Pembuatan Pathfinding untuk Game Platformer dengan Memanfaatkan Library A* Karya Aron Granberg pada Unity Engine

David Theosaksomo
13515131
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]*davidtsaksomo@students.itb.ac.id*

***Abstract*— Makalah ini memberikan alternatif pembuatan pathfinding pada sebuah game platformer dengan menggunakan library A* yang bernama Astar Pathfinding Project karya Aron Granberg dengan Unity Engine. Pada makalah ini juga akan dibahas mengenai cara membuat algoritma A* untuk digunakan pada game platformer. Makalah ini dibuat berdasarkan proyek game debut yang dibuat oleh penulis. yang berjudul Garbageman Battle. Makalah ini mengasumsikan pembaca sudah mengenal dan memahami algoritma A*, serta game engine Unity.**

Keywords—A*; platformer; pathfinding; unity; Aron Granberg; Garbageman Battle

I. PENDAHULUAN

A* adalah algoritma yang banyak digunakan dalam pathfinding. Pathfinding adalah proses mencari jalan yang dapat dilalui untuk pergi dari satu titik ke titik yang lain. Pathfinding diperlukan apabila terdapat rintangan yang menyebabkan pergerakan lurus saja tidak cukup, sehingga diperlukan keputusan untuk memilih jalur tercepat yang dapat dilalui tanpa menabrak rintangan. Pathfinding banyak dibutuhkan dalam pembuatan game. Semisal kita ingin membuat sebuah karakter yang digerakkan oleh AI, maka pembuatan pathfinding diperlukan agar karakter tersebut dapat berjalan dengan 'normal' layaknya seperti karakter yang digerakkan oleh pemain.

Makalah ini mengasumsikan pembaca sudah mengenal dan paham tentang algoritma A*. Namun akan diberikan sedikit pengertian umum dari algoritma A*. A* bekerja dengan cara mengunjungi simpul tetangga dari simpul, atau grid apabila anda menggunakan tiles dalam game anda, yang sedang kita kunjungi sekarang , mulai dari titik awal hingga ditemukan titik tujuan. Saat mencari simpul baru untuk dikunjungi, kita menghitung nilai f-value untuk setiap simpul tetangga. F-value dihitung dengan rumus:

$$f(n) = g(n) + h(n)$$

Dimana n adalah nomor simpul, g(n) adalah cost dari simpul awal ke simpul ke-n, dan h(n) adalah heuristik, yaitu perkiraan cost dari simpul ke-n ke simpul tujuan. Dalam menentukan heuristic terdapat berbagai metode, semisal euclidean, yaitu menghitung panjang garis lurus dari titik sekarang ke titik akhir, atau manhattan, yang menghitung step/langkah yang dibutuhkan untuk sampai ke titik akhir. Euclidean baik apabila pergerakan dapat ke segala arah, sedangkan manhattan cocok digunakan apabila representasi simpul dalam bentuk grid dan pergerakan dapat dilakukan ke 4 arah atau 8 arah.

Game platformer adalah sebuah jenis game dimana perspektif dari game adalah side-view, dan pemain bergerak secara 2D dalam axis x dan y, dimana axis y adalah ketinggiannya. Dalam game platformer klasik, terdapat platform-platform dengan berbagai ketinggian, dimana pemain dapat mencapainya dengan melompat. Sekarang game platformer sudah bervariasiu dan tidak terpaku pada definisi diatas. Contoh dari game platformer yang terkenal adalah Mario Bros.

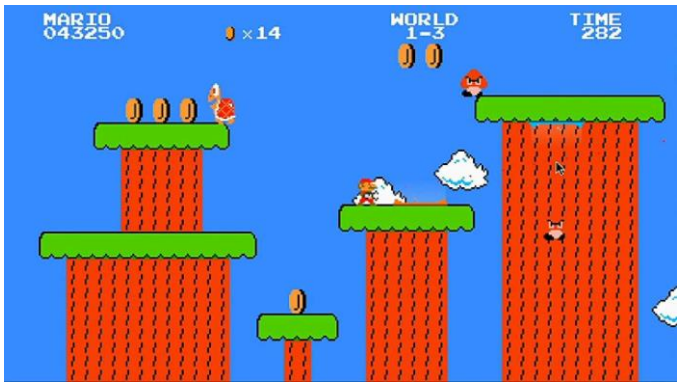

Figure 1. Classic Mario Bros Game sebagai contoh Game Platformer. Source: http://originalconsolegames.com

Dalam game Mario Bros terdapat musuh yang dapat bergerak ke kiri dan kanan, sesuai dengan arah posisi pemain. Pergerakan yang sederhana seperti ini tidak membutuhkan algoritma pathfinding. Namun bagaimana jika kita ingin membuat musuh yang lebih pintar? Semisal musuh tersebut dapat mengikuti pemain kemanapun ia pergi, meskipun berada di platform yang berbeda? Kita harus membuat musuh dapat mengetahui jalur yang dapat dilalui untuk sampai ke pemain dengan melompati platform-platform yang ada. Untuk dapat melakukan hal itu harus mengetahui platform mana yang dapat dijangkau dengan melompat, dan mencari jalur terpendek yang dapat dipilih. Untuk persoalan seperti ini, kita dapat memodifikasi algoritma A*, dengan mempertimbangkan

batasan pada y-axis, yaitu kemampuan dari musuh untuk melompat. A* untuk game platformer akan dibahas pada bab II makalah ini.

Pembuatan algoritma pathfinding untuk game platformer cukup kompleks dan sulit, dan membutuhkan waktu untuk mempelajari dan membuatnya. Pada makalah ini, penulis memberikan alternatif untuk membuat algoritma pathfinding dalam sebuah game platformer dengan memanfaatkan library A* karya Aron Granberg, yang bernama Astar Pathfinding Project. Library ini dapat digunakan pada Unity Engine dan dapat diunduh di situs Asset Store milik Unity. Library ini berisi fungsi lengkap untuk A*.

Unity Engine merupakan game engine yang banyak digunakan, dan menyediakan kemampuan untuk membangun game 2D dan 3D, dan dapat di bangun ke banyak patform diantaranya Windows, Android, dan iOS. Makalah ini mengasumsikan pembaca sudah memiliki pengetahuan terhadap Unity Engine, bila anda belum familiar maka dapat membuka situs Unity di https://unity3d.com/.

Makalah ini dibuat berdasarkan proyek game yang dimiliki penulis, yang berjudul Garbageman Battle. Garbageman Battle adalah game multiplayer dengan genre capture the flag, yang meyediakan pilihan untuk dapat bermain dengan AI.

II. A* Untuk Pathfinding Pada Game Platformer

Untuk membuat algoritma pathfinding pada game platformer, kita harus memodifikasi algoritma A* yang konvensional. Hal ini karena karakter tidak dapat bergerak dengan bebas pada y-axis, karena y-axis merupakan ketinggian sehingga kemampuan gerak pada y-axis bersesuaian dengan kemampuan melompat dari karakter bersangkutan.

Kebanyakan game platformer, termasuk yang klasik, menggunakan grid sebagai representasi map atau levelnya. Game seperti ini pada umumnya menggunakan pixel graphic atau tile-based graphic. Penggunaan grid dalam representasi map memudahkan pembuatan algoritma pathfinding. Pada bagian ini yang dibahas adalah algoritma pathfinding pada game dengan representasi map berupa grid.

Sebelumnya, kita harus menentukan hal-hal apa saja yang dapat dilakukan oleh objek/karakter yang ingin kita buat. Apabila karakter dapat terbang dengan bebas, maka algoritma A* biasa dapat langsung diaplikasikan. Jika tidak, kita harus memperhitungkan batasan pergerakannya. Pembahasan ini mengambil acuan dari artikel karya Daniel Branicki yang berjudul 'How to Adapt A* Pathfinding to a 2D Grid-Based Platformer: Theory'. Anda dapat membaca artikel tersebut di laman berikut: https://gamedevelopment.tutsplus.com/tutorials/how-to-adapt-a-pathfinding-to-a-2d-grid-based-platformer-theory--cms-24662. Pada pembahasan ini, kita ambil misalnya karakter dapat melompat setinggi 3 grid, hanya dapat bergerak ke 4 arah, yaitu atas, kanan, kiri, dan bawah dan tidak dapat bergerak secara diagonal.

Karena karakter hanya dapat melompat sejauh 3 grid, maka setiap karakter melompat dan sudah bergerak keatas sejauh 3 grid, maka karakter tidak dapat naik lagi, melainkan harus turus turun ke bawah hingga bertemu lantai. Saat melompat, karakter juga dapat bergerak ke kiri dan ke kanan. Ilustrasi contoh lompatan adalah sebagai berikut:

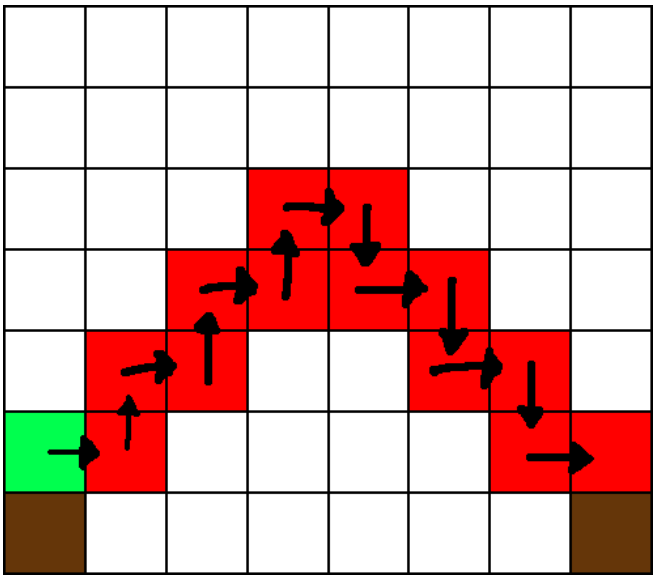

Figure 2. Maximum Distance Jump. Source: https://gamedevelopment.tutsplus.com/tutorials/how-to-adapt-a-pathfinding-to-a-2d-grid-based-platformer-theory--cms-24662

Untuk mencatat berapa ketinggian lompatan karakter pada grid tertentu, kita harus memberikan jump-value pada setiap grid yang kita kunjungi. Hal ini dilakukan agar algoritma dapat memutuskan apakah karakter dapat naik lebih tinggi atau mulai jatuh ke bawah. Pada saat melompat, semakin jauh karakter bergerak, nilai jump value akan bertambah. Kita ambil contoh dari gambar 2. Kotak hijau adalah kotak dimana karakter mulai melompat. Kotak setelah kotak hijau memiliki nilai jump value 1. Kotak setelahnya lagi memiliki nilai jump value 2, dan seterusnya. Hingga pada kotak terakhir nilai jump value adalah 13. Namun saat karakter menyentuh tanah kembali, nilai jump value dari kotak tersebut di ubah menjadi 0. Sehingga nilai jump value dari kotak terakhir adalah 0. Perhatikan bahwa pada saat berada di puncak lompatan, nilai jump value adaalah 6, yaitu dua kali lipat dari kemampuan lompat karakter. Pertambahan nilai jump value sebanyak dua kali lipat ini diakibatkan selain melakukan gerak keatas, karakter juga melakukkan gerak ke kanan. Bagaimana jika karakter hanya melakukan gerakan ke atas saja dan tidak bergerak ke kanan? Untuk menghitung jump value pada setiap kasus, kita menggunakan aturan berikut.

1. Pada saat di lantai, jump value bernilai 0.
2. Apabila karakter bergerak secara horizontal, nilai jump value bertambah 1.
3. Apabila karakter bergerak secara vertikal, tambah nilai jump value ke angka genap berikutnya yang terdekat.
4. Apabila bila nilai jump value sudah sama dengan atau lebih dari dual kali kemampuan lompat karakte, karakter hanya bisa bergerak secara horizontal atau bergerak ke bawah. Pada keadaan ini, grid atas diabaikan.

Mari kita ambil contoh pada suatu kasus.

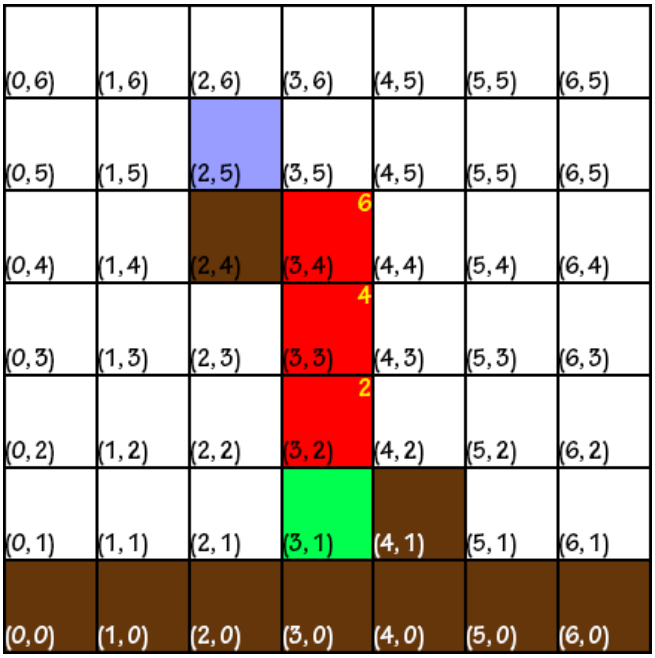

Figure 3. Contoh Kasus A Platformer. Source: https://gamedevelopment.tutsplus.com/tutorials/how-to-adapt-a-pathfinding-to-a-2d-grid-based-platformer-theory--cms-24662*

Titik awal kita adalah titik yang berwarna hijau yaitu titik (7,1). Titik tujuan kita adalah titik berwarna biru yaitu titik (2,5). Berdasarkan algoritma A*, kita mengunjungi grid yang sekiranya dapat membawa kita mendekati tujuan, dengan bantuan heuristik. Kita akan mengunjungi titik (3,2), (3,3), lalu ke (3,4). Namun pada titik (3,4), nilai jump value nya adalah 6, yaitu sebesar dua kali kemampuan lompat karakter, sehingga karakter tidak dapat naik lagi, dan tujuan tidak dapat dituju melalui rute ini. A* kemudian akan mencari rute lain. Rute lain tersebut adalah misalnya melalui grid (3,2), grid (4,2), grid (4,3), grid (4,4), hingga ke grid (3,4). Perhatikan bahwa saat karakter berada di grid (3,4), nilai jump-value nya sekarang adalah 4, berbeda dengan nilai jump value sebelumnya yaitu 6. Karakter masih dapat bergerak keatas menuju grid (3,5), hingga akhirnya sampai ditujuan yaitu grid (2,5). Satu grid yang sama bisa memiliki nilai jump value lebih dari satu, dan kita harus menyimpan semua nilai jump value agar rute dapat tersimpan dengan benar. Dalam algoritma A*, kita menganggap grid yang sama dengan jump value yang berbeda sebagai node yang berbeda, sehingga pada satu grid, dapat terdapat lebih dari 1 node. Sehingga bidang grid 2 dimensi dapat digambarkan sebagai grid 3 dimensi, dengan kedalaman z nya merupakan jump value dari grid tersebut.

Pada algoritma A* biasa, setelah suatu node dikunjung, node tersebut akan masuk ke daftar closed yaitu node yang sudah dikunjungi, dan tidak akan dikunjungi lagi. Namun pada kasus platformer, grid yang sama dikunjungi bisa lebih dari sekali, karena grid tersebut bisa saja memiliki jump value yang berbeda.

Harus menghitung jump value dan membuat representasi node 3 dimensi, persoalan A* untuk platformer merupakan persoalan yang cukup kompleks untuk diimplementasikan.

III. Library Astar Pathfinding Karya Aron Granberg

Astar Pathfinding Project merupakan library untuk Unity Engine karya Aron Granberg. Library ini dapat diunduh disitus Asset Store Unity. Library ini menyediakan struktur data, fungsi-fungsi dan lainnya yang dibutuhkan dalam membuat sebuah algoritma A* baik untuk game 2D atau game 3D.

Astar Pathfinding Project memiliki insterface yang baik melalui custom inspector. Pengguna dapat mengubah-ubah variabel yang digunakan dalam pathfinding, seperti jumlah node, letak-letak node, algoritma yang digunakan, melalui inspector dan interface pada unity editor.

Library ini dapat digunakan untuk berbagai macam representasi map/graf, yaitu Navmeshes, dimana pathfinding memperhitungkan adanya objek-objek mesh yang menghalangi, Grid Graph, dimana representasi map dalam grid dan setiap grid adalah node, dan Point Graph, dimana setiap map mempunyai graf sendiri yang diatur oleh pengguna. Pathfinding dengan weight cost/penalty juga dapat memanfaatkan library ini.

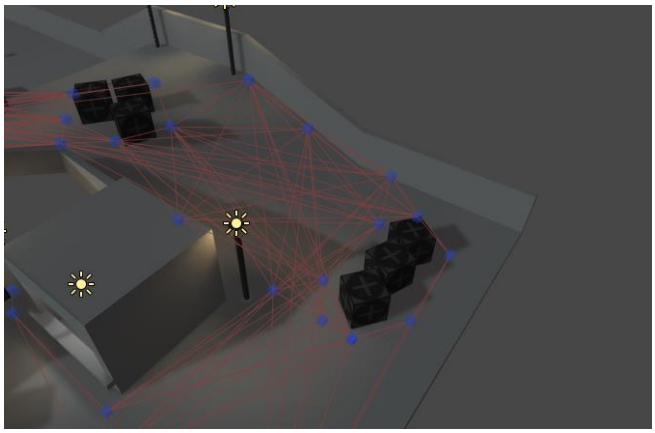

Figure 4. Point Graph pada Astar Pathfiinding Project. Source: https://arongranberg.com/astar/features

Untuk menggunakan A* Pathfinding Project, kita perlu membuat sebuah GameObject yang diberi nama "A*" pada scene kita dan menambahkan komponen "AstarPath" (Components–>Pathfinding–>Pathfinder). Komponen Astar Path memiliki bagian Graphs, dimana kita bisa membuat graf yang akan kita pakai. Kita bisa memilih satu dari tiga jenis graf yaitu Grid Graph, NavMeshGraph, PointGraph. Kita dapat membuat lebih dari satu graf. Selanjutnya ada bagian Settings, dimana kita bisa mengubah pengaturan untuk A* kita. Kita dapat mengatur jumlah thread yang digunakan, metode Heuristic yang digunakan, dan pengaturan spesifik lain. Terdapat juga bagian Save Load untuk menyimpan graf ke file eksternal, serta Optimization bagi yang membeli versi pro.

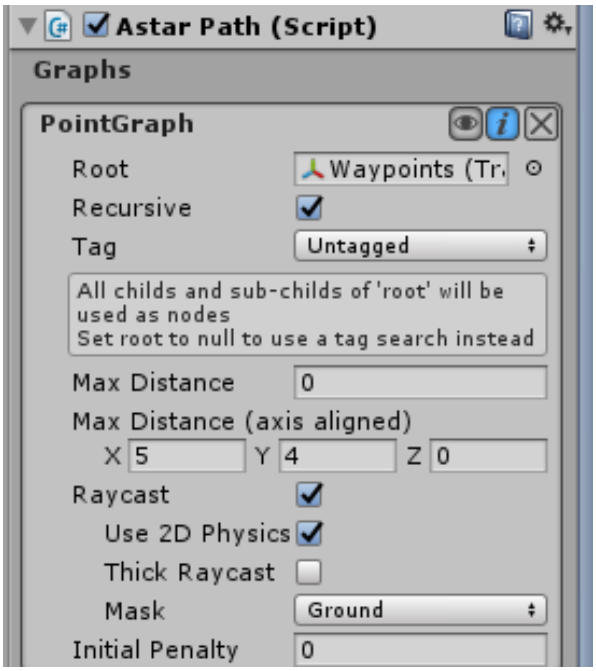

Figure 5. Komponen Astar Path Pada Unity

Untuk membuat AI yang bergerak pada graf yang kita buat, kita perlu membuat GameObject untuk AI tersebut dan menambahkan komponen Seeker pada AI tersebut. Seeker adalah komponen yang menjadikan GameObject dapat mengikuti nodes-nodes yang ada pada graf pada Objek A* sesuai dengan algoritma A*. Baru pada GameObject AI kita kita tambahkan script kita sendiri untuk mengatur bagaimana logika AI milik kita dalam mengikuti pathfinding.

Pathfinding untuk game platformer belum didukung oleh library ini secara langsung. Agar kita dapat menggunakan library ini untuk membuat pathfinding untuk game platformer, kita harus menambahkan algoritma kita sendiri.

IV. Menggunakan Library Astar Pathfinding Karya Aron Granberg Untuk Membuat Pathfinding Pada Game Platformer

Setelah membuat GameObject A* dan GameObject AI kita dengan komponen seeker di dalamnya, saatnya menambahkan script untuk mengatur logika dari AI kita dan membuat pengaturan pada GameObject A* agar pathfinding dapat digunakan sesuai dengan rules dari game platformer.

Pertama mari buat graf pada GameObject A* kita. Kita dapat memilih NavMeshGraph, PointGraph, atau GridGraph. NavMeshGraph baik apabila level dari game anda luas dengan hanya sedikit rintangan, seperti pada padang rumput dengan hanya satu atau dua rumah. GridGraph bagus apabila representasi map level anda berupa grid-grid atau pixel-based. PointGraph bagus apabila keakuratan tidak terlalu diperlukan sehingga hanya diperlukan jumlah node yang tidak terlalu banyak, PointGraph dapat lebih cepat karena jumlah node jauh lebih sedikit dibandingkan dengan grid. Namun anda harus membuat node PointGraph secara manual, sehingga sedikit merepotkan jika game anda memiliki sangat banyak level. Untuk game platformer dengan graphic pixel-based dan map yang terkotak-kotak, representasi GridGraph baik dipakai. Untuk game platformer selain itu dapat menggunakan PointGraph. NavMeshGraph lebih cocok ke game open world dan kurang cocok untuk game platformer.

Untuk GridGraph, pada pengaturan graf atur jumlah nodes pada graf, dalam representasi width * height, dan ukuran node. Untuk PointGraph kita harus membuat GameObject yang menjadi parent GameObject lain yang posisinya merepresentasikan posisi node. Jadi, apabila kita ingin membuat PointGraph dengan 100 node, maka kita harus membuat 100 GameObject + 1 GameObject sebagai parent. Jadikan GameObject parent sebagai variabel Root pada PointGraph di objek A* kita.

Lalu pada bagian setting, atur berapa jumlah thread yang kita inginkan. Pathfinding kadang sangat memakan resources, sehingga agar game tetap berjalan lancar, perhitungan algoritma pathfinding perlu dilakukan di thread yang berbeda. Bayangkan semisal ada 30 AI dalam satu level yang semuanya menggunakan pathfinding. Semakin banyak AI yang menggunakan pathfinding, diperlukan semakin banyak thread agar game berjalan lancar.

Di bagian setting juga kita dapat memilih metode heuristic yang ingin digunakan. Untuk GridGraph metode manhattan lebih baik, untuk PointGraph metode Euclidean lebih baik. Gunakan Diagonal Euclidean jika anda memperbolehkan gerakan diagonal. Di komponen ini juga banyak pengaturan lain yang dapat anda *tweak* sesuai dengan kebutuhan game anda.

Setelah graf pada map selesai, sekarang kita buat AI yang akan bergerak sesuai dengan graf yang kita buat. Pada

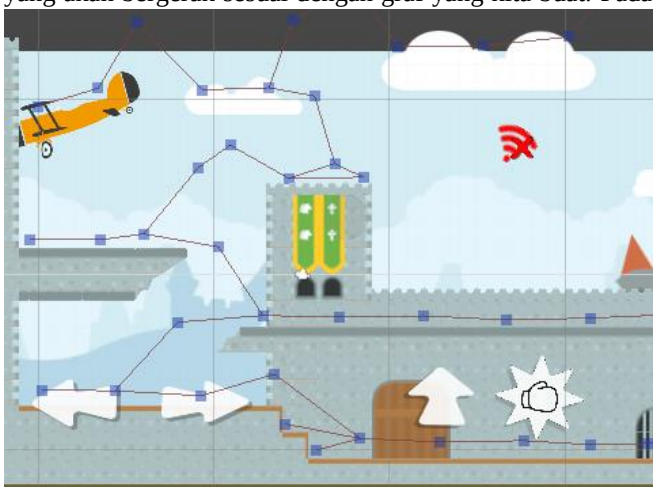

Figure 6. Point Graph pada game Garbageman Battle

GameObject AI, tambahkan komponen Seeker. Pada komponen Seeker kita dapat mengatur perilaku umum dari AI untuk mengikuti jalur pada graf, semisal kita dapat mengatur penalty apabilai AI melewati daerah dengan tag

tertentu. Misalnya, AI robot akan lebih memilih jalan yang datar dan apabila melewati jalur dengan tag berbatu misalnya, maka pada jalur tersebut akan diberi penalti sehingga AI lebih cenderung memilih jalan yang tidak berbatu.

Selanjutkan adalah memberikan logika pada AI kita dan mengatur bagaimana perilaku AI kita dalam mengikuti path dari graf yang dibuat sesuai dengan Game Platformer.

Hal pertama yang perlu dibuat adalah bagaimana AI kita bergerak. Kita perlu menambahkan script pada AI yang mengatur bagaimana cara AI bergerak. Misalnya, dengan membuat prosedur MoveLeft(), MoveRight(), dan Jump(), yang membuat AI bergerak ke kiri, ke kanan, dan melompat, atau Fly() jika AI yang kita buat memiliki kemampuan untuk terbang. Pada Unity pergerakan AI dapat menggunakan translate atau AddForce. Kita harus mengetahui berapa tinggi maksimum dari AI dapat melompat, dan berapa jarak horizontal maksimum yang dapat dicapai dari AI apabila AI melakukan lompatan. Nilai-nilai ini akan digunakan untuk mengisi variabel pada algoritma kita nanti. Jangan lupa untuk memperhatikan hal-hal kecil semisal AI tidak dapat melompat selama masih di udara, dan sebagainya.

Sekarang saatnya membuat algoritma pathfinding untuk AI. Algoritma yang dibuat tidak kompleks, karena sudah menggunakan pathfinding yang disediakan oleh library A*. Kita hanya membuat penyesuaian sesuai dengan game kita dan sesuai dengan perilaku AI pada game platformer. Cara berikut ini adalah algoritma dengan menggunakan PointGraph. Untuk GridGraph dapat dilakukan penyesuaian sesuai dengan representasi grid. Pertama adalah kita perlu menyimpan komponen seeker dalam suatu variabel dengan menggunakan generic method GetComponent dari Unity. Buat juga sebuah variabel bertipa Path. Saat kita ingin memulai pathfinding, kita hanya perlu memanggil method StartPath dari seeker, dengan 3 parameter, yaitu posisi awal dan posisi akhir (vektor) dan callback saat path selesai. Callback adalah method yang akan dijalankan saat path selesai. Method akan dipanggil dengan parameter path yang telah selesai. Buatlah Method tersebut, semisal bernama OnPathComplete. Pada OnPathComplete kita mengassign variable Path kita dengan path yang berhasil digenerate.

AI kita perlu menyimpan data sudah di node ke berapa dia berada dalam waypoint, semisal dalam variabel currentWayPoint. Waypoint adalah jalur yang telah ditemukan melalui algoritma A*, dalam hal ini disimpan dalam variabel path. Dalam method Update (method yang dijalankan dalam loop) cek apakah currentWayPoint sudah sama dengan jumlah dari node dalam path, dengan variabel path.vectorPath.Count. Apabila ya, maka karakter telah sampai di node terakhir, atau node tujuan. AI tinggal menunggu target selanjutnya dan membuat path baru. Apabila tidak, maka AI perlu mendekati waypoint selanjutnya. Kita dapat mengetahui posisi waypoint selanjutnya, dan dengan menguranginya dengan posisi karakter kita sekarang, kita akan mendapatkan selisih antara waypoint selanjutnya dengan AI kita sekarang.

Cek apabila selisih posisi tersebut dalam axis y cukup tinggi sehingga harus dicapai dengan melompat, namun masih dapat dicapai (yaitu kurang dari maksimum tinggi lompatan) dan selisih x axis nya masih kurang dari jarak maksimum yang dapat dicapai AI, maka panggil method Jump agar AI melompat. Perhatikan bahwa semakin tinggi platform tujuan, semakin kecil jarak dalam x axis yang dapat dicapai AI sehingga dibuat perhitungannya sendiri agar AI dapat memilih jalan dengan tepat. Pastikan juga diatas AI tidak ada platform dengan jarak dekat, agar AI tidak terbentur platform. Selanjutnya kita periksa selisih axis x dari kedua posisi tersebut, apabila < 0 maka panggil method MoveLeft, dan apabila > 0 maka MoveRight.

Apabila AI sudah berada di waypoint yang dituju, inkremen variable currentWayPoint. Dengan ini AI akan terus mengikuti waypoint selanjutnya.

AI yang kita buat sekarang sudah bisa menemukan jalan untuk pergi ke titik tujua. AI kita akan melompat apabila node tujuan cukup tinggi. AI akan mengikuti waypoint hingga ke waypoint terakhir. Untuk membuat path, kita tinggal memanggil method StartPath pada komponen seeker dengan mem-pass parameter posisi titik yang dituju.

V. CONTOH PENGAPLIKASIAN: GARBAGEMAN BATTLE

Pada Garbageman Battle, pemain dapat bermain dengan AI untuk memperebutkan suatu objektif. Karakter yang berhasil mendapatkan objektif harus meletakkannya di pos tujuan. Namun karakter lain dapat menyerang kita dan merebut objektif yang kita bawa. Apabila kita berhasil membawa objektif ke pos, kita akan mendapatkan poin. Game ini menggunakan level map seperti pada game platformer sehingga dibutuhkan pathfinding untuk game platformer.

Pada game ini, AI harus dapat menemukan jalan menuju objektif dengan rute tercepat karena harus berlomba dengan pemain dan AI yang lainnya. Untuk itu dibutuhkan algoritma pathfinding yang tepat agar AI dapat bermain dengan baik dan cukup menantang bagi pemain.

Figure 7. AI pada Garbageman Battle yang berebut objektif

Untuk dapat bergerak AI kita butuh suatu titik tujuan sebagai target algoritma pathfinding. Untuk membuatnya, dibuat kelas AILogic yang dipasang sebagai komponen pada GameObject AICharacter. AI memiliki beberapa state yang

menentukan keputusan dari AI. State yang ada yaitu state SearchFlag, state GoToFlag, state GuardFlag, dan terakhir state RetrieveFlag.

SearchFlag adalah state saat AI masih belum tau lokasi objektif. Pada state ini AI akan memilih suatu titik random yang dijadikan sebagai tujuan, hingga ditemukan objektif dalam jangkauan penglihatan AI. Sehingga AI akan terlihat seperti mencari, seperti apa yang akan dilakukan pemain saat belum tau dimana lokasi objektif berada.

GoToFlag adalah state saat AI sudah mengetahui posisi objektif. Pada saat ini AI membuat path baru dengan posisi objektif sebagai titik tujuannya. Pembuatan path dilakukan secara berkala, sehingga walaupun objektif bergerak (misal karena dibawa pemain lain) path dari AI tetap terupdate. Sehingga pada state ini AI akan bergerak menuju objektif dan mengikutinya kemanapun ia bergerak, semisal objektif sedang terpental ke tempat yag jauh.

GuardFlag adalah state dimana objektif sedang dibawa oleh pemain lain atau AI lainnya. Pada saat ini AI akan menjadikan karakter yaag membawa objektif sebagai titik tujuan. Pembuatan path juga dilakukan secara berkala, karena posisi titik tujuan yang dapat berubah. AI juga dapat memperkirakan kemungkinan untuk dapat melakukan *ambush* atau pencegatan, yaitu dengan memilih rute lain yang bersilangan dengan rute karakter yang membawa objektif agar dapat mencegat karakter tersebut. Namun AI hanya sekedar memperkirakan rute karakter lain tersebut, jadi apabila karakter tersebut tiba-tiba merubah rutenya, pencegatan akan gagal dilakukan.

RetrieveFlag aktif apabila AI kita sedang membawa objektif. AI akan mencari path baru dengan titik tujuan adalah pos tempat menaruh objektif. Dengan ini maka AI akan mencari jalur tercepat untuk sampai di pos. Namun karakter lain dapat melakukan pencegatan untuk merebut objektif kita. Apabila AI mengetahui ada karakter lain yang berniat melakukan pencegatan, AI dapat mengubah rutenya untuk menghindar dari pencegatan.

Antara AI dan pemain dapat saling menyerang, dan apabila terserang, maka karakter akan terpental. Apabila AI terpental, AI akan melakukan pencarian path ulang. Objektif yang dibawa karakter juga akan terpental dan dapat diambil pemain lain.

Pada game ini juga terdapat pickups yang dapat diambil oleh karakter. Pickups ini berupa senjata yang dapat digunakan untuk menyerang musuh atau membantu dalam mobilitas. Pickups yang ada di game ini adalah Baseball Bat, Sword, Bazooka, dan Jetpack. Jetpack akan memberikan kemampuan terbang bagi AI untuk beberapa saat. Apabila AI melihat ada powerup dalam jangkauan, maka AI akan mengubah titik yang dituju menjadi posisi pickup sehingga AI akan bergerak menuju pickup. Apabila AI telah mendapatkan pickup (atau pickkup diambil karakter lain) maka posisi tujuan AI akan kembali seperti semula sehingga AI akan bergerak menuju tujuan semua.

Perubahan state pada AI terjadi sesuai dengan status yang dimiliki AI. Misal AI sedang membawa objektif dan berada pada state RetrieveFlag, namun diserang oleh karakter lain dan objektifnya direbut, maka AI akan berganti state menjadi GoToFlag dan memperbarui path berdasarkan titik tujuan yang baru. Saat permainan baru dimulai dan posisi objektif tidak diketahui, AI akan berada pada state SearchFlag. Apabila AI telah mendeteksi posisi objektif, maka state AI akan menjadi GoToFlag.

Jenis graf yang digunakan pada game ini adalah PointGraph, karena ukuran map tidak terlalu besar dan tidak terlalu kompleks sehingga node yang dibutuhkan tidak terlalu banyak untuk dapat meng-cover seluruh map. Jumlah node juga perlu dibatasi karena terdapat tiga AI di dalam satu level sehingga performansi dibutuhkan. Jumlah level yang dimiliki untuk game ini juga tidak perlu banyak, sesuai jumlah variasi yang diinginkan saja, tidak seperti game seperti Mario Bros dimana jumlah level menentukan play length sehingga untuk mendapatkan game yang dapat dimainkan lebih lama dibutuhkan lebih banyak level.

VI. KESIMPULAN

Alternatif yang diberikan pada makalah ini cukup mudah untuk diimplementasikan dalam membuat pathfinding untuk game platformer. Namun alternatif ini spesifik hanya untuk game yang dibuat dengan Unity Engine. Keefektifan dari alternatif ini juga bergantung dari jenis game yang anda buat. Alternatif ini lebih cocok untuk game dengan jumlah dan ukuran map yang tidak terlalu besar.

REFERENSI

[1] D. Branicki, "How to Adapt A* Pathfinding to a 2D Grid-Based Platformer: Theory", 24 August 2015, https://gamedevelopment.tutsplus.com/tutorials/how-to-adapt-a-pathfinding-to-a-2d-grid-based-platformer-theory--cms-24662

[2] A. Granberg, "A* Pathfinding Project", https://arongranberg.com/astar/features

[3] A. Granberg, "Get Started with the A* Pathfinding Project", May 1 2017, https://arongranberg.com/astar/docs/getstarted.php

PERNYATAAN

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

Bandung, 16 Mei 2016

David Theosaksomo/ 13515131