TUGAS AKHIR – EF234801

PENGEMBANGAN *PACKAGE ARTIFICIAL INTELLIGENCE PATHFINDING* UNTUK GIM *PLATFORMER* 2D

BRIAN AKBAR WICAKSANA

NRP 5025201207

Dosen Pembimbing

Dr. Eng Darlis Herumurti, S.Kom, M.Kom

NIP 197712172003121001

Dosen Pembimbing II

Imam Kuswardayan, S.Kom., MT

NIP 197612152003121001

Program Studi S1 Teknik Informatika

Departemen Teknik Informatika

Fakultas Teknologi Elektro dan Informatika Cerdas

Institut Teknologi Sepuluh Nopember

Surabaya

2024

(Halaman ini sengaja dikosongkan)

TUGAS AKHIR – EF234801

PENGEMBANGAN *PACKAGE ARTIFICIAL INTELLIGENCE PATHFINDING* UNTUK GIM *PLATFORMER* 2D

BRIAN AKBAR WICAKSANA

NRP 5025201207

Dosen Pembimbing

Dr. Eng Darlis Herumurti, S.Kom, M.Kom

NIP 197712172003121001

Dosen Pembimbing II

IMAM KUSWARDAYAN, S.Kom., MT.

NIP 197612152003121001

Program Studi S1 Teknik Informatika

Departemen Teknik Informatika

Fakultas Teknologi Elektro dan Informatika Cerdas

Institut Teknologi Sepuluh Nopember

Surabaya

2024

(Halaman ini sengaja dikosongkan)

FINAL PROJECT – EF234801

DEVELOPMENT OF PATHFINDING ARTIFICIAL INTELLIGENCE PACKAGE FOR 2D PLATFORMER GAME

BRIAN AKBAR WICAKSANA

NRP 5025201207

Advisor

Dr. Eng Darlis Herumurti, S.Kom, M.Kom

NIP 197712172003121001

Co-Advisor

IMAM KUSWARDAYAN, S.Kom., MT.

NIP 197612152003121001

Undergraduate Study Program of Informatics

Department of Informatics

Faculty of Intelligent Electrical and Informatics Technology

Institut Teknologi Sepuluh Nopember

Surabaya

2024

(Halaman ini sengaja dikosongkan)

LEMBAR PENGESAHAN

PENGEMBANGAN *PACKAGE ARTIFICIAL INTELLIGENCE PATHFINDING* UNTUK GIM *PLATFORMER* 2D

TUGAS AKHIR

Diajukan untuk memenuhi salah satu syarat
Memperoleh gelar Sarjana Komputer pada
Program Studi S-1 Teknik Informatika
Departemen Teknik Informatika
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember

Oleh : BRIAN AKBAR WICAKSANA
NRP. 5025201207

Disetujui oleh Tim Penguji Tugas Akhir:

1. Dr. Eng Darlis Herumurti, S.Kom, M.Kom — Pembimbing
2. Imam Kuswardayan, S.Kom., MT. — Ko-pembimbing
3. Dr. Anny Yuniarti, S.Kom., M.Comp.Sc. — Penguji
4. Hadziq Fabroyir, S.Kom., Ph.D. — Penguji

SURABAYA
Juli, 2024

(Halaman ini sengaja dikosongkan)

APPROVAL SHEET

DEVELOPMENT OF PATHFINDING ARTIFICIAL INTELLIGENCE PACKAGE FOR 2D PLATFORMER GAME

FINAL PROJECT

Submitted to fulfill one of the requirements
for obtaining a degree Bachelor of Computer at
Undergraduate Study Program of Informatics
Department of Informatics
Faculty of Intelligent Electrical and Informatics Technology
Institut Teknologi Sepuluh Nopember

By: BRIAN AKBAR WICAKSANA
NRP. 5025201207

Approved by Final Project Examiner Team:

1. Dr. Eng Darlis Herumurti, S.Kom, M.Kom — Advisor
2. Imam Kuswardayan, S.Kom., MT. — Co-Advisor
3. Dr. Anny Yuniarti, S.Kom., M.Comp.Sc. — Examiner
4. Hadziq Fabroyir, S.Kom., Ph.D. — Examiner

SURABAYA
July, 2024

(Halaman ini sengaja dikosongkan)

PERNYATAAN ORISINALITAS

Yang bertanda tangan di bawah ini:

Nama mahasiswa / NRP : Brian Akbar Wicaksana / 5025201207
Departemen : Teknik Informatika
Dosen Pembimbing / NIP : Dr. Eng Darlis Herumurti, S.Kom, M.Kom / 197712172003121001
Dosen Ko-Pembimbing / NIP : Imam Kuswardayan, S.Kom., MT. / 197612152003121001

Dengan ini menyatakan bahwa Tugas Akhir dengan judul "**PENGEMBANGAN *PACKAGE ARTIFICIAL INTELLIGENCE PATHFINDING* UNTUK GIM *PLATFORMER* 2D**" adalah hasil karya sendiri, bersifat orisinal, dan ditulis dengan mengikuti kaidah penulisan ilmiah. Bilamana di kemudian hari dietemukan ketidaksesuaian dengan pernyataan ini, maka saya bersedia menerima saksi sesuai dengan ketentuan yang berlaku di Institut Teknologi Sepuluh Nopember.

Surabaya, 15 Juli 2024

Mahasiswa

Brian Akbar Wicaksana
NRP. 5025201207

Mengetahui,

Dosen Pembimbing I	Dosen Pembimbing II
	
Dr. Eng Darlis Herumurti, S.Kom, M.Kom	Imam Kuswardayan, S.Kom., MT.
NIP. 197712172003121001	NRP. 197612152003121001

(Halaman ini sengaja dikosongkan)

STATEMENT OF ORIGINALITY

The undersigned below:

Name of student / NRP	: Brian Akbar Wicaksana / 5025201207
Department	: Informatics
Advisor / NIP	: Dr. Eng Darlis Herumurti, S.Kom, M.Kom / 197712172003121001
Co-Advisor / NIP	: Imam Kuswardayan, S.Kom., MT. / 197612152003121001

Hereby declare that Final Project with the title of "**DEVELOPMENT OF** DYNAMIC **PATHFINDING ARTIFICIAL INTELLIGENCE PACKAGE FOR 2D PLATFORMER GAME**" is the result of my own work, is original, and is written by following the rules of scientific writing. If in the future there is a discrepancy with this statement, then I am willing to accept sanctions in accordance with the provisions that apply at Institut Teknologi Sepuluh Nopember.

Surabaya, 15 July 2024

Student

Brian Akbar Wicaksana

NRP. 5025201207

Acknowledge,

Advisor	Co-Advisor
Dr. Eng Darlis Herumurti, S.Kom, M.Kom	Imam Kuswardayan, S.Kom., MT.
NIP. 197712172003121001	NRP. 197612152003121001

(Halaman ini sengaja dikosongkan)

PENGEMBANGAN *PACKAGE ARTIFICIAL INTELLIGENCE PATHFINDING* UNTUK GIM *PLATFORMER* 2D

Nama Mahasiswa / NRP : Brian Akbar Wicaksana / 5025201207
Departemen : Teknik Informatika FTEIC - ITS
Dosen Pembimbing : Dr. Eng Darlis Herumurti, S.Kom, M.Kom

Abstrak

Navigasi antar platform dalam pengembangan gim *platformer* merupakan salah satu tantangan besar yang dihadapi oleh pengembang. Kesulitan ini muncul karena kompleksitas dalam menentukan jalur bagi karakter untuk berpindah antar platform dengan mempertimbangkan berbagai variabel seperti kecepatan gerak, gravitasi, dan rintangan yang ada. Untuk mengatasi kendala ini, penelitian ini mengusulkan pengembangan sebuah *package pathfinding* yang dirancang khusus untuk mengatasi masalah tersebut.

Package ini mengimplementasikan dua fitur utama, yaitu pembuatan *platform graph* dan navigasi platform. *Platform graph* digunakan untuk memetakan platform permainan secara detail, sedangkan fitur navigasi platform memungkinkan karakter untuk berpindah antar platform dengan efisien. Pengujian dilakukan melalui tiga metode, yaitu pengujian waktu eksekusi pembuatan *platform graph*, pengujian fungsionalitas, dan pengujian prototipe pada dua gim *platformer*. Hasil pengujian menunjukkan bahwa *package* ini efektif dalam menyediakan solusi navigasi yang andal untuk gim *platformer*, dengan waktu eksekusi pembuatan *platform graph* yang bervariasi tergantung pada ukuran peta.

Dengan demikian, *package* ini dapat membantu pengembang gim dalam menciptakan navigasi karakter yang lebih baik dan meningkatkan kualitas *gameplay* pada gim *platformer*.

Kata kunci: *Platformer, Pathfinding, Package,* Unity.

(Halaman ini sengaja dikosongkan)

DEVELOPMENT OF PATHFINDING ARTIFICIAL INTELLIGENCE PACKAGE FOR 2D PLATFORMER GAME

Student Name / NRP : Brian Akbar Wicaksana / 5025201207
Department : Informatics FTEIC - ITS
Advisor : Dr. Eng Darlis Herumurti, S.Kom, M.Kom

Abstract

Platform navigation in the development of platformer games is one of the major challenges faced by developers. This difficulty arises due to the complexity of determining the path for characters to move between platforms, considering various variables such as movement speed, gravity, and obstacles. To address this issue, this research proposes the development of a pathfinding package specifically designed to tackle this problem.

This package implements two main features: platform graph creation and platform navigation. The platform graph is used to map the game platform in detail, while the platform navigation feature enables characters to move between platforms efficiently. Testing was conducted using three methods: execution time testing for platform graph creation, functionality testing, and prototype testing on two platformer games. The test results show that this package is effective in providing a reliable navigation solution for platformer games, with the execution time for platform graph creation varying depending on the map size.

Thus, this package can assist game developers in creating better character navigation and improving the quality of gameplay in platformer games.

Kata kunci: Platformer, Pathfinding, Package, Unity.

(Halaman ini sengaja dikosongkan)

KATA PENGANTAR

Puji dan syukur penulis panjatkan kepada Tuhan Yang Maha Esa atas segala rahmat dan karunia-Nya, sehingga penulis dapat menyelesaikan Tugas Akhir ini dengan judul:

" PENGEMBANGAN *PACKAGE ARTIFICIAL INTELLIGENCE PATHFINDING* UNTUK GIM *PLATFORMER* 2D"

Penulis menyadari bahwa tanpa bantuan, dukungan, dan bimbingan dari berbagai pihak, Tugas Akhir ini tidak akan terselesaikan dengan baik. Oleh karena itu, pada kesempatan ini penulis ingin menyampaikan rasa terima kasih yang sebesar-besarnya kepada:

1. Bapak Dr. Eng Darlis Herumurti, S.Kom, M.Kom dan Bapak Imam Kuswardayan, S.Kom, MT. selaku dosen pembimbing penulis, yang telah memberikan bimbingan, saran, dan motivasi selama penyusunan Tugas Akhir ini.
2. Orang tua dan keluarga, yang telah memberikan doa, dukungan moral, dan material yang tiada henti.
3. Elthan Ramanda Berlindo sebagai teman yang telah menemani penulis selama perkuliahan di Teknik Informatika ITS.
4. Teman-teman dan semua pihak yang telah memberikan bantuan, dukungan, serta kebersamaan selama masa studi dan penyusunan Tugas Akhir ini.

Penulis menyadari bahwa Tugas Akhir ini masih jauh dari sempurna. Oleh karena itu, penulis mengharapkan saran dan kritik yang membangun dari berbagai pihak demi perbaikan di masa mendatang. Akhir kata, penulis berharap semoga Tugas Akhir ini dapat memberikan manfaat bagi pembaca dan pengembangan ilmu pengetahuan.

Surabaya, 15 Juli 2024

Brian Akbar Wicaksana

(Halaman ini sengaja dikosongkan)

DAFTAR ISI

DAFTAR GAMBAR/GRAFIK/DIAGRAM

DAFTAR TABEL

(Halaman ini sengaja dikosongkan)

DAFTAR PSEUDOCODE

(Halaman ini sengaja dikosongkan)

DAFTAR KODE PROGRAM

(Halaman ini sengaja dikosongkan)

DAFTAR SINGKATAN

AI	Artificial Intelligence
2D	Dua Dimensi
MCTS	Monte Carlo Tree Search
RRT	Rapidly-exploring Random Tree
NavMesh	Navigation Mesh
iOS	iPhone Operating System
HDRP	High Definition Render Pipeline
URP	Universal Render Pipeline
ms	milliseconds

(Halaman ini sengaja dikosongkan)

BAB I
PENDAHULUAN

1.1 Latar Belakang

Navigasi antar platform dalam pengembangan gim *platformer* merupakan salah satu tantangan utama yang sering dihadapi oleh para pengembang gim. Kesulitan ini muncul karena kompleksitas dalam menentukan jalur bagi karakter untuk berpindah antar platform. Beberapa faktor yang mempengaruhi navigasi ini meliputi kecepatan gerak karakter, efek gravitasi, serta adanya berbagai rintangan yang harus dihindari atau dilewati.

Penelitian ini bertujuan untuk mengembangkan sebuah *package pathfinding* yang dirancang khusus untuk mengatasi masalah navigasi antar platform dalam gim *platformer*. *Package* ini mengimplementasikan dua fitur utama, yaitu pembuatan *platform graph* dan navigasi platform. *Platform graph* digunakan untuk memetakan platform permainan secara detail, sehingga memudahkan dalam menentukan jalur terbaik bagi karakter. Sedangkan fitur navigasi platform memungkinkan karakter untuk berpindah antar platform dengan efisien dan alami.

Melalui pengembangan dan pengujian *package* ini, diharapkan dapat memberikan solusi yang handal dan efisien bagi para pengembang gim *platformer*, sehingga dapat meningkatkan kualitas gameplay dan memberikan pengalaman bermain yang lebih baik bagi para pemain.

Dengan adanya *package* ini, pengembang gim dapat lebih fokus pada aspek kreatif dan desain peta, sementara sistem navigasi yang kompleks dapat ditangani oleh *package* yang telah dikembangkan. Pengujian dilakukan melalui tiga metode, yaitu pengujian waktu eksekusi pembuatan *platform graph*, pengujian fungsionalitas, dan pengujian prototipe pada dua gim *platformer*, untuk memastikan keefektifan dan keandalan *package* dalam berbagai skenario permainan.

1.2 Rumusan Masalah

Rumusan masalah yang diajukan dalam Tugas Akhir ini adalah sebagai berikut.

1. Bagaimana pengembangan sebuah *package* kecerdasan buatan yang dapat dikustomisasi pada karakter untuk perpindahan antar platform?
2. Bagaimana pengujian efektivitas *package* ini dalam pengembangan gim yang berbeda?

1.3 Batasan Masalah

Terdapat beberapa batasan dalam perancangan permasalahan yang dibahas dalam Tugas Akhir ini, di antaranya sebagai berikut.

1. Pengembangan *package* dan prototipe akan menggunakan Unity.
2. Metode pengujian efektivitas *package* dilakukan terhadap dua prototipe gim. Prototipe pertama adalah prototipe yang dikembangkan sendiri, sedangkan prototipe kedua adalah prototipe yang sudah ada dan dikembangkan oleh pihak lain.

1.4 Tujuan

Tujuan yang diajukan dalam menyelesaikan rumusan permasalahan yang diajukan dalam Tugas Akhir ini adalah sebagai berikut.

1. Merancang dan mengembangkan sebuah *package* kecerdasan buatan yang adaptif untuk digunakan dalam gim *platformer* 2D.
2. Membuat dan menguji efektivitas *package* melalui pembuatan prototipe gim sebagai implementasi konkret dari konsep yang diusulkan.

1.5 Manfaat

Tugas akhir ini diharapkan dapat menjadi solusi bagi *game designer* untuk menyelesaikan permasalahan kecerdasan buatan pada *platformer* 2D. Pembuatan prototipe gim sebagai implementasi dari *package* diharapkan dapat menunjukkan keefektivitas *package* dalam membuat gim *platformer* 2D.

BAB II
TINJAUAN PUSTAKA

2.1 Penelitian Terkait

2.1.1 An Integrated Framework for AI Assisted Level Design in 2D Platformers

Pier Luca Lanzi dan Daniele Loiacono telah melakukan studi yang menghasilkan sebuah *framework* yang dirancang untuk membantu *game designer* dalam menciptakan level gim *platformer* 2D. *Framework* ini menyediakan berbagai alat yang mendukung proses pembuatan level, termasuk estimasi tingkat kesulitan dan probabilitas kesuksesan sekali aksi loncat, serta satu set metrik untuk mengevaluasi kesulitan dan kemungkinan penyelesaian seluruh level. Studi ini memberikan inspirasi untuk topik Tugas Akhir ini, yang bertujuan mengimplementasikan kecerdasan buatan dalam konteks gim *platformer* 2D. (Aramini et al., 2018)

2.1.2 I Can Jump! Exploring Search Algorithms for Simulating Platformer Players

Jonathan Tremblay, Alexander Borodovski, dan Clark Verbrugge telah menyajikan sebuah studi yang merinci tiga algoritma pencarian berbeda, yaitu A*, MCTS, dan RRT, yang dapat diterapkan untuk mensimulasikan perilaku pemain dalam domain *platformer*. Penelitian mereka melibatkan evaluasi dan perbandingan ketiga pendekatan tersebut pada tiga level, membuktikan kemungkinan workflow yang dapat digunakan oleh *game designer*. Hasil studi ini memberikan inspirasi untuk topik Tugas Akhir ini yang berkaitan dengan pengembangan algoritma untuk mengatur perilaku loncatan pemain dari satu platform ke platform lainnya. (Tremblay et al., 2021)

2.2 Dasar Teori

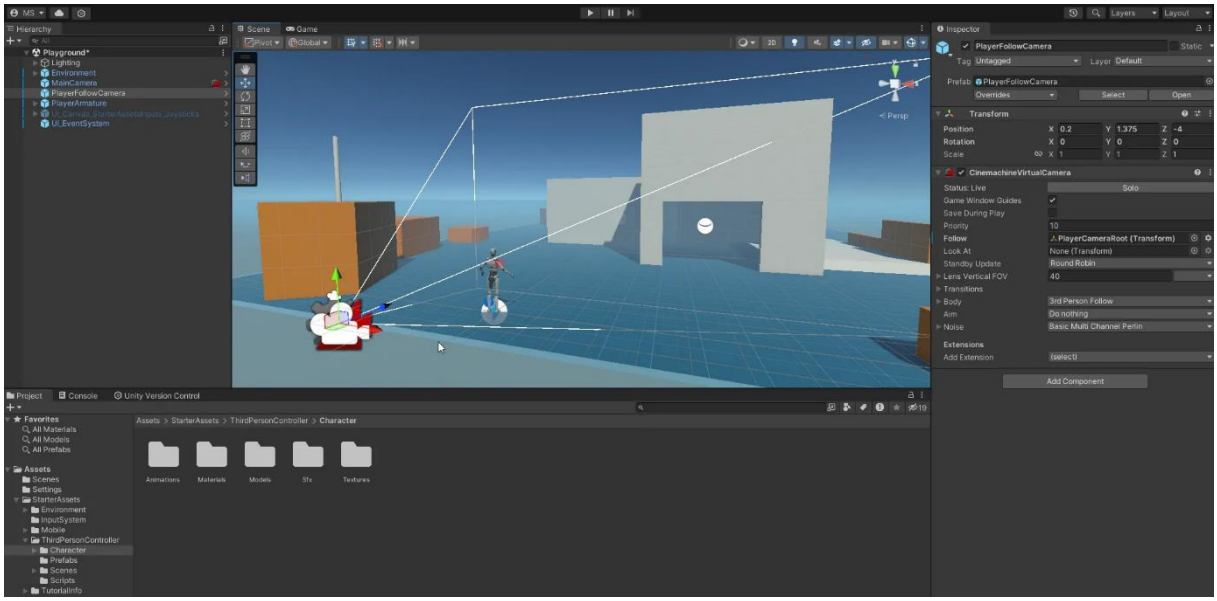
2.2.1 Kecerdasan Buatan

Kecerdasan Buatan adalah sistem komputer yang dirancang untuk melaksanakan tugas-tugas yang biasanya memerlukan kecerdasan manusia. Fokus utama dari pengembangan AI adalah menciptakan program-program yang mampu berpikir dan mengambil keputusan secara cerdas, meniru kemampuan berpikir manusia. Beberapa aplikasi kecerdasan buatan termasuk pengenalan suara, pengenalan wajah, permainan komputer, kendaraan otonom, dan sistem rekomendasi. Dalam lingkup Tugas Akhir ini, AI akan diimplementasikan untuk mensimulasikan proses berpikir dan pengambilan keputusan pemain dalam konteks permainan *platformer* 2D. Pemanfaatan AI di sini akan difokuskan pada kemampuan pemain untuk menentukan waktu yang optimal untuk meloncat dari satu platform ke platform lainnya.

2.2.2 Unity

Unity Game Engine adalah platform pengembangan permainan yang sangat populer, digunakan untuk menciptakan permainan 2D dan 3D. Unity mendukung seluruh proses pembuatan gim, termasuk desain level, animasi, fisika, dan skrip. Permainan yang dibangun dengan Unity dapat dijalankan di berbagai platform seperti Windows, macOS, Linux, Android, iOS, konsol gim seperti PlayStation dan Xbox, serta perangkat *virtual reality* seperti Oculus Rift dan HTC Vive. Dengan antarmuka grafis yang memudahkan pengembangan permainan secara visual, Unity memberikan fleksibilitas dan kenyamanan kepada pengembang. Bahasa pemrograman C# digunakan sebagai bahasa utama untuk scripting dalam Unity. Dengan mesin fisika dan grafis yang canggih, Unity memungkinkan pembuatan permainan dengan visual menarik dan respons yang realistis terhadap interaksi pemain. Dalam lingkup Tugas Akhir ini, Unity akan digunakan sebagai *game engine* untuk membuat prototipe gim *platformer*. Gambar 2.1 menunjukkan

contoh tampilan Unity Engine.



Gambar 2.1 Tampilan Unity Engine

2.2.2.1 *Package*

Dalam konteks Unity Engine, *package* adalah sekumpulan skrip, aset, dan data yang dikemas bersama untuk distribusi dan dapat digunakan ulang dalam proyek Unity. *Package* memungkinkan pengembang dengan mudah mengintegrasikan fitur-fitur yang sudah ada tanpa harus membuatnya dari awal. *Package* juga mempermudah pemeliharaan kode dan kolaborasi antar tim. Unity sendiri menyediakan beberapa jenis *package* yang dapat digunakan untuk pengembangan gim dan aplikasi, di antaranya:

1. *Asset Packages*
   Asset packages adalah kumpulan file dan data dari proyek Unity, atau elemen-elemen proyek, yang dikompres dan disimpan dalam satu file dengan ekstensi *.unitypackage*. Seperti file zip, paket aset mempertahankan struktur direktori aslinya saat dibongkar, serta metadata tentang asset, seperti pengaturan impor dan tautan ke aset lainnya.

2. *Unity Packages*
   Ini adalah *package* yang dikelola menggunakan *Unity Package Manager*. *Package* ini sering kali berisi fitur atau alat tambahan yang dikembangkan oleh Unity Technologies atau komunitas pengembang. Contohnya termasuk *package* untuk merender HDRP (*High Definition Render Pipeline*), URP (*Universal Render Pipeline*), dan berbagai layanan Unity lainnya.

2.2.2.2 *Scriptable Object*

ScriptableObject adalah kelas yang digunakan untuk menyimpan data yang dapat disesuaikan dan digunakan kembali. *ScriptableObject* memungkinkan pengembang untuk menghemat memori. Kelas ini sangat berguna untuk menyimpan data konstan yang tidak berubah selama *runtime*, seperti konfigurasi, pengaturan permainan, atau data statistik karakter. *ScriptableObject* dapat digunakan kembali di berbagai tempat dalam proyek, memungkinkan pengembang untuk mendefinisikan data sekali dan menggunakannya di banyak *GameObject* atau komponen. Dengan menggunakan *ScriptableObject*, pengembang dapat mengurangi

jumlah salinan data yang sama dalam memori, karena data disimpan di satu tempat dan dirujuk oleh banyak objek.

2.2.2.3 *Game Object*

GameObject adalah elemen dasar yang membentuk seluruh objek dalam sebuah scene di Unity. Sebagai wadah, *GameObject* mampu menyimpan berbagai komponen yang menetapkan perilaku, penampilan, dan interaksi objek tersebut dalam lingkungan gim. Dalam hierarki, *GameObject* dapat diatur untuk memudahkan pengelompokan dan pengelolaan. Objek-objek ini dapat memiliki hubungan induk-anak, di mana transformasi (posisi, rotasi, dan skala) objek anak relatif terhadap objek induknya. Setiap *GameObject* secara otomatis memiliki komponen *Transform* yang mengendalikan posisi, rotasi, dan skala objek dalam ruang 3D atau 2D, serta menentukan hubungan hierarkis antar *GameObject*.

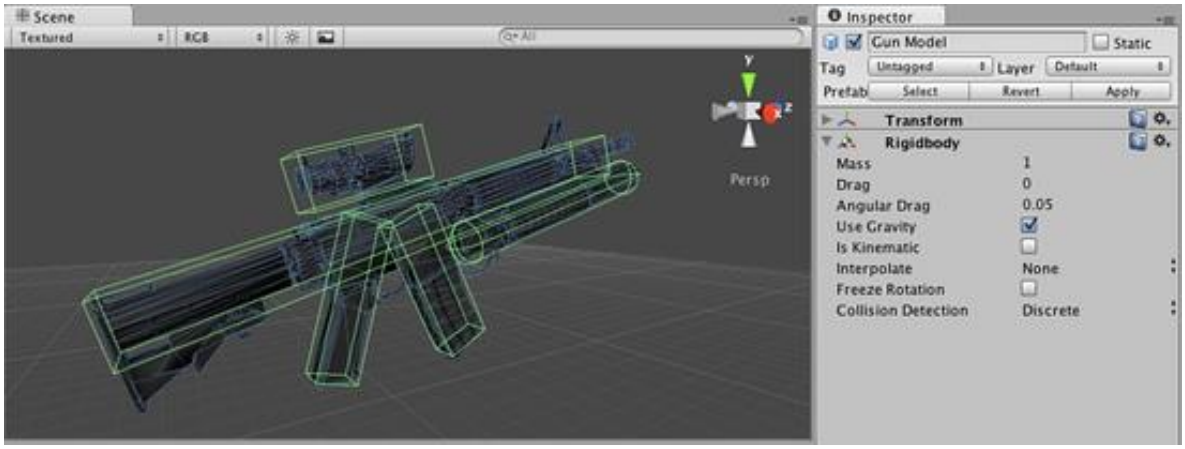
2.2.2.4 *Component*

Component atau komponen adalah elemen dasar yang menetapkan perilaku dan karakteristik *GameObject* dalam Unity. Awalnya, setiap *GameObject* adalah entitas kosong tanpa fungsi bawaan, namun dengan menambahkan komponen, *GameObject* dapat diperkaya dengan berbagai kemampuan dan fitur. Komponen berfungsi sebagai unit fungsional yang bisa digabungkan untuk membentuk perilaku yang kompleks. Komponen memungkinkan pengembang untuk menyisipkan berbagai fitur ke *GameObject* secara modular. Sebagai contoh, dengan menambahkan komponen *Rigidbody*, *GameObject* dapat berinteraksi dengan sifat fisika, sedangkan dengan komponen *AudioSource*, *GameObject* bisa memainkan suara.

2.2.2.5 *Collider*

Collider adalah komponen yang digunakan untuk menentukan area fisik atau bentuk interaktif dari sebuah *GameObject*. *Collider* menentukan bagaimana *GameObject* bereaksi terhadap interaksi fisik seperti tumbukan, deteksi tabrakan, atau perubahan dalam lingkungan simulasi fisika Unity. *Collider* mendefinisikan bentuk atau area fisik dari *GameObject* yang dapat berinteraksi dengan objek lain dalam scene.

Collider bekerja sama dengan komponen Rigidbody untuk memberikan simulasi fisika yang realistis. *Collider* memberi tahu Unity tentang area di mana gaya atau kecepatan akan berpengaruh pada *GameObject*, sementara *Rigidbody* menangani perhitungan fisika aktual seperti gravitasi, impuls, dan tumbukan. Gambar 2.2 menampilkan contoh *collider*.



Gambar 2.2 Contoh *Collider*

2.2.3 Platform

Dalam gim *platformer* 2D, platform merujuk pada permukaan datar yang menjadi pijakan bagi karakter atau objek dalam permainan. Platform memberikan landasan bagi karakter untuk bergerak, melompat, dan berinteraksi. Kemampuan karakter melompat dari satu platform ke platform lainnya menjadi keterampilan inti dalam gim ini, memungkinkan mereka mencapai tujuan, menghindari rintangan, atau mengumpulkan objek permainan. Platform dalam gim *platformer* 2D dapat memiliki berbagai karakteristik yang memengaruhi dinamika permainan. Gambar 2.3 menampilkan contoh dari gim *platformer*.

Gambar 2.3 Contoh Game *Platformer*

2.2.3.1 Platform Tetap

Platform Tetap adalah platform diam yang menjadi jalan yang aman bagi karakter untuk berjalan atau melompat.

2.2.3.2 Platform Bergerak

Platform Bergerak adalah platform yang bergerak dari satu lokasi ke lokasi lain, menuntut karakter untuk menyesuaikan pergerakan mereka.

2.2.3.3 Platform Jebakan

Platform Jebakan adalah platform yang dapat membahayakan karakter, seperti platform yang jatuh atau berubah bentuk tiba-tiba.

2.2.4 *Pathfinding*

Pathfinding adalah proses mencari jalur terbaik atau terpendek antara dua titik dalam suatu ruang, yang bisa berupa graf, grid, atau lingkungan lainnya. *Pathfinding* banyak digunakan dalam berbagai aplikasi, termasuk permainan video. Proses ini melibatkan penggunaan algoritma tertentu untuk menentukan jalur yang efisien dan menghindari rintangan atau hambatan yang ada di sepanjang jalur tersebut. Salah satu algoritma yang dapat digunakan adalah algoritma A*.

Pathfinding adalah aspek krusial dalam pengembangan gim, terutama dalam gim *platformer* di mana karakter harus menavigasi lingkungan yang kompleks. Sebuah perbandingan komprehensif berbagai algoritma *pathfinding* menyoroti pentingnya memilih algoritma yang tepat berdasarkan kriteria seperti waktu eksekusi dan panjang jalur terpendek (Lawande et al.,

2022). Penggunaan algoritma yang tepat dapat membuat gim menjadi lebih baik dalam hal proses komputasi, penggunaan memori, dan waktu komputasi (Handy Permana et al., 2018).

2.2.5 Algoritma A*

A* adalah algoritma traversal graf dan pencarian jalur, yang digunakan dalam banyak bidang ilmu komputer karena kelengkapannya, optimalitas, dan efisiensinya yang optimal (Russell & Norvig, 2021). Diberikan graf berbobot, simpul sumber, dan simpul tujuan, algoritma ini mencari jalur terpendek dari sumber ke tujuan.

A* menggunakan heuristik, biasanya jarak Euclidean atau Manhattan, untuk mengestimasi biaya dari suatu titik ke tujuan. Heuristik ini membantu algoritma memprioritaskan titik yang lebih dekat ke tujuan, mempercepat pencarian jalur terpendek. Algoritma A* mempertimbangkan biaya sejauh ini ditambah dengan estimasi biaya tersisa atau heuristik, untuk memilih titik berikutnya dalam pencarian. Hal ini meminimalkan jumlah simpul yang harus diperiksa.

(Halaman ini sengaja dikosongkan)

BAB III
METODOLOGI

3.1 Metode yang Digunakan

Pada subbab ini akan dijelaskan mengenai berbagai metode yang digunakan dalam perancangan dan pengembangan *package*. Metode yang diterapkan mencakup studi literatur, perancangan *package*, pengembangan prototipe, pengujian dan evaluasi *package*, serta penyusunan Buku Tugas Akhir.

3.1.1 Studi Literatur

Pada bagian ini, pengembangan dimulai dengan mengumpulkan dan mengkaji berbagai sumber literatur yang relevan. Tujuannya adalah untuk memahami konsep dan teknik yang sudah ada dalam bidang yang diteliti.

3.1.2 Perancangan *Package*

Bagian ini membahas tentang tahapan perancangan *package*, yang mencakup identifikasi fitur dan penentuan komponen-komponen utama yang akan dikembangkan. Perancangan yang baik akan menjadi dasar yang kuat untuk pengembangan *package* selanjutnya.

3.1.3 Pengembangan Prototipe

Pada bagian ini, akan dikembangkan dua buah prototipe. Prototipe pertama dikembangkan oleh penulis sendiri, sedangkan prototipe kedua adalah prototipe yang sudah jadi dan dikembangkan oleh pihak ketiga. Kedua prototipe ini digunakan untuk menguji efektivitas *package* nantinya.

3.1.4 Pengujian dan Evaluasi *Package*

Bagian ini menjelaskan metode pengujian dan evaluasi yang digunakan untuk menilai kinerja dan efektivitas *package*. Pengujian dilakukan dengan berbagai uji coba untuk memastikan bahwa *package* dapat berfungsi dengan baik dalam berbagai kondisi. Evaluasi dilakukan berdasarkan hasil pengujian.

3.1.5 Penyusunan Buku Tugas Akhir

Pada bagian ini, penulis akan menyusun Buku Tugas Akhir. Buku Tugas Akhir mencakup seluruh proses pengembangan dari awal hingga akhir. Penyusunan Buku Tugas Akhir dilakukan dengan memperhatikan kaidah penulisan ilmiah agar dapat dipahami dan digunakan sebagai referensi oleh pembaca.

3.2 Peralatan Pendukung

Terdapat beberapa peralatan hardware dan software pendukung yang digunakan dalam pengembangan Tugas Akhir ini adalah sebagai berikut.

- PC
- Unity
- Visual Studio Code

3.3 Implementasi

Pada subbab ini akan dijelaskan mengenai implementasi algoritma pencarian jalur pada gim *platformer* melalui penggunaan graf. Selain itu, akan dibahas juga perancangan *package* yang berisi detail-detail seperti komponen-komponen yang terdapat di dalamnya.

3.3.1 Definisi Umum

Pendekatan yang digunakan dalam tugas akhir ini adalah pembuatan *platform graph*, di mana graf ini menghubungkan titik-titik posisi lompatan dan posisi mendarat dengan lintasan. Lintasan tersebut dapat dihitung menggunakan persamaan gerakan satu dimensi. Persamaan ini melibatkan waktu sebagai input, sehingga posisi sebuah objek bergerak dapat diprediksi dalam rentang waktu yang diinginkan. Lintasan yang dihasilkan dari persamaan tersebut dapat digunakan untuk menentukan pergerakan yang akan diambil oleh sebuah objek bergerak saat berpindah antar platform.

Pergerakan horizontal dan vertikal dihitung secara terpisah untuk memudahkan implementasi. Persamaan 3.1 merupakan persamaan pergerakan horizontal yang digunakan.

$$s = s_0 + v_0 t \tag{3.1}$$

dimana:
s adalah posisi akhir
s_0 adalah posisi awal
v_0 adalah kecepatan awal
t adalah waktu

Persamaan 3.2 merupakan persamaan pergerakan vertikal yang digunakan.

$$s = s_0 + v_0 t + \frac{1}{2} a t^2 \tag{3.2}$$

dimana:
s adalah posisi akhir
s_0 adalah posisi awal
v_0 adalah kecepatan awal
a adalah percepatan
t adalah waktu

Dengan menggunakan kedua persamaan ini, kita dapat memprediksi lintasan objek yang bergerak, baik dalam arah horizontal maupun vertikal. Hal ini memungkinkan pergerakan antar platform menjadi lebih realistis dan akurat dalam implementasi algoritma *pathfinding* pada gim *platformer*.

Graf yang dibentuk dalam penelitian ini memiliki titik-titik di sepanjang platform sebagai *node*. *Node-node* ini merupakan titik di mana objek akan melompat dan mendarat. Lintasan yang dibentuk dari persamaan gerak digunakan sebagai *edge*. *Node* dan *edge* diperoleh dengan memparsing platform ke dalam wadah data, sehingga data tersebut dapat digunakan untuk mencari jalur yang optimal saat *runtime*. Dengan meningkatnya ukuran peta dan jumlah agen, penting untuk mengembangkan algoritma pencarian jalur yang efisien dalam penggunaan memori dan waktu (Lee & Lawrence, 2013).

Algoritma pencarian jalur yang digunakan dalam tugas akhir ini adalah algoritma A*. Algoritma ini dipilih karena kemampuannya dalam menemukan jalur terpendek secara efisien. Dengan menggabungkan metode graf berbasis *node* dan *edge* dengan algoritma A*, sistem

dapat menghitung jalur optimal bagi objek untuk bergerak dari satu platform ke platform lainnya.

3.3.2 Implementasi Parse Platform ke dalam Graf

Pada subbab ini akan dijelaskan mengenai cara memparsing platform ke dalam bentuk graf. Graf ini kemudian dapat digunakan untuk mencari jalur dengan menggunakan algoritma pencarian.

3.3.2.1 Mendapatkan Titik–Titik di Sudut Platform

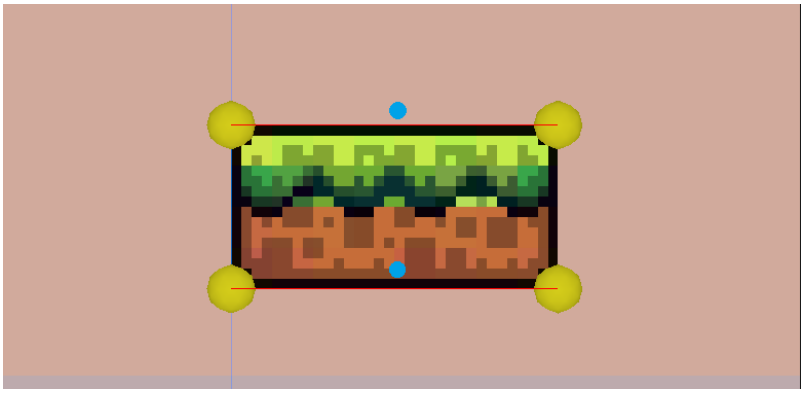
Sebelum dapat menghitung *node* dan *edge* dari graf, kita perlu menentukan titik-titik di sudut platform terlebih dahulu. Di dalam Unity, titik-titik ini dapat ditentukan dengan menggunakan *Composite Collider*. *Composite Collider* adalah jenis *collider* yang menggabungkan beberapa *collider* menjadi satu bentuk yang lebih kompleks. Dengan menggunakan *Composite Collider*, kita dapat memperoleh koordinat titik-titik yang membentuk platform di dalam gim.

Gambar 3.1 Titik–titik di sudut platform

Lingkaran berwarna kuning pada Gambar 3.1 menunjukkan titik-titik di sudut platform. Titik-titik ini kemudian dapat digunakan untuk menentukan sisi mana yang berada di atas, kanan, kiri, ataupun bawah pada suatu platform.

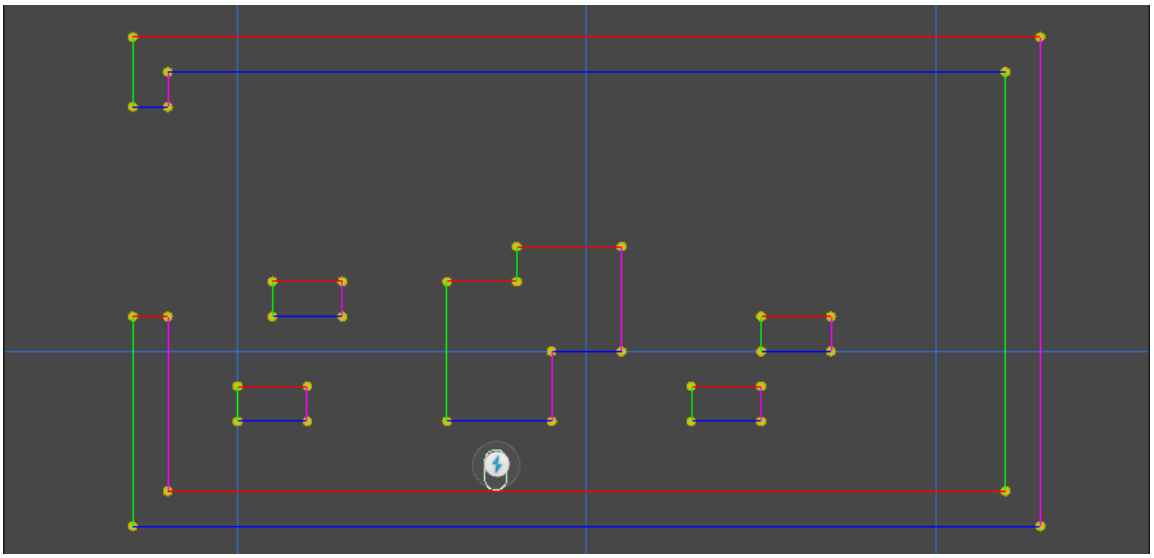
3.3.2.2 Mendapatkan Sisi Platform

Langkah selanjutnya adalah menentukan sisi pada setiap platform. Hal ini penting dilakukan karena lompatan-lompatan pada gim *platformer* umumnya dilakukan dari sisi atas sebuah platform. Di dalam Unity, *collider* memiliki fungsi untuk menguji apakah sebuah titik berada di dalam *collider* atau tidak. Gambar 3.2 menunjukkan visualisasi pengujian sisi platform.



Gambar 3.2 Pengujian sisi platform

Titik tengah dapat ditentukan pada sebuah sisi dan menggesernya ke atas pada sumbu y jika sisi tersebut merupakan sisi horizontal. Kemudian, uji apakah titik tersebut berada di dalam *collider* atau tidak. Jika tidak, maka sisi tersebut adalah sisi atas. Jika ya, maka sisi tersebut adalah sisi bawah. Hal yang sama berlaku untuk sisi vertikal, dengan menggeser titik tengah ke kanan pada sumbu x untuk mengidentifikasi sisi kanan dan kiri. Gambar 3.3 menunjukkan visualisasi sisi platform dari peta yang disediakan.



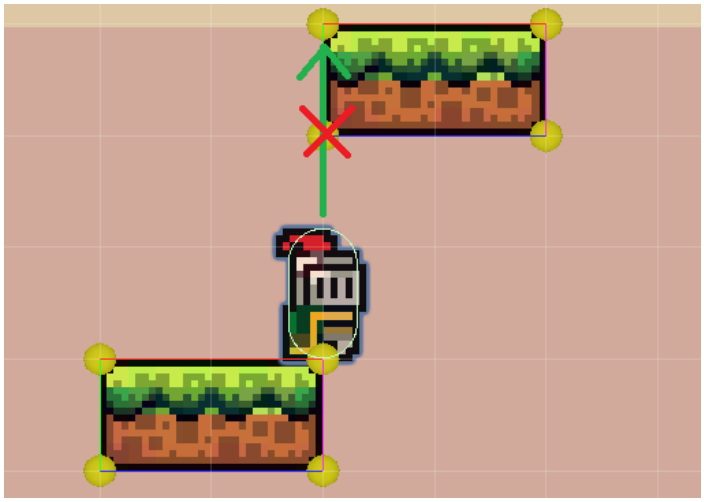
Gambar 3.3 Visualisasi sisi platform

3.3.2.3 Menentukan Titik Lompat dan Mendarat

Pada subbab ini akan dijelaskan secara mendetail mengenai konsep titik lompat dan titik mendarat. Titik lompat merupakan lokasi tertentu di mana karakter memulai lompatan dari satu platform ke platform lainnya. Sebaliknya, titik mendarat adalah lokasi di mana karakter mendarat setelah melakukan lompatan.

3.3.2.3.1 Titik di Sudut Platform yang Digeser

Titik lompat dan titik mendarat yang pertama terletak pada titik di sudut platform yang telah digeser. Titik ini digeser untuk menghindari potensi tabrakan pergerakan antar platform, seperti yang dapat dilihat pada Gambar 3.4.



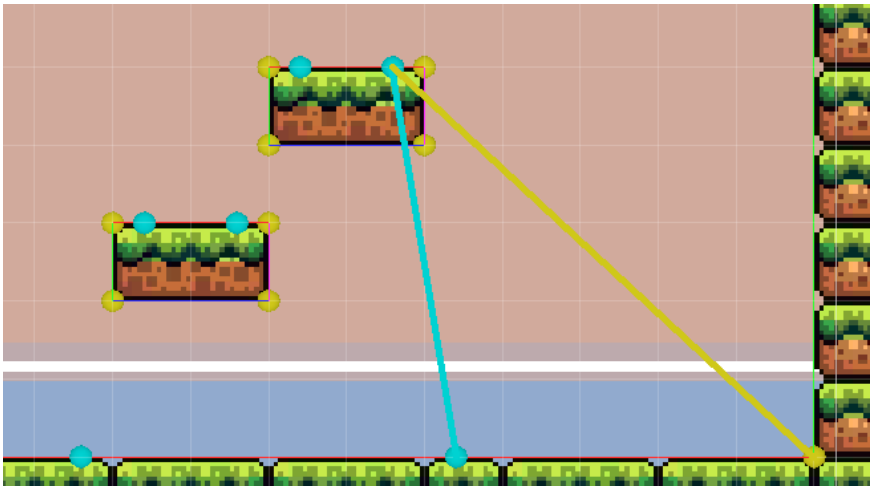
Gambar 3.4 Skenario melompat di sudut platform

Seperti yang dapat dilihat pada Gambar 3.4, ketika objek hendak melompat ke atas, objek tersebut bertabrakan dengan platform. Oleh karena itu, titik ini harus digeser setidaknya sejauh setengah dari lebar *collider* pada objek untuk memberikan ruang yang cukup saat melompat, seperti yang dapat dilihat pada Gambar 3.5.

Gambar 3.5 Skenario melompat di sudut platform yang digeser

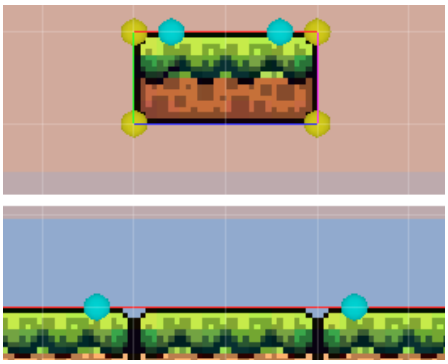
3.3.2.3.2 Titik di Platform yang Berada di Bawah Sudut Platform yang Lain

Titik lompat dan titik mendarat yang kedua terletak pada titik di sepanjang platform yang berada tepat di bawah sudut platform lainnya. Titik ini sangat cocok dipertimbangkan sebagai titik lompat dan mendarat karena memiliki kemungkinan lintasan yang lebih pendek dibandingkan dengan titik yang berada di sudut platform.



Gambar 3.6 Perbandingan titik di bawah platform dan titik di sudut platform

Gambar 3.6 menunjukkan perbandingan antara titik yang berada di bawah platform lain dan titik di sudut platform. Dalam gambar tersebut, dapat dilihat bahwa garis berwarna biru memiliki jarak yang lebih pendek dibandingkan dengan garis berwarna kuning. Oleh karena itu, titik pada garis biru lebih cocok untuk dijadikan sebagai pasangan titik lompat dan titik mendarat. Namun, kita juga harus mempertimbangkan ketika kita memiliki dua pasang titik lompat dan titik mendarat. Gambar 3.7 menunjukkan contoh dua pasang titik lompat dan titik mendarat.



Gambar 3.7 Dua pasang titik lompat dan titik mendarat

3.3.2.4 Kalkulasi Jalur Antar *Node*

Seperti yang dijelaskan sebelumnya, kalkulasi jalur antar *node* dilakukan dengan menghitung posisi objek pada suatu waktu, baik itu *node* pada satu platform ataupun *node* antar platform. Pergerakan objek dibedakan menjadi tiga jenis, yaitu pergerakan horizontal, pergerakan vertikal, dan pergerakan gabungan antara pergerakan horizontal dan vertikal.

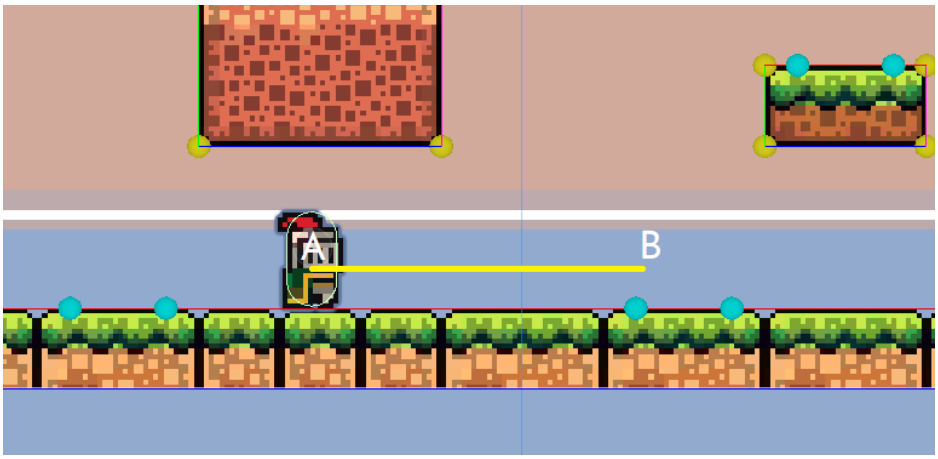
Di dalam Unity, perhitungan yang berhubungan dengan fisika dilakukan di dalam metode *FixedUpdate*. *FixedUpdate* dijalankan berdasarkan pengaturan yang telah ditentukan di dalam Unity. Secara default, *FixedUpdate* dijalankan sebanyak 50 kali per detik atau sekali setiap 0,02 detik. Kita dapat menggunakan interval waktu ini untuk mendapatkan jeda waktu antar perpindahan posisi saat mengkalkulasi jalur, yaitu 0,02 detik.

Interval waktu tetap sebesar 0,02 detik ini sangat berguna dalam kalkulasi jalur. Misalnya, saat menghitung lintasan atau jalur yang akan diambil objek, kita dapat memastikan bahwa setiap

langkah perhitungan dilakukan setiap 0,02 detik, sehingga menghasilkan simulasi pergerakan yang lebih halus dan realistis.

3.3.2.4.1 Pergerakan Horizontal

Pergerakan horizontal dilakukan ketika ingin menghubungkan *node* yang memiliki koordinat sumbu y yang sama, seperti yang ditunjukkan pada Gambar 3.8. Dalam gambar tersebut, objek hendak bergerak dari titik A ke titik B. Pergerakan horizontal hanya digunakan saat menghitung pergerakan dari satu *node* ke *node* yang lain dalam satu platform. Untuk menghitung pergerakan horizontal, diperlukan input kecepatan awal dan waktu.



Gambar 3.8 Simulasi pergerakan horizontal

```
Function CalculateHorizontalMovement(startNode, endNode, initialVelocity, timeStep):
    currentPosition = startNode.position

    distanceToCover = endNode.position.x - startNode.position.x

    If distanceToCover > 0:
        direction = 1  # Bergerak ke kanan
    Else:
        direction = -1  # Bergerak ke kiri

    totalTime = distanceToCover / (initialVelocity * direction)

    currentTime = 0

    positionTimeList = []

    While currentTime < totalTime:
        currentPosition.x = currentPosition.x + initialVelocity * direction * timeStep

        positionTimeList.append((currentPosition.x, currentTime))

        currentTime = currentTime + timeStep

    currentPosition.x = endNode.position.x
    positionTimeList.append((currentPosition.x, totalTime))
```

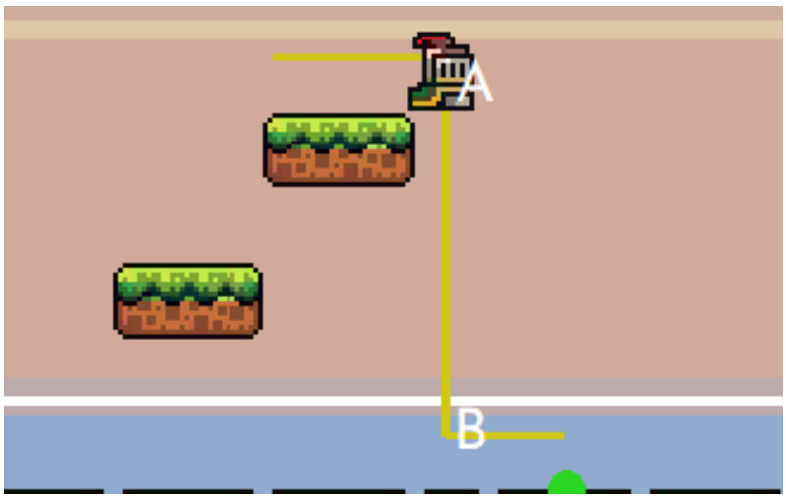
```
    Return positionTimeList
End Function
```

Pseudocode 3.1 Kode program pergerakan horizontal

Pseudocode 3.1 adalah fungsi pergerakan horizontal. Fungsi ini menerima parameter *startNode* (titik awal), *endNode* (titik tujuan), *initialVelocity* (kecepatan awal), dan *timeStep* (selang waktu untuk setiap langkah pergerakan). Daftar yang berisi posisi dan waktu dikembalikan sebagai hasil dari fungsi.

3.3.2.4.2 Pergerakan Vertikal

Pergerakan vertikal dilakukan ketika ingin menghubungkan *node* yang memiliki koordinat sumbu x yang sama. Berbeda dengan pergerakan horizontal, pergerakan vertikal melibatkan perpindahan dari satu platform ke platform lainnya. Oleh karena itu, setiap pergerakan vertikal harus memperhitungkan deteksi tumbukan. Untuk menghitung pergerakan vertikal, diperlukan beberapa input, yaitu kecepatan awal, percepatan, yang dalam kasus ini adalah gravitasi, dan waktu.



Gambar 3.9 Simulasi pergerakan vertikal

Gambar 3.9 menunjukkan contoh pergerakan vertikal. Misalkan objek ingin bergerak dari titik A ke titik B. Dalam kasus ini, kita dapat mengatur kecepatan awal menjadi 0 jika titik B berada di bawah titik A, yang berarti objek sedang jatuh.

Kecepatan awal ditentukan berdasarkan kebutuhan spesifik dari pergerakan yang diinginkan. Jika titik yang dituju berada di atas titik awal, maka kita dapat mengatur kecepatan awal sesuai dengan kecepatan lompat yang telah ditetapkan di inspektor Unity.

Misalnya, jika karakter melompat ke platform yang lebih tinggi, kecepatan awal harus cukup besar untuk mengatasi gravitasi dan mencapai ketinggian yang diinginkan. Sebaliknya, jika karakter sedang jatuh, kecepatan awal bisa diatur menjadi 0 dan biarkan gravitasi bekerja untuk menarik karakter ke bawah.

Dalam matematika, Persamaan 3.3 adalah persamaan kuadratik dalam bentuk umum.

$$\boldsymbol{ax^2 + bx + c = 0} \tag{3.3}$$

Persamaan 3.4 adalah rumus kuadratik untuk menyelesaikan persamaan kuadratik.

$$\boldsymbol{x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}} \tag{3.4}$$

Rumus ini memungkinkan kita untuk mencari akar-akar dari persamaan kuadratik tersebut, yang merupakan solusi dari persamaan. Dalam konteks perhitungan waktu perpindahan, kita dapat memanfaatkan Persamaan 3.2. Untuk memfasilitasi pencarian waktu yang diperlukan untuk bergerak dari posisi awal ke posisi akhir, kita dapat menyusun ulang persamaan menggunakan aljabar. Persamaan 3.5 adalah susunan ulang persamaan tersebut.

$$\boldsymbol{0 = \frac{1}{2}at^2 + vt + (s_0 - s)} \tag{3.5}$$

Rumus ini merupakan bentuk persamaan yang dapat dipecahkan untuk mencari waktu yang diperlukan untuk perpindahan yang diinginkan.

```
Function CalculateMovementDuration(displacement, velocity, acceleration,
returnLowerResult = true):
    If acceleration == 0 Then
        If velocity == 0 Then
            Return PositiveInfinity
        Else If (displacement > 0) != (velocity > 0) Then
            Return PositiveInfinity
        Else
            Return displacement / velocity
        End If
    End If

    discriminant = velocity * velocity + 2 * acceleration * displacement

    If discriminant < 0 Then
        Return PositiveInfinity
    End If

    discriminantSqrt = SquareRoot(discriminant)
    t1 = (-velocity + discriminantSqrt) / acceleration
    t2 = (-velocity - discriminantSqrt) / acceleration

    If t1 < 0 And t2 < 0 Then
        Return PositiveInfinity
    End If

    If returnLowerResult Then
        Return Min(t1, t2)
    Else
        Return Max(t1, t2)
    End If
```

```
End Function
```

Pseudocode 3.2 Kode program untuk menghitung durasi lompat

Pseudocode 3.2 adalah fungsi untuk menghitung durasi lompat. Fungsi ini menerima parameter *displacement* (perpindahan yang ingin dicapai), *velocity* (kecepatan awal), *acceleration* (percepatan), dan *returnLowerResult* (boolean yang menentukan apakah hasil yang lebih rendah atau lebih tinggi yang akan dikembalikan). Fungsi mengembalikan waktu yang diperlukan untuk mencapai perpindahan yang diberikan. Fungsi ini dapat digunakan untuk menghitung durasi maksimum untuk melompat dari satu titik ke titik yang lain.

```
Function CalculateVerticalMovement(startNode, endNode, initialVelocity, gravity, timeStep):
    currentPosition = startNode.position

    verticalDisplacement = endNode.position.y - startNode.position.y

    estimatedDuration = CalculateMovementDuration(verticalDisplacement, initialVelocity, gravity, False)

    positionTimeList = []

    currentTime = 0

    While currentTime < estimatedDuration:
        If currentPosition == endNode.position:
            Break

        nextPosition = startNode.position
        nextPosition.y = nextPosition.y + initialVelocity * currentTime + 0.5 * gravity * currentTime * currentTime

        If DetectCollisionAtPosition(nextPosition):
            Return null

        currentPosition = nextPosition
        positionTimeList.append((currentPosition.y, currentTime))

        currentTime = currentTime + timeStep

    currentPosition.y = endNode.position.y
    positionTimeList.append((currentPosition.y, currentTime))

    Return positionTimeList
End Function
```

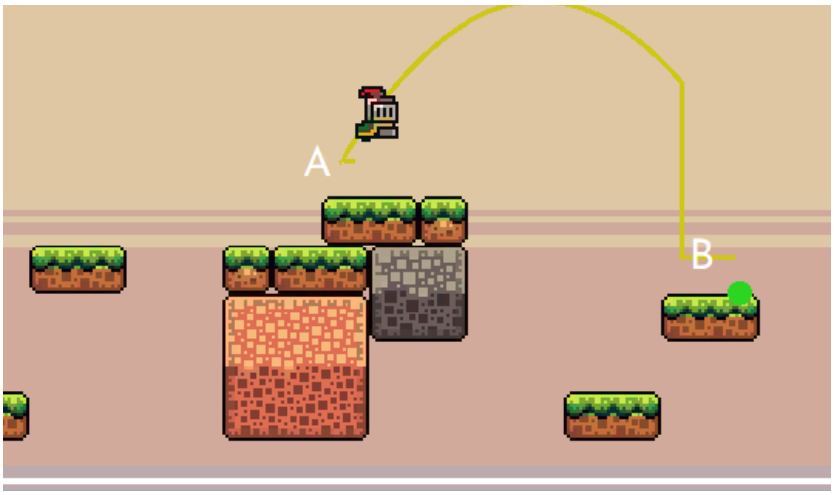
Pseudocode 3.3 Kode program pergerakan vertikal

Pseudocode 3.3 adalah fungsi untuk pergerakan vertical. Fungsi ini menerima parameter *startNode* (titik awal), *endNode* (titik tujuan), *initialVelocity* (kecepatan awal), gravity (gravitasi), dan *timeStep* (selang waktu untuk setiap langkah pergerakan). Daftar yang berisi posisi dan waktu dikembalikan sebagai hasil dari fungsi. Jika objek bertabrakan dengan objek

lain selama pergerakan, fungsi akan mengembalikan *null*.

3.3.2.4.3 Pergerakan Gabungan

Pergerakan gabungan dilakukan ketika ingin menghubungkan *node* yang memiliki koordinat x dan y yang berbeda atau menghubungkan *node* dengan sumbu x yang sama tetapi berada pada platform yang berbeda. Pergerakan gabungan mengombinasikan pergerakan horizontal dan vertikal, memungkinkan objek untuk bergerak dalam lintasan yang lebih kompleks. Setiap pergerakan ini harus memperhitungkan deteiksi tumbukan. Gambar 3.10 adalah simulasi pergerakan gabungan.



Gambar 3.10 Simulasi pergerakan gabungan

```
Function CalculateCombinedMovement(startNode, endNode, initialHorizontalVelocity,
initialVerticalVelocity, gravity, timeStep, minJumpHeight):
    currentPosition = startNode.position
    positionTimeList = []

    horizontalDisplacement = endNode.position.x - startNode.position.x

    If horizontalDisplacement > 0:
        direction = 1  # Bergerak ke kanan
    Else:
        direction = -1  # Bergerak ke kiri

    verticalDisplacement = endNode.position.y - startNode.position.y
    estimatedDuration = CalculateMovementDuration(verticalDisplacement,
initialVerticalVelocity, gravity)
    maxJumpHeight = -(initialVerticalVelocity * initialVerticalVelocity) / (2 * gravity)

    currentTime = 0
    hasPassedMinJumpHeight = False

    While currentTime < estimatedDuration:
        If currentPosition == endNode.position:
            Break
```

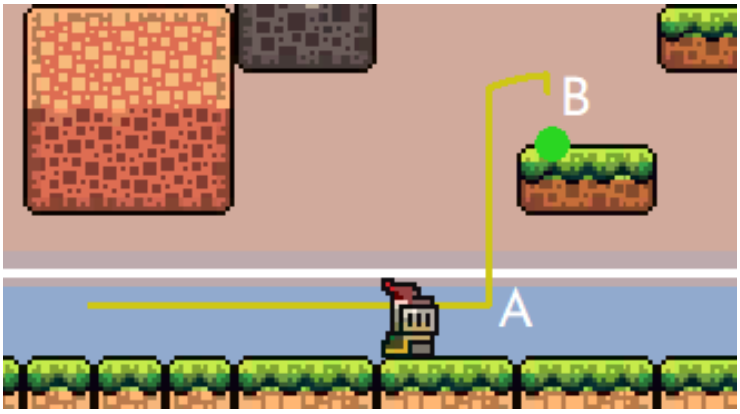
```
        nextPosition = startNode.position
        If currentPosition.y >= startNode.position.y + minJumpHeight AND NOT
hasPassedMinJumpHeight:
            hasPassedMinJumpHeight = True

        If hasPassedMinJumpHeight:
            If currentPosition.x != endNode.position.x:
                nextPosition.x = currentPosition.x + initialHorizontalVelocity * direction *
timeStep
                If direction == 1 AND nextPosition.x > endNode.position.x:
                    nextPosition.x = endNode.position.x
                Else If direction == -1 AND nextPosition.x < endNode.position.x:
                    nextPosition.x = endNode.position.x
            Else:
                nextPosition.x = endNode.position.x

        nextPosition.y = startNode.position.y + initialVerticalVelocity * currentTime + 0.5 *
gravity * currentTime * currentTime

        currentPosition = nextPosition
        positionTimeList.append((currentPosition, currentTime))

        If DetectCollisionAtPosition(nextPosition):
            If minJumpHeight >= maxJumpHeight:
                Return null
            newMinJumpHeight = Min(minJumpHeight + 1, maxJumpHeight)
            Return CalculateCombinedMovement(startNode, endNode,
initialHorizontalVelocity, initialVerticalVelocity, gravity, timeStep, newMinJumpHeight)

        currentTime = currentTime + timeStep

    currentPosition = endNode.position
    positionTimeList.append((currentPosition, currentTime))

    Return positionTimeList
End Function
```

Pseudocode 3.4 Kode program pergerakan gabungan

Pseudocode 3.4 adalah fungsi pergerakan gabungan. Fungsi ini menerima beberapa parameter, yaitu *startNode* yang merupakan posisi awal objek, *endNode* yang merupakan posisi tujuan objek, *initialHorizontalVelocity* sebagai kecepatan awal horizontal, *initialVerticalVelocity* sebagai kecepatan awal vertikal, gravity sebagai percepatan gravitasi, *timeStep* sebagai interval waktu untuk setiap langkah pergerakan, dan *minJumpHeight* sebagai ketinggian minimum lompatan. Fungsi ini mengembalikan sebuah daftar yang berisi pasangan posisi dan waktu, yang menunjukkan jalur pergerakan objek dari posisi awal ke posisi akhir dengan memperhitungkan deteksi tumbukan di sepanjang jalur.

Pada fungsi ini, objek akan bergerak ke atas pada sumbu y terlebih dahulu sebelum melakukan pergerakan horizontal. Nilai ini disebut sebagai *minJumpHeight* yang bernilai 0 pada saat fungsi

dijalankan. Ketika objek mendeteksi tumbukan, objek akan kembali ke posisi awal dan bergerak ke atas pada sumbu y yang lebih tinggi sebelum melakukan pergerakan horizontal. Tujuannya adalah agar objek dapat melakukan ini secara rekursif hingga mencapai ketinggian lompatan maksimum. Jika objek masih mendeteksi tumbukan setelah mencapai ketinggian maksimum, maka fungsi akan mengembalikan nilai *null*. Gambar 3.11 adalah contoh pergerakan dimana karakter bergerak horizontal setelah melewati *minJumpHeight* tertentu.



Gambar 3.11 Contoh pergerakan yang memiliki *minJumpHeight* lebih dari nol

Maksimum lompatan diperoleh dari persamaan kinematika dasar. Persamaan 3.6 adalah salah satu persamaan kinematika tersebut.

$$v_f{}^2 = v_i{}^2 + 2ad \tag{3.6}$$

dimana:
v_f adalah kecepatan akhir
v_i adalah kecepatan awal
a adalah percepatan
d adalah perpindahan

Ketika objek mencapai ketinggian maksimum, kecepatan akhir menjadi 0 karena objek akan berhenti sejenak sebelum berbalik arah menuju bawah. Dengan mensubstitusikan kecepatan akhir menjadi 0 dan mengisolasikan perpindahan, kita dapat menyelesaikan persamaan untuk mencari perpindahan yang merupakan ketinggian maksimum yang dapat dicapai oleh objek saat melompat. Persamaan 3.7 adalah rumus untuk mencari ketinggian maksimum.

$$d = -\frac{v_i{}^2}{2a} \tag{3.7}$$

3.3.3 Rancangan *Package*

Pada subbab ini akan dijelaskan mengenai rancangan *package* yang dikembangkan. Rancangan *package* ini mencakup komponen-komponen utama dan fungsionalitas yang diimplementasikan untuk mendukung navigasi dalam gim *platformer*.

3.3.3.1 Definisi Umum

Package pathfinding ini dikembangkan untuk memfasilitasi pencarian jalur pada gim *platformer*. *Package* ini bertujuan untuk mengatasi masalah navigasi karakter dari satu titik ke

titik lain dengan mempertimbangkan platform dan rintangan yang ada.

Package ini memiliki komponen utama bernama *Platform Navigator*, yang berfungsi sebagai antarmuka untuk mengakses berbagai fungsi dan kemampuan yang disediakan oleh *package*. *Platform Navigator* adalah komponen yang dapat dipasang pada *Game Object* yang diinginkan untuk mendapatkan kemampuan navigasi antar platform.

Untuk memanfaatkan kemampuan yang disediakan oleh *package* ini, diperlukan pembuatan skrip khusus yang akan memanggil fungsi-fungsi dari *Platform Navigator*. Skrip ini memungkinkan pengguna untuk mengatur dan mengontrol pergerakan karakter secara dinamis sesuai dengan kebutuhan gim yang sedang dikembangkan.

3.3.3.2 Komponen

Pada subbab ini akan dijelaskan mengenai komponen-komponen yang terdapat dalam *package* yang dikembangkan. Setiap komponen memiliki peran dan fungsi spesifik yang berkontribusi terhadap keseluruhan fungsi dari *package* ini.

3.3.3.2.1 *Platform Navigator*

Gambar 3.12 menunjukkan diagram *PlatformNavigator*, yang mencakup atribut, fungsi, serta relasinya dengan kelas-kelas lain. Tabel 3.1 menyajikan deskripsi dan fungsi utama dari *PlatformNavigator* secara rinci.

Gambar 3.12 Diagram *platform navigator*

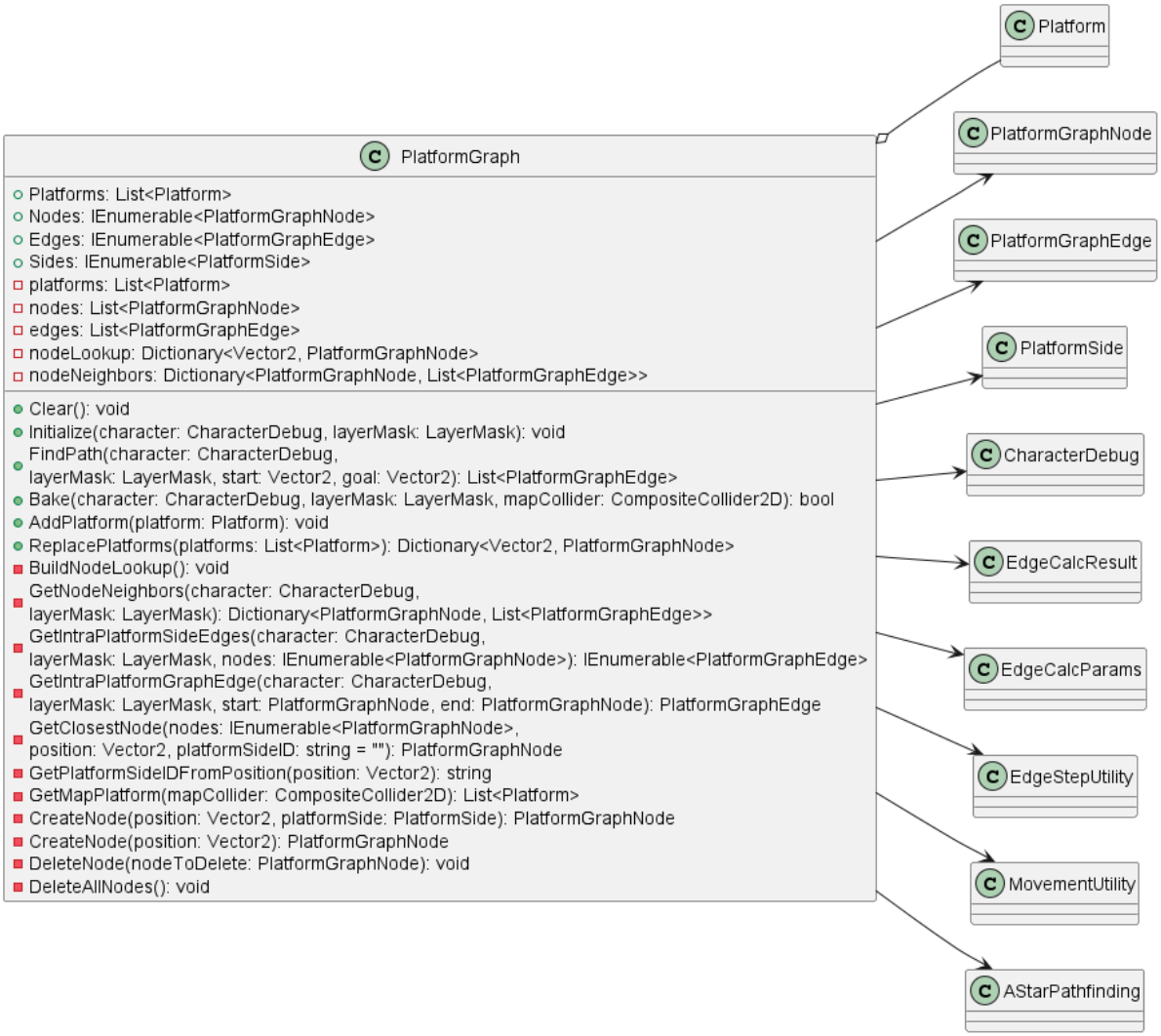
Tabel 3.1 Komponen *Platform Navigator*

ID	**K-01**
Nama	*Platform Navigator*
Deskripsi	*Platform Navigator* adalah komponen utama dalam *package* ini. Komponen ini dapat dipasang pada *game object* yang ingin diberi fitur bergerak antar platform. *Platform Navigator* berfungsi untuk mengelola navigasi antar platform, termasuk melakukan *raycast* pada platform dan bergerak ke titik tertentu. Dengan menggunakan komponen ini, *game object* dapat menentukan jalur dari posisi awal menuju destinasi
Fungsi Utama	1. *RecalculateGraph* Fungsi ini digunakan untuk menghitung ulang graf berdasarkan platform yang diberikan. 2. *MoveTo* Fungsi ini digunakan untuk bergerak dari titik di mana

	objek berada menuju posisi tujuan. Fungsi ini hanya dapat menerima posisi yang telah di-*raycast* terhadap platform. 3. *RaycastToPlatform* Fungsi ini digunakan untuk melakukan *raycast* ke bawah dari suatu posisi terhadap platform. Posisi yang diperoleh dari fungsi ini dapat digunakan untuk fungsi *MoveTo*. 4. *Stop* Fungsi ini digunakan untuk menghentikan pergerakan.

3.3.3.2.2 *Platform Graph*

Gambar 3.13 menunjukkan diagram *PlatformGraph*, yang mencakup atribut, fungsi, serta relasinya dengan kelas-kelas lain. Tabel 3.2 menyajikan deskripsi dan fungsi utama dari *PlatformGraph* secara rinci.



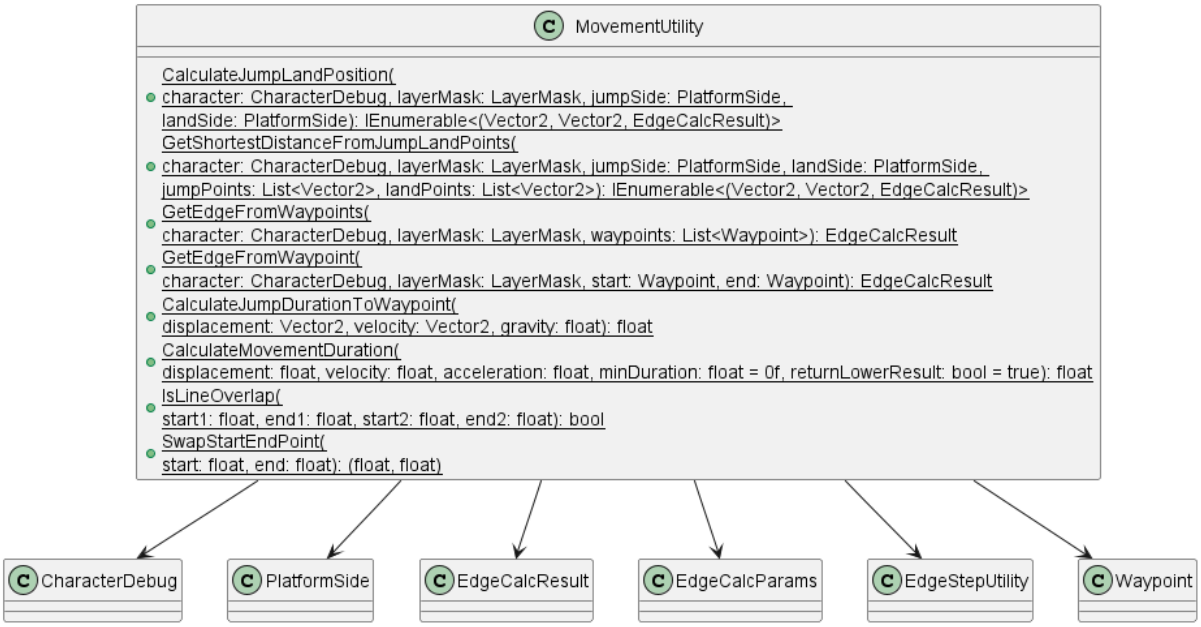
Gambar 3.13 Diagram *platform graph*

Tabel 3.2 Komponen *Platform Graph*

ID	**K-02**
Nama	*Platform Graph*
Deskripsi	*Platform Graph* adalah komponen yang menyimpan informasi graf dari platform. Informasi yang disimpan dalam komponen ini meliputi *node*, *edge*, dan platform yang ada di dalam peta. Komponen ini berupa *scriptable object* sehingga dapat dibuat sebelum *runtime* dan digunakan saat *runtime*.
Fungsi Utama	1. *FindPath* Fungsi ini digunakan untuk mencari jalur dari titik awal menuju titik akhir. 2. *GetClosestNode* Fungsi ini digunakan untuk mencari *node* terdekat berdasarkan input posisi. *Node* yang dikembalikan dari fungsi ini hanya *node* yang berada di platform yang sama dengan input. 3. *Bake* Fungsi ini digunakan untuk menghitung graf berdasarkan platform yang diberikan dan menyimpannya dalam *instance* baru dari *Platform Graph*. 4. *GetMapPlatform* Mendapatkan platform dari peta yang diberikan.

3.3.3.2.3 *Movement Utility*

Gambar 3.14 menunjukkan diagram *MovementUtility*, yang mencakup atribut, fungsi, serta relasinya dengan kelas-kelas lain. Tabel 3.3 menyajikan deskripsi dan fungsi utama dari *MovementUtility* secara rinci.



Gambar 3.14 Diagram *movement utility*

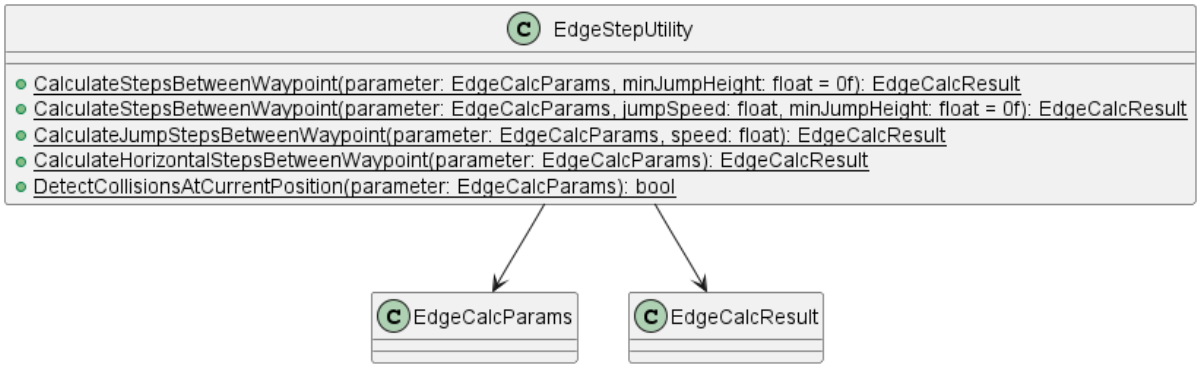
Tabel 3.3 Komponen *Movement Utility*

ID	**K-03**
Nama	*Movement Utility*
Deskripsi	*Movement Utility* adalah komponen yang menyediakan berbagai metode utilitas untuk mendukung perhitungan navigasi karakter. Ini termasuk perhitungan posisi lompatan dan mendarat, pergerakan antar *waypoints*, dan durasi pergerakan.
Fungsi Utama	1. *CalculateMovementDuration* Fungsi ini menghitung durasi waktu yang dibutuhkan untuk perpindahan berdasarkan perpindahan, percepatan, dan durasi minimum. Fungsi ini juga dapat mengembalikan durasi yang lebih rendah atau lebih tinggi tergantung pada parameter *returnLowerResult*. 2. *GetEdgeFromWaypoint* Fungsi ini digunakan untuk mendapatkan *edge* dari titik awal dan titik akhir. Fungsi ini mempertimbangkan apakah titik-titik tersebut sejajar secara horizontal, vertikal, atau tidak sejajar untuk menentukan gerakan yang valid. 3. *GetEdgeFromWaypoints* Fungsi ini digunakan untuk mendapatkan *edge* dari beberapa titik yang diberikan. Fungsi ini menggabungkan hasil kalkulasi *edge* dari setiap pasangan titik berurutan untuk menghasilkan rute lengkap dari titik awal ke titik akhir.

	4. *GetShortestDistanceFromJumpLandPoints* Fungsi ini mencari jarak terpendek antara titik-titik lompatan dan titik-titik pendaratan yang diberikan. Fungsi ini mempertimbangkan berbagai kemungkinan rute dan memilih rute dengan jarak terpendek, mengembalikan pasangan titik lompatan dan pendaratan beserta hasil kalkulasi *edge*-nya. Fungsi ini dapat mengembalikan lebih dari satu pasangan titik lompatan dan pendaratan. 5. *CalculateJumpLandPosition* Fungsi ini menghitung dan mengembalikan posisi lompatan dari satu platform ke platform lain. Fungsi ini memperhitungkan berbagai titik dari kedua platform dan mengembalikan pasangan titik lompatan dan pendaratan beserta hasil kalkulasi *edge*-nya. Fungsi ini dapat mengembalikan lebih dari satu pasangan titik lompatan dan pendaratan.

3.3.3.2.4 *Edge Step Utility*

Gambar 3.15 menunjukkan diagram *EdgeStepUtility*, yang mencakup atribut, fungsi, serta relasinya dengan kelas-kelas lain. Tabel 3.4 menyajikan deskripsi dan fungsi utama dari *EdgeStepUtility* secara rinci.



Gambar 3.15 Diagram *edge step utility*

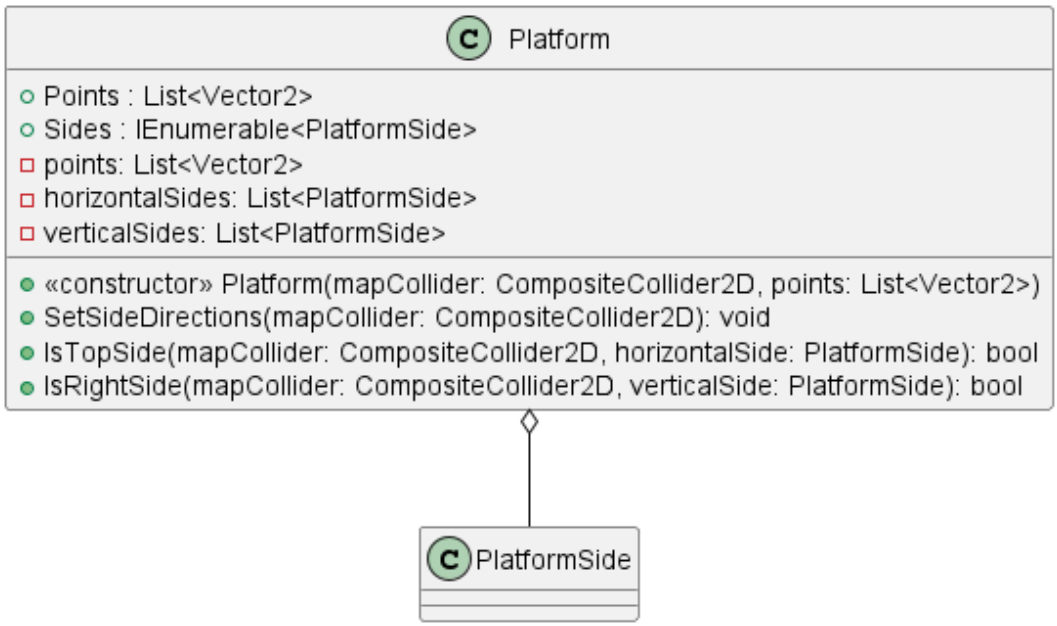
Tabel 3.4 Komponen *Edge Step Utility*

ID	**K-04**
Nama	*Edge Step Utility*
Deskripsi	*Edge Step Utility* adalah komponen yang menyediakan berbagai metode utilitas untuk menghitung jalur antara titik-titik *waypoint*, termasuk jalur horizontal, vertikal, dan kombinasi keduanya.
Fungsi Utama	1. *DetectCollisionAtCurrentPosition* Fungsi ini digunakan untuk mendeteksi tumbukan pada karakter dengan parameter posisi yang diberikan. 2. *CalculateHorizontalStepsBetweenWaypoint*

	Fungsi ini digunakan untuk menghitung langkah-langkah horizontal antara dua titik *waypoint* yang sejajar secara vertikal. 3. *CalculateJumpStepsBetweenWaypoint* Fungsi ini digunakan untuk menghitung langkah-langkah loncat antara dua titik *waypoint* yang sejajar secara horizontal. Fungsi ini tidak dapat digunakan untuk menghitung langkah-langkah yang membutuhkan pergerakan horizontal. 4. *CalculateStepsBetweenWaypoint* Fungsi ini digunakan untuk menghitung langkah-langkah antara dua titik *waypoint* dengan mempertimbangkan kemampuan lonpat dan pergerakan horizontal karakter. Fungsi ini merupakan gabungan dari pergerakan horizontal dan kemampuan lompat.

3.3.3.2.5 Platform

Gambar 3.16 menunjukkan diagram Platform, yang mencakup atribut, fungsi, serta relasinya dengan kelas-kelas lain. Tabel 3.5 menyajikan deskripsi dan fungsi utama dari Platform secara rinci.



Gambar 3.16 Diagram platform

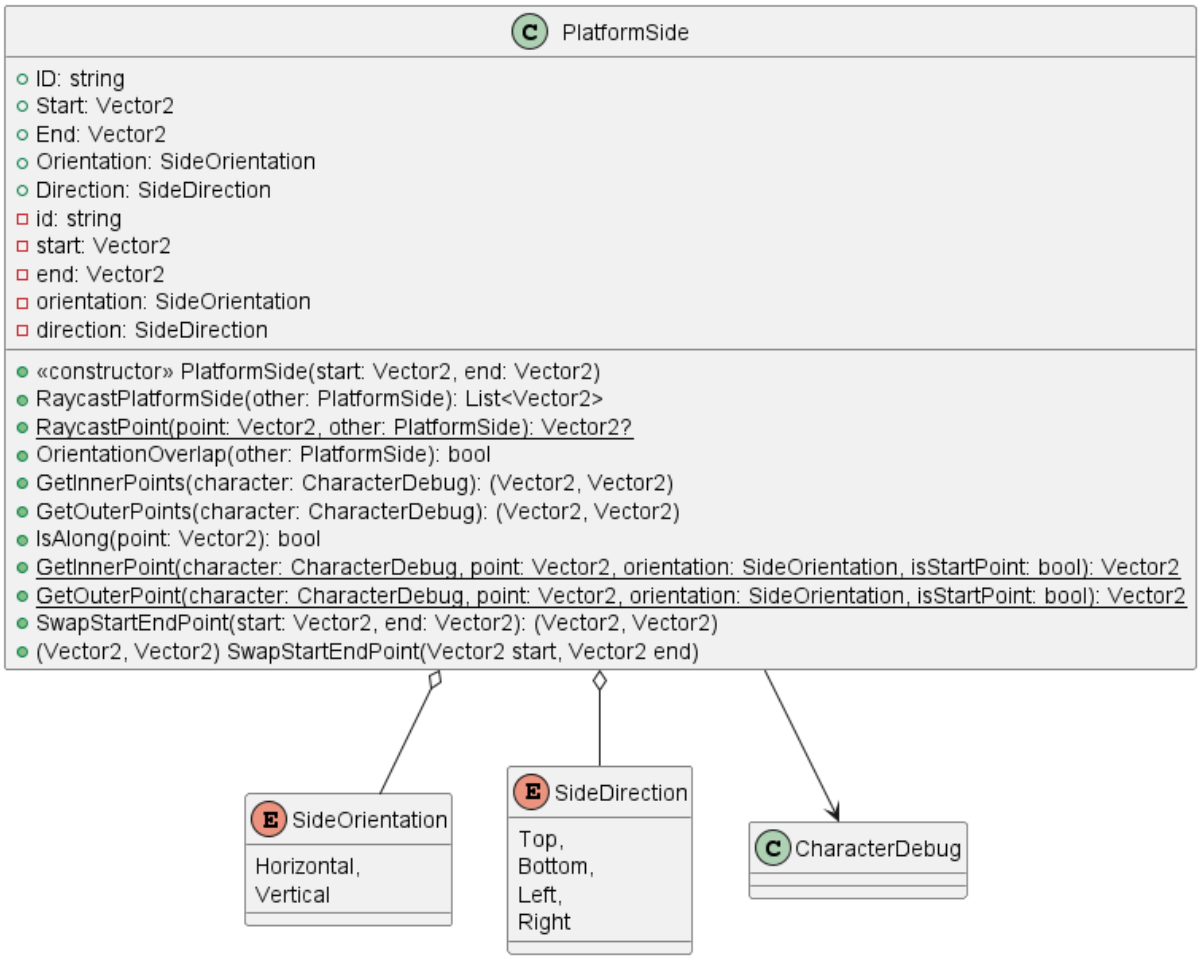
Tabel 3.5 Komponen Platform

ID	**K-05**
Nama	Platform
Deskripsi	Platform adalah komponen yang merepresentasikan sebuah platform dalam gim yang terdiri dari beberapa titik

	dan sisi-sisi platform. Setiap sisi platform dapat berorientasi horizontal atau vertikal.
Fungsi Utama	1. *SetSideDirections* Fungsi ini digunakan untuk menetapkan arah dari setiap sisi platform berdasarkan *collider* platform yang diberikan. Fungsi ini memeriksa setiap sisi platform horizontal dan vertikal. Fungsi ini akan menentukan apakah sisi tersebut adalah sisi atas, bawah, kanan atau kiri dari platform.

3.3.3.2.6 *Platform Side*

Gambar 3.17 menunjukkan diagram *PlatformSide*, yang mencakup atribut, fungsi, serta relasinya dengan kelas-kelas lain. Tabel 3.6 menyajikan deskripsi dan fungsi utama dari *PlatformSide* secara rinci.



Gambar 3.17 Diagram *platform side*

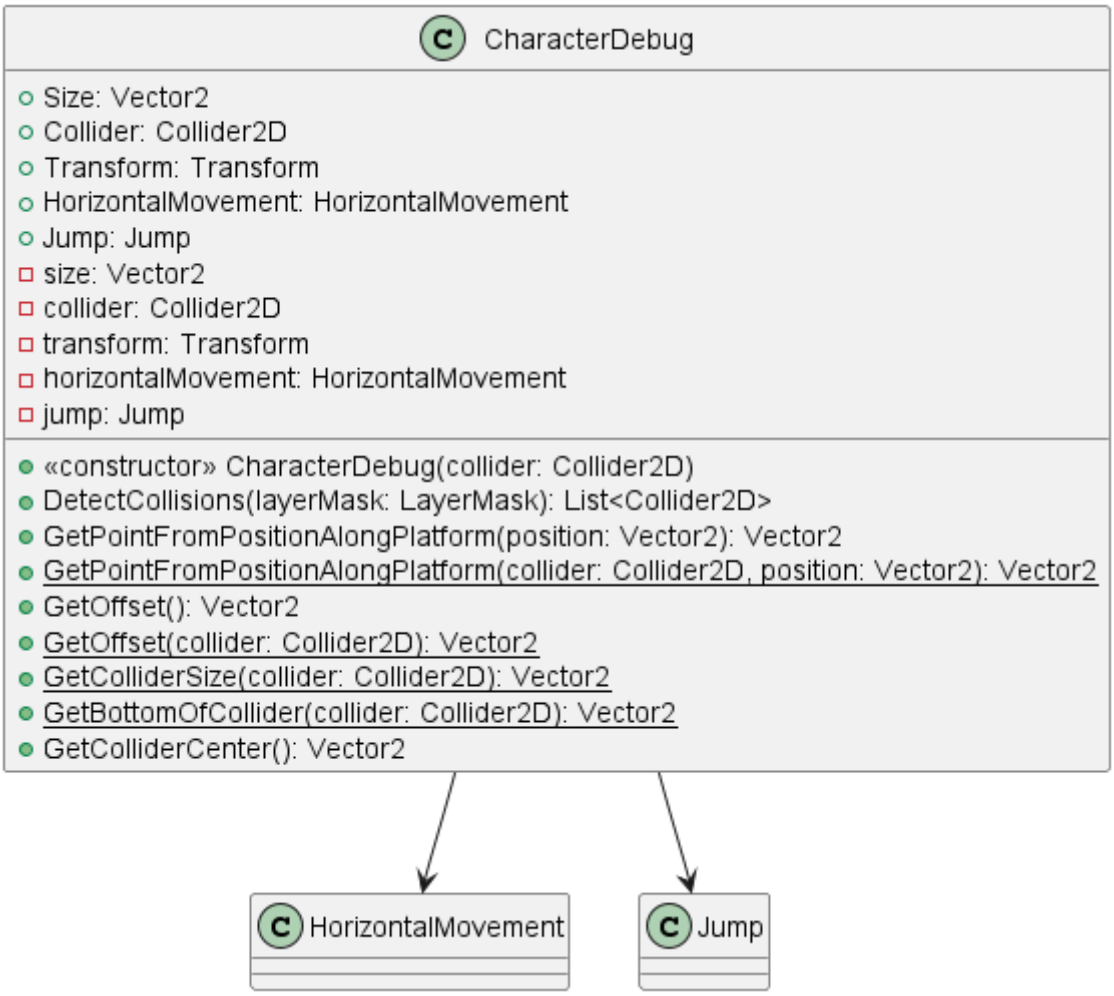
Tabel 3.6 Komponen *Platform Side*

ID	**K-06**
Nama	*Platform Side*
Deskripsi	*Platform Side* adalah komponen yang merepresentasi dari sisi platform dalam gim, yang terdiri dari dua titik, yaitu

	start dan *end*. Komponen ini juga memiliki orientasi, yaitu horizontal atau vertikal. Kelas ini menyediakan berbagai fungsi untuk melakukan operasi terkait sisi platform.
Fungsi Utama	1. *IsAlong* Fungsi ini digunakan untuk memeriksa apakah suatu titik berada sepanjang sisi platform. 2. *GetInnerPoint* Fungsi ini digunakan untuk mendapatkan titik dalam dari suatu titik, berdasarkan karakter dan orientasi sisi. 3. *GetOuterPoint* Fungsi ini digunakan untuk mendapatkan titik luar dari suatu titik, berdasarkan karakter dan orientasi sisi. 4. *SwapStartEndPoint* Fungsi ini digunakan untuk menukar posisi titik awal dan akhir jika diperlukan, berdasarkan orientasi sisi platform.

3.3.3.2.7 *Character Debug*

Gambar 3.18 menunjukkan diagram *CharacterDebug*, yang mencakup atribut, fungsi, serta relasinya dengan kelas-kelas lain. Tabel 3.7 menyajikan deskripsi dan fungsi utama dari *CharacterDebug* secara rinci.



Gambar 3.18 Diagram *character debug*

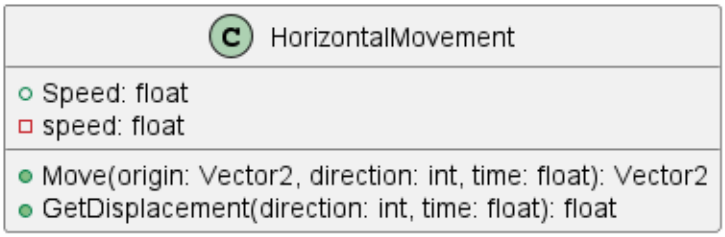
Tabel 3.7 Komponen *Character Debug*

ID	**K-07**
Nama	*Character Debug*
Deskripsi	*Character Debug* adalah komponen yang memberikan metode utilitas dan merepresentasikan karakter dengan kemampuan untuk mendeteksi tumbukan dan melakukan manipulasi terhadap *collider* yang dimilikinya.
Fungsi Utama	1. *DetectCollisions* Fungsi ini digunakan untuk mendeteksi tumbukan dengan kollidernya terhadap lapisan tertentu. Berdasarkan jenis *collider* yang dimiliki karakter, fungsi ini menggunakan metode Physics2D.Overlap untuk mencari tumbukan dengan *collider*nya dan mengembalikan daftar *collider* yang bertabrakan, kecuali *collider* karakter sendiri. 2. *GetPointFromPositionAlongPlatform* Fungsi ini digunakan untuk mendapatkan posisi transform dari karakter berdasarkan posisi platform

	yang diberikan.

3.3.3.2.8 *Horizontal Movement*

Gambar 3.19 menunjukkan diagram *HorizontalMovement*, yang mencakup atribut, fungsi, serta relasinya dengan kelas-kelas lain. Tabel 3.8 menyajikan deskripsi dan fungsi utama dari *HorizontalMovement* secara rinci.



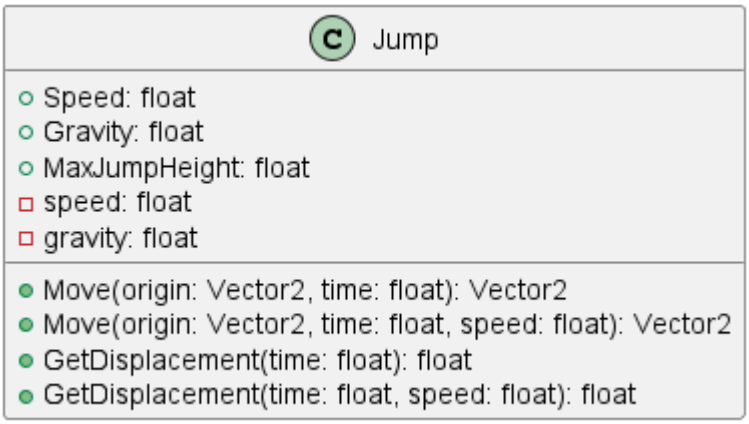
Gambar 3.19 Diagram *horizontal movement*

Tabel 3.8 Komponen *Horizontal Movement*

ID	**K-08**
Nama	*Horizontal Movement*
Deskripsi	*Horizontal Movement* adalah komponen yang merepresentasikan gerakan horizontal. Komponen ini memungkinkan pengaturan kecepatan gerakan horizontal dan menghitung posisi baru berdasarkan arah dan waktu.
Fungsi Utama	1. *GetDisplacement* Fungsi ini digunakan untuk menghitung perpindahan horizontal berdasarkan arah dan waktu. Fungsi menggunakan persamaan gerak satu dimensi untuk mengkalkulasi perpindahan. 2. *Move* Fungsi ini digunakan untuk menggerakkan karakter secara horizontal dari posisi asalnya. Fungsi menghitung dan mengembalikan posisi baru karakter dengan menjumlahkan perpindahan horizontal yang dihitung dari fungsi *GetDisplacement* dengan koordinat x posisi awal.

3.3.3.2.9 *Jump*

Gambar 3.20 menunjukkan diagram *Jump*, yang mencakup atribut, fungsi, serta relasinya dengan kelas-kelas lain. Tabel 3.9 menyajikan deskripsi dan fungsi utama dari *Jump* secara rinci.



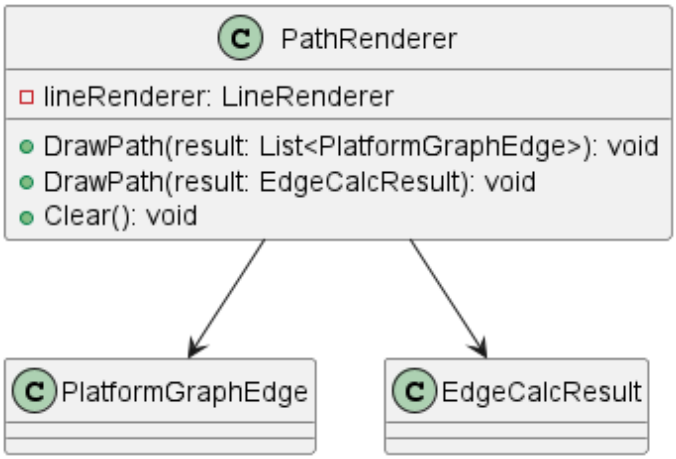
Gambar 3.20 Diagram *jump*

Tabel 3.9 Komponen *Jump*

ID	**K-09**
Nama	*Jump*
Deskripsi	*Jump* adalah komponen yang merepresentasikan gerakan lompatan. Komponen ini memungkinkan pengaturan kecepatan lompatan, gravitasi, dan menghitung posisi baru berdasarkan waktu.
Fungsi Utama	1. *GetDisplacement* Fungsi ini digunakan untuk menghitung perpindahan vertikal berdasarkan waktu dan kecepatan. Fungsi ini memiliki dua *overload*: satu menggunakan kecepatan default, dan yang lainnya menerima kecepatan khusus sebagai parameter tambahan. 2. *Move* Fungsi in digunakan untuk menggerakkan karakter secara vertikal atau melompat dari posisi asalnya. Fungsi ini menghitung dan mengembalikan posisi baru karakter dengan menjumlahkan perpindahan vertikal yang dihitung dari fungsi *GetDisplacement* dengan koordinat y posisi awal.

3.3.3.2.10 *Path Renderer*

Gambar 3.21 menunjukkan diagram *PathRenderer*, yang mencakup atribut, fungsi, serta relasinya dengan kelas-kelas lain. Tabel 3.10 menyajikan deskripsi dan fungsi utama dari *PathRenderer* secara rinci.



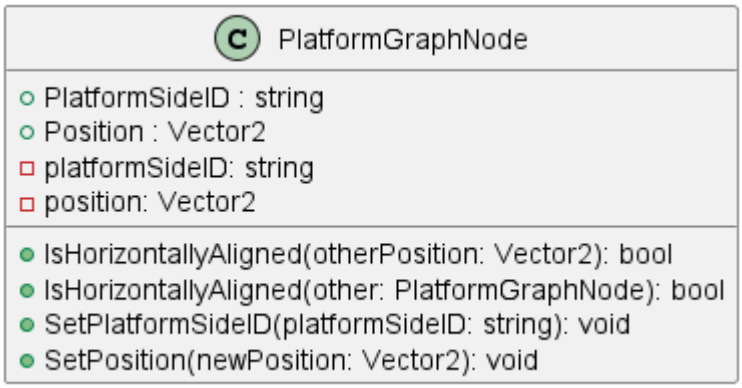
Gambar 3.21 Diagram *path renderer*

Tabel 3.10 Komponen *Path Renderer*

ID	**K-10**
Nama	*Path Renderer*
Deskripsi	*Path Renderer* adalah komponen yang digunakan untuk menggambar jalur berdasarkan hasil perhitungan *edge*. Komponen ini menggunakan komponen *LineRenderer* untuk menggambar jalur di dalam Unity. Ini memungkinkan jalur yang dihitung oleh algoritma navigasi untuk divisualisasikan di layar.
Fungsi Utama	1. *DrawPath* Fungsi ini digunakan untuk menggambar jalur berdasarkan daftar *edge*. Fungsi ini menggunakan *LineRenderer* untuk menggambar jalur.

3.3.3.2.11 *Platform Graph Node*

Gambar 3.22 menunjukkan diagram *PlatformGraphNode*, yang mencakup atribut, fungsi, serta relasinya dengan kelas-kelas lain. Tabel 3.11 menyajikan deskripsi dan fungsi utama dari *PlatformGraphNode* secara rinci.



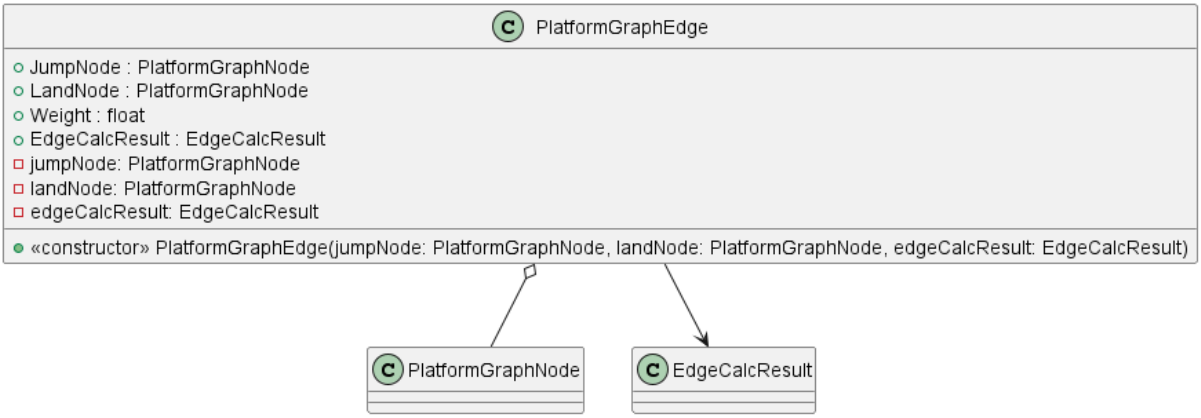
Gambar 3.22 Diagram *platform graph node*

Tabel 3.11 Komponen *Platform Graph Node*

ID	**K-11**
Nama	*Platform Graph Node*
Deskripsi	*Platform Graph Node* adalah komponen yang mewakili *node* di dalam *platform graph*. Komponen ini menyimpan informasi tentang posisi dan ID sisi platform terkait.
Fungsi Utama	1. *IsHorizontallyAligned* Fungsi ini digunakan untuk memeriksa apakah posisi *node* ini sejajar secara horizontal dengan posisi lain yang diberikan.

3.3.3.2.12 *Platform Graph Edge*

Gambar 3.23 menunjukkan diagram *PlatformGraphEdge*, yang mencakup atribut, fungsi, serta relasinya dengan kelas-kelas lain. Tabel 3.12 menyajikan deskripsi dan fungsi utama dari *PlatformGraphEdge* secara rinci.



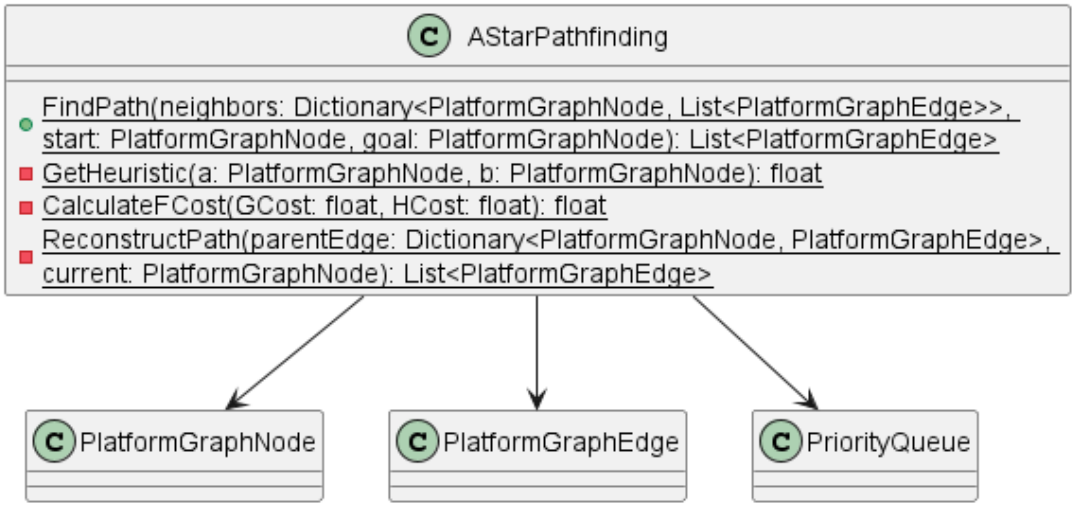
Gambar 3.23 Diagram *platform graph edge*

Tabel 3.12 Komponen *Platform Graph Edge*

ID	**K-12**
Nama	*Platform Graph Edge*
Deskripsi	*Platform Graph Edge* adalah komponen yang merepresentasikan *edge* dalam sebuah *platform graph*. *Edge* ini menghubungkan dua *node* dan menyimpan hasil perhitungan dalam bentuk *EdgeCalcResult*.
Fungsi Utama	-

3.3.3.2.13 *AStar Pathfinding*

Gambar 3.24 menunjukkan diagram *AStarPathfinding*, yang mencakup atribut, fungsi, serta relasinya dengan kelas-kelas lain. Tabel 3.13 menyajikan deskripsi dan fungsi utama dari *AStarPathfinding* secara rinci.



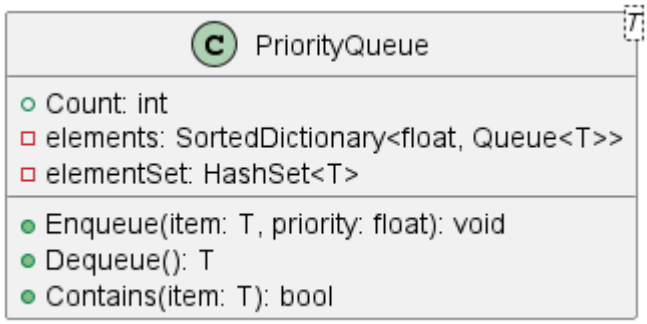
Gambar 3.24 Diagram *AStar Pathfinding*

Tabel 3.13 Komponen *AStar Pathfinding*

ID	**K-13**
Nama	*AStar Pathfinding*
Deskripsi	*AStar Pathfinding* adalah komponen statis yang menyediakan implementasi algoritma pencarian jalur A* untuk navigasi dalam *platform graph*. Algoritma ini digunakan untuk mencari jalur optimal antara dua *node* dalam graf.
Fungsi Utama	1. *FindPath* Fungsi ini digunakan untuk menemukan jalur dari *node start* ke *node goal* menggunakan algoritma A*. Fungsi ini mengembalikan list dari *PlatformGraphEdge* yang merupakan jalur dari *start* ke *goal*. 2. *GetHeuristic* Fungsi ini digunakan untuk menghitung nilai heuristik antara dua simpul berdasarkan jarak Euclidean antara posisi mereka.

3.3.3.2.14 *Priority Queue*

Gambar 3.25 menunjukkan diagram *PriorityQueue*, yang mencakup atribut, fungsi, serta relasinya dengan kelas-kelas lain. Tabel 3.14 menyajikan deskripsi dan fungsi utama dari *PriorityQueue* secara rinci.



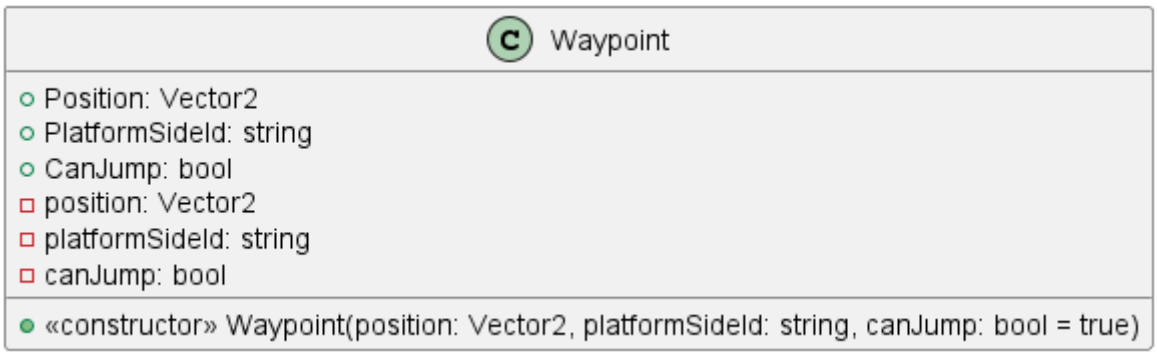
Gambar 3.25 Diagram *Priority Queue*

Tabel 3.14 Komponen *Priority Queue*

ID	**K-14**
Nama	*Priority Queue*
Deskripsi	*Priority Queue* adalah komponen yang mengimplementasikan antrian prioritas yang menggunakan nilai float sebagai prioritas untuk setiap elemen. Digunakan untuk menyimpan elemen-elemen dengan prioritas tertentu dan mengizinkan operasi *enqueue* dan *dequeue*.
Fungsi Utama	1. *Enqueue* Fungsi ini digunakan untuk memasukkan elemen baru ke dalam antrian dengan prioritas tertentu. 2. *Dequeue* Fungsi ini digunakan untuk menghapus dan mengembalikan elemen dengan prioritas tertinggi dari antrian. 3. *Contains* Fungsi ini digunakan untuk memeriksa apakah elemen tertentu ada dalam antrian.

3.3.3.2.15 *Waypoint*

Gambar 3.26 menunjukkan diagram *Waypoint*, yang mencakup atribut, fungsi, serta relasinya dengan kelas-kelas lain. Tabel 3.15 menyajikan deskripsi dan fungsi utama dari *Waypoint* secara rinci.



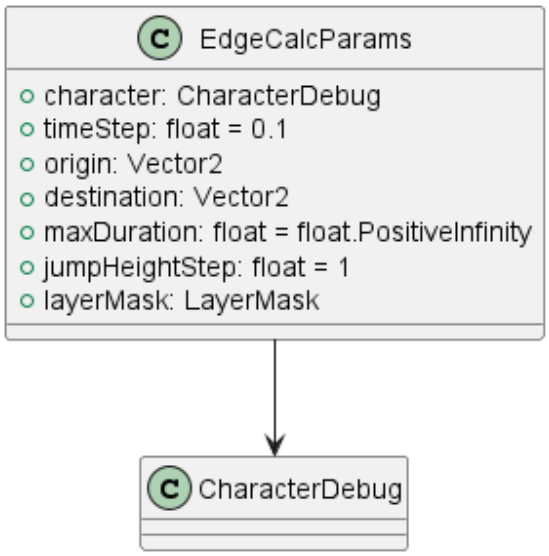
Gambar *3.26* *Diagram waypoint*

Tabel 3.15 Komponen *Waypoint*

ID	**K-15**
Nama	*Waypoint*
Deskripsi	*Waypoint* adalah komponen yang merepresentasikan posisi. Namun, berbeda dengan *node*, *waypoint* tidak harus merepresentasikan posisi melompat atau posisi mendarat.
Fungsi Utama	-

3.3.3.2.16 *Edge Calc Params*

Gambar 3.27 menunjukkan diagram *EdgeCalcParams*, yang mencakup atribut dan relasinya dengan kelas-kelas lain. Tabel 3.16 menyajikan deskripsi dan fungsi utama dari *EdgeCalcParams* secara rinci.



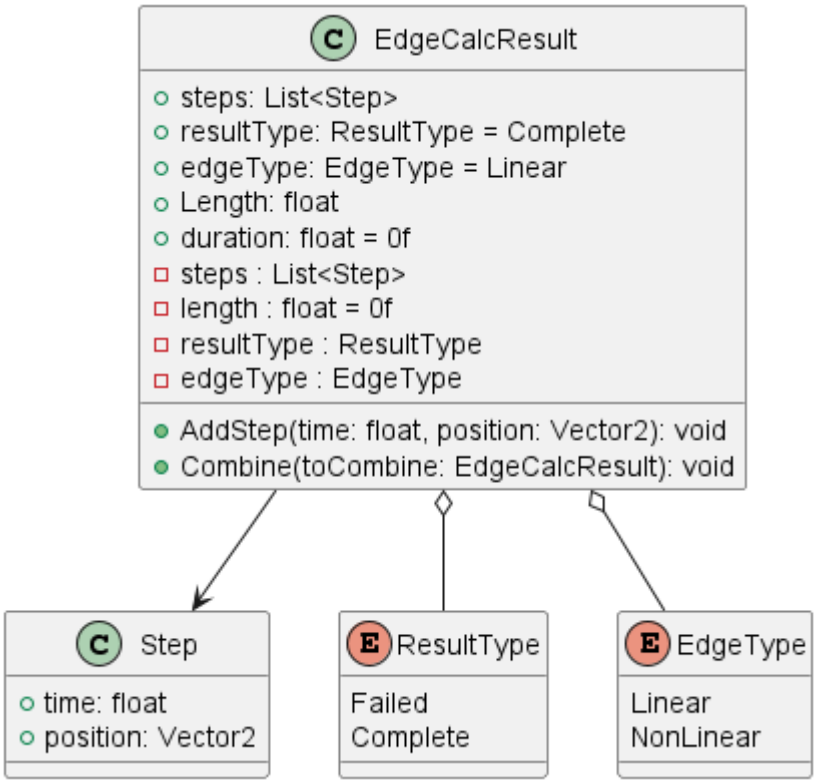
Gambar 3.27 Diagram *edge calc params*

Tabel 3.16 Komponen *Edge Calc Params*

ID	**K-16**
Nama	*Edge Calc Params*
Deskripsi	*Edge Calc Params* adalah komponen yang digunakan untuk input dalam komponen *edge step utility*.
Fungsi Utama	-

3.3.3.2.17 *Edge Calc Result*

Gambar 3.28 menunjukkan diagram *EdgeCalcResult*, yang mencakup atribut, fungsi, serta relasinya dengan kelas-kelas lain. Tabel 3.17 menyajikan deskripsi dan fungsi utama dari *EdgeCalcResult* secara rinci.



Gambar 3.28 Diagram *edge calc result*

Tabel 3.17 Komponen *Edge Calc Result*

ID	**K-17**
Nama	*Edge Calc Result*
Deskripsi	*Edge Calc Result* adalah komponen yang merepresentasikan hasil perhitungan *edge*. Ini mencakup langkah-langkah pergerakan, tipe hasil, tipe *edge*, panjang, dan durasi dari *edge* tersebut.
Fungsi Utama	1. *AddStep* Fungsi ini digunakan untuk menambahkan langkah baru. Di dalam fungsi ini juga menghitung panjang *edge* yang baru. 2. *Combine* Fungsi ini digunakan untuk menggabungkan *EdgeCalcResult*.

3.3.3.3 Fitur Utama

Pada subbab ini akan dijelaskan secara rinci mengenai fitur utama dari *package* ini, yang meliputi pembuatan *platform graph* dan navigasi antar platform.

3.3.3.3.1 Pembuatan *Platform Graph*

Fitur ini memungkinkan pembuatan sebuah graph yang mewakili semua platform dalam peta permainan. *Platform graph* digunakan untuk menentukan rute navigasi bagi karakter. *Package* ini dapat mendeteksi semua platform yang tersedia di peta dan menghubungkannya dalam bentuk graph. Setiap simpul mewakili titik lompat atau titik mendarat pada platform, sementara

tepi mengbubungkan simpul-simpul tersebut berdasarkan kemungkinan jalur yang dapat diambil karakter.

3.3.3.3.2 Navigasi Platform

Fitur navigasi platform menyediakan mekanisme bagi karakter untuk berpindah dari satu platform ke platform lainnya dengan memanfaatkan *platform graph* yang telah dibuat. *Package* ini mengatur pergerakan karakter baik secara horizontal maupun vertikal, termasuk lompatan dan pendaratan. Algoritma navigasi mempertimbangkan berbagai parameter seperti kecepatan gerak, gravitasi, dan tinggi lompatan.

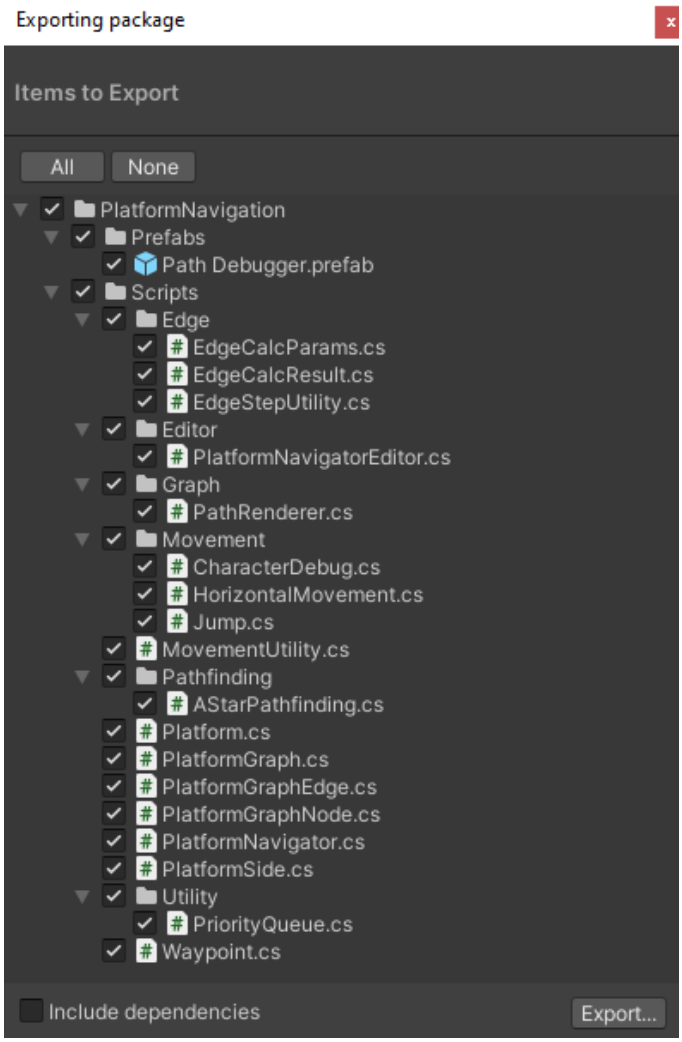
BAB IV
HASIL DAN PEMBAHASAN

4.1 Hasil

Pada subbab ini akan dijelaskan tentang hasil yang telah didapatkan dari implementasi dan pengembangan *package pathfinding* untuk gim *platformer*.

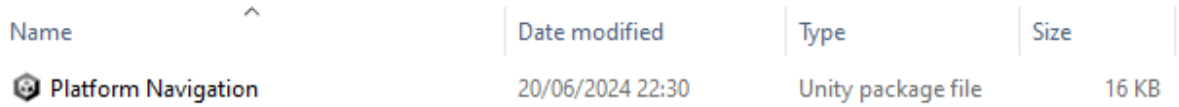
4.1.1 *Package*

Package di Unity dapat diekspor dengan mengklik kanan pada mouse, lalu memilih opsi "*Export Package*." Setelah itu, kita dapat memilih komponen apa saja yang akan disertakan dalam *package* tersebut. Gambar 4.1 adalah contoh tampilan saat mengekspor *package* di Unity.



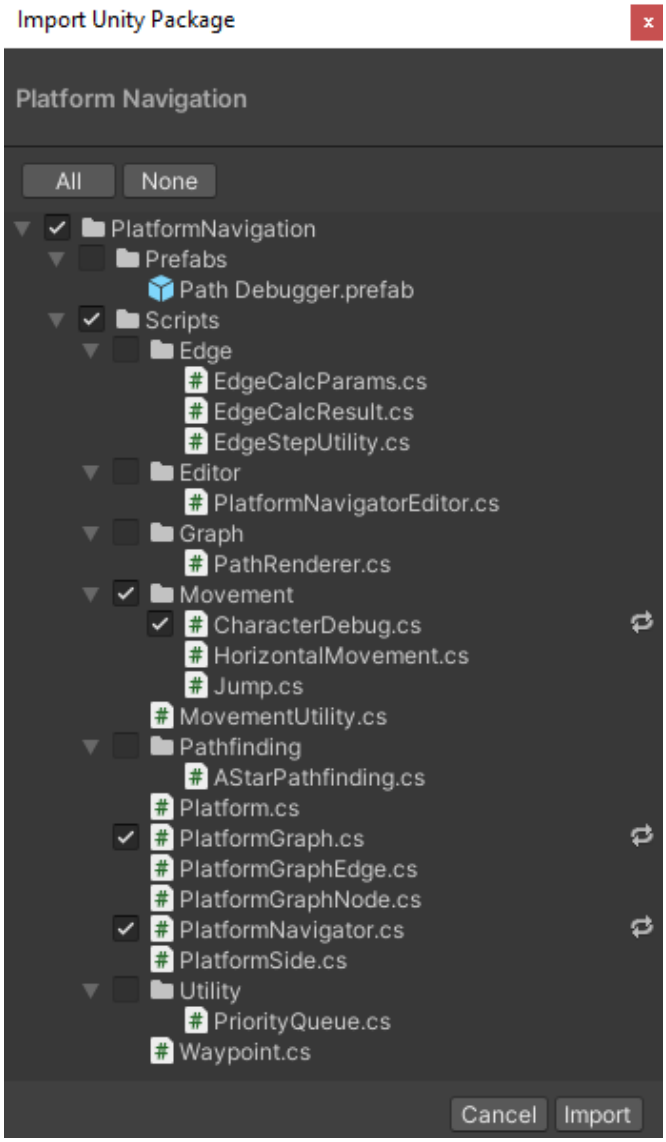
Gambar 4.1 Tampilan ketika mengekspor *package*

Package yang dihasilkan berupa file yang dapat digunakan dalam proyek mana pun. File ini berisi semua komponen yang diperlukan untuk mengintegrasikan *package* ke dalam proyek lain, termasuk skrip, aset, dan konfigurasi yang telah dipilih saat proses ekspor. Gambar 4.2 adalah contoh file *package* yang dihasilkan.



Gambar 4.2 *Package platform navigation*

Gambar 4.3 adalah tampilan saat mengimpor *package* ke dalam proyek. *Package* dapat diimpor dengan menarik file ke dalam Unity.



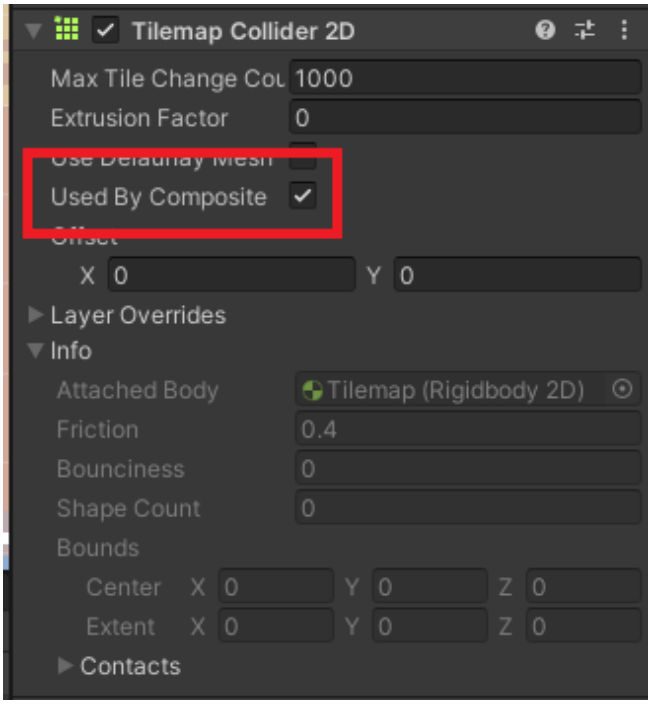
Gambar 4.3 Tampilan ketika mengimpor *package*

4.1.2 Panduan Penggunaan *Package*

Pada subbab ini akan dijelaskan cara menggunakan *package* ini setelah diimport ke dalam proyek.

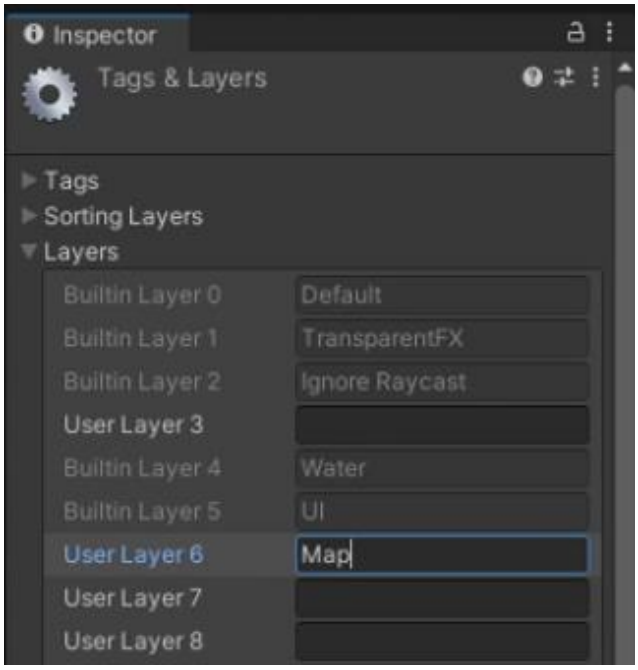
4.1.2.1 Mengatur *GameObject* Peta

Tambahkan *CompositeCollider2D* pada *GameObject* peta, lalu centang opsi *Used By Composite* pada *collider* yang lain. Misalnya, jika menggunakan tilemap, centang opsi tersebut pada *TilemapCollider2D*. Gambar 4.4 adalah contoh tampilan *TilemapCollider2D*.



Gambar 4.4 Tampilan tilemap *collider* 2d

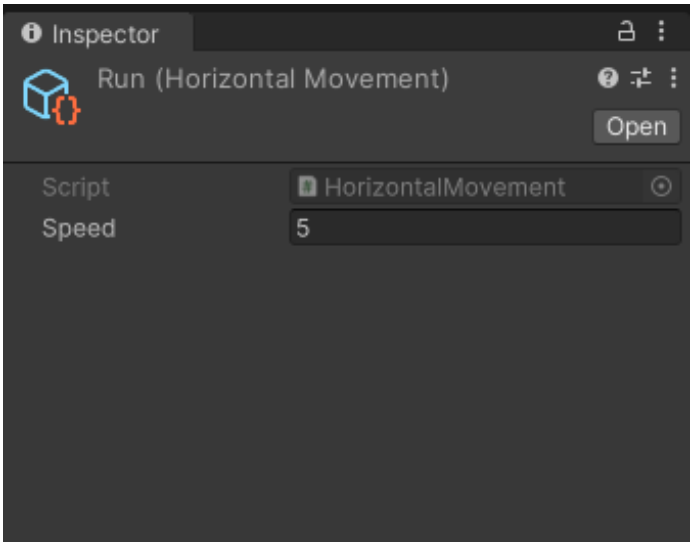
CompositeCollider2D digunakan untuk menggabungkan *collider-collider* lain yang ada pada *GameObject* tersebut. Setelah itu, pastikan *Geometry Type* pada *CompositeCollider2D* diubah menjadi *Polygons*. Hal ini penting agar *collider* dapat mendeteksi interior dari *outline* composite-nya, yang diperlukan untuk mengecek letak sisi platform. Kemudian buat layer baru dan atur layer tersebut pada *GameObject*. Gambar 4.5 adalah contoh tampilan saat mengatur layer untuk peta.



Gambar 4.5 Tampilan saat mengatur layer peta

4.1.2.2 Membuat *Scriptable Object Horizontal Movement*

Untuk membuat *scriptable object horizontal movement*, klik kanan pada Project window, lalu pilih Create -> *PlatformNavigation* -> Horizontal. Setelah itu, atur parameter speed di inspektor dengan nilai yang diinginkan. Gambar 4.6 adalah contoh tampilan *scriptable object* untuk pergerakan horizontal.



Gambar 4.6 Tampilan *scriptable object horizontal movement*

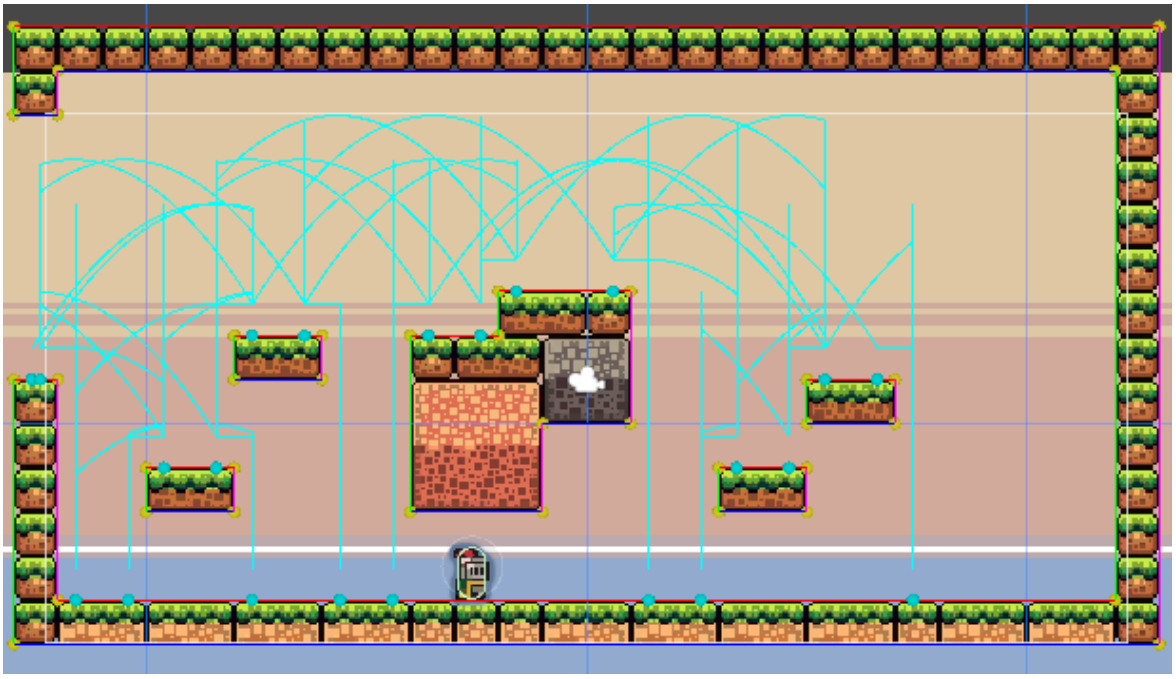
4.1.2.3 Membuat *Scriptable Object Jump*

Untuk membuat *scriptable object jump*, klik kanan pada *project window*, lalu pilih Create -> *PlatformNavigation -> Jump*. Setelah itu, atur parameter speed dan gravity di inspektor dengan nilai yang diinginkan. Gambar 4.7 adalah contoh tampilan *scriptable object* untuk lompat.

Gambar 4.7 Tampilan *scriptable object jump*

4.1.2.4 Tambahkan Komponen *PlatformNavigator* Pada *GameObject*

Tambahkan komponen *PlatformNavigator* pada *GameObject* yang ingin dilengkapi dengan kemampuan navigasi antar platform. Setelah itu, atur *layer mask*, *horizontal movement*, *jump*, dan *collider* peta. Kemudian, tekan tombol *Bake* dan pilih tempat untuk menyimpan hasil *platform graph*.



Gambar 4.8 Contoh *platform graph*

Gambar 4.8 menunjukkan *platform graph* yang telah dibentuk dari platform yang disediakan. Graf ini merepresentasikan berbagai *node* dan *edge* yang digunakan dalam algoritma *pathfinding* untuk menentukan jalur antara titik awal dan titik tujuan. Graf ini disimpan dalam sebuah *scriptable object*, yang berfungsi sebagai wadah data agar informasi ini dapat dihitung sebelumnya dan digunakan saat permainan dimulai.

Graf ini hanya menyimpan pergerakan antar platform, karena pergerakan dalam satu platform tidak memerlukan deteksi tumbukan saat dikalkulasi, sehingga tidak perlu dimasukkan ke dalam *scriptable object*. Pergerakan dalam satu platform dapat dihitung pada saat permainan dimulai, sehingga mengoptimalkan penggunaan sumber daya dan mempercepat waktu muat permainan.

4.1.2.5 Menggunakan Fungsi Publik *PlatformNavigator*

Untuk menggunakan *PlatformNavigator*, perlu membuat skrip khusus untuk memanggil fungsi-fungsi publik dari *PlatformNavigator*. Salah satu fungsi yang dapat dipanggil adalah fungsi *MoveTo*. Fungsi ini menggerakkan karakter ke koordinat posisi yang dituju dengan memperhatikan platform dan parameter yang diberikan. Contoh implementasi skrip khusus ini dapat ditemukan pada prototipe.

4.2 Pembahasan

Tujuan dari pengujian ini adalah untuk mengevaluasi performa dan keandalan *package pathfinding* yang telah dikembangkan. Oleh karena itu, akan diadakan tiga jenis pengujian untuk menguji *package* ini, yaitu uji coba waktu eksekusi pembuatan *platform graph*, uji coba fungsionalitas, dan uji coba prototipe.

4.2.1 Uji Coba Waktu Eksekusi

Uji coba ini akan dilakukan pada peta yang memiliki ukuran berbeda-beda. Tujuan dari uji coba ini adalah untuk mengetahui performa pembuatan *platform graph*. Uji coba ini akan mengukur waktu yang dibutuhkan untuk membuat *platform graph*. Hasil dari uji coba ini akan menentukan apakah pembuatan *platform graph* layak dilakukan saat *runtime* atau sebaiknya dilakukan sebelum *runtime*.

4.2.1.1 Lingkungan Pengujian

Lingkungan yang digunakan dalam uji coba ini memiliki spesifikasi perangkat keras dan perangkat lunak yang ditunjukkan pada Tabel 4.1.

Tabel 4.1 Lingkungan Uji Coba

Perangkat	Jenis Perangkat	Spesifikasi
Perangkat Keras	Prosesor	Intel Core i3 10105F CPU @ 3.70 GHz
	Memori	8 Gb
Perangkat Lunak	Sistem Operasi	Windows 10
	Perangkat Pengujian	Visual Studio Code
	Game Engine	Unity 2022.3.27f1

4.2.1.2 Hasil Uji Coba

Uji coba ini memanfaatkan kelas *Stopwatch* dari namespace *System.Diagnostics* untuk mengukur waktu eksekusi fungsi *Bake*. Fungsi *Bake* digunakan untuk membuat *Platform Graph*. *Stopwatch* diinisialisasi dan dimulai di awal fungsi. Setelah fungsi menyelesaikan semua tugasnya, *Stopwatch* dihentikan dan waktu eksekusi ditampilkan pada log konsol. Contoh implementasinya dapat dilihat pada Kode Program 4.1.

```
public bool Bake(CharacterDebug character, LayerMask layerMask,
CompositeCollider2D mapCollider) {
    System.Diagnostics.Stopwatch stopwatch = new();
    stopwatch.Start();
```

```
    List<Platform> platforms = GetMapPlatform(mapCollider);
    if (platforms == null) return false;
    Dictionary<Vector2, PlatformGraphNode> nodeLookup =
ReplacePlatforms(platforms);

    foreach(PlatformSide jumpSide in Sides) {
        foreach(PlatformSide landSide in Sides) {
            foreach((Vector2, Vector2, EdgeCalcResult) result in
MovementUtility.CalculateJumpLandPosition(character, layerMask, jumpSide,
landSide)) {
                if (!nodeLookup.ContainsKey(result.Item1)) {
                    nodeLookup[result.Item1] = CreateNode(result.Item1,
jumpSide);
                }
                if (!nodeLookup.ContainsKey(result.Item2)) {
                    nodeLookup[result.Item2] = CreateNode(result.Item2,
landSide);
                }

                PlatformGraphEdge edge = new(nodeLookup[result.Item1],
nodeLookup[result.Item2], result.Item3);
                edges.Add(edge);
            }
        }
    }

    stopwatch.Stop();
    Debug.Log("Waktu eksekusi fungsi Bake: " +
stopwatch.ElapsedMilliseconds + " ms");

    return true;
}
```

Kode Program 4.1 Pengujian waktu eksekusi

Uji coba dilakukan pada peta dengan berbagai ukuran yang berbeda, di mana ukuran peta diukur dalam satuan Unity unit. Hasil dari uji coba ini dapat dilihat pada Tabel 4.2.

Tabel 4.2 Hasil Uji Coba Waktu Eksekusi

Ukuran Peta	Waktu Eksekusi
12 x 10	173 ms
24 x 20	988 ms
36 x 30	2135 ms
48 x 40	7250 ms
60 x 50	17375 ms
72 x 60	37025 ms

4.2.2 Uji Coba Fungsionalitas

Uji coba ini akan dilakukan pada platform yang dapat dicapai, platform yang tidak dapat dicapai, dan platform dengan jalur terbaik. Tabel 4.3 menunjukkan parameter karakter yang digunakan untuk uji coba ini.

Tabel 4.3 Parameter Karakter

Parameter	**Nilai**
Kecepatan horizontal	5
Kecepatan lompat	8
Gravitasi	-9.8

4.2.2.1 Platform yang Dapat Dicapai

Algoritma yang digunakan untuk mencari jalur dalam *package* ini adalah algoritma A*. Algoritma A* dianggap optimal apabila heuristik yang digunakan memenuhi kondisi *admissible*. Sebuah heuristik dikatakan *admissible* jika tidak pernah melebih-lebihkan biaya sebenarnya untuk mencapai tujuan.

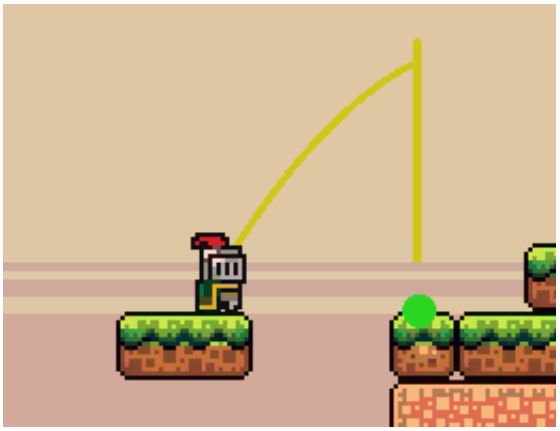
$$h(n) \leq h^*(n) \ (4.1)$$

dimana:
$h(n)$ adalah heuristik dari *node* n ke tujuan
$h^*(n)$ adalah biaya sebenarnya dari *node* n ke tujuan

Dalam konteks ini, biaya sebenarnya yang digunakan adalah panjang lintasan antar *node*, sedangkan heuristik menggunakan jarak garis lurus antar *node*. Jarak garis lurus antara dua titik selalu merupakan jalur terpendek yang mungkin antara titik-titik tersebut. Oleh karena itu, heuristik yang dipilih memenuhi kondisi *admissible*, sehingga algoritma A* akan selalu mengembalikan jalur yang optimal.

4.2.2.1.1 Uji Coba 1

Gambar 4.9 menunjukkan skenario uji coba platform yang berhasil dicapai pada percobaan pertama. Tabel 4.4 menyajikan pembahasan yang mencakup hasil uji coba tersebut.



Gambar 4.9 Skenario uji coba platform yang dapat dicapai 1

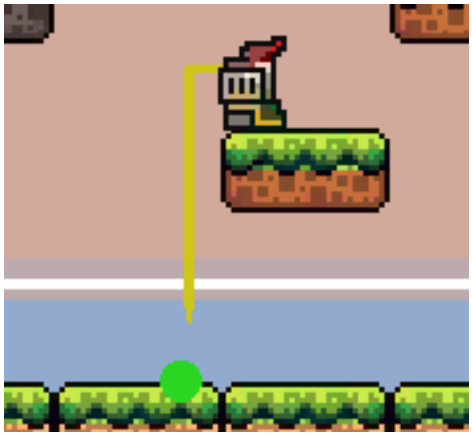
Tabel 4.4 Uji Coba Platform yang Dapat Dicapai 1

ID	**UJ-A01**
Hasil	Berhasil

4.2.2.1.2 Uji Coba 2

Gambar 4.10 menunjukkan skenario uji coba platform yang berhasil dicapai pada percobaan

kedua. Tabel 4.5 menyajikan pembahasan yang mencakup hasil uji coba tersebut.



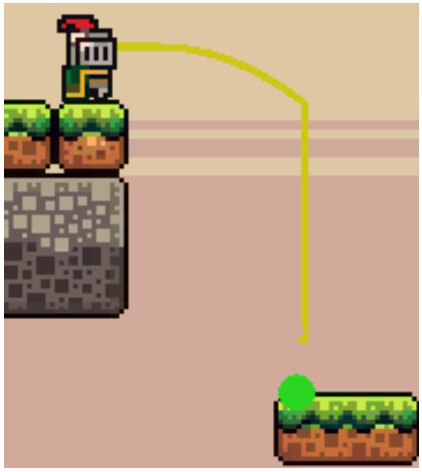
Gambar 4.10 Skenario uji coba platform yang dapat dicapai 2

Tabel 4.5 Uji Coba Platform yang Dapat Dicapai 2

ID	**UJ-A02**
Hasil	Berhasil

4.2.2.1.3 Uji Coba 3

Gambar 4.11 menunjukkan skenario uji coba platform yang berhasil dicapai pada percobaan ketiga. Tabel 4.6 menyajikan pembahasan yang mencakup hasil uji coba tersebut.



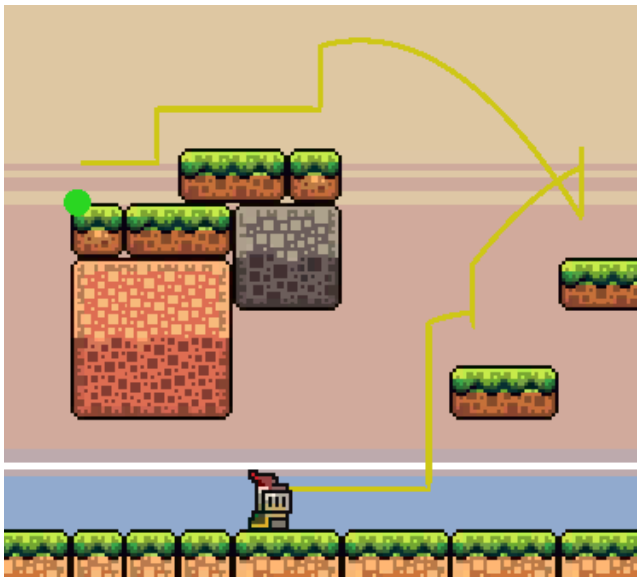
Gambar 4.11 Skenario uji coba platform yang dapat dicapai 3

Tabel 4.6 Uji Coba Platform yang Dapat Dicapai 3

ID	**UJ-A03**
Hasil	Berhasil

4.2.2.1.4 Uji Coba 4

Gambar 4.12 menunjukkan skenario uji coba platform yang berhasil dicapai pada percobaan keempat. Tabel 4.7 menyajikan pembahasan yang mencakup hasil uji coba tersebut.



Gambar 4.12 Skenario uji coba platform yang dapat dicapai 4

Tabel 4.7 Uji Coba Platform yang Dapat Dicapai 4

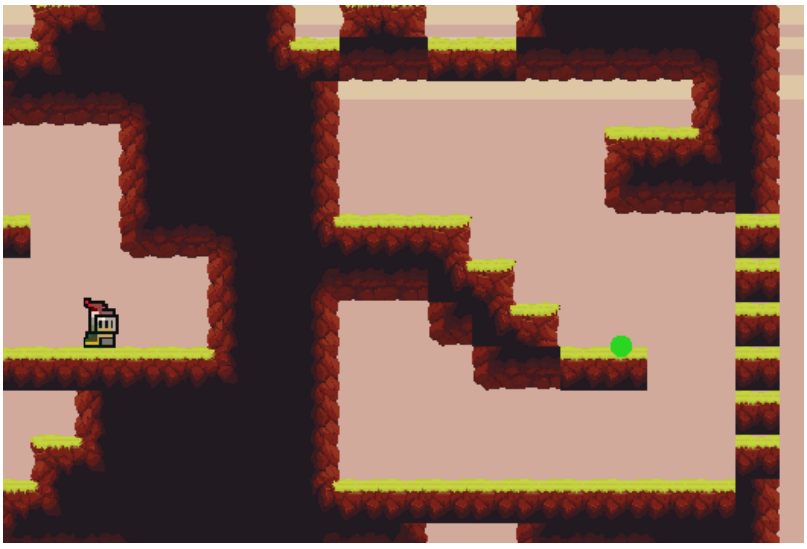
ID	**UJ-A04**
Hasil	Berhasil

4.2.2.2 Platform yang Tidak Dapat Dicapai

Pada subbab ini akan dibahas mengenai uji coba terhadap platform yang tidak dapat dicapai, termasuk penjelasan alasan mengapa platform tersebut tidak dapat dicapai.

4.2.2.2.1 Uji Coba 1

Gambar 4.13 menunjukkan skenario uji coba platform yang tidak dapat dicapai pada percobaan pertama. Tabel 4.8 menyajikan pembahasan yang mencakup hasil uji coba dan alasan mengapa uji coba tersebut tidak berhasil.



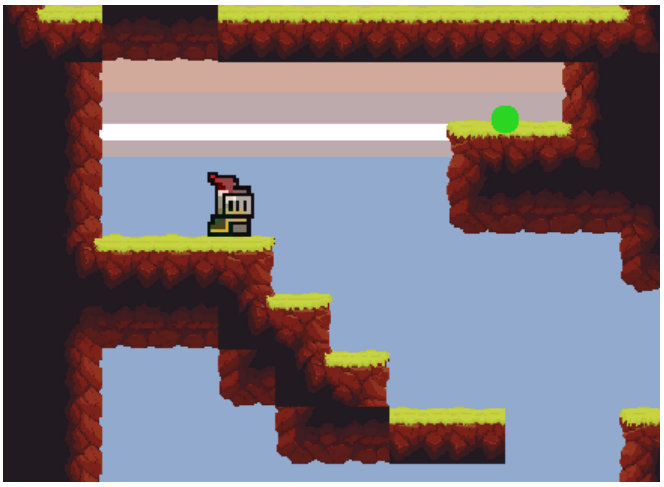
Gambar 4.13 Skenario uji coba platform yang tidak dapat dicapai 1

Tabel 4.8 Uji Coba Platform yang Tidak Dapat Dicapai 1

ID	**UJ-B01**
Hasil	Tidak berhasil
Pembahasan	Uji coba ini tidak berhasil karena posisi destinasi terletak pada ruang tertutup.

4.2.2.2.2 Uji Coba 2

Gambar 4.14 menunjukkan skenario uji coba platform yang tidak dapat dicapai pada percobaan kedua. Tabel 4.9 menyajikan pembahasan yang mencakup hasil uji coba dan alasan mengapa uji coba tersebut tidak berhasil.



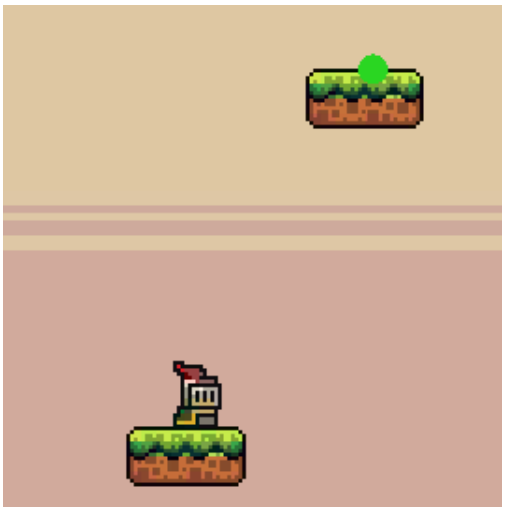
Gambar 4.14 Skenario uji coba platform yang tidak dapat dicapai 2

Tabel 4.9 Uji Coba Platform yang Tidak Dapat Dicapai 2

ID	**UJ-B02**
Hasil	Tidak berhasil
Pembahasan	Uji coba ini tidak berhasil karena karakter tidak muat pada posisi destinasi.

4.2.2.2.3 Uji Coba 3

Gambar 4.15 menunjukkan skenario uji coba platform yang tidak dapat dicapai pada percobaan ketiga. Tabel 4.10 menyajikan pembahasan yang mencakup hasil uji coba dan alasan mengapa uji coba tersebut tidak berhasil.



Gambar 4.15 Skenario uji coba platform yang tidak dapat dicapai 3

Tabel 4.10 Uji Coba Platform yang Tidak Dapat Dicapai 3

ID	**UJ-B03**
Hasil	Tidak berhasil
Pembahasan	Uji coba ini tidak berhasil karena karakter tidak memiliki kecepatan lompat yang cukup untuk mencapai posisi destinasi.

4.2.2.2.4 Uji Coba 4

Gambar 4.16 menunjukkan skenario uji coba platform yang tidak dapat dicapai pada percobaan keempat. Tabel 4.11 menyajikan pembahasan yang mencakup hasil uji coba dan alasan mengapa uji coba tersebut tidak berhasil.

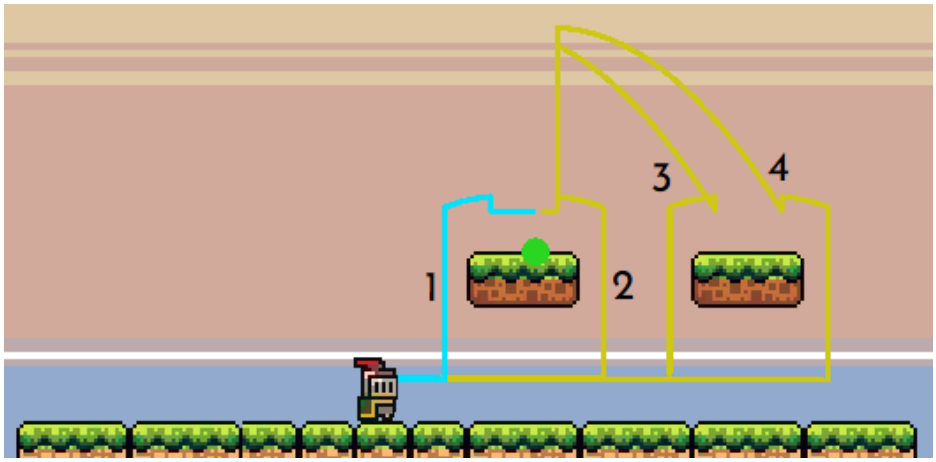
Gambar 4.16 Skenario uji coba platform yang tidak dapat dicapai 4

Tabel 4.11 Uji Coba Platform yang Tidak Dapat Dicapai 4

ID	**UJ-B04**
Hasil	Tidak berhasil
Pembahasan	Uji coba ini tidak berhasil karena karakter tidak memiliki kecepatan horizontal yang cukup untuk mencapai posisi destinasi.

4.2.2.3 Platform dengan Jalur Terbaik

Gambar 4.17 menunjukkan skenario uji coba platform dengan jalur terbaik. Lingkaran berwarna hijau menandakan posisi destinasi dari pergerakan. Terdapat 1.539 jalur yang dapat ditempuh untuk mencapai posisi destinasi. Dari jalur-jalur tersebut, diambil empat sampel untuk uji coba, masing-masing diberi nomor dari 1 hingga 4 berdasarkan urutan panjang jalur. Jalur yang ditandai dengan warna biru merupakan jalur terbaik. Tabel 4.12 menyajikan deskripsi uji coba yang mencakup hasil, jumlah jalur, dan panjang setiap jalur.



Gambar 4.17 Skenario uji coba platform dengan jalur terbaik

Tabel 4.12 Uji Coba Platform dengan Jalur Terbaik

ID	**UJ-C01**
Hasil	Berhasil
Jumlah jalur	1539
Panjang jalur	1. 6.080682 2. 8.380652 3. 17.17953 4. 21.10904

4.2.3 Uji Coba Prototipe

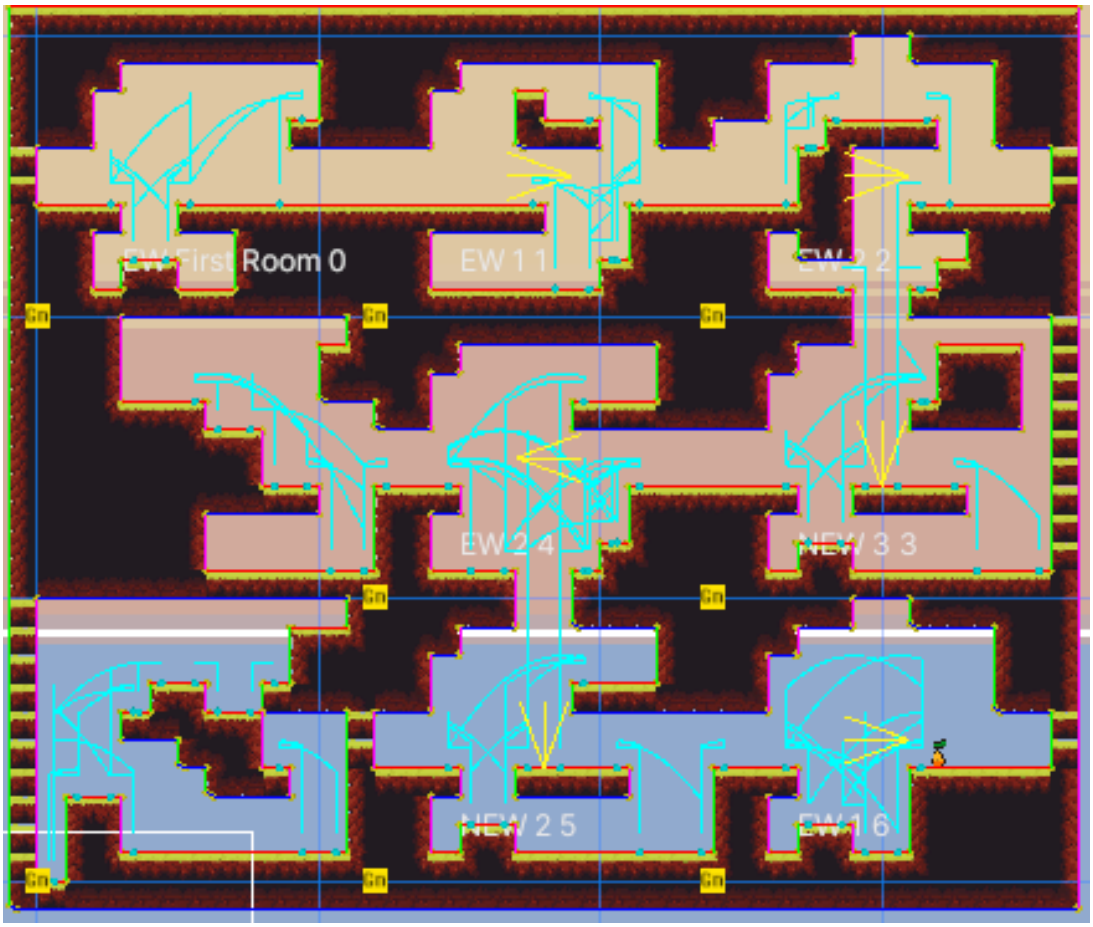
Uji coba ini akan dilakukan dalam dua buah prototipe gim *platformer*. Pengujian ini bertujuan untuk memastikan bahwa *package* dapat mengatasi skenario pergerakan karakter pada kedua prototipe.

Prototipe pertama adalah prototipe yang dikembangkan sendiri dengan *package* yang sudah diintegrasikan sejak awal. Prototipe kedua adalah prototipe yang telah jadi dan dikembangkan oleh pihak lain tanpa *package* yang sudah diintegrasikan sejak awal, namun *package* diintegrasikan setelah prototipe selesai dibuat. Tujuan dari pengujian ini adalah untuk menunjukkan bahwa *package* ini dapat diintegrasikan ke dalam gim yang dikembangkan sendiri maupun oleh pihak lain, sehingga menunjukkan fleksibilitas dan generalisasi dari *package* tersebut.

4.2.3.1 Prototipe 1

Prototipe ini adalah prototipe yang dikembangkan sendiri. Prototipe ini memiliki mekanik di mana pemain harus menekan area yang ingin dituju. Posisi klik tersebut kemudian di*raycast*

terhadap platform. Setelah itu, pemain dapat menemukan jalur dari posisi awal menuju ke posisi yang telah diklik tersebut. Peta dalam prototipe ini dirandomisasi setiap kali prototipe dijalankan, dan *platform graph*-nya dibuat ulang setiap kali. Tujuan dari permainan ini adalah agar pemain menyentuh buah nanas untuk menang. Gambar 4.18 adalah tampilan prototipe pertama.



Gambar 4.18 Tampilan prototipe 1

Integrasi *package* pada prototipe ini melibatkan pencarian jalur dari posisi pemain ke titik yang diklik. Kode Program 4.2 adalah contoh kode program untuk mengintegrasikan prototipe dengan *package*.

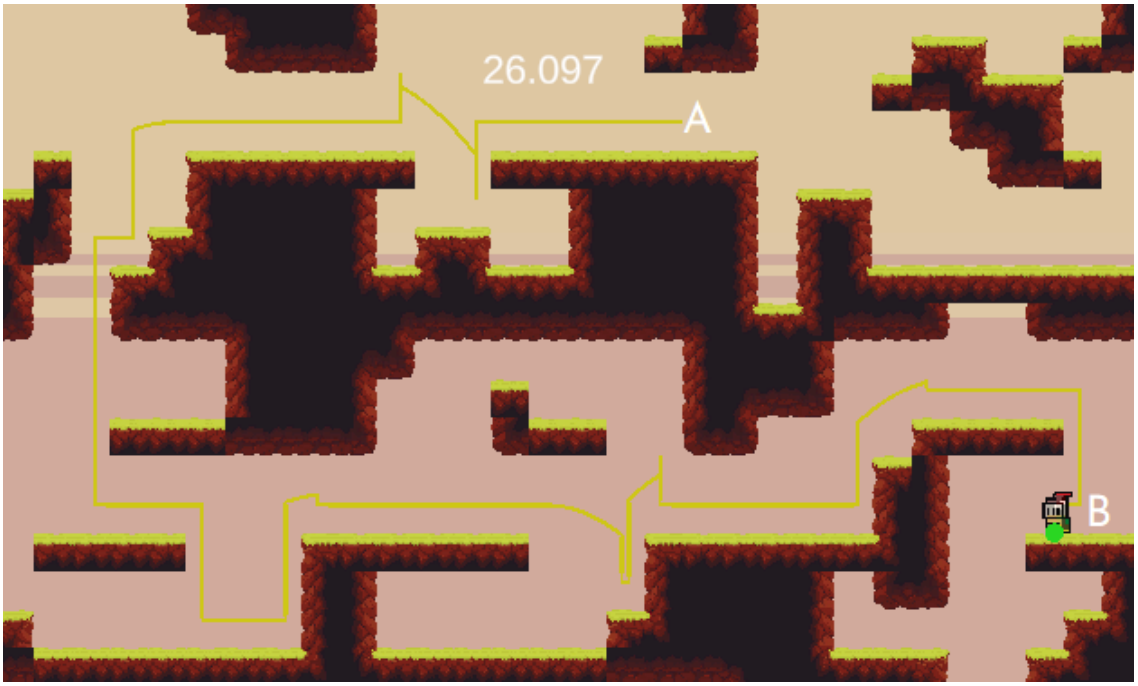
```
public class PlayerController : MonoBehaviour {
    [SerializeField] GameObject clickedPositionMarkerPrefab = null;

    GameObject clickedPositionMarkerInstance = null;
    Vector2 raycastedClickPosition;

    PlatformNavigator platformNavigator;

    void Awake() {
        platformNavigator = GetComponent<PlatformNavigator>();
    }

    void Update() {
        if (Mouse.current.leftButton.wasPressedThisFrame) {
```

```
            Vector2 worldPosition = GetMousePosition();
            MoveTo(worldPosition);
        }
    }

    void MoveTo(Vector2 clickedPosition) {
        RaycastHit2D? hit =
platformNavigator.RaycastToPlatform(clickedPosition);
        if (hit == null) return;
        raycastedClickPosition = hit.Value.point;

        if (clickedPositionMarkerPrefab == null) return;

        if (clickedPositionMarkerInstance == null) {
            clickedPositionMarkerInstance =
Instantiate(clickedPositionMarkerPrefab);
        }
        clickedPositionMarkerInstance.transform.position =
raycastedClickPosition;

        platformNavigator.MoveTo(raycastedClickPosition);
    }

    Vector3 GetMousePosition() {
        return
Camera.main.ScreenToWorldPoint(Mouse.current.position.ReadValue());
    }
}
```

Kode Program 4.2 Integrasi dengan prototipe 1

Kode program ini berfungsi untuk mengendalikan karakter pemain dalam gim. Kode ini menggerakkan karakter menuju posisi yang diklik oleh mouse, dengan bantuan *PlatformNavigator*, yang merupakan bagian dari *package* yang dikembangkan, untuk melakukan *raycast* dan navigasi. Integrasi dengan *package* dilakukan melalui *PlatformNavigator*, yang mengatur pergerakan karakter berdasarkan posisi klik yang valid.

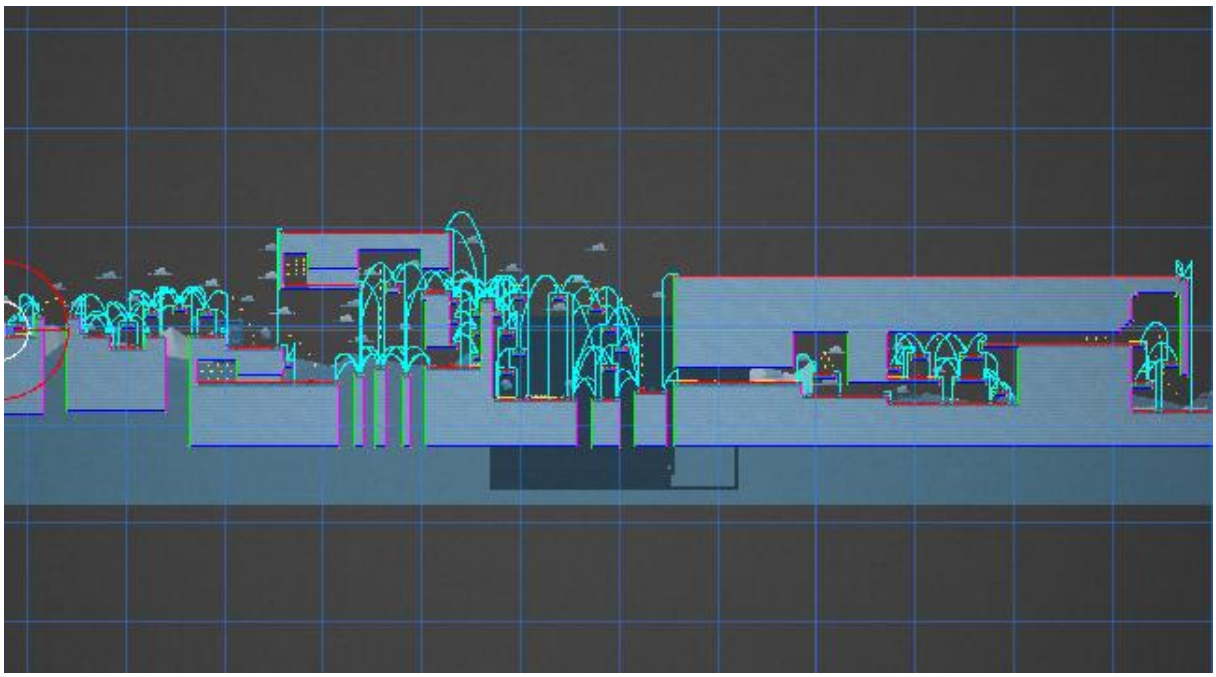
PlatformNavigator juga dapat menunjukkan jalur yang akan diambil dari posisi awal menuju posisi tujuan. Gambar 4.19 adalah contoh jalur yang dibentuk.



Gambar 4.19 Jalur dari posisi awal menuju posisi destinasi

4.2.3.2 Prototipe 2

Prototipe ini dikembangkan oleh pihak resmi Unity. Pada prototipe ini, pemain mengendalikan karakter menggunakan *keyboard* untuk melewati rintangan dan musuh menuju tujuan akhir peta. Musuh dalam prototipe dapat mengejar pemain jika pemain masuk ke dalam radius kejar musuh tersebut. Gambar 4.20 adalah tampilan prototipe kedua.



Gambar 4.20 Tampilan prototipe 2

Integrasi *package* pada prototipe ini dilakukan dengan menyisipkan fungsionalitas pada setiap musuh dalam peta sehingga mereka dapat bergerak antar platform untuk mengejar pemain. Kode Program 4.3 adalah contoh cuplikan kode program untuk mengintegrasikan prototipe

dengan *package*.

```
…
void Update() {
    switch (currentState)
    {
        case State.Patrol:
            Patrol();
            break;
        case State.Chase:
            ChasePlayer();
            break;
        case State.ReturnToOriginalPosition:
            ReturnToOriginalPosition();
            break;
    }

    CheckPlayerDistance();
}

void Patrol() {
    if (!IsGrounded()) return;

    animationController.enabled = true;
    enemyController.enabled = true;
}

void ChasePlayer() {
    bool isGrounded = IsGrounded();
    animator.SetFloat("57elocity", Mathf.Abs(1));
    animator.SetBool("grounded", isGrounded);
    if (platformNavigator.IsFollowingPath) return;
    if (!isGrounded) return;

    animationController.enabled = false;
    enemyController.enabled = false;
    GetComponent<Rigidbody2D>().isKinematic = true;

    RaycastHit2D? hit =
platformNavigator.RaycastToPlatform(player.position);
    if (hit == null) return;

    platformNavigator.MoveTo(hit.Value.point);
}

void ReturnToOriginalPosition() {
    if (platformNavigator.IsFollowingPath) return;
    if (!IsGrounded()) return;

    RaycastHit2D? hit =
platformNavigator.RaycastToPlatform(teritoryCenter);
    if (hit == null) return;

    platformNavigator.MoveTo(hit.Value.point);
```

```
    if (Vector2.Distance(transform.position, teritoryCenter) < 0.1f) {
        currentState = State.Patrol;
    }
}

void CheckPlayerDistance() {
    float distanceToPlayer = Vector2.Distance(teritoryCenter,
player.position);

    if (currentState == State.Chase) {
        if (distanceToPlayer > teritoryRadius) {
            currentState = State.ReturnToOriginalPosition;
        }
    } else {
        if (distanceToPlayer < chaseRadius) {
            currentState = State.Chase;
        }
    }
}

bool IsGrounded() {
    return Physics2D.Raycast(transform.position, Vector2.down,
groundCheckDistance, groundLayer);
}
...
```

Kode Program 4.3 Integrasi dengan prototipe 2

Cuplikan kode program ini mengendalikan perilaku musuh dalam prototipe. Setiap musuh memiliki tiga keadaan (state), yaitu state patroli, state kejar, dan state kembali ke posisi awal. Ketika dalam state patroli, musuh akan mengikuti jalur yang telah ditetapkan oleh desainer sebelumnya, yang merupakan mekanik dari pihak resmi Unity. Jika pemain masuk ke dalam radius pengejaran, musuh akan masuk ke dalam state kejar dan mengejar pemain. Saat berada dalam state kejar, musuh akan mencari jalur dari posisi musuh ke posisi pemain, yang merupakan implementasi tambahan. Jika pemain keluar dari radius teritori, musuh akan masuk ke dalam state kembali ke posisi awal dan kemudian kembali ke state patroli setelah sampai di posisi awal. Gambar 4.21 adalah contoh tampilan saat musuh mengejar pemain.

Gambar 4.21 Tampilan saat musuh mengejar pemain

4.3 Evaluasi Hasil Uji Coba

Pada bagian sebelumnya, telah dilakukan pengujian terhadap *package* yang telah dikembangkan. Pengujian ini mencakup uji coba waktu eksekusi pembuatan *platform graph*, uji coba fungsionalitas, dan uji coba prototipe. Kedua prototipe dalam pengujian prototipe berhasil mengintegrasikan kemampuan navigasi pada karakter dalam gim, menunjukkan bahwa *package* tersebut siap digunakan untuk pengembangan gim.

Sementara itu, hasil uji coba waktu eksekusi menunjukkan bahwa waktu eksekusi pembuatan *platform graph* relatif cepat untuk peta dengan ukuran kecil dan lebih lambat untuk peta dengan ukuran besar. Hal ini berarti pembuatan *platform graph* dapat dilakukan saat *runtime* untuk peta berukuran kecil, namun sebaiknya dilakukan di luar *runtime* untuk peta berukuran besar.

(Halaman ini sengaja dikosongkan)

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Dalam Tugas Akhir ini, telah dilakukan pengembangan dan pengujian *package* kecerdasan buatan *pathfinding* yang dapat digunakan untuk pergerakan antar platform. Kesimpulan dari pengembangan dan pengujian *package* ini adalah sebagai berikut.

1. *Package* yang dikembangkan berhasil menyediakan solusi navigasi untuk gim *platformer*. *Package* ini menggunakan *platform graph* untuk memetakan platform dan menggunakan persamaan kinematika untuk pergerakan karakter.
2. Hasil pengujian menunjukkan bahwa waktu eksekusi untuk pembuatan *platform graph* bervariasi tergantung pada ukuran peta. Untuk peta kecil, waktu eksekusi cepat, sehingga pembuatan *platform graph* dapat dilakukan saat *runtime*. Namun, untuk peta besar, waktu eksekusi lebih lama, sehingga sebaiknya dilakukan sebelum *runtime*.
3. Pengujian menggunakan dua prototipe gim *platformer* menunjukkan bahwa *package* ini efektif dan siap digunakan dalam pengembangan gim.

5.2 Saran

Berdasarkan hasil pengembangan dan pengujian yang telah dilakukan, berikut adalah beberapa saran untuk pengembangan lebih lanjut dan penerapan *package* ini.

1. Memperbanyak titik lompat dan titik mendarat agar pergerakan menjadi lebih natural.
2. Menambahkan berbagai jenis pergerakan selain melompat, seperti *wall jump* atau berlari di sepanjang dinding untuk meningkatkan variasi permainan.
3. Mengembangkan kemampuan *package* untuk dapat melakukan navigasi terhadap platform yang dinamis.

(Halaman ini sengaja dikosongkan)

DAFTAR PUSTAKA

Aramini, A. U., Lanzi, P. L., & Loiacono, D. (2018). *An Integrated Framework for AI Assisted Level Design in 2D Platformers*. https://www.researchgate.net/publication/324745087

Handy Permana, S. D., Bayu, K., Bintoro, Y., Arifitama, B., & Syahputra, A. (2018). Comparative Analysis of *Pathfinding* Algorithms A *, Dijkstra, and BFS on Maze Runner Game. *International Journal Of Information System & Technology*, *1*(2), 1–8.

Lawande, S. R., Jasmine, G., Anbarasi, J., & Izhar, L. I. (2022). A Systematic Review and Analysis of Intelligence-Based *Pathfinding* Algorithms in the Field of Video Games. *Applied Sciences (Switzerland)*, *12*(11). https://doi.org/10.3390/app12115499

Lee, W., & Lawrence, R. (2013). Fast grid-based path finding for video games. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 7884 LNAI*. https://doi.org/10.1007/978-3-642-38457-8_9

Russell, S., & Norvig, P. (2021). Artificial Intelligence A Modern Approach (4th Edition). In *Pearson Series*.

Tremblay, J., Borodovski, A., & Verbrugge, C. (2021). *I Can Jump! Exploring Search Algorithms for Simulating Platformer Players*. www.aaai.org

(Halaman ini sengaja dikosongkan)

LAMPIRAN-LAMPIRAN ATAU APPENDIKS

PlatformNavigator

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.IO;
using Unity.VisualScripting;

public class PlatformNavigator : MonoBehaviour {
    [SerializeField] LayerMask layerMask;
    public LayerMask LayerMask => layerMask;
    [SerializeField] HorizontalMovement horizontalMovement;
    public HorizontalMovement HorizontalMovement => horizontalMovement;
    [SerializeField] Jump jump;
    public Jump Jump => jump;
    [SerializeField] bool showPath = false;
    [SerializeField] PathRenderer pathDebuggerPrefab;

    [Header("Editor")]
    [SerializeField] CompositeCollider2D mapCollider;
    public CompositeCollider2D MapCollider => mapCollider;
    [SerializeField] PlatformGraph platformGraph;
    public PlatformGraph PlatformGraph { get { return platformGraph; }
set { platformGraph = value; } }

    bool isFollowingPath = false;
    public bool IsFollowingPath => isFollowingPath;
    public bool IsMoving => Velocity != Vector2.zero;
    public bool IsJumping => Velocity.y > 0f;
    public bool IsFalling => Velocity.y < 0f;
    int direction = 1;
    public int Direction => direction;

    Vector2 velocity = Vector2.zero;
    public Vector2 Velocity => velocity;

    PathRenderer pathRenderer;
    public PathRenderer PathRenderer => pathRenderer;

    Coroutine followPath = null;

    void Awake() {
        pathRenderer = Instantiate(pathDebuggerPrefab);
    }

    public bool RecalculateGraph() {
        if (!TryGetComponent<Collider2D>(out var collider)) return false;
        PlatformGraph.Clear();
        CharacterDebug character = new(collider) {
            transform = collider.transform,
            horizontalMovement = horizontalMovement,
            jump = jump
```

```
        };
        if(!PlatformGraph.Bake(character, layerMask, MapCollider)) return
false;
        PlatformGraph.Initialize(character, layerMask);

        return true;
    }

    public void MoveTo(Vector2 endPosition) {
        CharacterDebug character =
new(transform.GetComponent<Collider2D>()) {
            transform = transform,
            horizontalMovement = horizontalMovement,
            jump = jump
        };

        Vector2 startPosition = (Vector2)transform.position -
character.GetOffset();

        List<PlatformGraphEdge> path = platformGraph.FindPath(character,
layerMask, startPosition, endPosition);
        if (showPath) pathRenderer.DrawPath(path);

        if (followPath != null) StopCoroutine(followPath);
        followPath = StartCoroutine(FollowPath(path));
    }

    public RaycastHit2D? RaycastToPlatform(Vector2 position) {
        if (mapCollider.OverlapPoint(position)) return null;

        RaycastHit2D hit = Physics2D.Raycast(position, Vector2.down,
float.PositiveInfinity, layerMask);
        if (hit.collider == null) return null;
        return hit;
    }

    public void Stop() {
        if (followPath == null) return;

        StopCoroutine(followPath);
        velocity = Vector2.zero;
        isFollowingPath = false;
        followPath = null;
    }

    IEnumerator FollowPath(List<PlatformGraphEdge> path) {
        if (path == null) {
            isFollowingPath = false;
            yield break;
        }

        isFollowingPath = true;
        foreach(PlatformGraphEdge edge in path) {
            foreach(var step in edge.EdgeCalcResult.Steps) {
                float threshold = 0.001f;
```

```
                Vector2 positionDifference = step.position -
(Vector2)transform.position;

                if (Mathf.Abs(positionDifference.x) < threshold) {
                    positionDifference.x = 0f;
                }

                if (Mathf.Abs(positionDifference.y) < threshold) {
                    positionDifference.y = 0f;
                }

                velocity = (step.position -
(Vector2)transform.position)/Time.fixedDeltaTime;
                if (velocity.x > 0) direction = 1;
                else if (velocity.x < 0) direction = -1;

                transform.position = step.position;

                yield return new WaitForFixedUpdate();
            }
        }

        isFollowingPath = false;
        velocity = Vector2.zero;
    }

    void OnDrawGizmosSelected() {
        if (platformGraph == null) return;

        foreach (Platform platform in platformGraph.Platforms) {
            foreach (Vector2 point in platform.Points) {
                Gizmos.color = Color.yellow;
                Gizmos.DrawSphere(point, 0.15f);
            }
            foreach (PlatformSide side in platform.Sides) {
                if (side.Direction == PlatformSide.SideDirection.Top)
Gizmos.color = Color.red;
                else if (side.Direction ==
PlatformSide.SideDirection.Bottom) Gizmos.color = Color.blue;
                else if (side.Direction ==
PlatformSide.SideDirection.Left) Gizmos.color = Color.green;
                else Gizmos.color = Color.magenta;
                Gizmos.DrawLine(side.Start, side.End);
            }
        }

        foreach (PlatformGraphEdge edge in platformGraph.Edges) {
            Gizmos.color = Color.cyan;
            Gizmos.DrawSphere(edge.JumpNode.Position, 0.15f);
            Gizmos.DrawSphere(edge.LandNode.Position, 0.15f);

            if (edge.EdgeCalcResult != null) {
                for (int i = 0; i < edge.EdgeCalcResult.Steps.Count - 1;
i++) {
```

```
Gizmos.DrawLine(edge.EdgeCalcResult.Steps[i].position,
edge.EdgeCalcResult.Steps[i+1].position);
                }
            }
        }
    }
}
```

Platform Graph

```
using System.Collections.Generic;
using System.Linq;
using UnityEditor;
using UnityEngine;

public class PlatformGraph : ScriptableObject,
ISerializationCallbackReceiver {
    [SerializeField] List<Platform> platforms = new();
    public List<Platform> Platforms => platforms;
    [SerializeField] List<PlatformGraphNode> nodes = new();
    public IEnumerable<PlatformGraphNode> Nodes => nodes;
    [SerializeField] List<PlatformGraphEdge> edges = new();
    public IEnumerable<PlatformGraphEdge> Edges => edges;
    public IEnumerable<PlatformSide> Sides { get {
        foreach(Platform platform in platforms) {
            foreach(PlatformSide side in platform.Sides) yield return
side;
        }
    }}

    Dictionary<Vector2, PlatformGraphNode> nodeLookup = new();
    Dictionary<PlatformGraphNode, List<PlatformGraphEdge>> nodeNeighbors
= new();

    void OnEnable() {
#if UNITY_EDITOR
        if (!EditorApplication.isPlayingOrWillChangePlaymode) return;
#endif
        Debug.Log("OnEnable");
    }

    public void Clear() {
        platforms.Clear();
        DeleteAllNodes();
        edges.Clear();
#if UNITY_EDITOR
        EditorUtility.SetDirty(this);
        AssetDatabase.SaveAssets();
        AssetDatabase.Refresh();
#endif
    }

    public void Initialize(CharacterDebug character, LayerMask layerMask)
{
```

```
        if (nodeLookup.Count != 0 && nodeNeighbors.Count != 0) return;

        BuildNodeLookup();
        nodeNeighbors = GetNodeNeighbors(character, layerMask);
    }

    public List<PlatformGraphEdge> FindPath(CharacterDebug character,
LayerMask layerMask, Vector2 start, Vector2 goal) {
        if (nodeLookup.Count == 0 && nodeNeighbors.Count == 0) {
            Debug.Log("Initialize Platform Graph");
            Initialize(character, layerMask);
        }

        PlatformGraphNode startNode =
CreateInstance<PlatformGraphNode>();
        startNode.SetPosition(start);

startNode.SetPlatformSideID(GetPlatformSideIDFromPosition(start));

        PlatformGraphNode goalNode = CreateInstance<PlatformGraphNode>();
        goalNode.SetPosition(goal);
        goalNode.SetPlatformSideID(GetPlatformSideIDFromPosition(goal));

        var appendedNodeNeighbor = AppendStartGoalNode(character,
layerMask, nodeNeighbors, startNode, goalNode);

        List<PlatformGraphEdge> path = new();
        if (startNode.PlatformSideID == goalNode.PlatformSideID) {
            PlatformGraphEdge edge = GetIntraPlatformGraphEdge(character,
layerMask, startNode, goalNode);
            path.Add(edge);
        }
        else {
            path = AStarPathfinding.FindPath(appendedNodeNeighbor,
startNode, goalNode);
            if (path == null) return null;
        }

        return path;
    }

    public List<List<PlatformGraphEdge>> FindAllPath(CharacterDebug
character, LayerMask layerMask, Vector2 start, Vector2 goal) {
        if (nodeLookup.Count == 0 && nodeNeighbors.Count == 0) {
            Debug.Log("Initialize Platform Graph");
            Initialize(character, layerMask);
        }

        PlatformGraphNode startNode =
CreateInstance<PlatformGraphNode>();
        startNode.SetPosition(start);

startNode.SetPlatformSideID(GetPlatformSideIDFromPosition(start));

        PlatformGraphNode goalNode = CreateInstance<PlatformGraphNode>();
        goalNode.SetPosition(goal);
```

```
        goalNode.SetPlatformSideID(GetPlatformSideIDFromPosition(goal));

        var appendedNodeNeighbor = AppendStartGoalNode(character,
layerMask, nodeNeighbors, startNode, goalNode);

        List<List<PlatformGraphEdge>> allPath = new();
        allPath = PathfindingDebugger.FindAllPaths(appendedNodeNeighbor,
startNode, goalNode);
        if (allPath == null) return null;

        foreach(var path in allPath) Debug.Log(path.Sum(e => e.Weight));

        return allPath;
    }

    public Dictionary<PlatformGraphNode, List<PlatformGraphEdge>>
AppendStartGoalNode(CharacterDebug character, LayerMask layerMask,
Dictionary<PlatformGraphNode, List<PlatformGraphEdge>> nodeNeighbor,
PlatformGraphNode start, PlatformGraphNode goal) {
        Dictionary<PlatformGraphNode, List<PlatformGraphEdge>>
newNeighbor = new();
        foreach (var kvp in nodeNeighbor) {
            newNeighbor.Add(kvp.Key, new
List<PlatformGraphEdge>(kvp.Value));
        }

        foreach(PlatformGraphNode current in Nodes) {
            if (current.PlatformSideID == start.PlatformSideID) {
                PlatformGraphEdge edge =
GetIntraPlatformGraphEdge(character, layerMask, start, current);
                if (!newNeighbor.ContainsKey(edge.JumpNode))
newNeighbor[edge.JumpNode] = new();
                newNeighbor[edge.JumpNode].Add(edge);
                Debug.Log("start");
            }
            if (current.PlatformSideID == goal.PlatformSideID) {
                PlatformGraphEdge edge =
GetIntraPlatformGraphEdge(character, layerMask, current, goal);
                if (!newNeighbor.ContainsKey(edge.JumpNode))
newNeighbor[edge.JumpNode] = new();
                newNeighbor[edge.JumpNode].Add(edge);
                Debug.Log("goal");
            }
        }
        return newNeighbor;
    }

    void BuildNodeLookup() {
        nodeLookup.Clear();
        foreach(PlatformGraphNode node in Nodes) {
            nodeLookup[node.Position] = node;
        }
    }

    Dictionary<PlatformGraphNode, List<PlatformGraphEdge>>
GetNodeNeighbors(CharacterDebug character, LayerMask layerMask) {
```

```
        Dictionary<PlatformGraphNode, List<PlatformGraphEdge>>
nodeNeighbors = new();

        foreach(PlatformGraphEdge edge in Edges) {
            if (!nodeNeighbors.ContainsKey(edge.JumpNode))
nodeNeighbors[edge.JumpNode] = new();
            if (!nodeNeighbors.ContainsKey(edge.LandNode))
nodeNeighbors[edge.LandNode] = new();

            nodeNeighbors[edge.JumpNode].Add(edge);
        }

        foreach(PlatformGraphEdge edge in
GetIntraPlatformSideEdges(character, layerMask, nodeNeighbors.Keys)) {
            if (!nodeNeighbors.ContainsKey(edge.JumpNode))
nodeNeighbors[edge.JumpNode] = new();
            if (!nodeNeighbors.ContainsKey(edge.LandNode))
nodeNeighbors[edge.LandNode] = new();

            nodeNeighbors[edge.JumpNode].Add(edge);
        }

        return nodeNeighbors;
    }

    IEnumerable<PlatformGraphEdge>
GetIntraPlatformSideEdges(CharacterDebug character, LayerMask layerMask,
IEnumerable<PlatformGraphNode> nodes) {
        foreach(PlatformGraphNode start in nodes) {
            foreach(PlatformGraphNode end in nodes) {
                PlatformGraphEdge edge =
GetIntraPlatformGraphEdge(character, layerMask, start, end);
                if (edge == null) continue;

                yield return edge;
            }
        }
    }

    PlatformGraphEdge GetIntraPlatformGraphEdge(CharacterDebug character,
LayerMask layerMask, PlatformGraphNode start, PlatformGraphNode end) {
        if (string.IsNullOrEmpty(start.PlatformSideID)) return null;
        if (string.IsNullOrEmpty(end.PlatformSideID)) return null;
        if (start.PlatformSideID != end.PlatformSideID) return null;

        float maxDuration = Mathf.Abs(end.Position.x - start.Position.x)
/ character.HorizontalMovement.Speed;

        Vector2 fixedStart =
character.GetPointFromPositionAlongPlatform(start.Position);

        Vector2 fixedEnd =
character.GetPointFromPositionAlongPlatform(end.Position);
```

```
        EdgeCalcResult result =
EdgeStepUtility.CalculateHorizontalStepsBetweenWaypoint(new
EdgeCalcParams {
            character = character,
            origin = fixedStart,
            destination = fixedEnd,
            maxDuration = maxDuration,
            timeStep = Time.fixedDeltaTime,
            layerMask = layerMask
        });
        if (result == null) return null;

        return new(start, end, result);
    }

    PlatformGraphNode GetClosestNode(IEnumerable<PlatformGraphNode>
nodes, Vector2 position, string platformSideID = "") {
        float minDistance = float.PositiveInfinity;
        PlatformGraphNode closestNode = null;

        foreach (PlatformGraphNode node in nodes) {
            if (!string.IsNullOrEmpty(platformSideID)) {
                if (node.PlatformSideID != platformSideID) continue;
            }
            if (!node.IsHorizontallyAligned(position)) continue;

            float distance = Vector2.Distance(node.Position, position);
            if (distance < minDistance) {
                minDistance = distance;
                closestNode = node;
            }
        }

        return closestNode;
    }

    string GetPlatformSideIDFromPosition(Vector2 position) {
        foreach(Platform platform in platforms) {
            foreach(PlatformSide platformSide in platform.Sides) {
                if (platformSide.IsAlong(position)) return
platformSide.ID;
            }
        }
        return "";
    }

#if UNITY_EDITOR
    public bool Bake(CharacterDebug character, LayerMask layerMask,
CompositeCollider2D mapCollider) {
        System.Diagnostics.Stopwatch stopwatch = new();
        stopwatch.Start();

        this.nodeLookup = new();
        nodeNeighbors = new();

        List<Platform> platforms = GetMapPlatform(mapCollider);
```

```
        if (platforms == null) return false;
        Dictionary<Vector2, PlatformGraphNode> nodeLookup =
ReplacePlatforms(platforms);

        foreach(PlatformSide jumpSide in Sides) {
            foreach(PlatformSide landSide in Sides) {
                foreach((Vector2, Vector2, EdgeCalcResult) result in
MovementUtility.CalculateJumpLandPosition(character, layerMask, jumpSide,
landSide)) {
                    if (!nodeLookup.ContainsKey(result.Item1)) {
                        nodeLookup[result.Item1] =
CreateNode(result.Item1, jumpSide);
                    }
                    if (!nodeLookup.ContainsKey(result.Item2)) {
                        nodeLookup[result.Item2] =
CreateNode(result.Item2, landSide);
                    }

                    PlatformGraphEdge edge =
new(nodeLookup[result.Item1], nodeLookup[result.Item2], result.Item3);
                    edges.Add(edge);
                }
            }
        }

        stopwatch.Stop();
        Debug.Log("Waktu eksekusi fungsi Bake: " +
stopwatch.ElapsedMilliseconds + " ms");

        return true;
    }

    public void AddPlatform(Platform platform) {
        platforms.Add(platform);

        EditorUtility.SetDirty(this);
        AssetDatabase.SaveAssets();
        AssetDatabase.Refresh();
    }

    public Dictionary<Vector2, PlatformGraphNode>
ReplacePlatforms(List<Platform> platforms) {
        DeleteAllNodes();
        this.platforms = platforms;

        Dictionary<Vector2, PlatformGraphNode> nodeLookup = new();
        foreach(Platform platform in platforms) {
            foreach(Vector2 point in platform.Points) {
                nodeLookup[point] = CreateNode(point);
            }
        }

        EditorUtility.SetDirty(this);
        AssetDatabase.SaveAssets();
        AssetDatabase.Refresh();
```

```
        return nodeLookup;
    }

    List<Platform> GetMapPlatform(CompositeCollider2D mapCollider) {
        List<Platform> platforms = new();

        for (int i = 0; i < mapCollider.pathCount; i++) {
            Vector2[] points = new
Vector2[mapCollider.GetPathPointCount(i)];
            mapCollider.GetPath(i, points);
            for (int j = 0; j < points.Count(); j++) {
                points[j] += (Vector2)mapCollider.transform.position;
            }
            Platform platform = new(mapCollider, points.ToList());
            platforms.Add(platform);
        }

        return platforms;
    }

    PlatformGraphNode CreateNode(Vector2 position, PlatformSide
platformSide) {
        PlatformGraphNode node = CreateNode(position);
        node.SetPlatformSideID(platformSide.ID);
        EditorUtility.SetDirty(this);
        return node;
    }

    PlatformGraphNode CreateNode(Vector2 position) {
        PlatformGraphNode newNode = CreateInstance<PlatformGraphNode>();
        newNode.SetPosition(position);
        nodes.Add(newNode);
        EditorUtility.SetDirty(this);
        return newNode;
    }

    void DeleteNode(PlatformGraphNode nodeToDelete) {
        nodes.Remove(nodeToDelete);
        if (nodeToDelete == null) return;

        AssetDatabase.RemoveObjectFromAsset(nodeToDelete);
        Undo.DestroyObjectImmediate(nodeToDelete);
        EditorUtility.SetDirty(this);
    }

    void DeleteAllNodes() {
        if (nodes.Count <= 0) return;

        List<PlatformGraphNode> nodesToDelete = new(nodes);
        foreach (PlatformGraphNode node in nodesToDelete) {
            DeleteNode(node);
        }
        nodes = new();
    }
#endif
```

```
    public void OnBeforeSerialize() {
#if UNITY_EDITOR
        if(AssetDatabase.GetAssetPath(this) == "") return;

        foreach(PlatformGraphNode node in Nodes) {
            if(node == null) continue;
            if(AssetDatabase.GetAssetPath(node) != "") continue;

            AssetDatabase.AddObjectToAsset(node, this);
        }
#endif
    }

    public void OnAfterDeserialize() {}
}
```

PlatformGraphEdge

```
using System;
using UnityEngine;

[Serializable]
public class PlatformGraphEdge {
    [SerializeField] PlatformGraphNode jumpNode;
    public PlatformGraphNode JumpNode => jumpNode;
    [SerializeField] PlatformGraphNode landNode;
    public PlatformGraphNode LandNode => landNode;
    public float Weight => edgeCalcResult.Length;
    [SerializeField] EdgeCalcResult edgeCalcResult;
    public EdgeCalcResult EdgeCalcResult => edgeCalcResult;

    public PlatformGraphEdge(PlatformGraphNode jumpNode,
PlatformGraphNode landNode, EdgeCalcResult edgeCalcResult) {
        this.jumpNode = jumpNode;
        this.landNode = landNode;
        this.edgeCalcResult = edgeCalcResult;
    }
}
```

PlatformGraphNode

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEditor;
using UnityEngine;

public class PlatformGraphNode : ScriptableObject {
    [SerializeField] string platformSideID;
    public string PlatformSideID => platformSideID;
    [SerializeField] Vector2 position;
    public Vector2 Position => position;

    public bool IsHorizontallyAligned(Vector2 otherPosition) {
        if(Mathf.Abs(position.y - otherPosition.y) > 0.1) return false;
```

```
        return true;
    }

    public bool IsHorizontallyAligned(PlatformGraphNode other) {
        if(Mathf.Abs(position.y - other.position.y) > 0.1) return false;

        return true;
    }

    public void SetPlatformSideID(string platformSideID) {
        this.platformSideID = platformSideID;
#if UNITY_EDITOR
        EditorUtility.SetDirty(this);
#endif
    }

    public void SetPosition(Vector2 newPosition) {
        position = newPosition;
#if UNITY_EDITOR
        EditorUtility.SetDirty(this);
#endif
    }
}
```

AStarPathfinding

```
using System.Collections;
using System.Collections.Generic;
using PlatformNavigation.Utility;
using UnityEngine;

public static class AStarPathfinding {
    public static List<PlatformGraphEdge>
FindPath(Dictionary<PlatformGraphNode, List<PlatformGraphEdge>>
neighbors, PlatformGraphNode start, PlatformGraphNode goal) {
        if (start == null) return null;
        if (goal == null) return null;

        PriorityQueue<PlatformGraphNode> openList = new();
        HashSet<PlatformGraphNode> closedList = new();
        Dictionary<PlatformGraphNode, PlatformGraphEdge> parentEdge =
new();

        Dictionary<PlatformGraphNode, float> GCost = new();
        Dictionary<PlatformGraphNode, float> HCost = new();

        GCost[start] = 0f;
        HCost[start] = GetHeuristic(start, goal);
        openList.Enqueue(start, CalculateFCost(GCost[start],
HCost[start]));

        while(openList.Count > 0) {
            PlatformGraphNode current = openList.Dequeue();
```

```
            if (current == goal) {
                return ReconstructPath(parentEdge, current);
            }

            closedList.Add(current);

            foreach(PlatformGraphEdge edge in neighbors[current]) {
                PlatformGraphNode neighbor = edge.LandNode;

                if (closedList.Contains(neighbor)) continue;

                float tentativeGCost = GCost[current] + edge.Weight;

                if (!GCost.ContainsKey(neighbor) || tentativeGCost <
GCost[neighbor]) {
                    parentEdge[neighbor] = edge;
                    GCost[neighbor] = tentativeGCost;
                    HCost[neighbor] = GetHeuristic(neighbor, goal);
                    openList.Enqueue(neighbor,
CalculateFCost(GCost[neighbor], HCost[neighbor]));
                }
            }
        }

        return null;
    }

    static float GetHeuristic(PlatformGraphNode a, PlatformGraphNode b) {
        return Vector2.Distance(a.Position, b.Position);
    }

    static float CalculateFCost(float GCost, float HCost) {
        return GCost + HCost;
    }

    static List<PlatformGraphEdge>
ReconstructPath(Dictionary<PlatformGraphNode, PlatformGraphEdge>
parentEdge, PlatformGraphNode current) {
        var path = new List<PlatformGraphEdge>();
        while (parentEdge.ContainsKey(current)) {
            PlatformGraphEdge edge = parentEdge[current];
            path.Add(edge);
            current = edge.JumpNode;
        }
        path.Reverse();
        return path;
    }
}
```

MovementUtility

```
using System.Collections;
using System.Collections.Generic;
using Unity.VisualScripting;
using UnityEngine;
```

```
public static class MovementUtility {
    public static IEnumerable<(Vector2, Vector2, EdgeCalcResult)>
CalculateJumpLandPosition(CharacterDebug character, LayerMask layerMask,
PlatformSide jumpSide, PlatformSide landSide) {
        if (jumpSide == landSide) yield break;
        if (jumpSide.Direction != PlatformSide.SideDirection.Top) yield
break;
        if (landSide.Direction != PlatformSide.SideDirection.Top) yield
break;

        (Vector2 jumpStartInner, Vector2 jumpEndInner) =
jumpSide.GetInnerPoints(character);
        (Vector2 landStartInner, Vector2 landEndInner) =
landSide.GetInnerPoints(character);

        List<Vector2> jumpPoints = new() { jumpStartInner, jumpEndInner
};
        List<Vector2> landPoints = new() { landStartInner, landEndInner
};

        if (jumpSide.Start.y > landSide.Start.y) {
            foreach(Vector2 raycastedOuterPoint in
GetRaycastedOuterPoints(character, jumpSide, landSide)) {
                landPoints.Add(raycastedOuterPoint);
            }
        }
        else {
            foreach(Vector2 raycastedOuterPoint in
GetRaycastedOuterPoints(character, landSide, jumpSide)) {
                jumpPoints.Add(raycastedOuterPoint);
            }
        }

        Debug.Log("JumpPoints: " + jumpPoints.Count + " LandPoints: " +
landPoints.Count);

        foreach(var result in
GetShortestDistanceFromJumpLandPoints(character, layerMask, jumpSide,
landSide, jumpPoints, landPoints)) {
            yield return result;
        }
    }

    public static IEnumerable<(Vector2, Vector2, EdgeCalcResult)>
GetShortestDistanceFromJumpLandPoints(CharacterDebug character, LayerMask
layerMask, PlatformSide jumpSide, PlatformSide landSide, List<Vector2>
jumpPoints, List<Vector2> landPoints) {
        if (jumpPoints == null) yield break;
        if (landPoints == null) yield break;

        foreach (Vector2 jumpPoint in jumpPoints) {
            (Vector2, Vector2, EdgeCalcResult) jumpLandPair = default;
            float minDistance = float.PositiveInfinity;

            foreach (Vector2 landPoint in landPoints) {
```

```
                Waypoint start = new(jumpPoint, jumpSide.ID);
                Waypoint end = new(landPoint, landSide.ID);

                EdgeCalcResult result = null;
                if (jumpPoint.y > landPoint.y) {
                    (Vector2 startOuter, Vector2 endOuter) =
jumpSide.GetOuterPoints(character);
                    Vector2 outer = Vector2.Distance(jumpPoint,
startOuter) < Vector2.Distance(jumpPoint, endOuter) ? startOuter :
endOuter;
                    result = GetEdgeFromWaypoints(character, layerMask,
new(){ start, new(outer, jumpSide.ID, false), end});
                    if (result != null) {
                        if (result.Length < minDistance) {
                            jumpLandPair = (jumpPoint, landPoint,
result);
                            minDistance = result.Length;
                        }
                    }
                }

                result = GetEdgeFromWaypoints(character, layerMask,
new(){ start, end });
                if (result == null) continue;

                float distance = result.Length;
                if (distance < minDistance) {
                    jumpLandPair = (jumpPoint, landPoint, result);
                    minDistance = distance;
                }
            }

            if (jumpLandPair != default) yield return jumpLandPair;
        }
    }

    public static EdgeCalcResult GetEdgeFromWaypoints(CharacterDebug
character, LayerMask layerMask, List<Waypoint> waypoints) {
        EdgeCalcResult result = new();
        if (waypoints.Count <= 1) return null;

        for (int i = 0; i < waypoints.Count - 1; i++) {
            Waypoint start = waypoints[i];
            Waypoint end = waypoints[i + 1];

            EdgeCalcResult edge = GetEdgeFromWaypoint(character,
layerMask, start, end);
            if (edge == null) return null;

            result.Combine(edge);
        }

        return result;
    }
```

```
    public static EdgeCalcResult GetEdgeFromWaypoint(CharacterDebug
character, LayerMask layerMask, Waypoint start, Waypoint end) {
        Vector2 origin =
character.GetPointFromPositionAlongPlatform(start.Position);
        Vector2 destination =
character.GetPointFromPositionAlongPlatform(end.Position);

        EdgeCalcResult result;
        EdgeCalcParams parameter = new() {
            character = character,
            origin = origin,
            destination = destination,
            timeStep = Time.fixedDeltaTime,
            maxDuration = Mathf.Abs(origin.x - destination.x) /
character.HorizontalMovement.Speed,
            layerMask = layerMask
        };
        if (start.PlatformSideId == end.PlatformSideId) {
            result =
EdgeStepUtility.CalculateHorizontalStepsBetweenWaypoint(parameter);
            if (result != null) return result;
        }

        float maxDuration = CalculateJumpDurationToWaypoint(destination -
origin, Vector2.zero, character.Jump.Gravity);
        if (maxDuration != float.PositiveInfinity) {
            parameter.maxDuration = maxDuration;
            result =
EdgeStepUtility.CalculateJumpStepsBetweenWaypoint(parameter, 0f);
            if (result != null) {
                if (result.resultType !=
EdgeCalcResult.ResultType.Failed) return result;
            }
        }

        if (start.CanJump) {
            maxDuration = CalculateJumpDurationToWaypoint(destination -
origin, new(0f, character.Jump.Speed), character.Jump.Gravity);
            if (maxDuration == float.PositiveInfinity) return null;
            parameter.maxDuration = maxDuration;
            result =
EdgeStepUtility.CalculateStepsBetweenWaypoint(parameter);
        }
        else {
            maxDuration = CalculateJumpDurationToWaypoint(destination -
origin, Vector2.zero, character.Jump.Gravity);
            if (maxDuration == float.PositiveInfinity) return null;
            parameter.maxDuration = maxDuration;
            result =
EdgeStepUtility.CalculateStepsBetweenWaypoint(parameter, jumpSpeed: 0f);
        }
        if (result == null) return null;
        if (result.resultType == EdgeCalcResult.ResultType.Failed) return
null;

        return result;
```

```
    }

    public static float CalculateJumpDurationToWaypoint(Vector2
displacement, Vector2 velocity, float gravity) {
        return CalculateMovementDuration(displacement.y, velocity.y,
gravity, 0f, false);
    }

    public static float CalculateMovementDuration(float displacement,
float velocity, float acceleration, float minDuration = 0f, bool
returnLowerResult = true) {
        if (acceleration == 0) {
            if (velocity == 0) return float.PositiveInfinity;
            else if ((displacement > 0) != (velocity > 0)) return
float.PositiveInfinity;
            return displacement / velocity;
        }

        float discriminant = velocity * velocity + 2 * acceleration *
displacement;
        if (discriminant < 0) return float.PositiveInfinity;
        float discriminantSqrt = Mathf.Sqrt(discriminant);
        float t1 = (-velocity + discriminantSqrt) / acceleration;
        float t2 = (-velocity - discriminantSqrt) / acceleration;

        if (t1 < 0 && t2 < 0) return float.PositiveInfinity;

        if (t1 < minDuration) {
            if (t2 < minDuration) return float.PositiveInfinity;
            else return t2;
        }
        else if (t2 < minDuration) {
            if (t1 < minDuration) return float.PositiveInfinity;
            else return t1;
        }
        else {
            if (returnLowerResult) return Mathf.Min(t1, t2);
            else return Mathf.Max(t1, t2);
        }
    }

    public static bool IsLineOverlap(float start1, float end1, float
start2, float end2) {
        (start1, end1) = SwapStartEndPoint(start1, end1);
        (start2, end2) = SwapStartEndPoint(start2, end2);

        return start1 < end2 && start2 < end1;
    }

    public static (float, float) SwapStartEndPoint(float start, float
end) {
        if (start > end) return (end, start);
        return (start, end);
    }
```

```
    static IEnumerable<Vector2> GetRaycastedOuterPoints(CharacterDebug
character, PlatformSide top, PlatformSide bottom) {
        Vector2? raycastedStart = PlatformSide.RaycastPoint(top.Start,
bottom);
        if (raycastedStart != null) {
            Vector2 raycastedStartOuter =
PlatformSide.GetOuterPoint(character, raycastedStart.Value,
PlatformSide.SideOrientation.Horizontal, true);
            if (bottom.IsAlong(raycastedStartOuter)) yield return
raycastedStartOuter;
        }
        Vector2? raycastedEnd = PlatformSide.RaycastPoint(top.End,
bottom);
        if (raycastedEnd != null) {
            Vector2 raycastedEndOuter =
PlatformSide.GetOuterPoint(character, raycastedEnd.Value,
PlatformSide.SideOrientation.Horizontal, false);
            if (bottom.IsAlong(raycastedEndOuter)) yield return
raycastedEndOuter;
        }
    }
}
```

EdgeStepUtility

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public static class EdgeStepUtility {
    public static EdgeCalcResult
CalculateStepsBetweenWaypoint(EdgeCalcParams parameter, float
minJumpHeight = 0f) {
        return CalculateStepsBetweenWaypoint(parameter,
parameter.character.Jump.Speed, minJumpHeight);
    }

    public static EdgeCalcResult
CalculateStepsBetweenWaypoint(EdgeCalcParams parameter, float jumpSpeed,
float minJumpHeight = 0f) {
        EdgeCalcResult result = new();
        CharacterDebug character = parameter.character;
        Transform transform = character.Transform;
        Vector2 originalPosition = transform.position;
        transform.position = parameter.origin;

        int direction = parameter.origin.x > parameter.destination.x ? -1
: 1;
        float currentTime = 0f;
        float horizontalTimeLate = 0f;
        bool hasPassedMinJumpHeight = false;

        while (currentTime < parameter.maxDuration) {
            if ((Vector2)transform.position == parameter.destination)
break;
```

```
            Vector2 nextPos = parameter.origin;
            if ((transform.position.y >= parameter.origin.y +
minJumpHeight) && !hasPassedMinJumpHeight){
                hasPassedMinJumpHeight = true;
                horizontalTimeLate = currentTime;
            }

            if (hasPassedMinJumpHeight) {
                if (transform.position.x != parameter.destination.x) {
                    nextPos = character.HorizontalMovement.Move(nextPos,
direction, currentTime - horizontalTimeLate);
                    if (direction == 1 && nextPos.x >
parameter.destination.x) nextPos.x = parameter.destination.x;
                    else if (direction == -1 && nextPos.x <
parameter.destination.x) nextPos.x = parameter.destination.x;
                }
                else nextPos.x = parameter.destination.x;
            }

            nextPos = character.Jump.Move(nextPos, currentTime,
jumpSpeed);
            if (currentTime + parameter.timeStep >=
parameter.maxDuration) nextPos.y = parameter.destination.y;

            result.AddStep(currentTime, nextPos);
            transform.position = nextPos;
            if (DetectCollisionsAtCurrentPosition(parameter)) {
                transform.position = originalPosition;
                if (minJumpHeight >= character.Jump.MaxJumpHeight ||
jumpSpeed == 0f) {
                    break;
                }
                float newMinJumpHeight = Mathf.Min(minJumpHeight +
parameter.jumpHeightStep, character.Jump.MaxJumpHeight);
                return CalculateStepsBetweenWaypoint(parameter,
newMinJumpHeight);
            }
            currentTime += parameter.timeStep;
        }

        if ((Vector2)transform.position == parameter.destination)
result.resultType = EdgeCalcResult.ResultType.Complete;
        else result.resultType = EdgeCalcResult.ResultType.Failed;
        result.edgeType = EdgeCalcResult.EdgeType.NonLinear;

        result.duration = currentTime;

        transform.position = originalPosition;
        return result;
    }

    public static EdgeCalcResult
CalculateJumpStepsBetweenWaypoint(EdgeCalcParams parameter, float speed)
{
```

```
        if(Mathf.Abs(parameter.origin.x - parameter.destination.x) > 0.1)
return null;

        EdgeCalcResult result = new();
        CharacterDebug character = parameter.character;
        Transform transform = character.Transform;
        Vector2 originalPosition = transform.position;
        transform.position = parameter.origin;
        float currentTime = 0f;

        while (currentTime < parameter.maxDuration) {
            if ((Vector2)transform.position == parameter.destination)
break;

            Vector2 nextPos = parameter.origin;

            nextPos = character.Jump.Move(nextPos, currentTime, speed);
            if (currentTime + parameter.timeStep >=
parameter.maxDuration) nextPos.y = parameter.destination.y;

            if (DetectCollisionsAtCurrentPosition(parameter)) {
                transform.position = originalPosition;
                break;
            }

            result.AddStep(currentTime, nextPos);
            transform.position = nextPos;

            currentTime += parameter.timeStep;
        }

        if (transform.position.x == parameter.destination.x)
result.resultType = EdgeCalcResult.ResultType.Complete;
        else result.resultType = EdgeCalcResult.ResultType.Failed;
        result.edgeType = EdgeCalcResult.EdgeType.NonLinear;

        result.duration = currentTime;

        transform.position = originalPosition;

        return result;
    }

    public static EdgeCalcResult
CalculateHorizontalStepsBetweenWaypoint(EdgeCalcParams parameter) {
        if(Mathf.Abs(parameter.origin.y - parameter.destination.y) > 0.1)
return null;

        EdgeCalcResult result = new();
        CharacterDebug character = parameter.character;
        Transform transform = character.Transform;
        Vector2 originalPosition = transform.position;
        transform.position = parameter.origin;

        int direction = parameter.origin.x > parameter.destination.x ? -1
: 1;
```

```
        float currentTime = 0f;

        while (currentTime < parameter.maxDuration) {
            if ((Vector2)transform.position == parameter.destination)
break;

            Vector2 nextPos = parameter.origin;

            if (transform.position.x != parameter.destination.x) {
                nextPos = character.HorizontalMovement.Move(nextPos,
direction, currentTime);
                if (direction == 1 && nextPos.x >
parameter.destination.x) nextPos.x = parameter.destination.x;
                else if (direction == -1 && nextPos.x <
parameter.destination.x) nextPos.x = parameter.destination.x;
            }
            else nextPos.x = parameter.destination.x;

            result.AddStep(currentTime, nextPos);
            transform.position = nextPos;

            currentTime += parameter.timeStep;
        }

        if (transform.position.x == parameter.destination.x)
result.resultType = EdgeCalcResult.ResultType.Complete;
        else result.resultType = EdgeCalcResult.ResultType.Failed;
        result.edgeType = EdgeCalcResult.EdgeType.Linear;

        result.duration = currentTime;

        transform.position = originalPosition;

        return result;
    }

    static bool DetectCollisionsAtCurrentPosition(EdgeCalcParams
parameter) {
        List<Collider2D> hitColliders =
parameter.character.DetectCollisions(parameter.layerMask);
        if (hitColliders != null) {
            if ((Vector2)parameter.character.transform.position ==
parameter.origin) return false;
            if ((Vector2)parameter.character.transform.position ==
parameter.destination) return false;

            return true;
        }
        else return false;
    }
}
```

EdgeCalcResult

```
using System;
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

[Serializable]
public class EdgeCalcResult {
    [Serializable]
    public class Step {
        public float time;
        public Vector2 position;
    }

    [Serializable]
    public enum ResultType {
        Failed,
        Complete
    }

    [Serializable]
    public enum EdgeType {
        Linear,
        NonLinear
    }

    [SerializeField] List<Step> steps = new();
    public List<Step> Steps => steps;
    public ResultType resultType = ResultType.Complete;
    public EdgeType edgeType = EdgeType.Linear;
    [SerializeField] float length = 0f;
    public float Length => length;
    public float duration = 0f;

    public void AddStep(float time, Vector2 position) {
        if (position == null) return;
        Step newStep = new() { time = time, position = position };
        if (steps.Count != 0) {
            length += Vector2.Distance(newStep.position,
steps[^1].position);
        }

        steps.Add(newStep);
    }

    public void Combine(EdgeCalcResult toCombine) {
        if (toCombine == null) return;
        foreach(Step step in toCombine.Steps) {
            AddStep(step.time, step.position);
        }
        duration += toCombine.duration;
        if (toCombine.resultType == ResultType.Failed) resultType =
ResultType.Failed;
        if (toCombine.edgeType == EdgeType.NonLinear) edgeType =
EdgeType.NonLinear;
    }
}
```

BIODATA PENULIS

Penulis dilahirkan di Surabaya, 28 Juli 2001, merupakan anak kedua dari 3 bersaudara. Penulis telah menempuh pendidikan formal yaitu di SDN Bunulrejo 2 Malang, SMPN 5 Malang dan SMAN 3 Madiun. Setelah lulus dari SMAN tahun 2020, Penulis mengikuti SBMPTN dan diterima di Departemen Teknik Informatika FTEIC - ITS pada tahun 2020 dan terdaftar dengan NRP 5025201207.