

Implementasi Fsm Pada Perilaku Musuh Game 2d Metroidvania Platformer Berbasis Unity

^{1*}Muhammad Daffa Aldiansyah, ²Muh.Aris Saputra, ³Risa Helilintar

^{1, 2, 3} Teknik Informatika, Universitas Nusantara PGRI Kediri

E-mail: ¹m.daffa.aladiansyah.com, ²aris.saputra@unpkdr.ac.id, ³risa.helilintar@unpkediri.ac.id

Penulis Korespondens : Muhammad Daffa Aldiansyah

Abstrak— Pergerakan musuh pada game platformer 2D sering mengalami kendala seperti terjebak pada rintangan atau transisi animasi yang tidak mulus, sehingga menurunkan kualitas pengalaman bermain. Penelitian ini mengusulkan implementasi metode Finite State Machine (FSM) berbasis Unity untuk mengatur perilaku musuh secara dinamis dan efisien. Metode pengembangan menggunakan pendekatan Research and Development dengan model ADDIE yang meliputi tahap analisis, desain, pengembangan, implementasi, dan evaluasi. Hasil pengujian menunjukkan bahwa FSM berhasil mengelola empat status perilaku musuh, yaitu diam, patroli, mengejar, dan menyerang secara responsif berdasarkan pembacaan sensor jarak. Kesimpulannya, penerapan logika FSM mampu meminimalkan malfungsi tabrakan antar objek dan meningkatkan stabilitas pergerakan kecerdasan buatan musuh yang lebih dinamis di dalam permainan.

Kata Kunci— ADDIE, FSM, game platformer, kecerdasan buatan, Unity.

Abstract— Enemy movement in 2D platformer games often experiences obstacles such as getting stuck on barriers or unsmooth animation transitions, thereby reducing the quality of the gameplay experience. This research proposes the implementation of the Unity-based Finite State Machine (FSM) method to regulate enemy behavior dynamically and efficiently. The development method employs a Research and Development (R&D) approach with the ADDIE model, which includes the stages of analysis, design, development, implementation, and evaluation. The test results indicate that the FSM successfully manages four enemy behavioral states, namely idle, patrol, chase, and attack, responsively based on distance sensor readings. In conclusion, the application of FSM logic is capable of minimizing object collision malfunctions and enhancing the stability of a more dynamic enemy artificial intelligence movement within the game.

Keywords— ADDIE, artificial intelligence, FSM, platformer game, Unity.

This is an open access article under the CC BY-SA License.



I. PENDAHULUAN

Perkembangan industri permainan video saat ini menuntut pengembang untuk menciptakan interaksi agen cerdas atau karakter non-pemain (NPC) yang responsif dan kompetitif [1]. Pada genre game *Metroidvania* berbasis *platformer* 2D, pergerakan karakter musuh memiliki kompleksitas tersendiri karena melibatkan eksplorasi peta yang non-linear serta interaksi dengan rintangan lingkungan yang dinamis [2]. Permasalahan umum yang sering terjadi pada pergerakan musuh konvensional adalah kegagalan NPC dalam mendeteksi batas platform atau kecenderungan untuk terjebak pada kolider dinding permainan [3]. Kondisi malfungsi mekanis

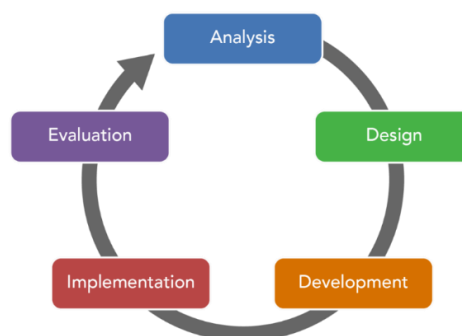
semacam ini secara langsung dapat menurunkan kualitas pengalaman bermain, sehingga dibutuhkan sistem kontrol kecerdasan buatan yang adaptif.

Berbagai penelitian telah dilakukan untuk mengatasi masalah navigasi dan perilaku NPC. Pendekatan komputasi yang terbukti efektif dan ringan dalam arsitektur logika agen cerdas adalah *Finite State Machine* (FSM) [4]. FSM sangat andal dalam mengatur perubahan status perilaku karakter berdasarkan parameter lingkungan yang dibaca oleh sensor sistem secara langsung [5]. Penggunaan *engine* Unity sebagai basis pengembangan memberikan dukungan penuh terhadap stabilitas logika FSM melalui efisiensi manajemen komponen fisik bawaan [6]. Meskipun FSM banyak digunakan, masih terdapat tantangan terkait malfungsi animasi saat karakter melakukan transisi cepat dari status mengejar ke menyerang akibat tumpang tindih kolider fisik.

Oleh karena itu, penelitian ini bertujuan untuk mengimplementasikan metode FSM dalam mengatur perilaku musuh pada game 2D *Metroidvania Platformer* berbasis Unity. Kebaruan yang ditawarkan dalam penelitian ini terletak pada penyesuaian parameter sensor jarak dan penguncian status aksi guna menghindari interupsi tabrakan fisik. Penelitian ini diharapkan dapat menghasilkan sistem perilaku musuh yang mulus, stabil, dan bebas dari cacat mekanis tanpa membebani performa komputasi perangkat.

II. METODE

Penelitian ini menggunakan pendekatan *Research and Development* (R&D) dengan memodelkan pengembangan perangkat lunak yang berfokus pada kecerdasan buatan dalam *game* [7]. Prosedur pengembangan mengadopsi model ADDIE (*Analysis, Design, Development, Implementation, Evaluation*) guna memastikan setiap tahap perancangan logika agen cerdas terstruktur dengan baik. Subjek penelitian ini adalah objek karakter musuh (*enemy*) pada *game* 2D *Metroidvania*, sedangkan alat yang digunakan meliputi PC pengembangan dan *game engine* Unity [8]. Teknik pengambilan data dilakukan melalui observasi *real-time* pada jendela *Unity Editor* untuk mengukur tingkat responsivitas transisi status (*state*) karakter berdasarkan variabel input jarak dan deteksi fisik.



Gambar 2. Model ADDIE

Desain eksperimen diawali dengan merumuskan logika penguncian status aksi menggunakan komponen bawaan Unity, lalu dilanjutkan dengan integrasi skrip C# untuk metode *Finite State*

Machine (FSM) [9]. Setiap perubahan status (dari patroli hingga menyerang) dihitung secara matematis menggunakan sensor visual buatan yang mengkalkulasi jarak antara musuh dan pemain. Logika ini diadopsi agar agen cerdas dapat mempertahankan status eksekusinya tanpa mengalami malfungsi meskipun terjadi perubahan variabel lingkungan yang cepat [10].

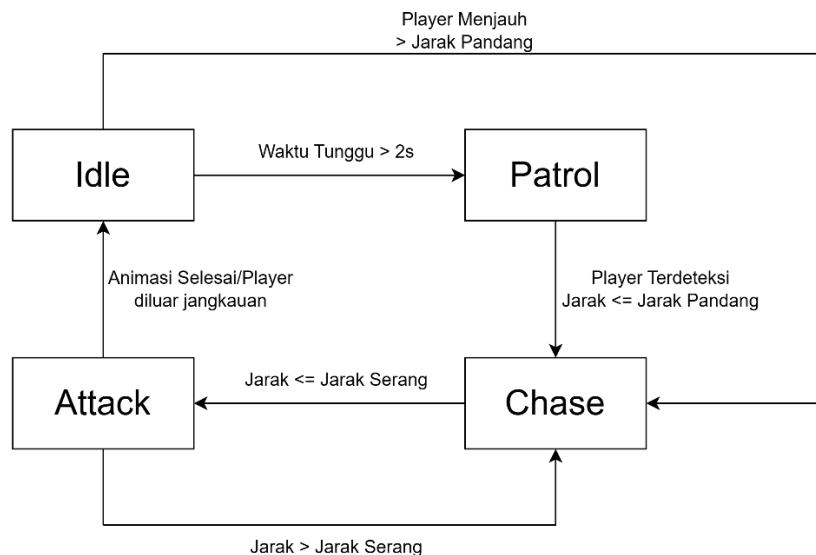
A. Rumus Matematika Kalkulasi Jarak

Kalkulasi Jarak di antara pemain dan posisi musuh dalam bidang 2D dihitung menggunakan persamaan Euclidean distance.

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \quad (1)$$

B. Desain Transisi FSM

Semua gambar yang anda masukkan dalam dokumen harus disesuaikan dengan urutan, agar memudahkan bagi reviewer untuk mencermati makna gambar. Gambar harus dijelaskan dengan detail dan nomor gambar harus disebutkan dalam penjelasan tersebut.



Gambar 1. Diagram Transisi *Finite State Machine* pada Karakter Musuh

Berdasarkan Gambar 1, alur logika pergerakan musuh dimulai dari kondisi Idle sebagai status default saat game pertama kali dijalankan. Karakter musuh akan berada dalam status Idle (diam) selama batas waktu tertentu, kemudian secara otomatis berpindah ke status Patrol (berpatroli) secara horizontal. Transisi dari Patrol menuju Chase (mengejar) dipicu apabila komponen Raycast atau sensor visual musuh mendeteksi posisi pemain yang berada dalam jangkauan jarak pandang tertentu.

Selama dalam status *Chase*, musuh akan terus memperbarui koordinat posisinya untuk mendekati pemain. Jika jarak antara musuh dan pemain sudah berada dalam area jangkauan serang, maka status akan berubah menjadi *Attack* (menyerang). Sebaliknya, jika pemain berhasil menjauh dari jangkauan pandang musuh, sistem FSM secara otomatis akan mengembalikan

status musuh dari *Chase* kembali ke *Patrol*. Setelah animasi pada status *Attack* selesai dieksekusi dan objek pemain tidak lagi terdeteksi di area sekitar, musuh akan kembali ke status *Idle* untuk mereset seluruh variabel kondisi.

Untuk memberikan batasan nilai (*threshold*) parameter yang jelas pada setiap perubahan status yang diilustrasikan di Gambar 1, detail parameter pemicu dari masing-masing transisi dirangkum secara sistematis pada Tabel 1.

Tabel 1. Parameter Batas Transisi State FSM

No	Parameter Batas Transisi State FSM		
	Status Awal	Kondisi Pemicu	Status tujuan
1	Idle	Variabel idleTimer > 2.0 Detik	<i>Patrol</i>
2	Patrol	Variabel Jarak Pemain $\leq$ visualRange	<i>Chase</i>
3	Chase	Variabel Jarak Pemain $\leq$ attackRange	<i>Attack</i>
4	Chase	Variabel Jarak Pemain > visualRange	<i>Patrol</i>
5	Attack	Variabel Jarak Pemain > attackRange	<i>Chase</i>
6	Attack	Kondisi isAttackFinished = <i>True</i>	<i>Idle</i>

Aturan transisi pada Tabel 1 menunjukkan mekanisme pembacaan kondisi yang dilakukan oleh *game engine* Unity secara *real-time*. Pada baris nomor 1, transisi murni menggunakan kalkulasi waktu tunggu internal sistem tanpa dipengaruhi objek luar. Baris nomor 2 dan 3 memanfaatkan rumus kalkulasi jarak Euclidean yang dideklarasikan pada Persamaan 1 untuk mendeteksi batas koordinat pemain.

Adanya kondisi interupsi pada baris nomor 4 dan 5 pada Tabel 1 berfungsi sebagai pengaman agar musuh tidak melakukan tindakan di luar jangkauan fisiknya, sehingga masalah animasi tumpang tindih (*overlapping animation*) dapat dihindari. Terakhir, pada baris nomor 6, penggunaan nilai boolean isAttackFinished memastikan bahwa skrip C# akan menyelesaikan seluruh *frame* animasi serangan terlebih dahulu sebelum sistem FSM diperbolehkan mereset status musuh kembali ke kondisi awal.

III. HASIL DAN PEMBAHASAN

Pembahasan terhadap hasil pengembangan sistem kecerdasan buatan (*Artificial Intelligence*) pada perilaku musuh game 2D *Metroidvania Platformer* disajikan secara bertahap mengikuti alur model ADDIE. Melalui kelima tahapan ini, data pengujian fungsional dan teknis diproduksi secara dinamis di dalam lingkungan kerja *game engine* Unity.

Tahapan pengembangan dengan model ADDIE memastikan proses pembuatan kecerdasan buatan berjalan secara sistematis dan terukur. Siklus ini tidak hanya berfokus pada tahap penulisan kode program, tetapi juga sangat menekankan pada identifikasi kelemahan logika pergerakan sejak awal melalui tahap analisis, hingga validasi fungsionalitas secara menyeluruh pada tahap evaluasi akhir.

A. Analisis

Pada tahap ini, dilakukan identifikasi masalah penjelajahan karakter musuh pada purwarupa game 2D platformer. Hasil analisis menunjukkan bahwa algoritma pergerakan musuh berbasis garis lurus (*linear movement*) konvensional sering mengalami malfungsi mekanis berupa kondisi musuh yang terjebak (*stuck*) pada sudut kolider dinding dan melayang di ujung platform jurang. Dari aspek estetika permainan, transisi perpindahan status aksi (misalnya dari mengejar menjadi menyerang) kerap kali terputus secara mendadak tanpa menyelesaikan siklus *frame* animasi. Oleh karena itu, ditetapkan kebutuhan fungsional berupa sebuah sistem kontrol berbasis *state* yang mampu mengisolasi fungsi gerak fisik (Rigidbody2D) dan sinkronisasi komponen logika animasi secara terpadu.

B. Perancangan

Di tahap ini perancangan menghasilkan cetak biru (*blueprint*) arsitektur logika FSM yang memisahkan perilaku musuh ke dalam sub-rutin otonom. Pemodelan ini mengacu pada visualisasi Gambar 1 dan matriks keputusan Tabel 1 yang telah dideklarasikan pada Bab II. Pada tahap ini, parameter fisis seperti *visualRange* dan *attackRange* ditentukan sebagai pembatas zona deteksi berbasis lingkaran (*bounding circle radius*) menggunakan sistem koordinat lokal Unity, sehingga musuh memiliki sensitivitas ruang yang akurat terhadap keberadaan objek pemain.

C. Pengembangan

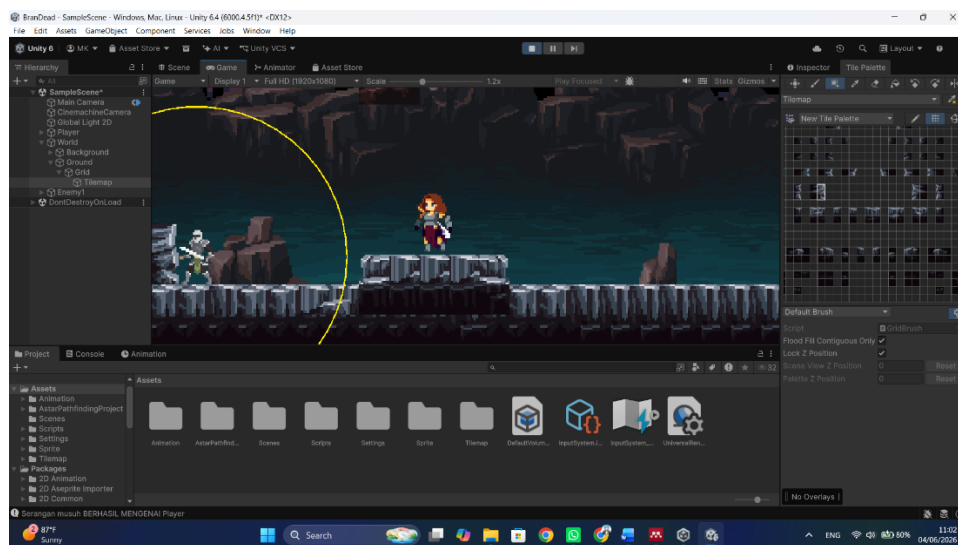
Di tahap pengembangan ini berfokus pada penerjemahan rancangan desain ke dalam baris kode program C# di Unity. Struktur kode dibangun menggunakan arsitektur pemrograman berbasis objek (*Object-Oriented Programming*) dengan memanfaatkan fungsi *Update()* untuk melakukan *scanning* status FSM secara *real-time* [7]. Sensor mekanis pendeteksi rintangan lingkungan diimplementasikan melalui skrip berikut:

```
void CheckObstacles() {  
  
    isGrounded      = Physics2D.Raycast(groundCheck.position,  
Vector2.down, rayDistance, groundLayer);  
  
    isWallBlocked    = Physics2D.Raycast(wallCheck.position,  
transform.right, rayDistance, wallLayer);  
  
}
```

Pada implementasi fungsi Physics2D.Raycast di atas bertindak sebagai pemicu (*trigger*) eksternal yang akan memaksa FSM membalikkan arah gerak patroli secara otonom sebelum menyentuh batas kolider objek lain [8]. Hal ini secara efektif menyelesaikan masalah *stuck* yang ditemukan pada tahap analisis awal.

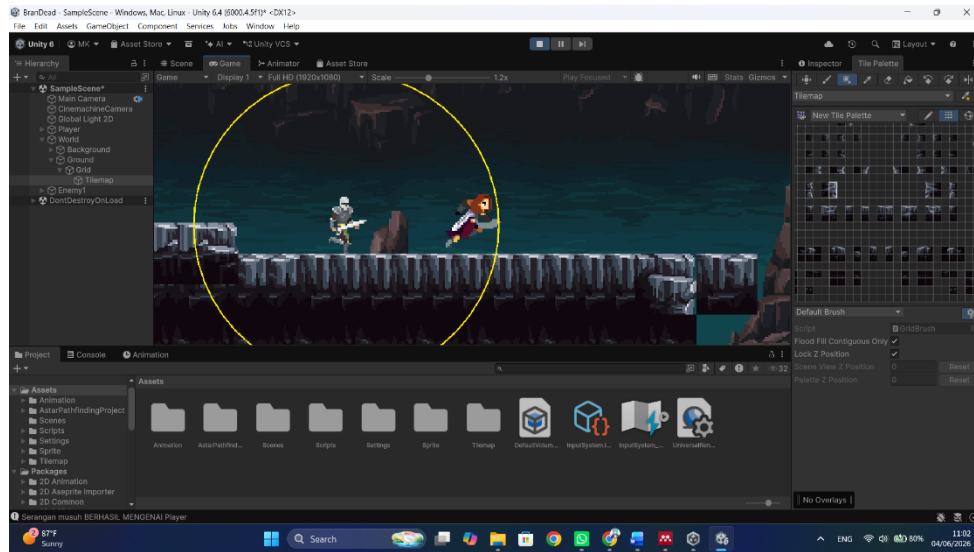
D. Implementasi

Tahap implementasi merupakan visualisasi nyata dari hasil kompilasi skrip logika FSM ke dalam objek *gameplay* 2D. Karakter musuh berhasil mengeksekusi transisi perilaku secara mulus sesuai dengan perubahan posisi pemain di dalam peta permainan [9].



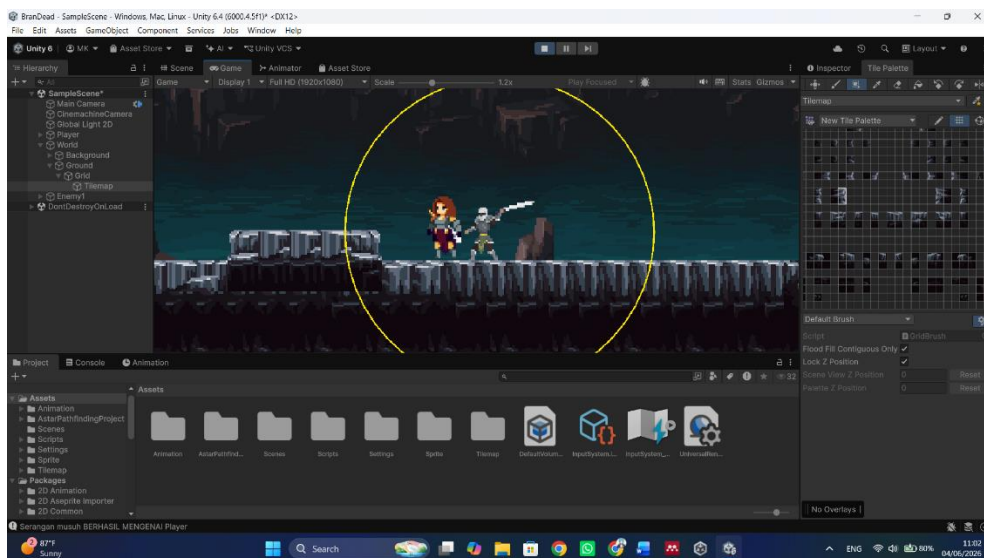
Gambar 3. Musuh dalam keadaan idle

Di gambar 3, karakter musuh berada dalam kondisi *Idle* atau berdiam diri pada titik koordinat awal, dengan indikator garis kuning berupa radius sensor *Raycast* yang aktif memantau area di sekitarnya. Ketika objek pemain mendekat dan memotong batas garis kuning sensor visual tersebut, sistem secara otomatis merubah status musuh menjadi *State Chase* seperti yang diilustrasikan pada Gambar 4.



Gambar 4. Musuh dalam State Chase

Pada Gambar 4, musuh memperbarui koordinat transformasinya untuk terus bergerak mengejar pemain secara dinamis.



Gambar 5. Musuh di State Attack

Selanjutnya, pada Gambar 5, ketika jarak antara musuh dan pemain sudah semakin dekat dan memenuhi prasyarat *attackRange*, sistem FSM menghentikan fungsi pergerakan mengejar dan beralih ke *State Attack*. Pada kondisi di Gambar 5 ini, musuh mengeksekusi animasi serangan tanpa mengalami malfungsi tumpang tindih kolider dengan objek pemain, membuktikan bahwa isolasi *state* berhasil berjalan dengan sangat mulus di dalam peta permainan.

E. Evaluasi

Tahap evaluasi dilakukan menggunakan metode pengujian fungsional (*black-box testing*) untuk menguji tingkat keberhasilan transisi setiap *state* perilaku musuh berdasarkan skenario permainan yang sesungguhnya. Pengujian dilakukan sebanyak 30 kali simulasi interaksi acak, dan hasilnya dicatat secara sistematis pada Tabel 2.

Tabel 2. Black Box Testing ke Perilaku Musuh

No	Black Box Testing ke Perilaku Musuh		
	Skenario Pemain	Prilaku Musuh yang Diharapkan	Status
1	Pemain berada di luar jangkauan	Musuh berjalan bolak-balik di platform	Sukses
2	Pemain mendekati jangkauan deteksi	Musuh berhenti patroli dan mengejar	Sukses
3	Pemain melompati karakter musuh	Musuh membalikkan arah hadap (<i>flip</i>)	Sukses
4	Pemain berada di jarak serang	Musuh berhenti bergerak dan memukul	Sukses
5	Pemain menjauh saat dipukul	Variabel Jarak Pemain > attackRange	Sukses
6	Pemain bersembunyi/me nghilang	Musuh diam sebentar lalu patroli lagi	Sukses

Data yang disajikan pada Tabel 2 membuktikan bahwa keberhasilan fungsional dari logika FSM yang dapat diimplementasikan dengan sukses. Batasan penelitian saat ini berfokus pada lingkungan platform statis, sehingga untuk penelitian di masa depan, sistem FSM ini dapat dikombinasikan dengan algoritma pencarian jalur dinamis seperti *A* Pathfinding* agar NPC musuh dapat melewati rintangan jebakan secara lebih fleksibel. Hubungan antara temuan ini dengan konsep dasar membuktikan bahwa pembagian status aksi yang terisolasi mampu mengeliminasi konflik tumpang tindih kolider fisik (*collider overlapping*) secara total, sehingga pergerakan musuh menjadi jauh lebih mulus, stabil, dan meningkatkan kualitas pengalaman bermain secara signifikan

IV. KESIMPULAN

Berdasarkan hasil perancangan dan pengujian yang telah dilakukan pada *Device* laptop Asus nitro 15, dapat disimpulkan bahwa implementasi metode *Finite State Machine* (FSM) berbasis Unity berhasil mengatasi kendala malfungsi pergerakan mekanis karakter musuh pada game 2D *Metroidvania Platformer*. Penerapan algoritma logika FSM yang dipecah menjadi empat status perilaku utama—yaitu diam (*idle*), patroli (*patrol*), mengejar (*chase*), dan menyerang (*attack*)—terbukti mampu mengelola transisi animasi secara mulus dan responsif berdasarkan kalkulasi sensor jarak terhadap posisi pemain. Peningkatan stabilitas navigasi kecerdasan buatan ini semoga memberikan kontribusi yang berarti bagi pengembangan ilmu teknik informatika, khususnya dalam memproduksi arsitektur agen cerdas yang adaptif, terstruktur, ringan dari sisi komputasi, dan secara langsung mampu meningkatkan kualitas pengalaman bermain secara keseluruhan.

DAFTAR PUSTAKA

- [1] R. K. Gea, I. H. . Manurung, A. F. . Sibero, and R. Sitanggang, “Perancangan Aplikasi Game 2D Kill Ghost Menggunakan Unity,” *J. Inform. Kaputama*, vol. 7, no. 1, pp. 100–112, 2023, doi: 10.59697/jik.v7i1.12.
- [2] M. Umaryanto, H. Zulfia Zahro’, and R. Primaswara Prasetya, “the Legend of the Knight 2D Menggunakan Metode Hierarchical Finite State Machine (Hfsm) Berbasis Android,” *JATI (Jurnal Mhs. Tek. Inform.*, vol. 8, no. 6, pp. 12408–12416, 2024, doi: 10.36040/jati.v8i6.10571.
- [3] I. Satrio, F. Santi Wahyuni, and D. Rudhistiar, “Penerapan a* Pathfinding Dan Fsm (Finite State Machine) Pada Game ‘Lost Civilization’ Berbasis Android,” *JATI (Jurnal Mhs. Tek. Inform.*, vol. 6, no. 2, pp. 1192–1199, 2023, doi: 10.36040/jati.v6i2.5402.
- [4] M. Hasan Syu’aibi, A. Mahmudi, and K. Auliasari, “PERANCANGAN DAN IMPLEMENTASI METODE FSM (FINITE STATE MACHINE) PADA GAME MILITARY DEFENCE 2D BERBASIS ANDROID,” *JATI (Jurnal Mhs. Tek. Inform.*, vol. 7, no. 4, pp. 2349–2357, Dec. 2023, doi: 10.36040/jati.v7i4.7508.
- [5] F. Hasfa, A. Panji Sasmito, and F. Santi Wahyuni, “Penerapan Metode Finite State Machine (Fsm) Dan Line of Sight Pada Game Petualangan the Sultan Hasanuddin Berbasis Android,” *JATI (Jurnal Mhs. Tek. Inform.*, vol. 8, no. 5, pp. 8284–8291, 2024, doi: 10.36040/jati.v8i5.10514.
- [6] F. R. Firmansyah and A. Rahmatulloh, “Perancangan Mekanisme Interaksi Pemain Dan Bot Di Game Sumoball Dengan Unity 2020 Menggunakan Addforce,” *J. Animat. Games Stud.*, vol. 11, no. 1, pp. 49–60, 2025, doi: 10.24821/jags.v11i1.11347.
- [7] B. Wicaksono and Y. Akbar, “Implementasi Finite State Machine (Fsm) Pada Game Edukasi 2D Untuk Pengenalan Fasilitas Umum Anak Usia Dini,” *JATI (Jurnal Mhs. Tek. Inform.*, vol. 9, no. 6, pp. 9715–9722, 2025, doi: 10.36040/jati.v9i6.15736.
- [8] N. B. Nugraha, Y. M. Santosa, and E. Mulyani, “Implementation of Finite State Machine Algorithm for Interactive Physics Learning in a 3D Game,” *J. Appl. Informatics Comput.*, vol. 7, no. 2, pp. 278–283, 2023, doi: 10.30871/jaic.v7i2.6738.
- [9] M. B. Firdaus, A. Z. Waksito, A. Tejawati, M. Taruk, M. K. Anam, and A. Irsyad, “Finite state machine for retro arcade fighting game development,” *Int. J. Informatics Commun. Technol.*, vol. 14, no. 1, pp. 102–110, 2025, doi: 10.11591/ijict.v14i1.pp102-110.
- [10] K. Awalludin, S. Busono, U. Indahyanti, and Y. Rahmawati, “Pengembangan Game 2D ‘Cat Collection Coin’ Berbasis Single Player Menggunakan Metode Finite State Machine untuk Manajemen State Karakter,” *J. Algoritma*, vol. 22, no. 2, pp. 140–151, 2025, doi: 10.33364/algoritma/v.22-2.2424.