Penerapan Minimax dengan Alpha-Beta Pruning untuk Optimasi Strategi pada Permainan Quoridor

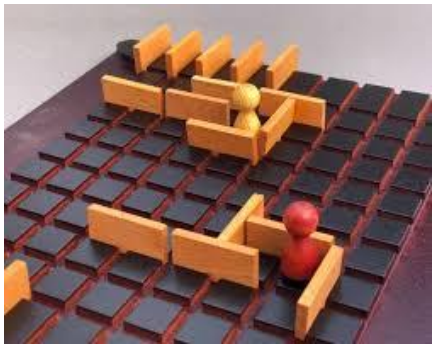
Wafiq Hibban Robbany - 13524016
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: wafiq.robbany@gmail.com , 13524016@std.stei.itb.ac.id

***Abstract* — Makalah ini membahas penerapan algoritma Minimax dengan Alpha-Beta Pruning untuk optimasi strategi pada permainan Quoridor. Quoridor memiliki ruang pencarian yang besar karena pemain dapat memilih untuk menggerakkan pion atau memasang wall pada setiap giliran. Untuk membantu pemilihan langkah, dibuat program pendukung yang menghitung tiga rekomendasi langkah terbaik berdasarkan skor evaluasi. Minimax digunakan untuk mengevaluasi kemungkinan langkah beberapa giliran ke depan, sedangkan Alpha-Beta Pruning digunakan untuk memangkas cabang pencarian yang tidak memengaruhi keputusan akhir. Fungsi evaluasi mempertimbangkan jarak terpendek menuju goal row, sisa wall, dan mobilitas pemain. BFS digunakan untuk menghitung jalur terpendek serta memastikan pemasangan wall tetap legal. Hasil pengujian menunjukkan bahwa Alpha-Beta Pruning mampu mengurangi jumlah node yang dikunjungi secara signifikan. Pada state awal, jumlah node turun dari 2284 menjadi 582 node, sedangkan pada mid game turun dari 2214 menjadi 754 node. Algoritma juga berhasil mengenali langkah kemenangan pada kondisi satu langkah sebelum menang. Dengan demikian, Minimax dengan Alpha-Beta Pruning efektif digunakan untuk mendukung optimasi strategi pada permainan Quoridor.**

Keywords* — *Quoridor; Minimax; Alpha-Beta Pruning; game tree search; heuristik; BFS; rekomendasi langkah

I. PENDAHULUAN

Quoridor merupakan permainan papan strategi yang dimainkan pada area berukuran 9x9 [4]. Dalam varian dua pemain, setiap pemain memulai langkah dari sisi yang saling berlawanan dan berkompetisi untuk mencapai sisi seberang terlebih dahulu. Pada setiap giliran, pemain dihadapkan pada pilihan strategis: menggerakkan pion atau menempatkan dinding penghalang. Penempatan dinding berfungsi untuk menghambat pergerakan lawan, dengan catatan aturan permainan mewajibkan setiap pemain tetap memiliki minimal satu jalur yang valid menuju garis finis [4]. Kombinasi dinamis antara pergerakan fisik pion dan manipulasi spasial lewat dinding ini menjadikan Quoridor sebagai studi kasus yang kaya untuk pemodelan strategi.



Gambar 1. Papan Permainan Quoridor (sumber: https://www.meeplemountain.com/reviews/quoridor/)

Dalam konteks permainan dua pemain dengan informasi sempurna (perfect information game), setiap pemain dipandang sebagai agen rasional yang berusaha mengambil keputusan terbaik berdasarkan proyeksi beberapa langkah ke depan. Salah satu metode yang sangat populer digunakan untuk memodelkan pengambilan keputusan pada kondisi deterministik ini adalah algoritma Minimax [1]. Algoritma ini bekerja dengan asumsi dasar bahwa pihak lawan juga akan selalu memilih langkah yang paling optimal untuk meminimalkan keuntungan pemain [1].

Kendati demikian, besarnya faktor percabangan (branching factor) pada Quoridor menyebabkan jumlah kemungkinan state membengkak secara eksponensial seiring bertambahnya kedalaman pencarian. Untuk mengatasi kendala tersebut, Alpha-Beta Pruning diterapkan sebagai metode optimasi [1, 2]. Berlandaskan prinsip branch and bound, teknik ini berfungsi memangkas (pruning) cabang-cabang pada pohon permainan (game tree) yang dipastikan tidak akan memengaruhi keputusan akhir, sehingga dapat mempercepat proses pencarian secara signifikan. Selain pengoptimalan struktur pohon, evaluasi dalam permainan ini juga memanfaatkan algoritma Breadth-First Search (BFS). Algoritma BFS diintegrasikan secara khusus untuk menghitung jarak terdekat dari posisi pion menuju sisi tujuan sekaligus memvalidasi ketersediaan jalur agar tidak melanggar aturan permainan.

Makalah ini bertujuan untuk merumuskan dan mengevaluasi penerapan algoritma Minimax yang dioptimalkan dengan Alpha-Beta Pruning untuk menghasilkan rekomendasi kandidat langkah terbaik pada permainan Quoridor dalam waktu yang terbatas. Penelitian ini

menitikberatkan pada efisiensi pencarian strategi melalui penerapan kompromi teknis, seperti pembatasan kedalaman pencarian, perancangan heuristik evaluasi, pembatasan kandidat dinding, serta mekanisme caching BFS dan move ordering sebagai kompromi antara kualitas rekomendasi dan waktu komputasi. Melalui pendekatan tersebut, makalah ini menyajikan analisis perbandingan langsung antara metode Minimax standar dan versi pencarian yang telah dioptimalkan berdasarkan metrik jumlah node yang dikunjungi, cabang yang berhasil dipangkas, serta efisiensi waktu pencarian.

II. LANDASAN TEORI

A. *Quoridor sebagai Permainan Strategi*

Quoridor termasuk permainan deterministik dengan informasi sempurna. Deterministik berarti tidak terdapat unsur acak seperti dadu, sedangkan informasi sempurna berarti kedua pemain dapat melihat seluruh state permainan. State permainan dapat didefinisikan melalui posisi pion, kumpulan dinding horizontal, kumpulan dinding vertikal, sisa dinding masing-masing pemain, dan giliran pemain.

Dalam implementasi ini, papan direpresentasikan sebagai grid 9x9. Player 1 mulai pada koordinat (8, 4) dan menang ketika mencapai row 0. Player 2 mulai pada koordinat (0, 4) dan menang ketika mencapai row 8. Setiap pemain memiliki 10 wall. Pemain dapat menggerakkan pion satu langkah ortogonal atau memasang dinding horizontal maupun vertikal. Aturan penting yang harus selalu dipertahankan adalah bahwa pemasangan dinding tidak boleh membuat salah satu pemain kehilangan seluruh jalur menuju goal [4].

Sifat permainan ini menimbulkan tantangan algoritmik. Di satu sisi, langkah pion relatif sedikit karena hanya mencakup arah ortogonal dan lompatan tertentu. Di sisi lain, kemungkinan pemasangan dinding sangat banyak, terutama pada fase awal ketika papan masih kosong. Akibatnya, pencarian yang mencoba seluruh kombinasi langkah hingga permainan selesai tidak realistis untuk program interaktif.

B. *Pohon Permainan dan Minimax*

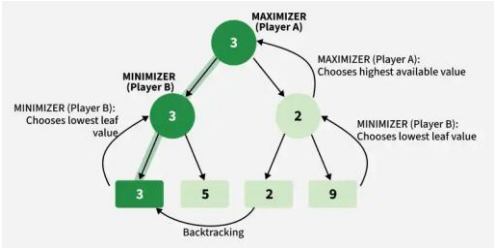
Pohon permainan (game tree) merepresentasikan state sebagai node dan move sebagai edge. Akar pohon adalah state saat ini. Anak dari suatu node adalah state baru yang dihasilkan setelah salah satu legal move diterapkan. Pada permainan dua pemain, level pohon bergantian antara pemain yang sedang dianalisis dan lawannya.

Minimax pada dasarnya adalah algoritma berbasis *backtracking* yang digunakan secara luas dalam teori permainan dan kecerdasan buatan untuk menentukan langkah optimal pada permainan kompetitif. Algoritma ini dirancang khusus untuk permainan dua pemain yang bersifat bergilir (turn-based) dan berjenis *zero-sum games*. Dalam *zero-sum games*, keuntungan yang didapat oleh satu pemain bernilai persis sama dengan kerugian di pihak lawan [5].

Algoritma ini membangun proses pengambilan keputusan dengan mengevaluasi pohon permainan menggunakan dua jenis peran: Maximizer dan Minimizer. Maximizer adalah pemain yang bertujuan untuk memaksimalkan hasil akhir atau skor permainan. Sebaliknya, Minimizer (direpresentasikan oleh minimizing node) merupakan pihak lawan yang berusaha meminimalkan skor tersebut atau menekan keuntungan dari Maximizer. Minimax beroperasi berlandaskan asumsi bahwa kedua pemain bertindak secara rasional dan selalu memainkan langkah yang paling optimal [1, 5].

Secara operasional, Minimax memperluas seluruh kemungkinan langkah masa depan dalam bentuk pohon permainan hingga mencapai akhir permainan atau batas kedalaman tertentu. Pada simpul daun , algoritma memberikan nilai utilitas. Nilai utilitas ini merupakan skor numerik yang merepresentasikan hasil absolut dari permainan, seperti menang, kalah, atau seri. Namun, karena pencarian hingga terminal penuh membutuhkan komputasi yang terlalu besar, fungsi evaluasi heuristik sering digunakan untuk mengestimasi nilai dari state non-terminal [5].

Setelah simpul daun atau batas kedalaman dievaluasi, nilai tersebut kemudian dirambatkan ke atas (*propagates upward*). Pada proses rambatan ini, simpul Maximizer akan selalu memilih nilai tertinggi dari anak-anaknya, sedangkan simpul Minimizer akan memilih nilai terendah. Rambatan ini terus berlanjut hingga mencapai *root node,* di mana keputusan akhir mengenai langkah terbaik diambil [5].

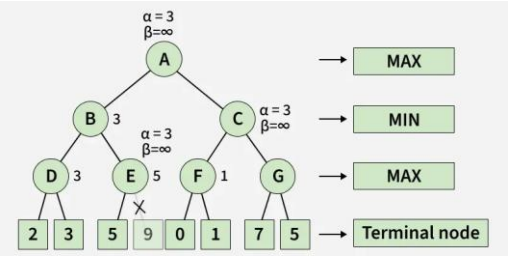


Gambar 2. Ilustrasi Minimax (sumber: [5])

Keunggulan utama algoritma Minimax adalah kemampuannya memberikan jaminan pemilihan langkah yang optimal di bawah asumsi perfect play. Meskipun demikian, algoritma ini memiliki keterbatasan teoretis dan praktis. Minimax mengeksplorasi seluruh kemungkinan langkah tanpa mempertimbangkan seberapa berguna cabang tersebut, yang pada akhirnya memicu kompleksitas waktu eksponensial. Oleh karena itu, Minimax standar sering kali tidak efisien dan sulit diskalakan untuk permainan yang sangat kompleks, serta membutuhkan sumber daya komputasi yang sangat tinggi apabila ingin melakukan pencarian yang lebih dalam [5]. Keterbatasan inilah yang menjadikan Minimax sekadar sebagai dasar (base) untuk teknik optimasi lanjutan, seperti Alpha-Beta Pruning.

C. *Alpha-Beta Pruning*

Alpha-Beta Pruning adalah optimasi terhadap Minimax. Algoritma ini mempertahankan dua batas nilai, yaitu alpha (α) dan beta (β). Alpha menyimpan nilai terbaik yang sejauh ini dapat dijamin oleh maximizing player, sedangkan beta menyimpan nilai terbaik yang sejauh ini dapat dijamin oleh minimizing player. Jika pada suatu titik diperoleh kondisi $\beta <= \alpha$, cabang tersebut tidak perlu dievaluasi lebih jauh karena tidak mungkin mengubah keputusan akhir [1, 2].



Gambar 3. Ilustrasi Minimax dengan Alpha-Beta Pruning (sumber: https://www.geeksforgeeks.org/artificial-intelligence/alpha-beta-pruning-in-adversarial-search-algorithms/)

Keunggulan penting Alpha-Beta Pruning adalah hasil akhir yang diperoleh tetap sama dengan Minimax biasa, selama urutan kandidat langkah dan fungsi evaluasi identik. Perbedaannya terletak pada jumlah node yang dievaluasi. Dengan move ordering yang baik, Alpha-Beta Pruning dapat memangkas lebih banyak cabang sehingga pencarian menjadi lebih cepat [2].

Dalam konteks Strategi Algoritma, Alpha-Beta Pruning dapat dikaitkan dengan prinsip branch and bound. Pencarian dilakukan pada ruang solusi berbentuk pohon. Ketika suatu cabang terbukti tidak dapat menghasilkan solusi yang lebih baik daripada batas yang sudah ditemukan, cabang tersebut tidak dilanjutkan [3]. Dengan demikian, kontribusi algoritma bukan hanya pada pemilihan langkah, tetapi juga pada efisiensi eksplorasi ruang solusi.

D. Breadth-First Search untuk Pencarian Jalur Terpendek

Breadth-First Search (BFS) merupakan salah satu algoritma traversal graf yang mengunjungi simpul-simpul graf secara sistematis. Traversal graf sendiri dapat dipahami sebagai proses mengunjungi simpul-simpul di dalam graf untuk melakukan operasi tertentu, misalnya membaca informasi pada simpul, memanipulasi nilai simpul, atau mencari solusi dari persoalan yang direpresentasikan sebagai graf [6]. Dalam konteks pencarian solusi berbasis graf, graf digunakan sebagai representasi persoalan, sehingga proses traversal pada graf dapat dipandang sebagai proses pencarian solusi pada ruang persoalan tersebut [6].

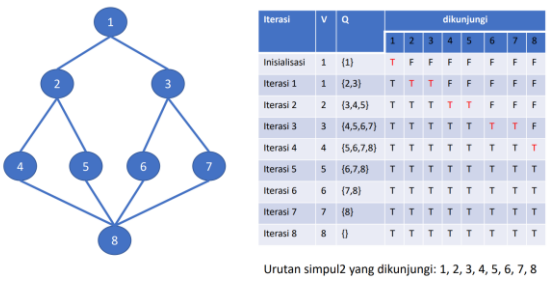
BFS termasuk ke dalam kategori pencarian tanpa informasi atau uninformed search. Pada jenis pencarian ini, algoritma tidak menggunakan informasi tambahan berupa fungsi heuristik untuk memperkirakan seberapa dekat suatu simpul dengan tujuan. Informasi yang digunakan hanya berasal dari struktur graf itu sendiri, yaitu simpul dan sisi yang menghubungkan simpul-simpul tersebut [6]. Hal ini membedakan BFS dari pencarian berbasis heuristik seperti Greedy Best First Search atau A*, yang menggunakan informasi tambahan untuk menentukan simpul mana yang lebih menjanjikan untuk dikunjungi lebih dahulu.

Prinsip utama BFS adalah mengunjungi simpul berdasarkan tingkat kedalaman atau jarak dari simpul awal. Traversal dimulai dari sebuah simpul awal v. Setelah simpul v dikunjungi, BFS akan mengunjungi seluruh simpul yang bertetangga langsung dengan v terlebih dahulu. Setelah semua tetangga langsung dikunjungi, pencarian dilanjutkan ke simpul-simpul yang bertetangga dengan simpul-simpul yang telah dikunjungi sebelumnya. Dengan demikian, BFS melakukan pencarian secara melebar, yaitu menyelesaikan semua simpul pada kedalaman tertentu sebelum berpindah ke kedalaman berikutnya [6].

Secara konseptual, urutan kunjungan BFS membentuk lapisan-lapisan pencarian. Simpul awal berada pada lapisan ke-0. Semua simpul yang bertetangga langsung dengan simpul awal berada pada lapisan ke-1. Simpul yang dapat dicapai melalui dua sisi dari simpul awal berada pada lapisan ke-2, dan seterusnya. Karena BFS mengunjungi simpul berdasarkan urutan lapisan tersebut, simpul yang pertama kali ditemukan pada suatu lapisan merupakan simpul yang dicapai dengan jumlah sisi minimum dari simpul awal. Oleh karena itu, pada graf tak berbobot, BFS dapat digunakan untuk mencari jalur terpendek berdasarkan jumlah sisi yang dilalui.

Dalam implementasinya, BFS membutuhkan beberapa struktur data utama. Pertama, graf dapat direpresentasikan menggunakan matriks ketetanggaan atau struktur ketetanggaan lain yang menyatakan hubungan antar simpul. Pada matriks ketetanggaan A, elemen aij bernilai 1 jika simpul i dan simpul j bertetangga, dan bernilai 0 jika keduanya tidak bertetangga [6]. Kedua, BFS menggunakan antrian atau queue untuk menyimpan simpul yang telah ditemukan tetapi belum sepenuhnya diproses. Ketiga, BFS menggunakan tabel Boolean dikunjungi untuk menandai apakah suatu simpul telah dikunjungi atau belum [6].

Penggunaan queue merupakan ciri penting BFS. Ketika sebuah simpul dikunjungi, simpul tersebut dimasukkan ke dalam queue. Selanjutnya, simpul diambil dari bagian depan queue, kemudian seluruh tetangganya yang belum dikunjungi dimasukkan ke bagian belakang queue. Mekanisme first-in first-out ini menyebabkan simpul yang ditemukan lebih awal akan diproses lebih awal. Akibatnya, BFS menjaga urutan pencarian tetap melebar dari simpul awal menuju lapisan-lapisan berikutnya [6].



Iterasi	V	Q	dikunjungi							
			1	2	3	4	5	6	7	8
Inisialisasi	1	{1}	T	F	F	F	F	F	F	F
Iterasi 1	1	{2,3}	T	T	T	F	F	F	F	F
Iterasi 2	2	{3,4,5}	T	T	T	T	T	F	F	F
Iterasi 3	3	{4,5,6,7}	T	T	T	T	T	T	T	F
Iterasi 4	4	{5,6,7,8}	T	T	T	T	T	T	T	T
Iterasi 5	5	{6,7,8}	T	T	T	T	T	T	T	T
Iterasi 6	6	{7,8}	T	T	T	T	T	T	T	T
Iterasi 7	7	{8}	T	T	T	T	T	T	T	T
Iterasi 8	8	{}	T	T	T	T	T	T	T	T

Urutan simpul2 yang dikunjungi: 1, 2, 3, 4, 5, 6, 7, 8

Gambar 4. Ilustrasi proses BFS menggunakan antrian dan larik dikunjungi (sumber: [6])

Selain menghasilkan urutan kunjungan, BFS juga dapat membentuk pohon BFS. Pohon BFS menunjukkan hubungan penemuan antar simpul selama proses traversal. Akar pohon BFS adalah simpul awal, sedangkan anak-anaknya adalah simpul yang pertama kali ditemukan dari simpul tersebut. Pohon BFS berguna untuk merekonstruksi jalur dari simpul awal menuju simpul tujuan. Jika setiap simpul menyimpan informasi parent atau simpul pendahulunya, maka jalur terpendek menuju simpul tujuan dapat diperoleh dengan menelusuri parent dari simpul tujuan kembali ke simpul awal.

Pada permainan Quoridor, BFS digunakan karena papan permainan dapat dimodelkan sebagai graf tak berbobot. Setiap petak pada papan 9x9 direpresentasikan sebagai simpul. Sisi antara dua simpul menyatakan bahwa pion dapat berpindah dari satu petak ke petak bertetangga dalam satu langkah. Namun, keberadaan wall dapat menghapus sisi tertentu karena pion tidak dapat melewati wall. Dengan demikian, graf papan Quoridor bersifat dinamis karena hubungan antar simpul dapat berubah setelah pemain memasang wall.

E. Fungsi Evaluasi Heuristik

Fungsi evaluasi heuristik diperlukan karena Minimax dibatasi pada depth tertentu. Dalam implementasi solver ini, evaluasi state dikalkulasi dari sudut pandang pemain yang sedang meminta rekomendasi. Formula matematis yang digunakan untuk menghitung bobot nilai state adalah sebagai berikut:

*score = (opponent_distance - player_distance) * 10 + (player_walls - opponent_walls) * 2 + mobility_score.*

Komponen opponent_distance - player_distance menjadi faktor dominan karena tujuan utama Quoridor adalah mencapai sisi seberang lebih cepat daripada lawan. Komponen sisa wall digunakan sebagai faktor tambahan karena wall adalah sumber daya strategis. Komponen mobility_score memperhitungkan perbedaan jumlah legal pawn moves antara pemain perspektif dan lawan. Jika pemain perspektif telah menang, nilai evaluasi dibuat sangat besar; jika lawan telah menang, nilai dibuat sangat kecil.

III. IMPLEMENTASI PROGRAM

Program pendukung dibuat menggunakan bahasa pemrograman Python dengan library Pygame sebagai antarmuka visual. Python digunakan karena mendukung implementasi algoritma secara ringkas dan mudah dibaca, sedangkan Pygame digunakan untuk menampilkan papan Quoridor, pion, wall, tombol kontrol, preview langkah, dan panel rekomendasi.

Program memungkinkan permainan Quoridor dilakukan oleh dua pemain secara manual. Pada suatu giliran, program dapat menghitung tiga rekomendasi langkah terbaik berdasarkan skor untuk pemain yang sedang berjalan. Setiap rekomendasi diperoleh melalui evaluasi Minimax dengan opsi Alpha-Beta Pruning. Skor tersebut digunakan untuk mengurutkan langkah dari yang paling menguntungkan hingga yang kurang menguntungkan bagi current player.

Gambar 5. Tampilan GUI Program (sumber: dokumen penulis)

A. Representasi State dan Move

State permainan direpresentasikan melalui posisi kedua pemain, kumpulan horizontal wall, kumpulan vertical wall, sisa wall masing-masing pemain, dan current player. Papan Quoridor direpresentasikan sebagai grid berukuran 9 x 9. Player 1 memulai permainan dari koordinat (8, 4) dan bertujuan mencapai row 0, sedangkan Player 2 memulai permainan dari koordinat (0, 4) dan bertujuan mencapai row 8.

Move dalam permainan dibagi menjadi dua jenis, yaitu pawn move dan wall move. Pawn move menyimpan koordinat tujuan pion, sedangkan wall move menyimpan orientasi wall dan posisi wall. Representasi ini membuat seluruh kandidat move dapat diproses secara seragam oleh algoritma pencarian.

Pemisahan antara state dan move penting karena Minimax perlu mensimulasikan banyak kemungkinan move. Setiap move diterapkan pada clone state, bukan langsung pada state asli. Dengan cara ini, program dapat mengevaluasi konsekuensi suatu move tanpa mengubah papan permainan yang sedang dimainkan.

B. Validasi Gerakan Pion dan Penempatan Wall

Setiap move yang akan dilakukan harus divalidasi terlebih dahulu. Pada gerakan pion, program memeriksa apakah koordinat tujuan masih berada di dalam papan, apakah petak tujuan dapat dicapai dari petak asal, dan apakah gerakan tersebut tidak melewati wall. Pion dapat bergerak satu langkah ke atas, bawah, kiri, atau kanan. Program juga mendukung jump lurus ketika pion lawan berada tepat di depan pemain dan tidak ada wall yang menghalangi lompatan.

Validasi wall dilakukan dengan beberapa syarat. Wall harus berada dalam batas papan, tidak boleh overlap dengan wall lain yang sudah ada, tidak boleh crossing secara ilegal, dan tidak boleh menutup seluruh jalur salah satu pemain menuju goal row. Syarat terakhir penting karena dalam Quoridor pemain boleh menghambat lawan menggunakan wall, tetapi tidak boleh membuat lawan sama sekali tidak memiliki jalan menuju tujuan.

Validasi ini membuat setiap state yang dievaluasi oleh Minimax tetap berada dalam aturan legal permainan Quoridor. Dengan demikian, rekomendasi langkah yang dihasilkan tidak hanya memiliki skor tinggi, tetapi juga dapat benar-benar dilakukan pada permainan.

C. Pencarian Jalur Terpendek dengan BFS

Breadth-First Search digunakan untuk mencari jalur terpendek dari posisi pemain menuju goal row. Papan Quoridor dimodelkan sebagai graf tak berbobot. Setiap petak papan merupakan simpul, sedangkan sisi menghubungkan dua petak yang bertetangga dan tidak dipisahkan oleh wall. Karena setiap perpindahan pion memiliki biaya yang sama, BFS dapat digunakan untuk memperoleh jarak terpendek berdasarkan jumlah langkah.

Dalam implementasi, BFS dimulai dari posisi pion pemain. Pencarian dilakukan dengan queue. Setiap petak yang dapat dicapai dan belum pernah dikunjungi dimasukkan ke dalam queue. Jika BFS menemukan salah satu petak pada goal row, maka jalur menuju petak tersebut dikembalikan. Jika tidak ada jalur yang ditemukan, maka pemain dianggap tidak memiliki akses menuju tujuan.

```python
def _shortest_path_cached(
    player: int,
    player_one_pos: Position,
    player_two_pos: Position,
    horizontal_walls_key: WallsKey,
    vertical_walls_key: WallsKey,
) -> Optional[tuple[Position, ...]]:
    horizontal_walls = frozenset(horizontal_walls_key)
    vertical_walls = frozenset(vertical_walls_key)
    start = player_one_pos if player == PLAYER_1 else player_two_pos
    goal_row = GOAL_ROWS[player]
    queue = deque([start])
    parent: dict[Position, Optional[Position]] = {start: None}

    while queue:
        current = queue.popleft()
        if current[0] == goal_row:
            path = []
            while current is not None:
                path.append(current)
                current = parent[current]
            return tuple(reversed(path))

        for dr, dc in DIRECTIONS:
            neighbor = (current[0] + dr, current[1] + dc)
            if (
                neighbor not in parent
                and in_bounds(neighbor)
                and _can_move_between_walls(
                    horizontal_walls, vertical_walls, current, neighbor
                )
            ):
                parent[neighbor] = current
                queue.append(neighbor)

    return None
```

Gambar 6. Source Code Pencarian Jalur Terpendek (sumber: dokumen penulis)

BFS digunakan pada dua bagian penting. Pertama, BFS menghitung panjang jalur terpendek pemain dan lawan menuju tujuan. Nilai ini digunakan dalam fungsi evaluasi heuristik. Kedua, BFS digunakan untuk memvalidasi penempatan wall. Jika setelah suatu wall dipasang salah satu pemain tidak lagi memiliki jalur menuju tujuan, maka wall tersebut tidak valid.

D. Fungsi Evaluasi Heuristik

Karena pencarian Minimax tidak dilakukan sampai akhir permainan, diperlukan fungsi evaluasi untuk menilai state pada kedalaman tertentu. Fungsi evaluasi menghitung skor dari sudut pandang pemain yang sedang diberi rekomendasi. Semakin tinggi skor, semakin menguntungkan state tersebut bagi pemain tersebut.

Evaluasi dilakukan dengan mempertimbangkan tiga komponen utama. Komponen pertama adalah selisih jarak terpendek lawan dan pemain menuju tujuan. Jika pemain lebih dekat ke tujuan dibanding lawan, maka skor menjadi lebih tinggi. Komponen kedua adalah selisih sisa wall. Pemain yang masih memiliki lebih banyak wall memiliki peluang lebih besar untuk mengatur jalur permainan. Komponen ketiga adalah mobility score, yaitu selisih jumlah gerakan pion legal antara pemain dan lawan.

```python
def evaluate(state: GameState, perspective_player: int) -> float:
    winner = state.winner()
    opponent = state.opponent(perspective_player)

    if winner == perspective_player:
        return WIN_SCORE
    if winner == opponent:
        return -WIN_SCORE

    player_distance = shortest_path_length(state, perspective_player)
    opponent_distance = shortest_path_length(state, opponent)

    if math.isinf(player_distance):
        return -WIN_SCORE
    if math.isinf(opponent_distance):
        return WIN_SCORE

    player_walls = state.walls_remaining[perspective_player]
    opponent_walls = state.walls_remaining[opponent]
    player_mobility = len(get_legal_pawn_moves(state, perspective_player
))
    opponent_mobility = len(get_legal_pawn_moves(state, opponent))

    return (
        (opponent_distance - player_distance) * 10
        + (player_walls - opponent_walls) * 2
        + (player_mobility - opponent_mobility)
    )
```

Gambar 7. Source Code Fungsi Evaluasi Heuristik (sumber: dokumen penulis)

E. Minimax dengan Alpha-Beta Pruning

Minimax digunakan untuk mengevaluasi konsekuensi langkah beberapa giliran ke depan. Jika giliran berada pada pemain yang sedang dievaluasi, simpul dianggap sebagai maximizing node. Sebaliknya, jika giliran berada pada lawan, simpul dianggap sebagai minimizing node. Pada depth 0 atau state terminal, fungsi evaluasi dipanggil untuk memberikan nilai terhadap state tersebut.

Alpha-Beta Pruning ditambahkan untuk mengurangi jumlah cabang yang perlu dievaluasi. Nilai alpha menyimpan nilai terbaik sementara bagi maximizing player, sedangkan beta menyimpan nilai terbaik sementara bagi minimizing player. Jika pada suatu cabang diperoleh kondisi beta <= alpha, maka cabang tersebut tidak perlu dilanjutkan karena tidak akan memengaruhi keputusan akhir.

```python
def minimax_alpha_beta(state: GameState, depth: int, alpha: float, beta: float,
    perspective_player: int, stats: SearchStats, use_alpha_beta: bool = True, ) -> float:
    stats.nodes_visited += 1

    if depth == 0 or state.is_terminal():
        return evaluate(state, perspective_player)

    legal_moves = _ordered_moves(state, perspective_player)
    if not legal_moves:
        return evaluate(state, perspective_player)

    maximizing = state.current_player == perspective_player

    if maximizing:
        value = float("-inf")
        for move in legal_moves:
            child = state.apply_move(move)
            value = max(
                value,
                minimax_alpha_beta(child, depth - 1, alpha, beta, perspective_player,
                    stats, use_alpha_beta, ),
            )
            if use_alpha_beta:
                alpha = max(alpha, value)
            if use_alpha_beta and beta <= alpha:
                stats.pruned_branches += 1
                break
        return value

    value = float("inf")
    for move in legal_moves:
        child = state.apply_move(move)
        value = min(
            value,
            minimax_alpha_beta(child,depth - 1, alpha, beta, perspective_player,
                stats, use_alpha_beta, ),
        )
        if use_alpha_beta:
            beta = min(beta, value)
        if use_alpha_beta and beta <= alpha:
            stats.pruned_branches += 1
            break
    return value
```

Gambar 8. Source Code Algoritma Minimax (sumber: dokumen penulis)

Jika Alpha-Beta Pruning dimatikan, algoritma tetap menjalankan Minimax biasa tanpa melakukan pemangkasan. Dengan demikian, program dapat membandingkan jumlah node yang dikunjungi ketika pemangkasan digunakan dan ketika pemangkasan tidak digunakan. Perbandingan ini menjadi dasar pengujian efektivitas Alpha-Beta Pruning pada permainan Quoridor.

F. Kompleksitas Pencarian

Kompleksitas pencarian Minimax dipengaruhi oleh branching factor dan kedalaman pencarian. Jika b adalah rata-rata jumlah langkah legal pada setiap state dan d adalah kedalaman pencarian, maka kompleksitas Minimax dapat dinyatakan sebagai $O(b^d)$. Pada Quoridor, nilai b dapat menjadi besar karena selain gerakan pion, pemain juga memiliki banyak kemungkinan penempatan wall.

Alpha-Beta Pruning tidak mengubah kompleksitas terburuk Minimax, tetapi dapat mengurangi jumlah node yang dikunjungi pada banyak kasus. Efektivitasnya bergantung pada urutan evaluasi langkah. Jika langkah yang baik dievaluasi lebih awal, maka nilai alpha dan beta dapat diperketat lebih cepat sehingga lebih banyak cabang dapat dipangkas.

Dalam program ini, kedalaman pencarian dibatasi pada nilai tertentu agar proses rekomendasi tetap berjalan dalam waktu yang wajar. Pembatasan kandidat wall, penggunaan BFS untuk menghitung jarak terpendek, dan move ordering digunakan sebagai kompromi antara kualitas rekomendasi dan waktu komputasi.

IV. HASIL DAN PEMBAHASAN

Pengujian dilakukan untuk melihat pengaruh Alpha-Beta Pruning terhadap proses pencarian langkah pada permainan Quoridor. Pengujian dilakukan dengan membandingkan dua mode pencarian, yaitu Minimax tanpa Alpha-Beta Pruning dan Minimax dengan Alpha-Beta Pruning. Kedua mode dijalankan pada state permainan yang sama, depth yang sama, fungsi evaluasi yang sama, kandidat move yang sama, dan urutan move yang sama. Dengan demikian, perbedaan jumlah node yang dikunjungi dapat dikaitkan dengan penggunaan Alpha-Beta Pruning.

Metrik yang diamati dalam pengujian meliputi nodes visited, pruned branches, search time, dan rekomendasi langkah terbaik berdasarkan skor. Nodes visited menunjukkan jumlah node yang dievaluasi selama pencarian. Pruned branches menunjukkan jumlah cabang yang dipangkas oleh Alpha-Beta Pruning. Search time menunjukkan waktu yang diperlukan program untuk menghasilkan rekomendasi. Rekomendasi terbaik digunakan untuk melihat apakah penggunaan Alpha-Beta Pruning mengubah hasil keputusan atau hanya mengoptimasi proses pencarian.

Pengujian dilakukan pada tiga skenario. Skenario pertama adalah state awal permainan, yaitu kondisi ketika belum ada pemain yang bergerak. Skenario kedua adalah mid game, yaitu kondisi ketika permainan sudah berjalan dan terdapat konfigurasi posisi serta wall tertentu. Skenario ketiga adalah kondisi satu langkah sebelum menang, yaitu ketika pemain memiliki move langsung menuju goal row.

Tabel I. Hasil Pengujian Performa Pencarian

Skenario	AB On/Off	Nodes Visited	Pruned Branches	Search Time
Awal permainan	Off	2284	0	0.804 s
Awal permainan	On	582	149	0.700 s
Mid game	Off	2214	0	0.588 s
Mid game	On	754	143	0.460 s
Satu langkah sebelum menang	Off	1	0	0.000 s
Satu langkah sebelum menang	On	1	0	0.000 s

Selain performa pencarian, hasil rekomendasi terbaik pada setiap skenario juga diamati. Rekomendasi terbaik merupakan move dengan skor evaluasi tertinggi pada state tersebut. Hasil rekomendasi terbaik ditunjukkan pada Tabel II.

Tabel II. Hasil Rekomendasi Langkah Terbaik

Skenario	Rekomendasi Terbaik	Skor
Mid game	Move pawn to (7, 4)	10.00
Satu langkah sebelum menang	V-wall at (5, 2)	8.00
Satu langkah sebelum menang	Move pawn to (0, 5)	100000.00

Berdasarkan Tabel I, penggunaan Alpha-Beta Pruning berhasil menurunkan jumlah node yang dikunjungi pada skenario awal permainan. Pada mode Minimax tanpa Alpha-Beta Pruning, jumlah node yang dikunjungi adalah 2284 node. Setelah Alpha-Beta Pruning diaktifkan, jumlah node turun menjadi 582 node. Penurunan ini setara dengan sekitar 74,5%. Selain itu, terdapat 149 cabang yang dipangkas. Hal ini menunjukkan bahwa Alpha-Beta Pruning mampu mengurangi jumlah cabang yang perlu dievaluasi tanpa mengubah proses evaluasi state.

Pada skenario mid game, Alpha-Beta Pruning juga menunjukkan pengurangan jumlah node yang signifikan. Jumlah node turun dari 2214 menjadi 754 node, atau berkurang sekitar 65,9%. Jumlah cabang yang dipangkas adalah 143. Hasil ini menunjukkan bahwa Alpha-Beta Pruning tetap efektif tidak hanya pada state awal, tetapi juga pada state ketika permainan sudah berjalan dan konfigurasi papan menjadi lebih kompleks.

Dari sisi waktu pencarian, Alpha-Beta Pruning juga mempercepat proses pencarian. Pada skenario awal permainan, search time turun dari 0.804 detik menjadi 0.700 detik. Pada skenario mid game, search time turun dari 0.588 detik menjadi 0.460 detik. Penurunan waktu pada mid game lebih terlihat dibandingkan skenario awal permainan. Hal ini menunjukkan bahwa pengurangan node dapat berdampak pada waktu pencarian, meskipun besar penurunannya tidak selalu sebanding secara langsung dengan jumlah node yang berkurang.

Perbedaan antara penurunan jumlah node dan penurunan waktu dapat terjadi karena biaya komputasi pada setiap node tidak selalu sama. Beberapa node membutuhkan proses tambahan seperti validasi wall, pencarian jalur terpendek dengan BFS, dan pembangkitan kandidat move. Oleh karena itu, nodes visited menjadi metrik yang lebih langsung untuk menunjukkan efek Alpha-Beta Pruning terhadap ruang pencarian, sedangkan search time menunjukkan dampaknya terhadap waktu eksekusi program.

Pada skenario satu langkah sebelum menang, kedua mode hanya mengunjungi 1 node, tidak melakukan pruning, dan memiliki search time 0.000 detik. Hal ini terjadi karena state tersebut sudah sangat dekat dengan terminal state. Move terbaik yang ditemukan adalah Move pawn to (0, 5) dengan skor 100000.00. Skor tersebut merupakan nilai kemenangan pada fungsi evaluasi. Dengan demikian, algoritma berhasil mengenali move yang langsung memenangkan permainan.

Hasil pada Tabel II juga menunjukkan bahwa rekomendasi terbaik tetap sama baik ketika Alpha-Beta Pruning dimatikan maupun diaktifkan. Pada skenario awal permainan, rekomendasi terbaik adalah Move pawn to (7, 4) dengan skor 10.00. Pada skenario mid game, rekomendasi terbaik adalah V-wall at (5, 2) dengan skor 8.00. Pada skenario satu langkah sebelum menang, rekomendasi terbaik adalah Move pawn to (0, 5) dengan skor 100000.00. Kesamaan rekomendasi ini penting karena Alpha-Beta Pruning seharusnya tidak mengubah hasil akhir Minimax, melainkan hanya memangkas cabang yang tidak memengaruhi keputusan akhir.

Pada skenario awal permainan, rekomendasi Move pawn to (7, 4) menunjukkan bahwa algoritma menilai langkah maju menuju goal row sebagai pilihan terbaik. Hal ini sesuai dengan fungsi evaluasi yang memberi bobot besar pada jarak terpendek menuju tujuan. Karena pada awal permainan belum terdapat wall yang menghalangi jalur, move yang memperpendek jarak pemain menuju goal row memperoleh skor tertinggi.

Pada skenario mid game, rekomendasi terbaik berubah menjadi V-wall at (5, 2). Hal ini menunjukkan bahwa pada kondisi tertentu, algoritma tidak selalu memilih gerakan pion. Ketika pemasangan wall dapat memperbaiki posisi strategis pemain atau menghambat jalur lawan, wall move dapat memperoleh skor lebih tinggi dibanding pawn move. Dengan demikian, fungsi evaluasi tidak hanya mempertimbangkan kedekatan pemain ke goal row, tetapi juga dampak langkah terhadap posisi lawan.

Pada skenario satu langkah sebelum menang, algoritma memberikan rekomendasi Move pawn to (0, 5) dengan skor 100000.00. Skor ini menunjukkan bahwa state setelah move tersebut merupakan state kemenangan. Dengan kata lain, algoritma berhasil menemukan langkah kemenangan ketika langkah tersebut tersedia. Hal ini memperlihatkan bahwa fungsi terminal pada Minimax bekerja dengan benar, karena state menang diberi nilai sangat besar sehingga selalu dipilih sebagai rekomendasi terbaik.

Secara keseluruhan, hasil pengujian menunjukkan bahwa Alpha-Beta Pruning efektif mengurangi jumlah node yang dikunjungi pada pencarian Minimax. Pada skenario awal dan mid game, jumlah node berkurang secara signifikan ketika Alpha-Beta Pruning diaktifkan. Pada saat yang sama, rekomendasi terbaik tetap sama. Hal ini membuktikan bahwa Alpha-Beta Pruning berperan sebagai optimasi pencarian, bukan sebagai pengubah hasil evaluasi.

Hasil pengujian juga menunjukkan bahwa algoritma berhasil memberikan rekomendasi strategis sesuai kondisi permainan. Pada awal permainan, algoritma memilih langkah maju. Pada mid game, algoritma memilih pemasangan wall. Pada kondisi satu langkah sebelum menang, algoritma memilih move yang langsung memenangkan permainan. Dengan demikian, penerapan Minimax dengan Alpha-Beta Pruning berhasil digunakan untuk mendukung optimasi strategi pada permainan Quoridor.

V. KESIMPULAN

Penerapan Minimax dengan Alpha-Beta Pruning berhasil digunakan untuk optimasi strategi pada permainan Quoridor. Minimax digunakan untuk mengevaluasi kemungkinan langkah berdasarkan konsekuensi beberapa giliran ke depan, sedangkan Alpha-Beta Pruning digunakan untuk mengurangi jumlah node yang perlu dikunjungi tanpa mengubah hasil keputusan akhir.

Hasil pengujian menunjukkan bahwa Alpha-Beta Pruning mampu meningkatkan efisiensi pencarian. Pada skenario awal permainan, jumlah node berkurang dari 2284 menjadi 582 node. Pada skenario mid game, jumlah node berkurang dari 2214 menjadi 754 node. Meskipun jumlah node yang dikunjungi berkurang signifikan, rekomendasi langkah terbaik yang dihasilkan tetap sama dengan Minimax tanpa pruning.

Algoritma juga berhasil memberikan rekomendasi langkah-langkah terbaik hingga Langkah kemenangan pada skenario satu langkah sebelum menang. Rekomendasi Move pawn to (0, 5) menghasilkan skor 100000.00, yang menunjukkan state kemenangan. Dengan demikian, Minimax dengan Alpha-Beta Pruning terbukti dapat digunakan untuk membantu memilih langkah strategis pada permainan Quoridor secara lebih efisien.

Secara keseluruhan, Alpha-Beta Pruning berperan sebagai optimasi terhadap Minimax dalam pencarian strategi Quoridor. Metode ini mampu memangkas cabang pencarian yang tidak relevan, mengurangi jumlah node yang dievaluasi, dan tetap mempertahankan kualitas keputusan yang dihasilkan.

LAMPIRAN

Link repositori GitHub program: https://github.com/wafhr/Simple-Quoridor-Solver

UCAPAN TERIMA KASIH

Penulis mengucapkan puji dan syukur kepada Tuhan Yang Maha Esa atas rahmat dan karunia-Nya sehingga makalah ini dapat diselesaikan dengan baik. Penulis juga mengucapkan terima kasih kepada orang tua penulis atas doa, dukungan, dan motivasi yang senantiasa diberikan selama proses penyusunan makalah ini.

Penulis menyampaikan terima kasih kepada dosen pengampu mata kuliah IF2211 Strategi Algoritma atas ilmu, bimbingan, dan materi perkuliahan yang menjadi dasar dalam penyusunan makalah ini. Selain itu, penulis juga berterima

kasih kepada sosok yang telah menjadi sumber semangat dan motivasi selama proses pengerjaan makalah ini.

REFERENSI

[1] S. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, 4th ed. Pearson, 2021.

[2] D. E. Knuth and R. W. Moore, "An analysis of alpha-beta pruning," Artificial Intelligence, vol. 6, no. 4, pp. 293-326, 1975, doi: 10.1016/0004-3702(75)90019-3.

[3] R. Munir, "Algoritma branch and bound (Bagian 1)," Bahan Kuliah IF2211 Strategi Algoritma, Program Studi Teknik Informatika, Institut Teknologi Bandung, 2026.

[4] Gigamic, "Quoridor," [Online]. Available: https://en.gigamic.com/index.php?controller=attachment&id_attachment=467. Accessed: Jun. 19, 2026.

[5] GeeksforGeeks, "Minimax Algorithm in Game Theory | Set 1 (Introduction)," GeeksforGeeks, May 27, 2026. [Online]. Available: https://www.geeksforgeeks.org/dsa/minimax-algorithm-in-game-theory-set-1-introduction/. Accessed: Jun. 19, 2026.

[6] R. Munir, "Breadth First Search (BFS) dan Depth First Search (DFS) - Bagian 1," Bahan Kuliah IF2211 Strategi Algoritma, Program Studi Teknik Informatika, Institut Teknologi Bandung, 2026.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 19 Juni 2026

Wafiq Hibban Robbany (13524016)