JITET (Jurnal Informatika dan Teknik Elektro Terapan)
Vol. 13 No. 3S1, pISSN: 2303-0577 eISSN: 2830-7062 http://dx.doi.org/10.23960/jitet.v13i3S1.7763

IMPLEMENTASI PATHFINDING DENGAN ALGORITMA A* PADA GAME 2D ROGUELIKE SURVIVAL MENGGUNAKAN UNITY AI NAVIGATION SYSTEM

Apdya Amrul Atqia[1*], Mamay Syani[2]

[1,2] Politeknik TEDC Bandung; Jl. Politeknik Jl. Pesantren No.2, Cibabat, Kec. Cimahi Utara, Kota Cimahi, Jawa Barat 40513; telp (022) 6645951

Keywords:
Pathfinding;
Algoritma A*;
NavMesh;
Roguelike 2D;
Unity AI.

Corespondent Email:
apdyaamrul@gmail.com

Abstrak. *Game merupakan sebuah struktur yang memiliki ketentuan, objektif, dan prosedur yang mengikat pemain terlepas dari konsekuensi di dunia nyata. Sebuah game harus menantang, juga menghibur pemain, serta memberikan hasil akhir yang pasti dan terukur seperti menang dan kalah. Game bergenre roguelike memberi tantangan kepada pemain untuk memenangkan permainan dalam sekali coba dan mengulanginya dari awal jika kalah. Dalam game roguelike 2d seperti Vampire Survivor, pergerakan karakter musuh dalam mengejar karakter pemain pada umunya linear dan tak mengenal rintangan yang dapat menghadang arah pengejaran karakter pemain. Metode pathfinding dengan Algoritma A* menjadi standar dalam mengatasi pergerakan karakter dalam game. Tetapi pathfinding statis belum cukup mengatasi pergerakan karakter yang dinamis, maka dalam penelitian ini akan digunakan Unity AI Navigation System bersama ekstensi NavMeshPlus untuk membuat beragam variasi area navigasi NavMesh 2D yang luasnya dinamis saat runtime, kemudian pathfinding karakter musuh yang dinamis dalam mendeteksi rintangan yang ada. Hasil penelitian dengan pembuatan tiga tipe area navigasi grass dan tiga tipe area navigasi sand, penerapan agen navigasi pada karakter pemain dan dua tipe karakter musuh berhasil membuat pergerakan karakter musuh game roguelike survival yang dinamis dalam mengenali rintangan yang berasal dari objek rintangan, area navigasi yang tak bisa dilalui, dan agen navigasi lain.*

Abstract. *A game is a structured system with rules, objectives, and procedures that bind players regardless of real-world consequences. A game must be challenging, entertaining, and provide a definite and measurable outcome, such as winning or losing. Roguelike games challenge players to win the game in a single attempt and restart from the beginning if they lose. In 2D roguelike games like Vampire Survivor, enemy character movement in pursuing the player character is generally linear and does not encounter obstacles that could block the player's path. The A* algorithm is the standard method for handling character movement in games. However, static pathfinding is insufficient for handling dynamic character movement. Therefore, this study will utilize the Unity AI Navigation System alongside the NavMeshPlus extension to create various dynamic 2D NavMesh navigation areas whose sizes change dynamically at runtime, followed by dynamic enemy character pathfinding to detect existing obstacles. The research results, which involved creating three types of grass navigation areas and three types of sand navigation areas, and applying navigation agents to player characters and two types of enemy characters, successfully enabled dynamic enemy character movement in roguelike survival games to recognize obstacles from obstacle objects, impassable navigation areas, and other navigation agents.*

1. PEfNDAHULUAN

Sejak video *game* pertama rilis di dunia, beragan genre *game* telah bermunculan, termasuk genre *role playing game* (RPG) di mana pemain akan mengendalikan karakter untuk berinteraksi dengan dunia *game* yang dimainkan [1]. Pada era awal kepopuleran RPG di dekade 1980-an, sebuah *game* RPG bernama *Rogue* muncul dan menjadi pionir genre *roguelike* yang memadukan elemen *permadeath* dengan *procedural generation* yang membuat *game* menjadi cukup menantang sehingga tingkat kesulitan genre ini ada diantara *midcore* dan *hardcore* [2].

Saat ini genre *roguelike* telah diintegrasikan dengan genre *game* lain, seperti genre *survival* dalam *game* berformat 2 dimensi (2D) bernama *Vampire Survivor* yang mengimplementasikan elemen tersebut bersama dengan elemen *permadeath* pada sesi permainannya yaitu bertahan dari serangan musuh yang muncul tanpa henti. Dalam menghadapi hal tersebut, pemain akan mengontrol pergerakan karakter pada area yang akan terus meluas seiring perpindahan posisi mereka. Sementara itu karakter pemain akan mengaktifkan serangan secara otomatis setelah waktu *cooldown* serangan berakhir. Musuh yang dikalahkan akan memberikan nilai *experience* (*exp*) yang ketika mencapai nilai tertentu membuat karakter pemain naik level dan memunculkan implementasi *procedural generation* berupa opsi serangan atau *powerup* yang dapat dipilih oleh pemain.

Setelah meninjau metode pergerakan karakter musuh pada *game* tersebut diketahui bahwa metode yang digunakan adalah pergerakan langsung pada fisik objek *game*. Dengan metode tersebut karakter musuh tidak dapat mengenali adanya objek yang dapat menjadi rintangan dalam pergerakan mereka ke arah karakter pemain.

Berdasarkan penelitian pengembangan *game roguelike* terdahulu oleh Hendi Hermawan dan Hari Setiyani yang berjudul "Implementasi Algoritma A-Star Pada Permainan Komputer *Roguelike* Berbasis Unity" dapat diketahui bahwa hasil penelitian membuktika pergerakan musuh ke arah karakter pemain menggunakan metode *pathfinding* dengan algoritma A* berhasil dilakukan. Karakter musuh dapat menghindari rintangan yang ada untuk mencari jalur terbaik ke arah karakter pemain [3]. Dikarenakan karakter musuh yang hanya akan bergerak setelah karakter pemain selesai bergerak, dan ukuran area permainan yang tetap membuat penerapan *pathfinding* pada penelitian ini bersifat statis.

Berdasarkan penelitian pengembangan *game* yang dilakukan oleh Moh. Zikky yang berjudul "*Review of A* (A Star) Navigation Mesh Pathfinding as the Alternative of Artificial Intelligent for Ghosts Agent on the Pacman Game*" menyatakan bahwa *pathfinding* karakter menggunakan *navigation mesh* (*NavMesh*) lebih efektif dibandingkan dengan menggunakan metode *pathfinding* biasa yang bergerak ke titik terdekat berdasarkan nilai *cost* yang sudah diatur terlebih dahulu seperti metode *waypoint graph.* Dengan *NavMesh* karakter akan bergerak lurus secara dinamis menghindari rintangan ke area atau poligon tujuan yang berbeda dengan poligon awal karakter berada. Untuk hasil penelitian ini, diketahui bahwa algoritma A* merupakan algoritma *pathfinding* pada *NavMesh* terbaik setelah dibandingkan dengan algoritma *Dijkstra* [4].

Berdasarkan temuan-temuan tersebut, penelitian ini akan mengimplementasikan metode *pathfinding* dengan algoritma A* menggunakan sistem navigasi yang dimiliki Unity *game engine* bersama dengan ekstensi sistem navigasinya *NavMeshPlus* untuk menggantikan mekanisme pergerakan karakter musuh yang sudah ada sebelumnya pada *game roguelike* bergaya *Vampire Survivor* sehingga memungkinkan navigasi karakter musuh dalam mengejar karakter pemain berjalan secara dinamis dengan kemampuan menghindari sumber rintangan secara otomatis, dan memungkinkan pembuatan area navigasi *NavMesh* 2D dengan luas yang dinamis saat *runtime* juga memiliki tipe area yang bervariasi.

2. TINJAUAN PUSTAKA

2.1. *Game*

Definisi *game* secara umum merujuk pada sebuah sistem formal yang bersifat tertutup. Sistem berarti sebuah struktur yang terdiri atas ketentuan, objektif, dan prosedur yang mengatur dan mengikat pemain. Sifat tertutup menunjukkan bahwa segala sesuatu yang terjadi

dalam *game* terlepas dari aturan dunia nyata, dan sebaliknya. Sistem *game* tersebut harus menghadirkan tantangan yang menghibur bagi pemain serta memiliki hasil akhir yang pasti dan terukur, seperti menang atau kalah [5].

Awalnya, definisi *game* mencakup baik *game* tradisional non-digital maupun *game* modern digital, namun kini istilah *"game"* lebih identik dengan segala bentuk *game* yang menggunakan perangkat *display* yang juga dikenal sebagai *video game.* Pergeseran makna ini terjadi karena perkembangan *video game* yang pesat yang merepresentasikan modernisasi *game* tradisional dengan berbagai genrenya termasuk RPG menjadi digital, tetapi juga menunjukkan perubahan minat bermain *game* yang sangat mudah ditemui dan diakses di berbagai media digital saat ini [1], [6], [7, Ch. 1–4].

2.2. *Pathfinding*

Pathfinding merupakan metode pencarian jalur—serangkaian *node* atau posisi yang saling terhubung—teroptimal dalam suatu *map*/*grid*—area navigasi—untuk dilalui agen ke *node* tujuan. Optimalitas suatu jalur ditentukan oleh: (1) Jarak tempuh, (2) Nilai *cost*, (3) Keberadaan rintangan yang berdampak signifikan dengan sifat statis (diam) atau dinamis (bergerak)—sifat ini menentukan klasifikasi *pathfinding* [8].

2.3. *Algoritma A**

A* (*A-Star*) merupakan algoritma *pathfinding* heuristis turunan algoritma *Dijkstra* yang mengalkulasikan nilai *cost* aktual dan *cost* heuristis ke *node* akhir [3], [8], [9], [10]. Kalkulasi perhitungan *cost* algoritma A* tersebut adalah sebagai berikut:

$$f(n) = g(n) + h(n) \qquad (1)$$

Keterangan:
$f(n)$: Notasi estimasi total *cost pathfinding*
$g(n)$: Notasi *cost* aktual.
$h(n)$: Notasi *cost* heuristis.

Cost aktual berasal dari perhitungan nilai *cost node n* dengan jarak *node n* ke *node* awal. Sedangkan *cost* heuristis didapatkan dari fungsi perhitungan prediksi jarak dari *node n* ke *node* tujuan berdasarkan arah pergerakan yang dapat dilakukan, jika hanya bergerak secara horizontal dan vertikal maka *Manhattan distance* dapat digunakan, jika pergerakan diagonal dibolehkan dan *cost* sama dengan pergerakan horizontal atau vertikal maka *Chebyshev distance* dapat digunakan, dan jika pergerakan dapat dilakukan ke segala arah maka *Euclidean distance* dapat digunakan [11].

Dengan perhitungan *cost* tersebut akan menghasilkan pencarian jalur yang cepat dan efisien, membuat algoritma A* sangat populer bahkan menjadi standar *de facto pathfinding* dalam pengembangan *game* [8], [10], [4].

2.4. *Navigation Mesh*

Navigation Mesh (*NavMesh*) merupakan struktur jaringan poligon cembung yang dibentuk oleh pemukaan objek 3D sebagai metode navigasi karakter yang menjamin pergerakan yang bebas selama tujuan navigasi berada pada poligon yang sama, dan jika tidak maka akan dilakukan proses *pathfinding* ke poligon tujuan Bentuk serta ukuran sebuah poligon ini dipengaruhi oleh keberadaan objek rintangan dan tipe area lain yang bersentuhan.

Beberapa *game engine* seperti Unity mengaplikasikan konsep *triangulation optimization* pada *NavMesh* sehingga poligon berbentuk kompleks akan dipecah menjadi beberapa bagian berbentuk segitiga sehingga membuat proses *pahfinding* yang dilakukan algoritma seperti A* hanya menyusuri *centroid*—titik tengah sebuah poligon—setiap poligon sebagai *node* pergerakan ke arah tujuan saja [4]. Setelah jalur teroptimal ditemukan akan dilakukan *post-processing path smoothing* atau *string pulling* sehingga membentuk sebuah jalur aktual yang lurus namun juga menghindari rintangan [12].

2.5. Unity *Game Engine*

Unity merupakan salah satu *game engine* terbaik dan terpopuler dalam pengembangan *game* 2D dan 3D [13]. Unity mendukung fitur-fitur yang lengkap lewat *Scripting* API miliknya yang mengintegrasikan dan mengatur seluruh struktur aset dalam *game*, terutama pada pembuatan *script mono behaviour* sebagai komponen yang dapat digunakan oleh objek *game* apa saja. Penambahan dan penggunaan komponen ini dapat dengan mudah diakses melalui panel *inspector* [14].

2.6. Unity *AI Navigation System*

Unity memiliki sistem navigasi yang mengimplementasikan *NavMesh* menggunakan algoritma *pathfinding* A*, yang memungkinkan

pergerakan agen navigasi yang dinamis di beragam area navigasi, dapat mengenali beragam sumber rintangan, dan pembuatan *NavMesh* saat *runtime* [10], [15], [4]. Dikarenakan pergerakan karakter pada *NavMesh* memungkinkan ke segala arah, maka perhitungan heuristis yang digunakan A* adalah *Euclidean distance.*

Berikut dijelaskan komponen-komponen sistem navigasi yang digunakan pada penelitian ini:

2.6.1. *Navigation Window*

Jendela untuk mengonfigurasi tipe agen dan area navigasi dalam *game.* Parameter pengaturan ukuran, ketinggian disediakan pada bagian agen tidak akan berdampak pada proses navigasi 2D, sehingga hanya perlu mengatur nama agennya saja.

Pada bagian area navigasi terdapat tiga tipe area bawaan: (1) *Walkable* sebagai area navigasi *default,* (2) *Not Walkable* sebagai area yang tidak bisa dilalui, (3) *Jump* yang digunakan untuk berpindah antar area navigasi yang terpisah oleh area *Not Walkable*. Selain tiga area tersebut terdapat 29 area kosong yang dapat diberikan nama sehingga membuat tipe area navigasi baru. Setiap area kosong memiliki nilai *cost* bernilai satu secara *default* untuk digunakan dalam perhitungan jarak aktual algoritma A*. Untuk keperluan *debugging*, setiap tipe area navigasi memiliki warna yang unik agar mudah dibedakan pada jendela *scene*.

2.6.2. *NavMesh Agent*

Komponen yang membuat sebuah objek menjadi agen navigasi dengan empat bagian parameter pengaturan. Pada bagian pertama terdapat parameter yang mengatur tipe agen navigasi, dan parameter yang berfungsi untuk mengatur nilai *offset* agar posisi agen sinkron terhadap posisi *transform*—komponen yang mengatur posisi, rotasi, dan *scale* sebuah objek—jika *anchor point* pada objek tidak berada di bagian dasar.

Bagian kedua *steering* yang mengatur parameter pergerakan agen, hanya parameter yang mengatur kecepatan dan akselerasi maksimal saja yang digunakan. Hal tersebut dikarenakan hanya perilaku agen karakter musuh yang mengajar karakter pemain tanpa henti saja yang dibutuhkan. Sedangkan parameter *speed* pada karakter pemain digunakan secara tidak langsung oleh metode pergerakannya

Bagian ketiga *obstacle avoidance* yang mengatur penghindaran rintangan, dengan parameter yang mengatur radius penghindaran sekaligus lebar agen di *scene* 2D, tinggi minimum untuk melewati bawah rintangan yang menjadi tinggi agen di *scene* 2D*,* kualitas akurasi penghindaran rintangan, dan nilai prioritas yang menjadi indikator apakah agen akan menghindari agen lain yang memiliki prioritas lebih tinggi.

Bagian terakhir yang mengatur *pathfinding,* terdapat parameter area *mask* yang mengatur area navigasi yang dapat dilalui oleh agen, dan parameter *auto repath* yang jika diaktifkan membuat agen mencari jalur baru ketika rintangan muncul pada jalur yang sudah ada.

2.7. *2D NavMeshPlus*

Merupakan ekstensi Unity AI *Navigation System* dari pihak ketiga yang didesain untuk kebutuhan navigasi dinamis pada *scene* 2D menggunakan objek 2D seperti *Tilemap, Sprite Renderer,* dan *Collider2D* [16].

Komponen-komponen ekstensi yang digunakan pada penelitian akan diletakan dalam objek *Grid* atau turunannya untuk mendukung penggunaan objek *Tilemap* sebagai sumber koleksi objek. Berikut komponen-komponen ekstensi ini yang digunakan pada penelitian:

2.7.1. *Navigation Surface*

Komponen yang melakukan proses pembuatan *NavMesh* (*baking*) dan perubahannya (*rebake*) dari objek 2D yang proses koleksinya ditangani oleh komponen *Navigation CollectSources2d.*

NavMesh yang dibuat berlaku untuk sebuah tipe agen tertentu dengan sebuah tipe area navigasi *default* dengan parameter koleksi objek seperti berikut: (1) Opsi asal objek yang akan dikoleksi seperti seluruh objek di *scene,* objek yang beririsan dengan area/agen navigasi saja atau objek turunan *Navigation Surface/Modifier,* (2) Tipe *layer mask* atau lapisan fisik objek, (3) Sumber geometri dengan opsi *render meshes* untuk objek berkomponen *Mesh Filter* atau *physical collider* untuk objek berkomponen *Collider2D.*

2.7.2. *Navigation CollectSources2d*

Selain berfungsi untuk mengoleksi objek 2D, komponen ini juga yang mengatur orientasi *area navigasi* dengan merotasikan objek pada sumbu x sebesar -90 unit sehingga perspektif *grid* menjadi *top down*. Oleh karenanya

komponen ini akan diletakan pada objek turunan *Grid* bersama komponen *Navigation Surface* utama. sehingga perspektif tidak berubah saat objek dirotasi dan komponen *surface* dapat mengoleksi objek 2D.

2.7.3. *Navigations Modifier*

Komponen untuk mengubah tipe area navigasi *default* di komponen *surface* menjadi tipe tertentu atau menghapus area navigasi di sekitar objek dengan mengaturnya menjadi tipe *Not Walkable.* Perubahan ini dapat diatur agar berdampak pada tipe agen navigasi tertentu saja, semua tipe agen, atau tidak sama sekali.

2.7.4. *Navigation Modifier Tilemap*

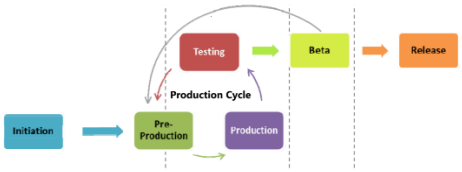
Komponen ini memiliki dependensi terhadap komponen *Navigation Modifier* dan *Navigation Surface* agar dapat digunakan. Dengan komponen ini, koleksi objek 2D *Tilemap* dapat dilakukan pada setiap *tileset* terdaftar yang dapat mengubah tipe area *default* jika diinginkan.

2.7.5. *Navigation CacheTileMapSources2d*

Berfungsi untung melakukan *cache* koleksi tiap *tileset* berdasarkan komponen *Tilemap, Navigation Modifier, dan Navigation Modifier Tilemap* yang direferensikan. *Cache* akan digunakan setiap perubahan area navigasi saat *runtime* sehingga proses lebih cepat dan efisien.

3. METODE PENELITIAN

Pengembangan *game* memerlukan metodologi yang fleksibel terhadap perubahan, maka dalam penelitian ini digunakan *Game Development Life Cycle* (GDLC) hasil rancangan Rido Ramadan dan Yani Widyani yang disusun dalam enam tahapan [17]. Ilustrasi setiap tahapan GDLC dapat dilihat pada Gambar 1.



Gambar 1. *Game Development Life Cycle* [17, p. 98]

3.1. *Initiation*

Tahap pertama adalah pembuatan konsep dasar dalam pengembangan *game* yang akan dikembangkan sepeti mekanik dasar *game roguelike* apa saja yang ingin diterapkan.

3.2. *Pre-production*

Tahap *pre-production* merupakan tahap pertama pada siklus *production* .yang aktivitas pada siklus pertamanya adalah pembuatan prototipe awal dari konsep dasar *game* berupa desain ilustrasi dalam *game* seperti desain level kasar, desain karakter, desain antarmuka, dan desain interaksi yang dapat terjadi dalam *game*,

Kemudian untuk setiap siklus berikutnya akan dilakukan revisi terhadap desain yang dibuat, sebagai panduan pada siklus *production* terkini. Lamanya durasi sebuah siklus dapat bervariasi untuk setiap siklusnya, tergantung dengan target yang direncanakan oleh pengembang *game*.

3.3. *Production*

Tahap ini akan mengimplementasikan desain *game* pada prototipe *game* yang dikembangkan, berupa pembuatan fitaur-fitur utama *game,* tampilan antarmuka, dan implementasi *pathfinding.*

Tahap ini juga akan dilakukan perbaikan fitur utama, mengatasi *bug* dan eror yang ditemukan, dan proses *refactoring* yang memperbaiki struktur kode agar lebih efisien setelah sistem dalam *game* sudah minim dengan *bug*.

3.4. *Testing*

Melakukan *playtesting* dan *debugging* secara rutin di akhir siklus *production* untuk menguji prototipe *game* berdasarkan implementasi desain yang dilakukan sehingga *bug* dan eror, fitur yang tak sesuai harapan, dan faktor yang membuat pengalaman bermain *game* tidak menyenangkan dapat ditemukan.

3.5. *Beta*

Saat mekanik dan fitur utama prototipe *game* sudah cukup siap untuk dimainkan, maka prototipe *game* dapat di rilis secara terbatas kepada responden yang berkeinginan mengikuti *beta test.* Kegiatan *beta test* ini bertujuan untuk mendapatkan *feedback* mengenai kualitas prototipe, fitur yang dapat dikembangkan dan ditambahkan, serta ketertarikan mereka terhadap *game* yang dikembangkan.

3.6. *Release*

Setelah prototipe *game* memenuhi setiap kriteria desain yang dirancang maka *game* dapat di rilis secara publik dengan versi *build* rilis

tertentu. Setelah ini, pengembang masih dapat melakukan siklus *production* lagi untuk memperbarui *game* atau sekedar mengeluarkan *patch* yang memperbaiki *bug* atau eksploit.

4. HASIL DAN PEMBAHASAN

Pada bagian ini akan dibahas hasil penelitian implementasi *pathfinding* pada *game roguelike survival* yang dimulai dengan pembahasan implementasi antarmuka *game,* dan kemudian dilanjutkan dengan pembahasan hasil implementasi *pathfinding.*

4.1. Tampilan Antarmuka *Game*

Game roguelike survival ini mengimplementasikan beberapa antarmuka yang dimulai pada Gambar 2 yang menampilkan antarmuka menu utama saat aplikasi *game* dijalankan atau pemain kembali ke menu ini dari antarmuka *gameplay.* Antarmuka ini memberikan beberapa opsi yang dapat dipilih oleh pemain, yang pertama adalah *Start Run* untuk berpindah memulai permainan di antarmuka *gameplay*, *Settings* untuk membuka antarmuka menu pengaturan, dan terakhir *Quit* untuk menutup aplikasi *game*.

Gambar 2. Antarmuka Menu Utama

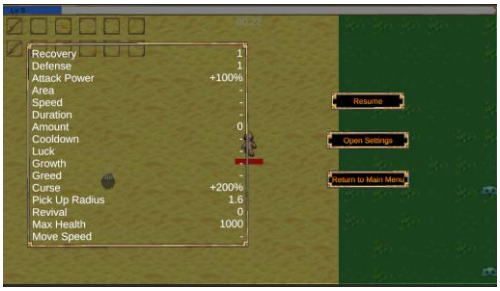
Antarmuka menu pengaturan dapat dilihat pada Gambar 3 dengan opsi pengaturan ukuran resolusi layar dan volume audio.

Gambar 3. Antarmuka Menu Pengaturan

Antarmuka *gameplay* seperti pada Gambar 4, menampilkan karakter pemain di tengah layar beserta *heads on display* (HUD) *health point* (HP) *bar* yang menempel di bawahnya. Kamera akan mempertahankan posisi karakter pemain terhadap kamera ini sambil terus mengikuti pergerakan karakter pemain di area *game*. Di bagian atas layar terdapat beberapa HUD, mulai dari *experience* (*exp*) *bar* yang memenuhi ujung atas layar berserta nilai level karakter pemain, lalu di pojok kiri sedikit di bawahnya terdapat *power slots* sebagai indikator *spell* dan *powerup* yang didapat, dan terakhir di bagian tengah terdapat *stopwatch* yang menjadi indikator lamanya permainan berlangsung.

Gambar 4. Antarmuka *Gameplay*

Gambar 5 menampilkan antarmuka menu *pause* yang terbuka untuk memberhentikan *gameplay* sementara waktu. Pada bagian kiri layar terdapat daftar indikator *stats* karakter pemain yang berpengaruh pada *gameplay*, dan di sebelah kanan terdapat pilihan untuk menutup antarmuka menu pause, membuka antarmuka menu pengaturan, dan kembali ke menu utama.



Gambar 5. Antarmuka Menu *Pause*

Ketika karakter pemain mendapat cukup *experience* (exp) setelah mengalahkan karakter musuh, maka karakter akan naik level dan *gameplay* diberhentikan sementara waktu untuk menampilkan antarmuka menu *level up* yang merupakan penerapan elemen *procedural generation* seperti pada Gambar 6. Pada antarmuka ini ditampilkan tiga atau empat opsi berupa *spell* atau *powerup* secara acak oleh

sistem. Setelah memilih salah satu opsi yang diberikan *gameplay* akan dilanjutkan kembali.

Gambar 6. Antarmuka Menu *Level Up*

Lalu pada gambar 7 ditampilkan salah satu *spell* karakter pemain yang didapat berhasil memberikan *damage* saat mengenai karakter musuh.

Gambar 7. *Spell* Karakter Pemain

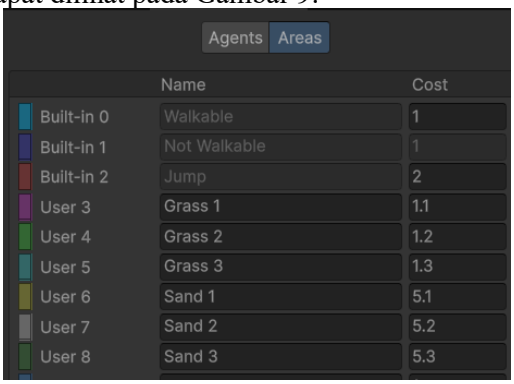
Jika HP karakter pemain menyentuh nilai nol, maka *gameplay* akan berakhir dan layar akan menampilkan antarmuka *game over* seperti pada gambar 8. Antarmuka ini menampilkan lama waktu *survive* permainan, level yang diraih, dan *power slots* beserta isinya yang didapat selama permainan. Pada bagian bawah layar diberikan dua pilihan yang diberikan yaitu *New Run* untuk memulai permainan lagi, dan *Return to Main Menu* untuk kembali ke menu utama.

Gambar 8. Antarmuka *Game Over*

4.2. Implementasi Unity AI *Navigation System*

4.2.1. Konfigurasi Area Navigasi

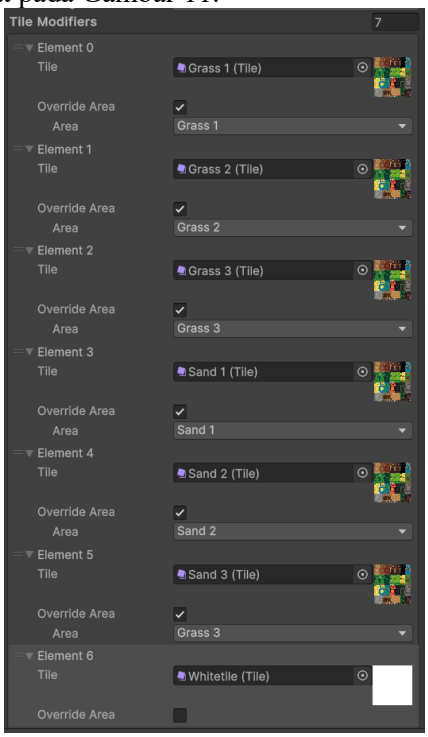
Dibuat enam buah tipe area navigasi dengan tiga area di antaranya merupakan variasi area *grass,* dan sisanya adalah variasi area *sand.* Ilustrasi pembuatan tipe area navigasi tersebut dapat dilihat pada Gambar 9.



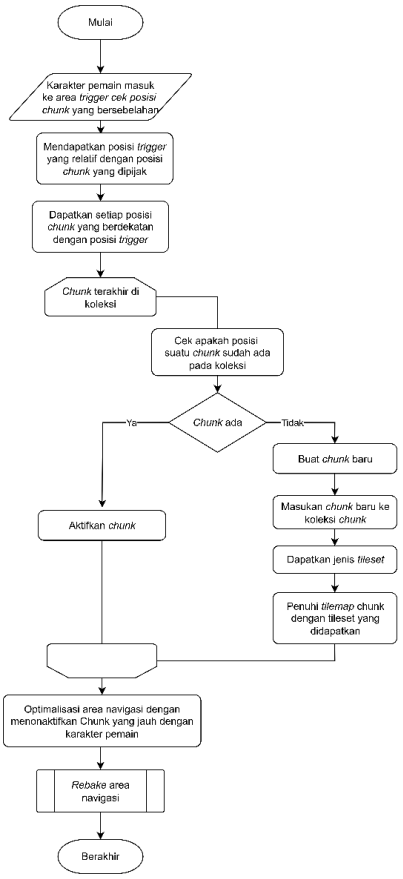
Gambar 9. Pembuatan Tipe Area Navigasi

Tipe area navigasi tersebut kemudian digunakan pada komponen *Navigation Modifier Tilemap* di *prefab* (objek *template*) *chunk tilemap* yang berisi *tileset default* berwarna putih seperti pada Gambar 10.

Prefab *chunk* ini digunakan sebagai metode penerapan elemen *procedural generation* berupa jenis *tile* yang pilih secara acak pada *script* komponen *Map Controller* di objek *surface* utama untuk selanjutnya di *generate* oleh komponen *Tile Generator* pada sebuah *chunk.* Setelah proses selesai maka akan dilanjutkan dengan proses optimasi *chunk* dan *rebake.* Ilustrasi proses-proses tersebut dapat dilihat pada Gambar 11.



Gambar 10. Daftar *Tileset Modifier*



Gambar 11. Proses *Tile Generation* dan *Rebake* Area Navigasi

4.2.2. Konfigurasi Objek Rintangan

Pembuatan objek rintangan dengan menambahkan komponen *Navigaion Modifier* dan mengatur perubahan area navigasi ke *Not Walkable*. Objek rintangan ini adalah batu dengan tiga variasi, kecil, sedang dan besar, yang tampilannya seperti pada gambar 12.

Gambar 12. Objek Rintangan

4.2.3. Konfigurasi Agen Navigasi

Dalam mengimplementasikan *pathfinding* pada area navigasi, agen navigasi *Character* dibuat terlebih dahulu untuk digunakan oleh karakter pemain dan karakter musuh.

Selanjutnya, mengonfigurasi komponen *NavMeshAgent* pada karakter pemain yang tampilannya seperti pada Gambar 12. Karakter pemain diatur sebagai agen agar dapat berinteraksi dengan komponen sistem navigasi lain, baik itu agen pada musuh, objek rintangan, dan juga area navigasi. Konfigurasi pertama adalah mengatur tipe agen dan nilai *offset* agennya. Kemudian seluruh parameter pada bagian *steering* dikosongkan dan dinonaktifkan. Pada bagian *obstacle avoidance,* sesuaikan parameter radius agar sesuai dengan ukuran *transform* objek, lalu biarkan kualitas penghindaran di *high quality*, sedangkan parameter prioritas ditetapkan ke tingkat tertinggi. Terakhir, area *mask* pada bagian *pathfinding* diatur ke *Everything* untuk memungkinkan akses ke seluruh tipe area navigasi.

Proses dilanjutkan dengan pembuatan *script* komponen yang mengatur pergerakan agen karakter pemain dengan memanfaatkan fungsi bawaan *Move*(). Fungsi ini menggerakkan agen secara relatif terhadap posisi *transform* objeknya, dengan arah gerak berasal dari *input keyboard* WASD (W = atas, A = kiri, S = bawah, D = kanan). Karena pergerakan agen yang bersifat relatif, posisi *transform* perlu disinkronkan mengikuti posisi aktual agen. Dengan menggunakan fungsi tersebut, karakter pemain dapat melintasi area navigasi tipe apa pun tanpa masalah.

Gambar 13. Karakter Pemain

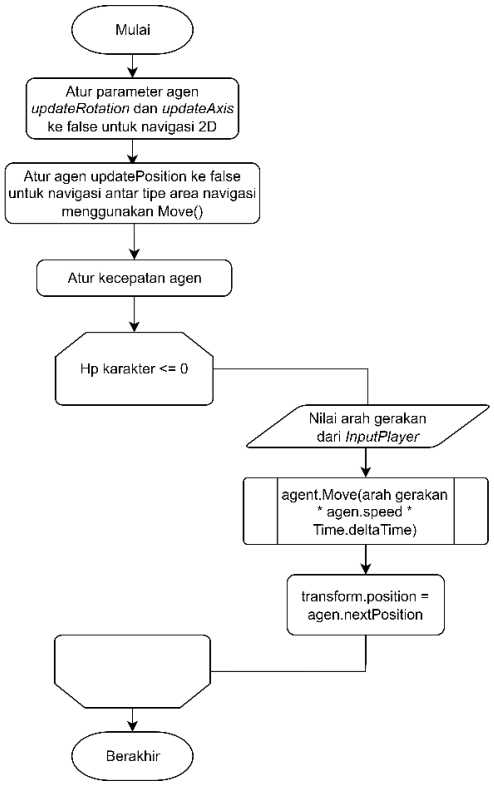
Selanjutnya konfigurasi karakter musuh yang terbagi atas dua tipe utama berdasarkan cara *pathinding* yang dilakukan. Tipe normal dengan tampilan seperti Gambar 14 (a) melakukan *pathfinding* di setiap *frame,* dan tipe *charging* bepenampilan seperti Gambar 15 (b) yang melakukannya hanya saat *spawn/respawn.* Konfigurasi pertama di bagian *steering* pada parameter *speed* dan *acceleration*. Kemudian pada bagian *obstacle avoidance* atur kualitas penghindaran tipe *charging* ke *none* agar menembus agen navigasi lain dan atur ke *high quality* pada tipe normal agar dapat menghindari agen lain, lalu berikan prioritas

penghindaran yang rendah untuk tipe normal. Terakhir atur area *mask* ke *everyhting,* untuk keduanya.

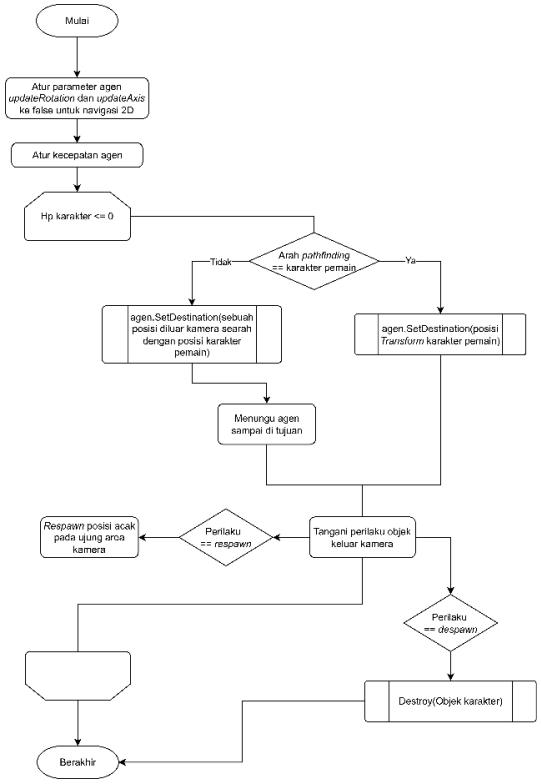
Pada pembuatan komponen *script pathfinding* untuk karakter musuh akan memanfaatkan fungsi *SetDestination*(). Fungsi inilah yang akan yang akan menjalankan *pathfinding* ke arah posisi *transform* karakter pemain sebagai tujuan navigasi dan bergerak . Fungsi ini menggunakan parameter menggunakan parameter *steering* yang telah dikonfigurasikan.

Gambar 14. Karakter Musuh

Alur proses pergerakan agen navigasi karakter pemain dan karakter musuh setelah dikonfigurasi masing-masing dapat dilihat pada Gambar 15, dan Gambar 16.



Gambar 15. Alur Navigasi Karakter Pemain

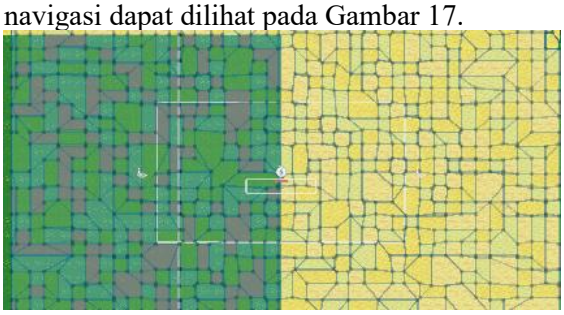


Gambar 16. Alur Navigasi Karakter Musuh

4.2.4. Tampilan *Debugging* Sistem Navigasi

Proses *debugging* dilakukan pada *scene editor* Unity agar dapat menampilkan *gizmo* visual sistem navigasi yang diimplementasikan.

Konfigurasi proses *tile generation* dan *baking* menghasilkan dua tipe *tilemap chunk*, *grass* dan *sand* yang masing-masing memiliki tiga tipe area navigasi. Poligon-poligon pada dua tipe *chunk* tersebut dan karakter pemain yang dapat bergerak ke seluruh tipe area navigasi dapat dilihat pada Gambar 17.

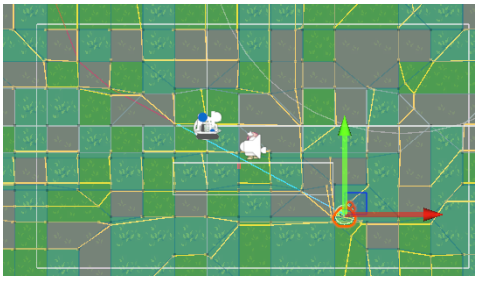


Gambar 17. *Tilemap Chunk Grass* dan *Sand*

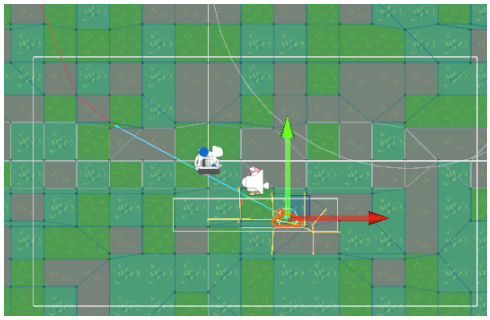
Pergerakan karakter musuh dimulai dengan proses *pathfinding* yang divisualisasikan oleh *gizmo* garis kuning di sekitar mereka yang

menuju poligon karakter pemain berada. Setelah jalur ditemukan, *gizmo* garis merah yang mengindikasikan jalur aktual dan garis *cyan* yang mengindikasikan arah dan kecepatan navigasi akan aktif. Karakter musuh tipe normal akan melakukan *pathfinding* secara terus menerus pada setiap *frame* sehingga bersifat dinamis seperti pada Gambar 18, sedangkan tipe *charge* akan melakukan secara *pathfinding* secara statis pada awal pergerakan saja seperti Gambar 19, dan pada Gambar 20 akan menyusuri jalur aktual yang aktif.

Gambar 18. *Pathfinding* dan Navigasi Karakter Musuh Tipe Normal

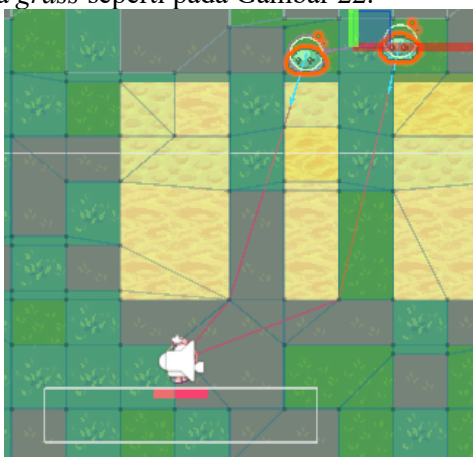


Gambar 19. *Pathfinding* Karakter Musuh Tipe *Charge*

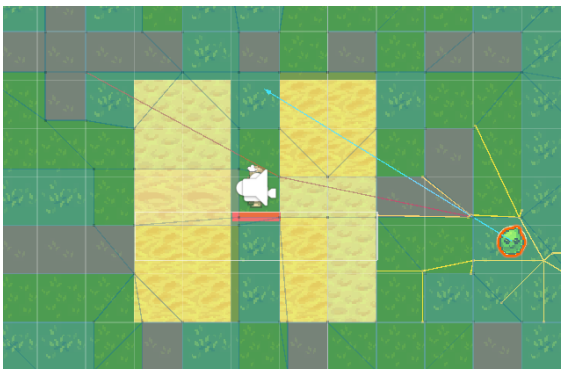


Gambar 20. Navigasi Karaker Musuh Tipe *Charge*

Pathfinding karakter musuh jika terdapat tipe area *sand* yang memiliki *cost* yang jauh lebih besar untuk tipe normal akan memilih tipe area *grass* pada jalur aktualnya seperti pada Gambar 21. Sedangkan tipe *charge* yang tujuannya *patfindingnya* adalah *offset* posisi karakter pemain akan di luar kamera, jalur aktualnya tetap lurus menembus area *sand* dengan sedikit penyesuaian ketika melewati area *grass* seperti pada Gambar 22.

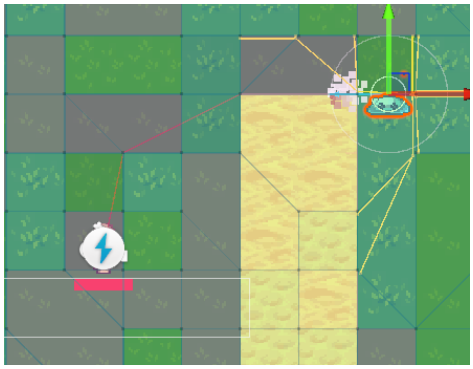


Gambar 21. *Pathfinding* Karakter Musuh Tipe Normal Dengan Area Navigasi *Sand*



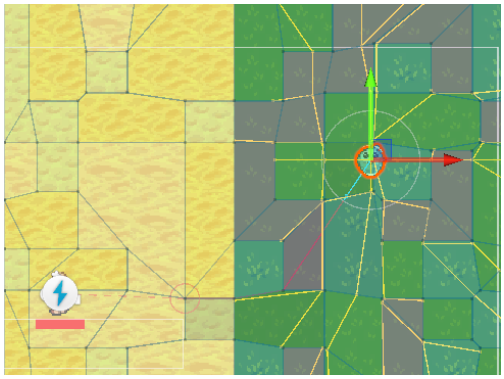
Gambar 22. *Pathfinding* Karakter Musuh Tipe *Charge* Dengan Area Navigasi *Sand*

Selanjutnya adalah pengujian *pathfinding* karakter musuh jika area *mask* diatur agar berisi tipe-tipe area *grass* saja, sehingga tidak dapat melewati tipe area *sand*. Hasilnya pada tipe normal akan mendeteksi tipe area *sand* sebagai rintangan lewat visual *gizmo* berbentuk sekumpulan persegi empat berwarna putih yang memerah ketika mendeteksi rintangan di sekitarnya seperti pada Gambar 23. Sementara pada tipe *charge* tidak jauh berbeda dengan sebelumnya selama posisi tujuan berada di area yang bisa dilalui tetap akan seperti pada Gambar 22, namun jika tidak maka jalur aktualnya akan berhenti pada awal area yang tidak bisa dilalui seperti yang terjadi pada stipe normal di Gambar 24.



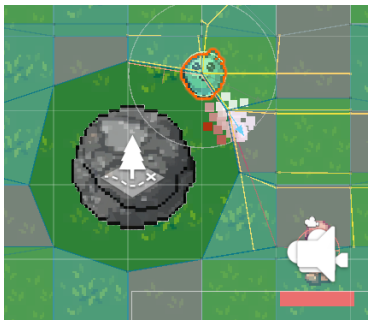
Gambar 23. *Pathfinding* Karaker Musuh Tipe Normal Menghindari Area Yang Tidak Bisa Dilalui

.



Gambar 24. *Pathfinding* Karakter Musuh Tipe Normal Ke Area Yang Tidak Bisa Dilalui

Pathfinding dengan adanya objek rintangan pada tipe normal akan terus melakukan deteksi rintangan seperti pada Gambar 25. Sedangkan tipe *charge* tidak mendeteksi adanya rintangan dikarenakan kualitas penghindaran diatur pada opsi *none,* namun jalur aktual tetap menyesuaikan ke posisi tujuan seperti pada Gambar 26.

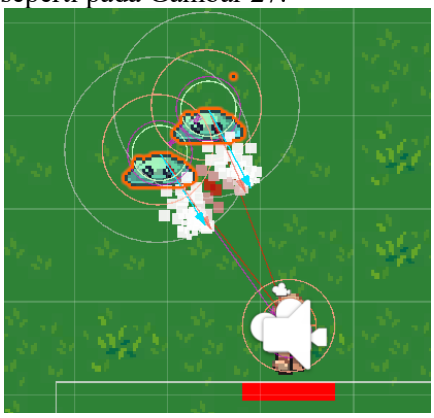


Gambar 25. *Pathfinding* Karakter Musuh Tipe Normal Dengan Rintangan

s

Gambar 26. *Pathfinding* Karakter Musuh Tipe *Charge* Dengan Rintangan

Interaksi penghindaran antar agen navigasi pada karakter musuh tipe normal dan karakter pemain terlihat pada *gizmo* pendeteksi rintangan, dan garis violet yang mendeteksi agen lain dengan prioritas sama (antar musuh) atau lebih tinggi (terhadap karakter pemain). *Gizmo* area navigasi dinonaktifkan agar *gizmo* sistem navigasi yang lain dapat terlihat lebih jelas seperti pada Gambar 27.



Gambar 27. Interaksi Penghindaran Antar Agen Navigasi

5. KESIMPULAN DAN SARAN

5.1. Kesimpulan

Berdasarkan hasil penelitian implementasi algoritma A* pada game 2D *roguelike survival* menggunakan Unity *AI Navigation System,* dapat diambil kesimpulan seperti berikut:

1 Implementasi metode *pathfinding* dengan algoritma A* menggunakan Unity *AI Navigation System* sebagai metode pergerakan karakter musuh *game roguelike survival* yang dinamis dilengkapi dengan kemampuan menghindari sumber rintangan dalam mengejar karakter pemain berhasil dilakukan. Karakter musuh dapat menghindari objek rintangan, area yang

tidak bisa dilalui, agen navigasi lain terutama tipe musuh yang sama.

2 Proses *pathfinding* karakter musuh dimulai dari mereka *spawn* di area navigasi, untuk tipe normal akan melakukannya terus menerus setiap *frame*, sedangkan tipe *charge* hanya di awal mereka *spawn/respawn.* Kedua tipe musuh yang konfigurasi agen berbeda yang dikombinasikan dengan sistem *spawn* membuat perilaku pergerakan musuh menjadi sangat berbeda.

3 Pembuatan tipe area navigasi *NavMesh* yang dinamis mengikuti *tilemap chunk* yang di *generate,* atau diaktifkan kembali saat *runtime* dapat dilakukan. Luas *NavMesh* akan menyesuaikan dengan *chunk* yang aktif berdasarkan posisi mereka terhadap karakter pemain

4 Tipe area navigasi yang bervariasi diterapkan melalui penggunaan *tileset* yang terdaftar pada komponen *Navigation Modifier Tilemap* yang ada pada objek *tilemap chunk.*

5.2. Saran

Berdasarkan potensi yang masih berlimpah dan perlu dieksplorasi lebih lanjut, juga kekurangan yang ada pada penelitian ini, terdapat beberapa saran yang dapat dipertimbangkan sebagai bentuk penelitian lebih lanjut seperti berikut:

1 Perlunya memperbanyak tipe musuh yang mungkin dapat memanfaatkan parameter komponen *NavMesh Agent* yang tidak dikonfigurasikan pada penelitian ini. Sehingga perilaku pergerakan musuh dapat lebih bervariasi.

2 Tipe area navigasi yang perlu diperbanyak sehingga dapat diklasifikasikan sebagai *biome*.

3 Membuat sebuah tipe musuh hanya muncul di tipe area navigasi atau *biome* tertentu saja.

4 Menerapkan *multithreading* agar proses *rebake* saat *runtime* dapat berjalan dengan lebih efisien sehingga tidak terjadi *spike* pada penggnaan *resource CPU*.

DAFTAR PUSTAKA

[1] R. Caesar, “Kajian Pustaka Perkembangan Genre Games Dari Masa Ke Masa,” *Journal of Animation and Games Studies*, vol. 1, no. 2, Oct. 2015, Accessed: Jul. 22, 2025. [Online]. Available: https://journal.isi.ac.id/index.php/jags/article/view/1301

[2] R. Cobbett, “The history of RPGs,” PC Gamer. Accessed: Feb. 06, 2025. [Online]. Available: https://www.pcgamer.com/the-complete-history-of-rpgs/

[3] H. Hermawan and H. Setiyani, “Implementasi Algoritma A-Star Pada Permainan Komputer Roguelike Berbasis Unity,” *Jurnal Algoritma, Logika dan Komputasi*, vol. 2, no. 1, pp. 111–120, May 2019, doi: 10.30813/j-alu.v2i1.1571.

[4] M. Zikky, “Review of A* (A Star) Navigation Mesh Pathfinding as the Alternative of Artificial Intelligent for Ghosts Agent on the Pacman Game,” *EMITTER International Journal of Engineering Technology*, vol. 4, no. 1, pp. 141–149, Jun. 2016, doi: doi.org/10.24003/emitter.v4i1.117.

[5] T. Fullerton, *Game Design Workshop a Playcentric Approach to Creating Innovative Games*, 5th ed. Boca Ratorn: CRC Press, 2024. doi: 10.1201/9781003460268.

[6] S. Khoiriyah, “Digitalisasi Permainan: Dampak dan Tren Pergeseran dari Permainan Tradisional ke Dunia Digital,” *Sinar Dunia: Jurnal Riset Sosial Humaniora dan Ilmu Pendidikan*, vol. 3, no. 4, pp. 320–333, Dec. 2024, doi: 10.58192/sidu.v3i4.2801.

[7] M. J. P. . Wolf, *The video game explosion : a history from PONG to Playstation and beyond*, 1st ed. Westport, Connecticut: Greenwood Press, 2008. Accessed: Jul. 23, 2025. [Online]. Available: https://www.scribd.com/document/866580788/The-Video-Game-Explosion-a-History-Mark-J-P-Wolf

[8] A. Rafiq, T. Asmawaty Abdul Kadir, and S. Normaziah Ihsan, “Pathfinding Algorithms in Game Development,” in *IOP Conference Series: Materials Science and Engineering*, Institute of

Physics Publishing, Jun. 2020. doi: 10.1088/1757-899X/769/1/012021.

[9] N. Nurhamni, "Geographic Information System (Gis) Uses A* Algorithm For Sorting Nearest Umkm Locations," *Jurnal Informatika dan Teknik Elektro Terapan*, vol. 13, no. 2, Apr. 2025, doi: 10.23960/jitet.v13i2.6256.

[10] Zhang He, Minyong Shi, and Chunfang Li, "Research and Application of Path-finding Algorithm Based on Unity 3D," in *2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS)*, Okayama: IEEE, Jun. 2016. doi: 10.1109/ICIS.2016.7550934.

[11] D. Monzonís Laparra, "Pathfinding Algorithms In Graphs And Applications," Barcelona, Jan. 2019.

[12] M. Janelly, S. Borbon, R. Zhen, M. Indol, R. M. Dioses, and K. E. Mata, "An Enhancement of A* Algorithm Applied in Automated Vehicle Parking," in *Proceeding of the International Conference on Electrical Engineering and Informatics Vol. 1 No. 1*, Demak: ARTEII, Feb. 2024, pp. 1–12. doi: 10.62951/iceei.v1i1.22.

[13] A. Sulaiman and Mamay Syani, "Development Of A 2d Educational Animal Card Game Using Unity," *NUANSA INFORMATIKA*, vol. 19, no. 2, pp. 35–48, Jul. 2025, doi: 10.25134/ilkom.v19i2.423.

[14] S. Singh and A. Kaur, "Game Development using Unity Game Engine," *2022 3rd International Conference on Computing, Analytics and Networks (ICAN)*, pp. 1–6, 2022, doi: 10.1109/ICAN56228.2022.10007155.

[15] Unity Technologies, "AI Navigation 2.0," Unity Documentation. Accessed: Jul. 16, 2025. [Online]. Available: https://docs.unity3d.com/Packages/com.unity.ai.navigation@2.0/manual/index.html

[16] h8man, "NavMeshPlus." Accessed: Feb. 28, 2025. [Online]. Available: https://github.com/h8man/NavMeshPlus/wiki

[17] R. Ramadan and Y. Widyani, "Game Development Life Cycle Guidelines," in *2013 International Conference on Advanced Computer Science and Information Systems (ICACSIS)*, Faculty of Computer Science, Universitas Indonesia, 2013, pp. 95–100. doi: 10.1109/ICACSIS.2013.6761558.