berlangsung, agen adalah Artificial Intelligence yang telah dibuat, dan lawannya adalah komputer [5]. Ketika pemain adalah agen, aksi $\in \{atas, bawah, kanan, kiri\}$,saat pemain lawan, aksi a akan secara acak menempatkan ubin berangka 2 atau 4 ke ubin kosong papan saat ini.

Pada pohon pencarian, *minimax* menghitung nilai keuntungan mencapai *leaf node*. Pada sebagian besar kasus, tidak mungkin mencapai "*real leaf node*" yaitu situasi dimana permainan berakhir, karena kedalamannya terlalu besar. Jadi, kadang-kadang algoritma minimax diimplementasikan dengan *depth-limited search*. Karena pada *boundless depth-first search*, tidak tahu nilai pasti perolehan pada *lead node*, sehingga hanya akan menghitung perkiraannya, menggunakan beberapa fungsi heuristik.

B. Algoritma Minimax dengan Alpha-Beta Prunning

Algorithm 1 Alpha-Beta Prunning

function ALPHABETA($node, depth, \alpha, \beta$)

if $depth = 0$ or node is terminal node **then**
return the heuristic value of node
if $turn == player1$ **then**
for each child of node **do**
α := max($\alpha, alphabeta(child, depth - 1, \alpha, \beta, player2)$)
if $\beta \leq \alpha$ **then break**($\beta cut - off$)

return α
for $eachchildofnode$ **do**
β := min($\beta, alphabeta(child, depth - 1, \alpha, \beta, player1)$)
if $\beta \leq \alpha$ **then break**($\alpha cut - off$)

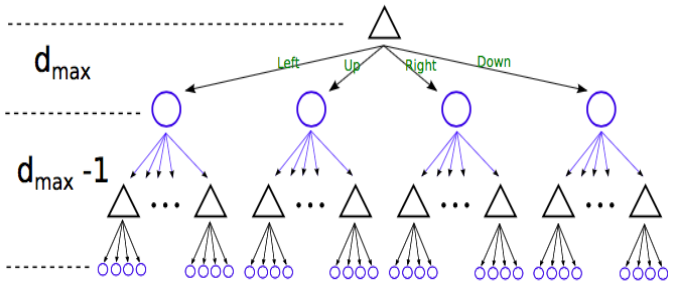
return β

Alpha-beta menggunakan nilai *minimax* untuk memangkas sub dari pohon, jika suatu aksi tidak akan mempengaruhi keputusan pada *root node*. [6] Hal ini terjadi ketika nilai *Minimax* dari subtree menunjukkan nilai *minimax* yang dicadangkan dari subtree yang berbeda. Hal tersebut sepenuhnya berhenti mengevaluasi tindakan, ketika setidaknya satu kemungkinan telah ditemukan yang membuktikan bahwa tindakan tersebut menjadi lebih buruk daripada tindakan yang sebelumnya diperiksa, jadi tidak perlu dievaluasi lebih lanjut.

Pada Algorithm 1, Istilah α adalah batas bawah perolehan, dan β adalah batas atasnya. Pada giliran *player1*, perolehan dimaksimalkan dengan mengevaluasi semua kemungkinan pergerakan. Nilai α diperbarui setiap saat. Bila pada titik tertentu β tidak lebih besar dari α, maka pencarian dapat berhenti di sini, karena telah dibuktikan bahwa ada pergerakan yang lebih baik. Ini disebut β cut-off, begitu pula dengan α.

C. Algoritma Expectimax

Algoritma *Expectimax* cukup sama dengan algoritma minimax, bedanya adalah menambahkan node peluang antara node agen dan node lawan. Dalam algoritma expectimax, saat node lawan di evaluasi, akan di hitung semua kemungkinan pergerakan, tertimbang oleh probabilitas terjadinya. [7] Dengan kata lain, kita akan menghitung nilai yang diharapkan dari keuntungan atas semua kemungkinan kasus. Oleh karena itu, pohon pencarian expectimax tumbuh sangat pesat, dan perhitungannya menjadi sangat mahal.



Gambar 2: Pohon pencarian *expectimax*

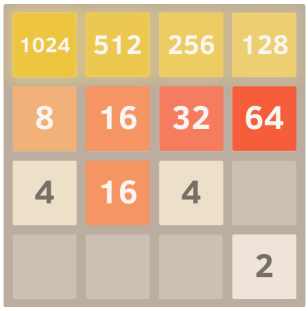
Pada pembuatan agen AI di tulisan ini, menggunakan pohon pencarian expectimax yang dibatasi kedalamannya. Nilai pada node daun diambil dari fungsi heuristik, yang nanti akan dijelaskan di bab berikutnya. Mengenai probabilitasnya, sudah diketahui yaitu, pilihan antara ubin kosong itu seragam, dan probabilitas terjadinya muncul angka 2 adalah 80%, sedangkan probabilitas muncul angka 4 adalah 20%. Pencarian expectimax sendiri dibuat sebagai pencarian rekursif yang bergantian antara "ekspetasi" setiap langkah (menguji semua lokasi dan nilai tangkapan ubin yang mungkin mempunyai peluang untuk mendapatkan skor lebih tinggi, dan membobot skor optimal dengan probabilitas setiap kemungkinan), dan langkah "maksimalisasi" (menguji semua kemungkinan pergerakan dan memilih satu dengan

skor terbaik). Pencarian pohon *expectimax* berakhir saat melihat posisi yang sebelumnya dilihat (menggunakan tabel transposisi), saat mencapai batas kedalaman yang telah ditentukan, atau saat mencapai *state* yang sangat tidak mungkin.

III. Implementasi Program

Pada pembuatan agen untuk menyelesaikan permainan 2048, algoritma yang telah disebutkan di bab sebelumnya, ditambahkan fungsi heuristik. Fungsi heuristik, adalah nilai yang memberi peringkat alternatif dalam algoritma pencarian. Nilai tersebut akan menjadi lebih baik jika kondisinya juga lebih baik dari sebelumnya.

Di permainan 2048, untuk mendapatkan ubin 2048, kita harus menggabungkan dua ubin 1024. Lalu, untuk mendapatkan dua 1024 ubin, kita harus memiliki ubin 1024 dengan cara mennggabungkan dua 512 ubin, dan seterusnya.



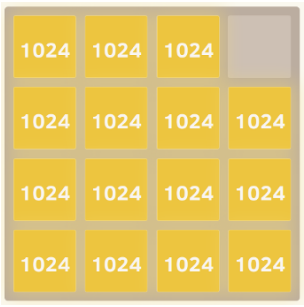
Gambar 3: Struktur ubin yang dapat mendapatkan skor tinggi

Berikut observasi untuk mengubah papan permainan menjadi seperti diagram di atas.

A. Ruang Kosong

Jika ada banyak ruang kosong pada papan permainan, maka, lebih mudah untuk bergerak dalam memindahkan ubin dan untuk mendapatkan jumlah yang lebih besar lagi. Ruang kosong ini, bisa ditentukan menjadi fungsi heuristik, dengan memberikan beberapa nilai besar pada semua ubin kosong yang telah ditemukan.

B. Kelancaran



Gambar 4: Contoh kedekatan ubin dengan kelancaran yang sangat baik

Tujuan permainan ini adalah untuk mendapatkan ubin 2048, dan untuk mendapatkan ubin tertentu, kita harus menggabungkan dua ubin dengan nilai lebih kecil, dan untuk menggabungkannya, kedua ubin ini harus berdekatan. Jadi, bila nilai ubin yang berdekatan dekat, kita katakan bahwa *grid* itu lancar. Istilah kelancaran disini ditentukan oleh perbedaan antara nilai-nilai ubin yang berdekatan. Semakin kecil perbedaannya, semakin lancar *grid*.

Pada fungsi heuristik, menggunakan faktor ini, dengan menurunkan nilai heuristik dengan total semua perbedaan dari semua ubin ubin yang berdekatan. Nilai ini dapat dibobot dengan mengalikannya dengan beberapa konstanta.

C. Ubin dengan Angka Besar di Pinggir Papan

Gambar 5: Ubin 512(paling besar) di sudut papan

Konsekuensi dari kelancaran *grid* adalah kita harus menjaga nilai-nilai yang besar bersama-sama, dan membiarkan ruang kosong untuk potongan kecil berubah menjadi ubin yang lebih besar. Tapi faktor kelancaran tidak cukup. Konsekuensi dari kelancaran *grid* pada papan permainan adalah, menjaga nilai-nilai yang besar tetap bersampingan, dan membiarkan ruang kosong untuk potongan kecil berubah menjadi ubin yang lebih besar. Tapi faktor kelancaran saja tidak cukup. Ubin bernilai besar harus disimpan di sisi papan yang sama, sehingga ubin bisa digabungkan dengan mudah. Bagaimanapun, dengan menggabungkan faktor kelancaran, ubin besar akan bertemu di perbatasan yang sama secara otomatis, jadi tidak perlu memeriksanya secara eksplisit.

Pada faktor ini, dalam permainan dapat dijadikan fungsi heuristik. Dengan cara, untuk setiap ubin di *grid*, diberikan penalti, jika ditempatkan di tengah *grid*. Jumlah penalti adalah dimana $C\ x\ jarak\ ke\ pinggir\ papan\ terdekat\ x\ nilai\ ubin$ dimana C adalah konstan. Memvariasikan nilai ini menghasilkan hasil yang berbeda. Atau bisa pula, memasukkan faktor ”nilai terbesar di sudut papan” dengan memberi *reward* pada posisi ubin dengan nilai terbesar yang ditempatkan di sudut.

Jika E adalah jumlah ubin kosong, D menunjukkan total semua selisih ubin yang pasangannya berdekatan, dan P adalah jumlah $jarak\ ke\ pinggir\ papan\ terdekat\ x\ nilai\ ubin$ dari semua ubin, maka nilai heuristik nya adalah:

$$H = A\ X\ E - B\ X\ D - C\ X\ P$$

Dimana A,B,C adalah konstan. Pada program implementasi A = 4096, B= 10 dan C= 10. Jika pada suatu saat, mencapai kondisi dimana tidak ada lagi gerakan yang tersedia, fungsi heuristik akan mengembalikan nilai yang sangat kecil ($-\infty$).

IV. Hasil Program

Integrasi antara *Artificial Intelligence* (AI) milik Robert [8] dan Ovolve [9] pada permainan 2048 ini, menggunakan bahasa pemograman javascript. Sehingga, pemrosesan komputasinya sedikit lebih lambat. Untuk mengambil nilai *depth* yang tinggi untuk pohon pencarian sangat terbatas. Padahal, semakin dalam pohon pencarian, semakin baik AI dalam mendapatkan ubin 2048 dan selebihnya.

Bila *depth* pohon pencarian *minimax* dibatasi hanya 4, tingkat kemenangan sangat rendah. Tapi performa (dalam hal kecepatan) sangat tinggi. Dalam satu detik AI bisa melakukan lebih dari 10 gerakan. Berikut data dalam penggunaan algoritma yang sama sebanyak 50 kali.

Ubin Tertinggi	Persentase
512	22%
1024	26%
2048	48%
4096	4%

TABLE I: algoritma minimax, depth = 4

Skor maksimum yang dicapai oleh algoritma *minimax* adalah 60744. Sebagian besar waktu AI bekerja, ia dapat mencapai ubin 2048 dengan persentase 48%.

Algoritma *Expectimax* tidak dapat dipangkas seperti *minimax* dengan *alpha-beta*, karena untuk menghitung nilai yang diharapkan, semua konfigurasi setiap *child-node* harus dihitung. Untuk mencapai kecepatan yang sama seperti algoritma minimax dengan batas *depth*= 4, batas kedalaman *expectimax* harus kurang dari 2. Bila batas *depth*> 2, jumlah pergerakan maksimum yang dapat dilakukan AI hanya tiga gerakan, dibutuhkan waktu lebih dari 10 detik untuk mengevaluasi suatu pergerakan, sehingga tidak efektif. Namun, bila batas *depth* = 2, tingkat kecepatan dan kemenangan tidak berbeda dengan algoritma minimax dengan batas *depth*= 4 .

Ubin Tertinggi	Persentase
512	12%
1024	32%
2048	48%
4096	8%

TABLE II: algoritma expectimax, depth = 2

Dibandingkan dengan algoritma *minimax*, algoritma *expectimax* sedikit lebih baik. Pada algoritma *expectimax*, tingkat kemenangan (untuk mencapai ubin 2048 atau lebih besar lagi) lebih tinggi, dilihat dari pencapaian ubin 4096 mengalami penaikan, yaitu 8%.

Jika meningkatkan batas *depth* algoritma *minimax* menjadi 8, AI masih bisa menghitung lebih dari 4 pergerakan per detik. Dari 10 permainan berturut-turut, berikut hasilnya.

Ubin Tertinggi	Persentase
512	10%
1024	20%
2048	50%
4096	20%

TABLE III: algoritma minimax, depth = 8

Tingkat kemenangan meningkat, dari hanya 52% menjadi 70%. Kesempatan mendapatkan ubin 4096 lebih besar lagi. Namun, 10 permainan tersebut memakan waktu lebih dari 4 jam, atau setara satu permainan membutuhkan waktu sekitar 24 menit. Ini sekitar 4-9 kali lebih lambat dari algoritma *minimax* dengan *depth* = 4.

Algoritma *expectimax* bahkan menunjukkan peningkatan yang lebih baik setelah meningkatkan batas *depth* = 3. Skor rata-rata adalah 40288, dan tingkat kemenangan 80%. Sebagai berikut.

Ubin Tertinggi	Persentase
1024	20%
2048	50%
4096	30%

TABLE IV: algoritma expectimax, depth = 3

Pada algoritma *expectimax* dengan batas *depth* = 3, permainan tidak pernah gagal untuk mendapatkan ubin 1024, dan kemungkinan untuk mendapatkan ubin 4096 sangat tinggi. Ini membuktikan bahwa tingkat kemenangan akan meningkat seiring dengan meningkatnya batas *depth*.

V. Kesimpulan

Algoritma pencarian *minimax* dan *expectimax* beserta fungsi heuristik yang digunakan cukup baik untuk memecahkan game 2048. Tingkat kemenangan lebih tinggi dan mudah diraih saat meningkatkan batas *depth* pada pohon pencarian. Namun, semakin batas *depth* tinggi, semakin buruk juga performa kecepatan AI dalam menyelesaikan permainan. Algoritma *minimax* dengan batas *depth* = 8 memiliki tingkat kemenangan sekitar 70%, sedangkan *expectimax* dengan batas *depth* = 3, memiliki tingkat kemenangan sekitar 80% dan memiliki sekitar 30% kemungkinan mendapatkan ubin 4069. Bahkan hasil yang lebih baik dapat dicapai dengan memperbaiki fungsi heuristik atau menerapkan algoritma dengan lebih baik.

Acknowledgment

Source Code yang digunakan untuk pengerjaan tulisan ini diunggah ke https://github.com/khenshaa/play2048AI.

References

[1] P. Norvig and S. J. Russell, *Artificial Intelligence: A Modern Approach (3rd edition)*. Prentice Hall, 2010.

[2] (2018, January) What is the optimal algorithm for the game, 2048? [Online]. Available: http:// stackoverflow.com/questions/22342854

[3] M. J. Osborne and A. Rubinstein, *A Course in Game Theory*. Cambridge, Massachusetts: The MIT Press, 1994.
[4] I. Adler, "The equivalence of linear programs and zero-sum games," *International Journal of Game Theory*, vol. 42, no. 1, pp. 165–177, 2013.
[5] G. Cirulli, "2048," 2014.
[6] A. Saffidine, H. Finnsson, and M. Buro, "Alpha-beta pruning for games with simultaneous moves." in *AAAI*, 2012.
[7] D. Michie and R. A. Chambers, "Boxes: An experiment in adaptive control," *Machine intelligence*, vol. 2, no. 2, pp. 125–153, 1968.
[8] R. Xiao, "Ai for the 2048 game," https://github.com/nneonneo/2048-ai.
[9] Ovolve, "Ai for the 2048 game," https://github.com/ovolve/2048-AI", 2014.