TUGAS KECIL 01

MINIMAX ALGORITHM AND ALPHA BETA PRUNNING IN DOTS AND BOXES BOARD GAME

IF3170 – INTELIGENSI BUATAN

Disusun oleh:

13520034 – Bryan Bernigen
13520055 – Christopher Jeffrey

Institut Teknologi Bandung
Jl. Ganesha NO. 10, Bandung 40132

2022

Struktur Program Game

Struktur Game ini dibuat berdasarkan https://github.com/AbdiHaryadi/Dots-and-Boxes

Nama		Keterangan
Main.py		File utama yang akan dijalankan. Berisi konfigurasi game seperti ukuran permainan dan bot
Class Dots_and_Boxes		Kelas utama yang akan menyajikan penampilan game
	__init__	Melakukan inisiasi permainan
	play_again	Mengulang permainan
	mainloop	Loop utama yang akan dijalankan terus menerus untuk memperbaharui keadaan
	is_grid_occupied	Menentukan apakah suatu lokasi tertentu sudah ada garisnya atau belum
	convert_grid_to_logical_position	Mengubah array 1 dimensi menjadi 2 dimensi agar mudah divisualisasikan
	pointScored	Menandakan apakah kita berhasil mencetak poin atau tidak
	mark_box	Mewarnai box dengan warna pemain yang berhasil mencetak poin
	update_board	Melakukan pembaharuan tampilan board
	Is_gameover	Mengecek apakah seluruh baris dan kolom sudah diberi garis
	Make_edge	Mewarnai garis sesuai warna pemain yang membuat garis
	Display_gameover	Menampilkan pesan jika permainan selesai
	Refresh_board	Memperbaharui tampilan papan
	Display_turn_text	Menampilkan pesan siapa pemain beriktunya
	Shade_box	Membuat persegi panjang untuk bagian luar papan
	Display_turn_text	Menampilkan pesan siapa pemain berikutnya
	Click	Mencatat jika pemain memilih suatu garis untuk dibuat
	Update	Melakukan update untuk seluruh game state
	Turn	Menandakan giliran siapa saat ini
	Bot_turn	Meminta bot untuk melakukan suatu aksi
Bot.py		File yang menyimpan algoritma bot
Class bot		Kelas dimana algoritma bot diimplementasikan
	Get_action	Melakukan suatu aksi berdasarkan algoritma yang diimplementasikan

RandomBot.py		File yang menyimpan algoritma bot yang memilih aksi secara random
Class RandomBot		Kelas untuk menyimpan algoritma bot yang memilih aksi secara random
	Get_action	Mendapatkan suatu aksi jika masih ada aksi yang dapat dilakukan. Penentuan aksi dilakukan pada fungsi random lainnya
	Get_random_action	Mendapatkan suatu aksi secara random
	Get_random_row_action	Mendapatkan suatu baris kosong secara random
	Get_random_position_with_zero_value	Mendapatkan suatu baris dan kolom kosong secara random
	Get_random_col_action	Mendapatkan suatu kolom kosong secara random
GameAction.py		File yang menyimpan aksi aksi yang dapat dilakukan pada game
Class GameAction		Kelas untuk menyimpan aksi-aksi yang dapat dilakukan pada game
GameState.py		File yang menyimpan status-status yang ada di game
Class GameState		Kelas untuk menyimpan status-status permainan

Rencana Kelas dan Fungsi

Untuk kelas, sesuai dengan struktur program game pada bagian sebelumnya. Berikut fungsi-fungsi yang dapat dibuat

```
Function miniMaxWithAlphaBeta(state State, isMaximizing bool,
alpha Integer, beta Integer) => State
```

Merupakan fungsi utama yang dipanggil untuk menentukan langkah terbaik yang dapat dilakukan pada state tertentu

```
Function terminalTest(state State) => Bool
```

Mengembalikan True jika state merupakan terminal

```
Function objective(state State)
```

Menghitung heuristik *score* suatu state input

```
Function generateNeighbour(state State) => State[]
```

Merupakan fungsi batuan dalam miniMaxWithAlphaBeta. Menerima suatu state, lalu mengembalikan semua state neighour dari state tersbut. Jika sudah tidak ada neighour lagi, fungsi akan mengembalikan array kosong.

Fungsi Objektif

Salah satu fungsi objektif yang dapat digunakan pada permainan ini adalah fungsi sederhana yang menghitung jumlah kotak yang dihasilkan oleh pemain 1 dikurang kotak yang dihasilkan pemain 2. Fungsi objektif tersebut didapat dari heuristik bahwa kemenangan seorang pemain dilihat dari banyaknya kotak yang ia buat. Karena terdapat 2 orang yang memiliki objektif yang sama, yakni memaksimalkan jumlah kotak yang ia buat, maka ada sedikit perubahan pada fungsi objektif dari yang tadinya hanya menghitung jumlah kotak yang dihasilkan menjadi jumlah kotak yang dihasilkan oleh pemain 1 dikurang jumlah kotak yang dihasilkan oleh pemain 2. Dengan mengubah fungsi tersebut, maka objektif tiap pemain juga sedikit berubah. Sebelumnya, kedua pemain sama-sama berusaha memaksimalkan *score* hasil fungsi objektif, namun setelah fungsi objektifnya berubah, tujuan pemain 1 menjadi mencari langkah yang menghasilkan *score* objektif tertinggi sedangkan pemain 2 mencari langkah yang menghasilkan *score* objektif terendah.

Proses Pencarian

Permainan dots and boxes dilakukan dalam sebuah papan atau *board.* Detiap kondisi unik *board* kita sebut sebagai suatu **state**. Suatu state terseebut memiliki ***score*** yang ditentukan berdasarkan bagus tidaknya suatu langkah untuk mendekatkan seorang player ke tujuan dari game (yaitu menang dengan membuat box sebanyak mungkin diakhir game). Kita sebut fungsi untuk menentukan *score* dengan **objective function.** Secara lebih formal, objective function adalah sebuah fungsi yang digunakan untuk menghitung *score* suatu state, berdasarkan bagus tidak nya suatu Langkah dalam mendekatkan seorang player ke kemenangan(yaitu membuat box sebanyak mungkin diakhir game).

Algoritma minimax digunakan untuk mencari langkah terbaik dalam permainan yang melibatkan 2 pemain, salah satunya dots and boxes. Algoritma minimax dapat digunakan untuk menghitung kedua objektif, objektif untuk memaksimalkan, dan objektif untuk meminimalkan. Dalam permainan dots and boxes sendiri, aplikasi algoritma minimax dapat diterapkan dengan memberikan *score* objektif yang berbeda kepada kedua pemain. Misalnya memberikan *score* positif untuk state yang menguntungkan pemain satu dan memberikan *score* negatif untuk state yang menguntungkan pemain dua. Dengan demikian, pemain satu akan berusaha untuk melakukan langkah yang membuat state permainan semaksimal mungkin sedangkan pemain dua akan melakukan langkah yang membuat state semiminal mungkin.

Penerapan algoritma minimax dilakukan dengan melakukan rekursif sampai state terminal, atau sampai kedalaman tertentu. Setiap state yang bukan terminal atau bukan kedalaman terdalam akan meng*enerate* seluruh kemungkinan *state tetangganya* (state yang dapat dicapai dalam 1 langkah) dan dengan memanfaatkan rekursif, setiap state tetangga tersebut akan meng*enerate* tetangga lainnya. Setiap state yang di*enerate* akan diberi *score* sesuai dengan *objective function*. Setelah program sampai di state terminal atau state paling dalam, program akan melakukan *trace back* sambil membandingkan *score* state tersebut dengan state lainnya. Pada saat inilah algoritma minimax dibedakan dengan algoritma pencarian jalan terbaik lainnya karena pada game ini, terdapat 2 objektif yang selalu berganti-ganti sesuai dengan giliran pemainnya. Jika giliran pemain 1, maka pemain 1 akan mencari langkah yang paling membuat state memiliki *score* paling besar sedangkan jika giliran pemain 2, maka pemain 2 akan mencari

langkah yang membuat state paling kecil. Dengan algoritma minimax, kita dapat mensimulasikan pemilihan langkah walaupun terdapat 2 pemain yang memiliki goal yang berbeda. Dengan demikian kita dapat memilih gerakan terbaik karena kita dapat sekaligus memprediksi pergerakan musuh.

Pada game dot and boxes dengan ukuran dot 4x4, state board kosong memiliki 24 neighbouring state. Masing-masing neighbouring state tersebut, memiliki 23 neighbouring state. Jika ini diteruskan, dapat menghasilkan 24! State yang perlu dihitung untuk menentukan suatu Langkah (sebenarnya ini kurang tepat. Seorang player dapat bergerak dua kali jika gerakan sebelumnya menghasilkan box, sehingga belum tentu tepat 24!).

Hal ini buruk dari segi komputasi. Terlebih untuk kasus yang lebih besar dan rumit.

Untuk menanggapi masalah ini, kita dapat melakukan dua hal. Pertama dengan memberikan Batasan *depth*. Yaitu Batasan kedalaman state yang dapat dicapai untuk menghitung *score* yang paling bagus. Hal ini menjadikan jumlah state yang perlu dihitung menjadi

$$24!/(24 - \text{maxDepth})!$$

Kedua, kita dapat tidak menghitung state yang sudah pasti tidak akan dipilih. Strategi ini yang kita sebut alpha beta prunning.

Misal ada dua buah neighbouring state yang dapat dipilih player satu. Neighbouring state satu memiliki *score* 10. Neighbouring state kedua, kita tahu maksimalnya adalah 4(dalam symbol matematika, <= 4). Tanpa perlu menghitung *score* sesungguhnya dari neighbouring state kedua, kita pilih neighbouring state pertama.

Kita dapat sebut 4 sebagai Alpha, yaitu *score* maksimal sementara. Untuk kasus giliran player dua, kita ingin score sekecil mungkin. Kita dapat buat sebuah holder *score* lagi, kita sebut dengan Beta, yaitu score minimal sementara.

Secara teori, prunning dilakukan jika $\beta \leq \alpha$ dimana α merupakan *score* maksimal dan β merupakan *score* minimal. α diperbaharui setiap kali pemain yang objektifnya memaksimalkan bergerak dan β diperbaharui setiap kali pemain yang objektifnya meminimalkan bergerak. $\beta \leq \alpha$ dapat berarti satu diantara 2 hal ini terjadi: Secara teori, prunning dilakukan jika $\beta \leq \alpha$ dimana α merupakan *score* maksimal dan β merupakan *score* minimal. α diperbaharui setiap kali pemain yang objektifnya memaksimalkan bergerak dan β diperbaharui setiap kali pemain yang objektifnya meminimalkan bergerak. $\beta \leq \alpha$ dapat berarti satu diantara 2 hal ini terjadi:

1. Jika terjadi pada giliran pemain yang memaksimalkan, berarti sebelumnya sudah ada pilihan gerakan lain yang hasilnya lebih negatif sehingga lebih disukai oleh lawan. Pilihan yang membuat alphanya besar tersebut tidak perlu ditinjau lebih lanjut karena lawan kita yang bergerak sebelum kita akan memilih langkah yang lebih negatif sehingga state ini di prune karena tidak akan pernah tercapai (lawan tidak akan memilih state ini).
2. Jika terjadi pada giliran pemain yang meminimalkan, berarti sebelumnya sudah ada pilihan gerakan yang menghasilkan state yang lebih positif sehingga lawan kita akan cenderung memilih state tersebut dibanding state yang sedang kita explore saat ini. Oleh karena itu state ini di prune karena tidak akan pernah tercapai (lawan lebih memilih state dengan *score* lebih besar).

Local Search

Penerapan local search dalam menentukan langkah terbaik tidak dapat menjamin bahwa langkah yang dipilih merupakan langkah terbaik yang menuju ke objektif. Hal tersebut terjadi karena local search hanya mencari state tetangga yakni state yang dapat dituju dengan satu langkah. Dengan demikian, langkah yang dipilih merupakan langkah terbaik pada state tersebut, namun tidak menjamin bahwa langkah tersebut akan menghasilkan state terbaik pada akhirnya.

Salah satu contoh penerapan local search pada game 4x4 dots and boxes adalah penerapan algoritma hill climbing. Pertama,dalam kita akan menghitung *score* berdasarkan fungsi objektif pada seluruh langkah yang mungkin untuk dilakukan pada state tersebut. Kedua, kita akan memilih langkah yang menghasilkan *score* tertinggi.

Dalam penerapannya, algoritma ini sering kali gagal menemukan global maximum apalagi jika fungsi objektifnya hanya memedulikan apakah sebuah langkah menghasilkan poin bagi kita atau tidak. Jika menggunakan fungsi objektif seperti itu, kita dapat dipastikan akan kalah walaupun sudah berada dalam posisi yang menguntungkan karena beberapa teknik yang tidak diterapkan jika menggunakan algoritma tersebut. Dengan algoritma tersebut, kita tidak akan dapat melakukan *2 or 4 sacrifice* yakni salah satu teknik yang mengorbankan 2 atau 4 kotak untuk mendapat kotak yang lebih banyak. Dengan demikian, kita dapat dipastikan kalah jika teknik *2 or 4 sacrifice* perlu dilakukan untuk menang karena bot kita tidak akan memilih hal tersebut. Bot kita akan lebih memilih untuk mengambil seluruh kotak (menambah 2 atau 4 poin) namun kehilangan kotak yang lebih banyak pada giliran-giliran selanjutnya.

Genetic Algorithm

Penerapan algoritma genetik pada permainan ini tidak mungkin dilakukan. Alasannya adalah karena algoritma genetik menghasilkan state-state yang tidak bertetangga dengan state saat ini dengan melakukan kawin silang dan mutasi. Hal tersebut tidak dapat diterapkan pada game 4x4 dots and boxes karena dalam 1 giliran, kita hanya dapat bergerak 1 kali atau dengan kata lain kita hanya dapat mencapai state tetangga. Jika menggunakan algoritma generik, kita harus melakukan beberapa gerakan secara langsung atau memaksa lawan untuk melakukan gerakan sesuai yang kita inginkan karena permainan dilakukan secara bergilir. Oleh karena itu, penggunaan algoritma generik tidak dapat digunakan dalam permainan ini.

Commented [CJ1]: Saraan, Mungkin depannya bisa ditambahin 'menurut algoritma local search yang digunakan'. Because yaa it's not objectively the best langkah. Itu quote on quote subjective menurut algo yang digunaka

Commented [CJ2]: Saraan, bisa paraphrase jadi 'Kita coba terapkan local search pada game 4x4 dots and boxes. Sebagai contoh, algoritma yang akan digunakan adalah *steepest hill climbing'*

Commented [CJ3]: Ada diksi tepatnya ngga ya? Kalo ada. Bisa diganti diksi tepatnya. Kalo gada, di italic mungkin?

Commented [CJ4]: Sama kek salah satu comment sebelumnya, mungkin ada diksi yang lebih bagos

Referensi

1. https://www.abelromer.com/static/dandb.pdf
2. https://www.researchgate.net/publication/338855656_INTELLIGENT_AGENTS_FOR_SOLVING_THE_GAME_DOTS_AND_BOXES
3. https://pats.cs.cf.ac.uk/@archive_file?p=1686&n=final&f=1-report.pdf&SIG=4eec909fce4fb2da94c4b05d1906f909face6aa1dc12dd5def36b33ff165eace
4. https://towardsdatascience.com/how-a-chess-playing-computer-thinks-about-its-next-move-8f028bd0e7b1
5. https://www.mygreatlearning.com/blog/alpha-beta-pruning-in-ai/