Richard Rivaldo
13519185 – K04

2048

1. Jelaskan secara umum algoritma AI yang kalian gunakan!

   Pada tugas kali ini, sebenarnya Saya sangat ingin dan cukup penasaran untuk menggunakan algoritma *Hill Climbing* karena menurut Saya algoritma tersebut cukup unik. Namun, karena keterbatasan sumber terhadap pengaplikasiannya, akhirnya Saya memilih algoritma lain, yaitu Algoritma *Expectimax*, yang sumbernya juga sedikit tapi paling tidak mencukupi.
   Algoritma yang sebelumnya digunakan pada permainan *Tic-Tac-Toe*, yaitu Algoritma Minimax memiliki banyak perkembangan dan versi yang memungkinkan pemrogram untuk mengadaptasikan lingkungan permainan dengan agen yang ingin dibangun. Dalam hal ini, algoritma *Expectimax* merupakan salah satu pengembangan lebih lanjut algoritma *Minimax*. Varian lain yang mungkin juga sering dipakai adalah *Expectiminimax*.
   Pada *Minimax*, kita tahu bahwa yang menjadi agen *minimizer* adalah agen yang memang juga memiliki tujuan untuk mengeluarkan gerakan paling optimal yang menghasilkan kekalahan bagi kita. Namun, kunci dasar perbedaan *Minimax* dengan *Expectimax* di sini adalah agen yang ada pada *Expectimax* merupakan agen yang tidak tahu gerakan paling optimal untuk meminimalisir kemenangan kita. Adapun dapat dibilang bahwa agen *minimizer* pada *Expectimax* merupakan agen yang mengeluarkan gerakan yang terbilang acak berdasarkan suatu distribusi probabilitas tertentu dan tidak bisa (sulit) diprediksi untuk ruang kemungkinan gerak yang besar.
   Secara struktural, algoritma *Expectimax* menggunakan struktur yang mirip dengan algoritma *Minimax*. Dengan representasi *tree*, di mana setiap simpulnya merupakan gerakan yang mungkin dilakukan, akan dilakukan evaluasi terhadap gerakan tersebut. Karena gerakan yang dihasilkan bersifat stokastik, hasil yang didapatkan dari pergerakan lawan belum tentu merupakan hasil yang paling minimum. Dengan demikian, evaluasi yang dilakukan terhadap setiap gerakan bukan lagi nilai minimum evaluasi, tetapi merupakan hasil rataan terhadap semua kemungkinan gerakan yang timbul dari suatu gerakan tertentu. Rataan ini berkaitan dengan konsep nilai ekspektasi E pada probabilitas.
   Hal yang mungkin membedakan *Expectimax* dengan *Minimax* adalah pada *Expectimax*, simpul *minimizer* akan diganti dengan simpul *chance*. Simpul *Chance* merupakan simpul yang akan menghitung nilai rataan utilitas untuk menghasilkan *expected utility*. Nilai ini akan digunakan untuk menentukan keputusan yang akan diambil oleh *maximizer* atau dalam kata lain adalah agen yang dibangun. Untuk menentukan nilai utilitas ini, maka dibutuhkan suatu fungsi yang disebut dengan fungsi evaluasi. Sesuai dengan namanya, fungsi evaluasi akan mengevaluasi suatu kondisi lingkungan permainan tertentu dan dengan demikian, menilai apakah suatu gerakan akan menghasilkan situasi yang menguntungkan bagi agen.
   Dalam menentukan fungsi evaluasi yang digunakan, biasanya digunakan berbagai fungsi heuristik yang mempertimbangkan berbagai faktor di dalam lingkungan permainan. Pada *Minimax*, skala fungsi evaluasi yang digunakan tidak terlalu penting dan dengan demikian algoritma ini dapat dikatakan bersifat insensitif terhadap transformasi monotonik. Berbeda dengan *Minimax* yang hanya menginginkan status yang lebih baik dan menguntungkan agen, *Expectimax* akan memerlukan bobot (*weight*) untuk setiap heuristik sehingga agen bisa mengetahui kondisi yang menjadi prioritasnya dalam pemilihan gerakan baru.
   Perbedaan lain antara *Expectimax* dan *Minimax* adalah masalah *completeness* pada pohon pencariannya. Pada *Minimax*, utilitas yang dihitung pada fungsi evaluasi tidak sensitif terhadap monotonitas, dan dengan demikian, pohon pencariannya bisa dipangkas

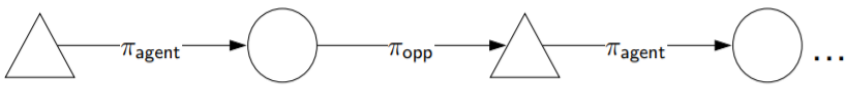
dengan menggunakan *Alpha-Beta Pruning*. Hal ini tidak berlaku pada algoritma *Expectimax*, karena evaluasi setiap gerakan akan memerlukan nilai utilitas lengkap sampai dengan simpul daun. Untuk mengatasi hal ini, maka biasanya sering digunakan *Depth-Limited Expectimax* yang membatasi jumlah kedalaman pohon pencariannya, mirip dengan IDS (*Iterative Deepening Search*) atau DLS (*Depth-Limited Search*).
Secara umum, algoritma *Expectimax* bisa diimplementasikan dengan prinsip rekursivitas sebagai berikut. Jika pada giliran tertentu merupakan giliran *maximizer* (agen), maka akan dikembalikan nilai maksimum dari pohon pencarian untuk mengembalikan gerakan yang menghasilkan status optimal bagi agen. Jika giliran tersebut merupakan giliran simpul *Chance*, maka akan dikembalikan nilai rataan pohon pencarian dengan mempertimbangkan nilai probabilitas yang dimiliki oleh setiap kemungkinan gerak. Kedua fungsi utama ini akan dipanggil terus-menerus sampai algoritma berhasil mencapai *terminal state*, atau dalam kata lain sudah mencapai simpul daun. Pada kasus *Depth-Limited Expectimax*, maka status terminal adalah ketika sudah dicapai batas kedalaman pohon yang ditentukan sebelumnya. Berikut merupakan ilustrasi *pseudocode* untuk algoritma *Expectimax* yang diambil dari bahan ajar *UC Berkeley* oleh Pieter Abbeel dan juga Stanford.

```
def value(state):
    if the state is a terminal state: return the state's utility
    if the next agent is MAX: return max-value(state)
    if the next agent is EXP: return exp-value(state)
```

```
def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor))
    return v
```

```
def exp-value(state):
    initialize v = 0
    for each successor of state:
        p = probability(successor)
        v += p * value(successor)
    return v
```

Analogy: recurrence for value iteration in MDPs



$$V_{\text{exptmax}}(s) = \begin{cases} \text{Utility}(s) & \text{IsEnd}(s) \\ \max_{a \in \text{Actions}(s)} V_{\text{exptmax}}(\text{Succ}(s,a)) & \text{Player}(s) = \text{agent} \\ \sum_{a \in \text{Actions}(s)} \pi_{\text{opp}}(s,a) V_{\text{exptmax}}(\text{Succ}(s,a)) & \text{Player}(s) = \text{opp} \end{cases}$$

Keuntungan algoritma *Expectimax* dibandingkan dengan algoritma *Minimax* adalah algoritma *Expectimax* sangat cocok digunakan untuk agen lawan yang tidak optimal dalam menentukan gerakan selanjutnya. Tidak seperti *Minimax*, *Expectimax* bisa mengambil gerakan yang memiliki resiko tinggi, karena lawan belum tentu memberikan gerakan yang optimal dan dengan demikian memungkinkan agen kita untuk mendapatkan status dengan utilitas yang jauh lebih tinggi dan menguntungkan.
Kelemahan dari algoritma *Expectimax* adalah algoritma ini tidak optimal, terutama apabila pohon pencariannya dipangkas dan dibatasi kedalamannya. Dengan keterbatasan tersebut pula, algoritma *Expectimax* merupakan algoritma yang jauh lebih lambat dibandingkan dengan *Minimax*. Terakhir, *Expectimax* sangat sensitif terhadap transformasi monotonik dan dengan demikian sangat bergantung pada pembobotan setiap nilai heuristik pada fungsi evaluasinya. Nilai heuristik yang belum tentu optimal ini menyebabkan algoritma *Expectimax* menjadi kurang stabil.

2. Jelaskan bagaimana algoritma AI mengambil langkah terbaik dalam program yang kalian buat!

Pada permainan 2048 yang dibuat, agen yang dibangun merupakan agen yang memindahkan atau menggerakan blok-blok angka pada setiap langkahnya sampai dengan *game over*. Adapun agen lawan yang menjadi *chance node* adalah papan yang secara otomatis menambahkan satu buah blok baru dengan nilai dan posisi yang acak. Nilai yang mungkin untuk blok baru ini hanyalah 2 dan 4, dengan nilai 2 memiliki 90% kemungkinan dan sisanya untuk nilai 4. Posisi blok diacak dari blok-blok yang masih kosong pada papan, sehingga tidak ada satupun *turn* yang tidak menghasilkan blok baru.
Seperti yang telah dijelaskan sebelumnya, agen sangat bergantung pada fungsi evaluasi yang digunakannya dalam menilai *environment* permainan. Dalam hal ini, nilai heuristik memegang peranan penting dalam penentuan gerakan tersebut. Pada algoritma yang dibangun, ada 4 buah nilai heuristik yang digunakan. Masih banyak nilai heuristik menarik lain yang bisa digunakan untuk permainan 2048 seperti yang terdapat pada diskusi StackOverflow ini.
Heuristik pertama yang digunakan adalah jumlah kuadrat dari setiap elemen yang ada pada papan permainan. Heuristik ini layaknya ketika kita memainkan permainan ini, di mana kita akan selalu menginginkan nilai yang semakin besar setiap blok pada papan permainan. Dengan nilai yang semakin besar, maka kita bisa mendapatkan poin yang lebih besar pula.
Fungsi heuristik kedua yang digunakan adalah banyaknya blok kosong yang ada pada papan permainan. Dengan menghitung banyaknya blok kosong ini, maka agen bisa mengetahui apakah dirinya berada dalam posisi yang aman. Hal ini mirip pada saat kita memainkan permainan tersebut dan kita menginginkan semakin banyak blok kosong pada permainan sehingga hal ini bisa meminimalisir kemungkinan gerakan yang berujung pada status buntu dan menyebabkan *game over* pada permainan kita.
Dalam memainkan permainan ini, seringkali kita ingin menempatkan blok-blok kita hanya pada satu sudut saja. Hal ini ditujukan supaya tidak ada blok-blok acak yang mengganggu susunan blok kita dan dengan demikian memudahkan kita ketika membangun angka yang lebih besar. Hal ini berkaitan dengan konsep monotonitas pada algoritma *Expectimax*. Karena algoritma ini sensitif terhadap transformasi ini, maka nilai monotonitas ini dijadikan sebagai heuristik ketiga. Untuk menentukan monotonitas ini, maka akan dihitung perbedaan nilai logaritma (*base 2*) untuk setiap blok yang bersisian terhadap 2 tanpa menghitung blok kosong. Perhitungan ini dilakukan untuk setiap arah monotonitas yang mungkin, yaitu keempat arah gerakan yang ada.

Untuk setiap arah vertikal dan arah horizontal, akan dicari nilai maksimal monotonitas di antara keduanya. Misalnya, monotonitas secara vertikal berarti membandingkan nilai monotonitas untuk arah atas dan bawah, begitu juga untuk nilai monotonitas horizontal untuk arah monotonitas kanan dan kiri. Kedua nilai maksimum ini kemudian akan ditambahkan dan dengan demikian, agen bisa menentukan dua buah sisi yang akan dipilih untuk menempatkan blok-blok secara menaik ataupun menurun dalam hal jumlahnya.
Heuristik sebelumnya memungkinkan blok-blok yang disusun untuk memiliki nilai yang sangat jauh berbeda dan dengan demikian tidak akan bisa digabungkan untuk membentuk blok baru yang lebih besar. Untuk mengatasi hal tersebut, maka akan digunakan heuristik keempat, yaitu *smoothness*. Heuristik ini akan menghitung perbedaan nilai-nilai dari blok yang bersisian, baik bersisian secara vertikal maupun secara horizontal. Dengan demikian, agen bisa memutuskan kapan waktu yang tepat untuk menggabungkan blok-blok dan membentuk blok yang lebih besar tanpa mengganggu monotonitas yang telah terbentuk sebelumnya.
Keempat nilai heuristik ini akan digabungkan dalam proses pengkalkulasian utilitas dengan bobotnya masing-masing. Bobot ini diperoleh secara acak, juga melalui berbagai percobaan dan eksperimen untuk melihat pasangan bobot mana yang memberikan dampak yang paling stabil terhadap permainan agen. Setelah dibobot, maka nilai hasil pembobotan tersebut akan dijumlahkan dan dengan demikian memberikan hasil akhir utilitas untuk suatu kondisi papan tertentu.
Untuk mempersingkat pemrosesan pohon pencarian, maka pohon pencarian yang dibentuk akan dipangkas dengan menggunakan kedalaman dan jumlah blok kosong sebagai parameternya. Dengan demikian, misalkan sudah tidak ada blok kosong yang tersisa pada papan permainan, maka akan langsung dikembalikan utilitas maksimum pada kedalaman berikutnya.
Untuk memaksimalkan utilitas pada setiap gerakannya, maka untuk setiap gerakan yang mungkin dan valid dilakukan oleh agen, maka akan dicari nilai ekspektasi utilitasnya. Gerakan terbaik dipilih dari nilai utilitas yang paling maksimum dan dapat memberikan status yang menguntungkan bagi agen. Adapun nilai ekspektasi utilitas dihitung rataannya untuk jumlah blok kosong yang ada, dengan cara yang sama seperti pada formula dan *pseudocode* sebelumnya. Perhitungan skor merupakan aproksimasi nilai dengan hanya mempertimbangkan blok acak berupa blok bernilai 2.

Referensi

Adversarial Search.
An artificial intelligence for the 2048 game.
Expectimax Algorithm in Game Theory.
Games I.
Lecture7: Expectimax and Utilities.
What is the optimal algorithm for the game 2048?
Writing a 2048 AI.