NAVIGASI *PLAYER* UNTUK PENCARIAN *OBSTACLE* PADA GAME SEPEDA MENGGUNAKAN METODE *PATHFINDING A**

SKRIPSI

Oleh:

HARIS BUDI ERWANTO

NIM. 10650006

JURUSAN TEKNIK INFORMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS ISLAM NEGERI
MAULANA MALIK IBRAHIM MALANG
2014

NAVIGASI *PLAYER* UNTUK PENCARIAN *OBSTACLE* PADA GAME SEPEDA MENGGUNAKAN METODE *PATHFINDING A**

SKRIPSI

Diajukan Kepada:

Fakultas Sains dan Teknologi
Universitas Islam Negeri Maulana Malik Ibrahim Malang
Untuk Memenuhi Salah Satu Persyaratan Dalam
Memperoleh Gelar Sarjana Komputer (S.Kom)

Oleh:

HARIS BUDI ERWANTO
NIM. 10650006

JURUSAN TEKNIK INFORMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS ISLAM NEGERI
MAULANA MALIK IBRAHIM MALANG
2014

SURAT PERNYATAAN

ORISINALITAS PENELITIAN

Saya yang bertanda tangan di bawah ini:

Nama : Haris Budi Erwanto

NIM : 10650006

Fakultas / Jurusan : Sains dan Teknologi / Teknik Informatika

Judul Penelitian : **NAVIGASI *PLAYER* UNTUK PENCARIAN *OBSTACLE* PADA GAME SEPEDA MENGGUNAKAN METODE *PATHFINDING A**

Menyatakan dengan sebenar-benarnya bahwa hasil penelitian saya ini tidak terdapat unsur-unsur penjiplakan karya penelitian atau karya ilmiah yang pernah dilakukan atau dibuat oleh orang lain, kecuali yang secara tertulis dikutip dalam naskah ini dan disebutkan dalam sumber kutipan dan daftar pustaka.

Apabila ternyata hasil penelitian ini terbukti terdapat unsur-unsur jiplakan, maka saya bersedia untuk mempertanggungjawabkan, serta diproses sesuai peraturan yang berlaku.

Malang, 1 Maret 2014

Yang Membuat Pernyataan

Haris Budi Erwanto
NIM. 10650006

NAVIGASI *PLAYER* UNTUK PENCARIAN *OBSTACLE* PADA GAME SEPEDA MENGGUNAKAN METODE *PATHFINDING A**

SKRIPSI

Oleh:

HARIS BUDI ERWANTO
NIM. 10650006

Telah disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Yunifa Miftachul A, MT	Fresy Nugroho, MT
NIP. 19830616 201101 1 004	NIP. 19710722 200101 1 001

Tanggal, 1 Maret 2014

Mengetahui

Ketua Jurusan Teknik Informatika

Dr. Cahyo Crysdian

NIP. 19740424 200901 1 008

NAVIGASI *PLAYER* UNTUK PENCARIAN *OBSTACLE* PADA GAME SEPEDA MENGGUNAKAN METODE *PATHFINDING A**

SKRIPSI

Oleh:

HARIS BUDI ERWANTO
NIM. 10650006

Telah Dipertahankan di Depan Dewan Penguji Skripsi dan
Dinyatakan Diterima Sebagai Salah Satu Persyaratan Untuk
Memperoleh Gelar Sarjana Komunikasi (S.Kom)

Tanggal 26 Januari 2010

Susunan Dewan Penguji		Tanda Tangan
1. Penguji Utama	: Fatchurrochman, Kom NIP. 19700731 200501 1 002	( )
2. Ketua	: Fachrul Kurniawan, M.MT NIP. 19771020 200901 1 001	( )
3. Sekretaris	: Yunifa Miftachul A, MT NIP. 19830616 201101 1 004	( )
4. Anggota	: Fresy Nugroho , MT NIP. 19710722 200101 1 001	( )

Mengetahui dan Mengesahkan
Ketua Jurusan Teknik Informatika

Dr. Cahyo Crysdian
NIP. 19740424 200901 1 008

PERSEMBAHAN

Saya persembahkan karya ini kepada :
Kedua orang tuaku yang aku sayangi,
Ayahku Nuri Arif, S.Pd dan ibuku Aliyah
yang telah membimbingku sampai saat ini

Adik adik ku,
Alfian edi mashudi, Andi syahrial Latifi, Helmi zulfan fanani,
Yang menjadi motivasiku
untuk menjadi teladan yang baik

Tim Game Gowes,
Pogal indra m,Oxali Triadmaja, M Agung Tarecha
Yang solid atas kerjasamanya selama ini

Teman se-kontrakan
Agus, Muiz, Paung, Susanto, Amrozi

Teman seperjuanganku, seluruh satu angkatan
Teknik informatika khususnya angkatan 2010
Semoga sukses menyertai kita

BARAKALLAH

KATA PENGANTAR

Assalamu'alaikum Wr. Wb.

Segala puja dan puji bagi Allah SWT, karena atas segala rahmat dan limpahan hidayah-Nya, skripsi yang berjudul "Navigasi *Player* Untuk Pencarian Obstacle pada Game Sepeda menggunakan metode *Pathfinding A**" ini dapat diselesaikan dengan baik. Sholawat serta salam semoga senantiasa terlimpahkan kepada Nabi Muhammad SAW beserta para sahabat dan seluruh umatnya yang talah mengubah dari jaman kegelapan , kepada jalan yang diridhai Allah SWT.

Dalam penulisan skripsi ini, penulis mengucapkan terima kasih kepada pihak pihak yang telah memberikan bantuan, baik motivasi ataupun ilmu. Terimakasih sedalam dalamnya kepada:

1. Prof. DR. H. Mudjia Rahardjo, selaku Rektor Universitas Islam Negeri (UIN) Maulana Malik Ibrahim Malang beserta seluruh karyawan pendukung Bapak dan Ibu sekalian terhadap Universitas Islam Negeri Malang turut memberikan pendidikan .
2. Dr. Bayyinatul Muchtaromah, drh. MSi, selaku Dekan Fakultas Sains dan Teknologi Universitas Islam Negeri Malang beserta staf. Bapak dan ibu sekalian sangat berjasa dalam mengembangkan fakultas saintek untuk menjadi maju dan lebih baik

3. Dr. Cahyo Crysdian selaku Ketua Jurusan Teknik Informatika, yang telah memberikan saran, membantu dan mengarahkan penulis menyelesaikan penulisan skripsi ini.
4. Ririen Kusumawati, M.Kom selaku Dosen Wali jurusan Teknik Informatika UIN Maliki Malang. Yang memberi saran baik pada waktu kuliah ,maupun konsultasi kepenasehatan studi
5. Yunifa Miftachul a,MT dan Fressi Nugroho,MT. Beliau selaku Dosen Pembimbing di bidang game teknologi , beliau yang telah membimbing dan mengarahkan penulis dalam menyusun skripsi ini sehingga pengerjaan berjalan dengan lancar.
6. Seluruh Dosen Universitas Islam Negeri (UIN) Maliki Malang, yang tidak bisa di sebutkan satu persatu, khususnya Dosen Teknik Informatika dan staf karyawan yang telah memberikan ilmu kepada penulis selama masa studi, dan motivasi untuk menyelesaikan penulisan skripsi ini.
7. Bapak dan Ibuku yang tercinta, adik-adikku dan seluruh keluarga besar di Gresik dan Lamongan yang telah banyak memberikan banyak doa, motivasi dan dorongan dalam penyelesaian skripsi ini.
8. Teman - temanku yang telah membantu, mensupport penulis hingga terselesaikannya skripsi ini, khususnya kepada teman-teman Teknik informatika angakatan 2010 semoga kalian dilancarkan dalam berbagai urusan oleh Allah SWT dan juga balasan yang setimpal atas jasa dan bantuan yang telah diberikan

Semoga dalam penulisan skripsi ini dapat bermanfaat bagi pembaca sekalian , kekurangan dalam penulisan mungkin di temukan dalam skripsi ini, kritik dan saran pembaca akan sangat membangun.

Wassalamualaikum Wr,Wb.

Malang ,1 Maret 2014

Penulis

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

ABSTRAK

Erwanto, Haris B. 2014. **Navigasi *Player* Untuk Pencarian *Obstacle* pada Game Sepeda menggunakan metode *Pathfinding A*.*** Pembimbing : Yunifa Miftachul A,MT (1) ,Fresy Nugroho,MT (2)

Kata Kunci : *Pathfinding A**, Navigasi, Game Sepeda.

Game pada saat ini sangat bervariasi dan banyak di jumpai hampir pada setiap perangkat elektronik,game saat ini sangatlah komplek dan membutuhkan teknologi di dalamnya

Pada game simulasi kebutuhan akan adanya navigasi sangatlah penting , sebagai penunjuk arah dari posisi sekarang sampai ke tujuan. Seperti halnya di dunia nyata yang mana kita tidak mengetahui arah. Dengan adanya tool atau alat tertentu untuk navigasi akan mempermudah untuk memngetahui arahan untuk menuju ke tujuan.

Berdasarkan latar belakang tersebut penulis ingin menerapkan navigasi pada game sepeda menggunakan algoritma pencarian Pathfinding A*, yang diterapkan pada sepeda (*player*) untuk menuju ke suatu tujuan (obstacle), dan di harapkan dapat membantu untuk menunjukkan rute tertentu agar tidak tersesat.

ABSTRACT

Erwanto, Haris B. 2014. ***Player* Navigation for a *obstacle* search using *Pathfinding A** in bike game simulation.** Supervisor : Yunifa Miftachul A,MT (1) ,Fresy Nugroho,MT (2)

Keywords : *Pathfinding A**, Navigation, Bike game.

Game is very varied and many encountered in almost every electronic device, this time the game is complicated and requires a new technology to play a heavy game.

In the simulation game navigation is very important, as a direction from the current position to the destination. like in the real world where we do not know the direction. Given a particular tool or tools will make it easier to navigate to find the direction to go to the destination.

Based on this background the author would like to apply to use the navigation on the bike game with pathfinding A * search algorithm, which is applied to the bike (the player) to get to a destination (obstacle), and is expected to help to indicate a best route in order not lost.

BAB I
PENDAHULUAN

1.1 Latar Belakang

Game saat ini banyak dimainkan di semua kalangan usia,baik muda maupun tua, karene sifat game itu sendiri yang menyenangkan, saat ini banyak game bermunculah setiap harinya, di dalam game terdapat beberapa genre game. contohnya seperti game simulasi, yaitu mensimulasikan suatu kegiatan tertentu pada komputer atau media lain yang seolah olah melakukan hal yang sama pada dunia nyata, game ini disimulasikan kedalam komputer dalam bentuk virtual Game 3D atau permainan di dalam komputer sebagian juga merupakan simulasi dari bentuk-bentuk nyata dalam kehidupan manusia.

Pandangan masyarakat tentang game masih sebagai media hiburan saja dibandingkan media pembelajaran. Karena sifat dasar game yang menantang,dengan Salah satu jenis permainannya yaitu game simulasi yaitu mengharuskan pemain untuk menggunakan sepeda tiruan yang digunakan untuk memainkannya.

Didalam game visualisasi membutuhkan alat dan juga game itu sendiri. Di dalam game tersebut kita dihadapkan pada alur cerita atau alur misi yang bersifat edukatif, yang sudah di desain oleh pembuat game , salah satunya adalah rintangan atau obtacle yang mana obstacle tersebut bisa saja harus di hindari atau harus di tabrak, untuk mengetahuin obstacle tersebut maka dibuatlah suatu model untuk menggambarkan arah untuk menunjukkan jalan yang akan di lalui *player* menuju *obstacle*, Metode *Pathfinding* menggunakan (Hart, Nilsson, & Raphael,

1968) dipilih untuk menyelesaikan masalah penentuan arah *player*. Dari berbagai hal diatas, peneliti tertarik untuk membuat game seperti dengan keadaan sesunggunya dengan menggunakan *Immersive Tool* (Ashaolu, 2012)dan juga dengan alur yang lebih edukatif.

1.2 Rumusan Masalah

Berdasarkan latar belakang yang telah dikemukakan dapat dirumuskan masalahnya adalah

1. Bagaimana membangun game simulasi untuk permainan edukatif yang menggunakan algoritma A* yang di terapkan pada payer?
2. Bagaimana cara *player* mengetahui letak obstacle untuk memperoleh misi?

1.3 Batasan Masalah

Mengingat banyaknya genre dan engine dalam game, maka dalam pembahasan dibatasi pada :

1. Genre game hanya berupa simulasi.
2. Engine dalam pembuatan menggunakan game engine, untuk mempercepat proses pembuatan game yang cukup kompleks.

1.4 Tujuan Penelitian

Tujuan dilakukannya penelitian ini adalah untuk

1. Untuk membuat game simulasi yang menarik menggunakan metode A* untuk menentukan jalur terpendek *player* menuju ke obstacle.
2. Menggunakan metode pathfinding menggunakan algorithma A* (Astar) sehingga *player* dapat mengetahui arah dari obstacle tersebut sehingga dapat sampai ke obstacle tanpa tersesat/ mengalami kebingungan.

1.5 Manfaat Penelitian

Hasil dari penelitian ini diharapkan akan memberikan manfaat terhadap pengembangan game di indonesia antara lain :

1. Mengembangkan teknologi virtual untuk simulasi di indonesia
2. Membuat media belajar berupa game interaktif

1.6 Metode Penelitian

Berikut adalah langkah-langkah metode yang yang di gunakan dalam penelitian,terdiri dari :

1. Analisis

   Pada tahap ini menganalisa setiap permasalahan yang akan muncul dalam pembuatan sistem ini,diantaranya:

   1. Identifikasi masalah

      Mengidentifikasi kelemahan dan kelebihan pada sistem.

   2. Analisis masalah

      Setelah semua masalah teridentifikasi , kemudian di analisis untuk menentukan solusi dari masalah tersebut.

   3. Analisis literatur

      Dalam memecahkan masalah,kita dapan mendapatkan solusi dari beberapa sumber referensi yang berkaitan dengan penelitian yang dilakukan, pada penelitian ini topik yang dikaji diantaranga : algorithma *Pathfinding A** , game engine , dan beserta materi pendukung dalam pembuatan game.

2. Desain

   Pada tahap ini membahas tentang desain sistem pada game, meliputi:

1. Pembuatan desain lapangan virtual yang digunakan.

   Pembuatan desain game berupa skenario yang akan di jalankan pada game pada waktu pembuatan nantinya.

2. Pembuatan desain output.

   Output yang akan dihasilkan adalah navigasi , berdasarkan node terbaik menggunakan algorithma *Astar*

3. Pembuatan desain input.

   Input pada game untuk di gunakan menggunakan *Astar*

4. Pembuatan desain proses.

   Tahapan pada sistem untuk menghasilkan navigasi pada game menggunakan *Astar*

5. Pembuatan desain antarmuka pada *Game engine*

   Rancangan desain game dan gui akan di gambarkan di sini.

3. Implementasi

   Pada tahap ini membahas tentang implementasi dari desain sistem pada tahapan sebelumnya.

   1. Implementasi algoritma

      Mengimplementasikan algoritma Astar pada game untuk proses pencarian obstacle.sehingga dapat menghasil kan navigasi bagi *player*

   2. Perancangan dan pembuatan game

      Merancang game menggunakan Astar pada navigasinya,kemudian membuatnya.

   3. Debugging

Melakukan pembenahan pada sistem yang mengandung *bug* agar tidak mengalami kesalahan ketika bermain.

4. Ujicoba

   Ujicoba game pada *player* scara langsung.

5. Pembuatan laporan

   Pembuatan laporan skripsi. Sebagai dokumentasi tugas akhir.

1.7 Sistematika Penelitian

Sistematika dalam penulisan skripsi ini akan dibagi menjadi beberapa bab sebagai berikut:

BAB I Pendahuluan

Pada bab ini berisi latar belakang, rumusan masalah, batasan masalah,tujuan penelitian, manfaat penelitian, metodologi penelitian, dan sistematika penulisan tugas akhir ini.

BAB II Tinjauan Pustaka

Pada Bab ini menjelaskan tentang teori-teori yang terkait dengan permasalahan yang diambil.

BAB III Analisis, dan Perancangan Sistem

Bab ini menjelaskan Perancangan pembuatan game yang dibuat, meliputi perancangan perangkat, IDE yang digunakan sampai pada pembuatan game.

BAB IV Hasil Dan Pembahasan

Bab ini membahas tentang implementasi dari algoritma pada sistem navigasi pada game. Serta melakukan pengujian pada permainan game untuk mengetahui Arah untuk menuju ke tujuan tersebut,dan apakah dapat menyelesaikan permasalahan yang dihadapi sesuai dengan yang diharapkan.

BAB V Penutup

Bab ini berisi kesimpulan dari keseluruhan dari lapotan tugas akhir dan saran yang diharapkan dapat bermanfaat untuk pengembangan pembuatan game selanjutnya.

Daftar Pustaka

Seluruh materi referensi dalam penulisan skripsi ini, dicantumkan dalam bab ini.

Lampiran

Data pendukung untuk melengkapi uraian yang telah disajikan dalam bagian utama di tempatkan di bagian ini.

BAB II
TINJAUAN PUSTAKA

2.1 Game Dalam Perspektif Islam

Game dalam islam di perbolehkan jika di lakukan sewajarnya dan tidak berlebihan, sehingga dapat melupakan kita dengan Allah.

يَا أَيُّهَا الَّذِينَ آمَنُوا لا تُحَرِّمُوا طَيِّبَاتِ مَا أَحَلَّ اللَّهُ لَكُمْ وَلا تَعْتَدُوا إِنَّ اللَّهَ لا يُحِبُّ الْمُعْتَدِينَ (٨٧)

"*Hai orang-orang yang beriman, janganlah kamu haramkan apa-apa yang baik yang telah Allah halalkan bagi kamu dan janganlah kamu melampaui batas, sesungguhnya Allah tidak menyukai orang yang melampaui batas.*" (Qs. al-Mâ'idah [5]: 87).

Dalam islam game di identikkan dengan olahraga yang menyehatkan dan menyenangkan, dan salah satu olah raga yang di sukai oleh nabi adalah mengendarai (kuda), hal ini terdapat pada hadits yang di riwayatkan oleh ibnu majah :

عَنْ عُقْبَةَ بْنِ عَامِرٍ الْجُهَنِيِّ قَالَ قَالَ رَسُولُ اللَّهِ صَلَّى اللَّهُ عَلَيْهِ وَسَلَّمَ ...ارْمُوا وَارْكَبُوا وَأَنْ تَرْمُوا أَحَبُّ إِلَيَّ مِنْ أَنْ تَرْكَبُوا وَإِنَّ كُلَّ شَيْءٍ يَلْهُو بِهِ الرَّجُلُ إِلَّا بَاطِلٌ رَمْيَةَ الرَّجُلِ بِقَوْسِهِ وَتَأْدِيبَهُ وَمُلَاعَفَرَسَهُ عَبَتَهُ امْرَأَتَهُ . ...رواه ابن ماجه

Artinya

"Memanahlah dan kenderailah olehmu (kuda). Namun, memanah lebih saya sukaidaripada berkuda. Sesungguhnya setiap hal yang menjadi permainan seseorang adalah batil kecuali yang memanah dengan busurnya, mendidik/ melatih kudanya dan bersenang-senang dengan istrinya".(Hr. Ibn Majah) (Al-Qazwiniy)

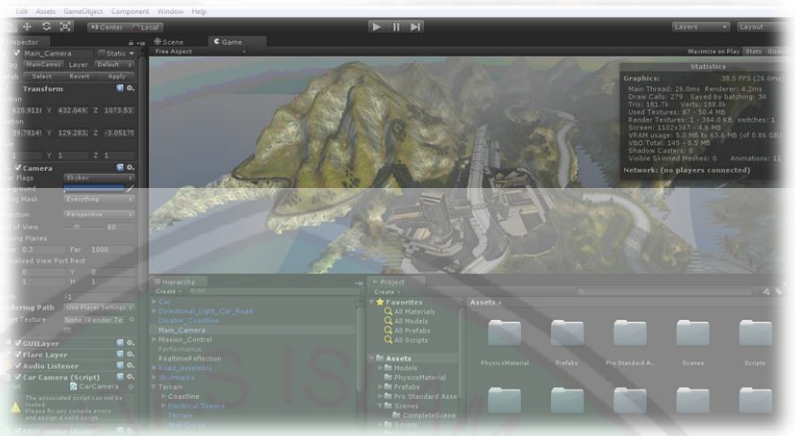
Dari Kutipan hadits di atas , berkendara adalah salah olah raga yang di sukai oleh nabi. sama halnya dengan berkendara dengan sepeda, pada game simulasi ini, menggunakan *Immersive tool* berupa Sepeda, sehingga untuk memainkan game ini dengan cara menggunakan sepeda yang sudah di rancang dengan penambahan sensor agar dapat terintegrasi dengan game.

2.2 Game Engine

Banyaknya game baru yang bermunculan banyak di pengaruhi beberapa faktor salah satunya adalah kemajuan teknologi di bidang game , membuat game saat ini saat ini berbeda dengan pada zaman dahulu ,kemudahan dalam membuat game pada saat ini di dukung oleh bermunculan *engine* untuk pembuatan game, selain dirasa mudah dan juga mempermudah untuk menganalisis kesalahan yang terjadi pada game tersebut.

Game engine yang digunakan Mendukung tiga bahasa dengan framework Mono open source, C#, JavaScript dan Phyton. Pada Game engine tersebut terdapat fasilitas yang memudahkan untuk penggolongan seperti, assest, animasi, tekstur, suara dan juga memungkinkan untuk mengimport model 3D dari aplikasi modeller lain, seperti Blender, untuk membuat real-time grafis menggunakan mesin rendering buatan sendiri dan juga dikombinasikan dengan *nVidia PhysX physics engine*

2.2.1 Antar muka dan control pada game Engine



Gambar 2. 1 Antar muka keseluruhan *Game engine*

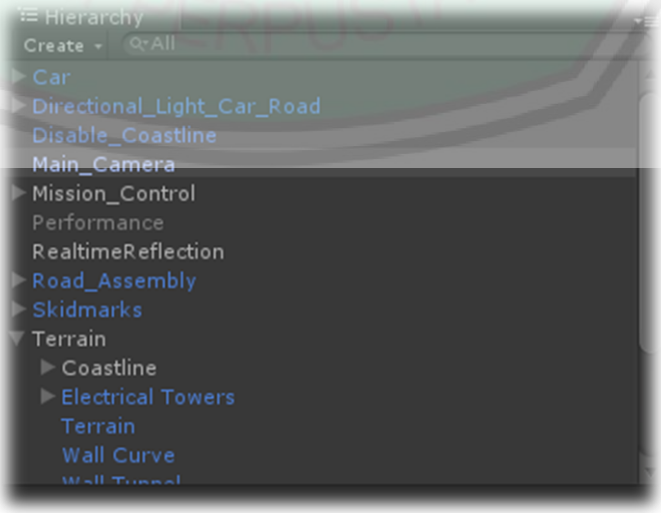
2.2.1.1 Inspector

Inspector berfungsi untuk melihat dan mengubah pengaturan dari semua objek yang ada pada game, komponen dari suatu game terdiri dari banyak aspek mulai dari *mesh,script,sound graphic*,dll. Untuk memanipulasi dari aspek tersebut kita bisa mengaksesnya melalui Inspektor,di dalamnya dapat mengubah pengaturan dari pada semua komponen dan material yang ada, seperti ukuran objek, letak koordinat objek, model kamera Dll. Control objek bisa menggunakan C# yang mana dapat mengendalikan dari inputan yang telah di berikan.

Gambar 2. 2 Tampilan *Inspector* Pada *Game engine*

2.2.1.2 Hierarchy

Hierarchy berisi tentang semua yang ada dalam suatu scene pada game, dan juga memberikan informasi tentang parenting pada suatu objek, seperti halnya suatu obek pohon yang mempunyai turunan objek berupa cabang.



Gambar 2. 3 Tampilan *Hierarchy* pada *Game engine*

2.2.1.3 Project

Project menampilkan semua file yang sudah kita buat pada proyek, pada tab ini menampilkan apa saja komponen yang ada pada game yang sedang di buat. pada gambar 4 dapat dilihat pada folder asset terdapat model, texture , scene,script dan lain lain.

Gambar 2. 4 Tampilan *Project* pada *Game engine*

2.2.1.4 Scene

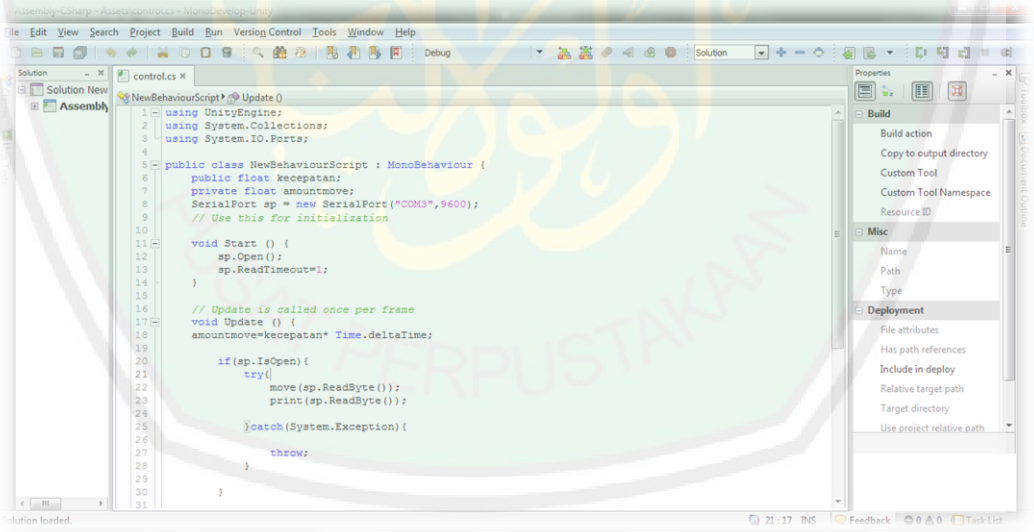
Scene menampilkan lembar kerja untuk memanipulasi objek,pada tab Scene kita dapat melakukan beberapa manipulasi game diantaranya: melakukan penggeseran objek, rotasi objek edit ukuran , perspektif objek, penambahan tekstur ,Dll

2.2.1.5 Game

Untuk menampilkan hasil dari manipulasi scene Pada game secara *realtime.* Pada tampilan tab Game kita dapat melihat hasil dari game yang telah di buat sebelum nantinya akan di build menjadi game final yang siap di gunakan.

2.2.2 Antarmuka Mono Develop

Pada *Game engine* untuk menggunakan Mono develop sebagai IDE untuk keperluan scripting pada game yang akan di buat, di dalam *game engine* tersedia 3 pilihan bahasa pemrograman diantaranya C# Script , JavaScript, dan Boo Script ,Fungsi Monodevelop adalah di gunakan untuk memudahkan kita dalam menulis sebuah program / script yang akan di masukkan pada game, kemudahan tersebut diantaranya adalah mengkoreksi adanya kesalahan penulisan kode, dalam penulian kode program biasanya tidak luput dalam penulisan huruf atau kekurangan suatu atribut, selain itu digunakan untuk debugging program, sehingga memudahkan proses penanganan error , dan mempermudah proses perbaikannya.



Gambar 2. 5 Antarmuka *MonoDevelop* untuk Kode Program

2.3 *PATHFINDING A* (Astar)*

Pathfinding adalah metode untuk menentukan arah pergerakan suatu object berdasarkan lokasi object yang ada di sekitarnya dari satu titik ke titik yang lain. Metode ini hampir dapat ditemukan pada semua game (Xiao Cui, 2011),terutama game 3d, permasalahan pergerakan *player* ataupun musuh dapat bergerak dengan lancar tanpa proses menunggu pergerakan manual dari inputan pemain.

2.3.1 Deskripsi *Astar (A*)*

Algoritma AStar adalah algoritma dari hasil perbaikan dari fungsi Heuristiknya metode *Best-First search* ,A* akan memberikan solusi yang terbaik dalam waktu yang optimal dengan meminimumkan total biaya litasan yang di laluinya. (Kusumadewi, 2003)

Algorithma AStar banyak di gunakan pada game dan aplikasi interaktif seperti halnya algorithma BFS karena sederhana dan efektif (Peters, Kyaw, & Swe, 2013) hanya saja berbeda pada pengurutan node yang di lakukan pada garis perbatasan (Galochkin, 2013).

Cara kerja algoritma ini adalah mencari jalur terpendek dari posisi awal sampai posisi tujuan,algoritma ini memiliki 2 fungsi untuk menentukan solusi yang terbaik.Fungsi yang pertama adalah *g(n)* ,adalah fungsi untuk menghitung total cost yang di butuhkan dari node awal (starting point) sampai ke tujuan (goal),node dalam Astar mengandung beberapa informasi diantaranya posisi,nilai dari node ke node sekitar, dan nilai node ke tujuan (Higgins, 2002). Fungsi yang kedua adalah *h(n)*, adalah fungsi untuk perkiraan total *cost* yang di perkirakan dari node awal

sampai akhir. (Patel, 2013), dan A * menyeimbangkan dua fungsi tersebut ketika bergerak dari titik awal ke tujuan. Setiap pergerakan meliha f(n) = g (n) + h (n) terendah.

g = biaya (cost) yang dikeluarkan dari keadaan awal sampai keadaan n

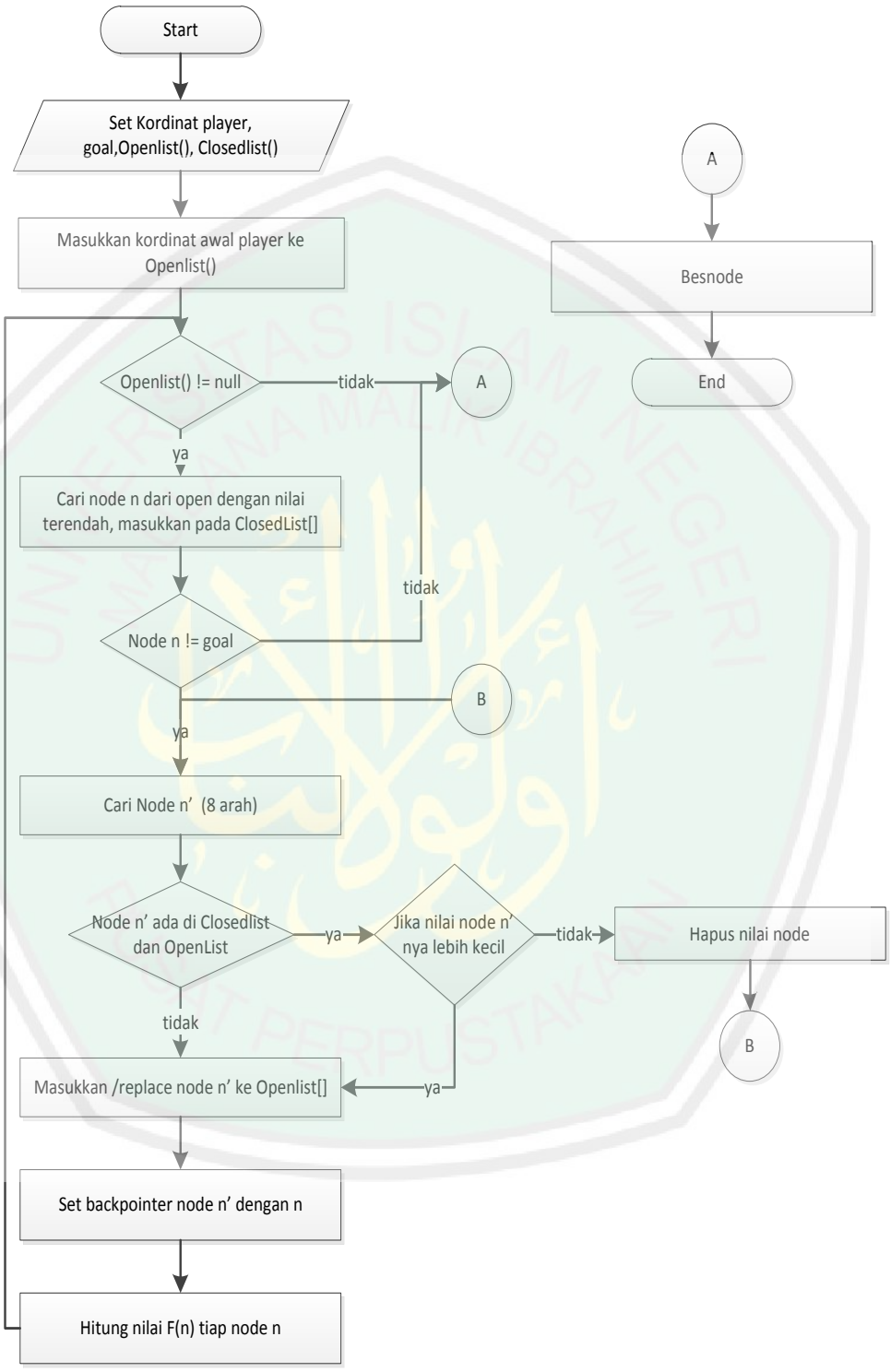
h = estimasi biaya untuk sampai pada suatu tujuan mulai dari n

Jadi untuk titik yang terbaik(dengan f terendah) di dalam open set(titik yang bisa di lalu), kemudian cek apakah itu posisi goal atau tidak,jika itu bukan goal dan *walkable(*dapat dilalui*)*, maka dihitung *cost* terbaik. (Matthews, 2002)

2.3.2 Manfaat *Astar (A*)*

Manfaat dari penggunaan algorithma *Pathfinding A** adalah kemampuanya untuk mencari suatu *goal* dengan memperhitungkan nilai jarak tiap node,antara posisi start sampai menuju ke goal sehingga menghasilkan node pilihan dan tercepat ke arah *Goal* , Pathfinding astar juga akan membarikan solusi yang terbaik dalam waktu yang optimal. (Kusumadewi, 2003)

2.4 Algoritma Astar



Gambar 2.6 Flowchart Proses Algoritma Astar

Keterangan :

1. Set posisi antara pemain dengan objek yang akan di cari pada map (x,y)
2. Set closedlist() dan openlist()
3. Memasukkan koordinat awal *player* ke Openlist[]
4. Jika Openlist[] kosong, return besnode , jika Openlist[] belum kosong maka lakukan :
   i. Cari node n pada openlist[] dengan f(n) minimal, kemudian masukkan node n tersebut pada ClosedList[]
   ii. Jika node n adalah goal, maka Selesai
   iii. Melihat ke node sekitar n (*neighbour node*)
   iv. Kerjakan pada tiap setiap node sekitar n, yaitu n' :
      1. Jika node n' belum ada pada Openlist / Closedlist maka :
         a. Masukkan node n' ke dalam Openlist, kemudian set *backpoint* (*parent node*) node n' dengan node n.
         b. Hitung fungsi heuristic f(n) = g(n) + h(n);
      2. Jika node n' sudah ada di open atau Closedlist dan nilainya lebih kecil, maka:
         a. Ganti dengan node n' terkecil
         b. Set Backpointer dari node n' (yang baru) dengan node n
5. Selesai

BAB III
ANALISIS DAN PERANCANGAN SISTEM

3.1 Analisa dan perancangan sistem

3.1.1 Keterangan Umum

Dalam suatu game simulasi, masalah yang sering muncul adalah masalah pada penerapan algoritma pada *player*, dari beberapa inisiasi input dari pemain hingga di proses oleh algoritma dan menghasilkan output. Dari masalah pertama dapat di selesaikan dengan mengkaji algoritma astar serta penerapannya pada *player*.

Dalam pembuatan sistem ini memiliki kelebihan dalam kompabilitas , yaitu game engine ini yang cross platform (unity3d, 2014),yang hasil ahir dalam pembuatan dapat di jalankan di sistem Windows,MAC,Android,IOS,dll. Akan tetapi akan di fokuskan pada platform Windows yang berbasis PC.

3.1.2 Analisis Kebutuhan Perangkat Keras

Spesifikasi Kebutuhan perangkat Keras yang di gunakan dalam penelitian ini adalah:

1. Prosesor intel Core2Duo T6570 @2.10 GHz
2. HardDisk 250 GB - 5600 RPM
3. RAM 4GB(2x2GB)
4. VGA IntelHD 45 series 1 Gb Shared memory
5. Keyboard
6. Mouse Laser
7. Monitor 14’’

3.1.3 Analisis Kebutuhan Perangkat Lunak

Kebutuhan perangkat lunak yang di gunakan dalam penelitian ini adalah :

1. Game Engine
2. Monodevelop
3. Google Sketchup v8.0
4. CorelDraw v.14
5. Windows 7 Ultimate Edition x64

Spesifikasi hardware dan software pada pc ataupun laptop,sudah mumpuni dan sudah di coba secara langsung untuk menjalankan game simulasi, hal yang berpengaruh dan menjadi prioritas adalah pada spesifikasi hardware, yang harus setara atau lebih pada spesifikasi di atas , karena hal ini berpengaruh pada pengolahan grafis yang menggunakan animasi 3D.

3.1.4. Analisis Literatur

Untuk membatu kelancaran dalam pembuatan sistem ini, maka di butuhkan pula literatur. Buku- buku yang berkaitan dengan algoritma, tentang pemrograman C# dan JavaScript.Buku tentang algorithma *Astar* juga menjadi rujukan untuk pengerjaan game ini, begitu pula dengan buku buku tentang *Script* bahasa pemrograman C# yang bisa di dapat pada literatur manual scripting yang di sediakan *Game engine*, khususnya dalam fungsi yang terintegrasi dengan game engine *Game engine*, literatur online yang berupa jurnal , *e-book* , *proceding* juga dilakukan untuk menunjang keterbatasan buku-buku yang membahas tentang *Astar*.

3.2 Rancangan Game

3.2.1 Keterangan umum Game

Game gowes ini adalah berupa game simulasi yang menggabungkan hardware dengan game, game simulasi sama halnya melakukan hal yang sebenarnya. Paga game simulasi ini menggunakan beberapa alat tambahan berupa *Immersive tool* Sepeda, yang berfungsi layaknya joystick pada game, sepeda yang di gunakan adalah sepeda biasa akan tetapi sudah mengalami modifikasi yang mana telah di pasang sensor untuk control pada game yang ada pada PC, Diantaranya Kecepatan, Rem, dan kontrol mundur.

Untuk pemrosesan data dari sensor dibutuhkan mikrokotroller yang menghubungkan antara beberapa sensor,yang kemudian di olah dan di transmisikan ke PC sebagai inputan game.

Pada sisi game yang di akan di dibangun, dengan background lingkungan kampus UIN Maliki Malang, dan juga dengan tambahan misi game untuk mengkoleksi point, dan misi tertentu,dengan model di buat mendekati dengan aslinya.

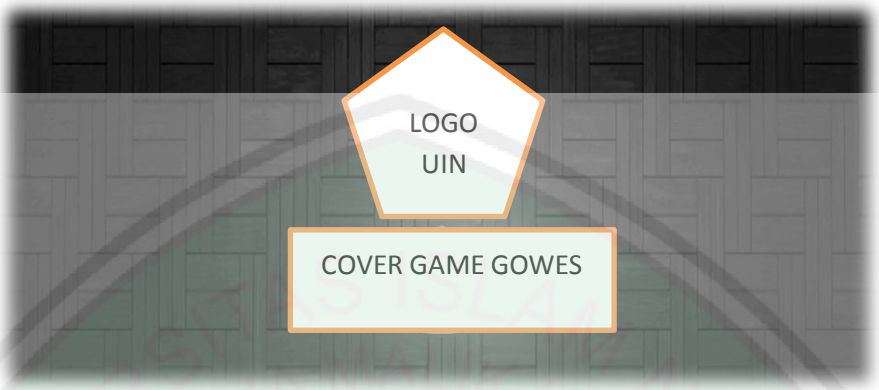
3.2.2 Desain Game

Rencana Scenee yang akan dibuat pada game ini adalah terdiri dari beberapa tampilan berikut:

a. Splash screen

Adalah tampilan awal ketika pertama kali membuka game, tampilan splash berdurasi dalam hitungan detik, berisi judul game dan

logo UIN malang dengan sedikit animasi transisi. Ilustrasi dapat di lihat pada gambar 3.1



Gambar 3. 1 *Splash Screen*

b. Main Menu

Adalah menu utama sebelum melakukan permainan , terdapat beberapa pilihan diantaranya sebagai berikut:

- Play

  Menu untuk memulai game

- Option

  Menu untuk beberapa pilihan yang telah disediakan, diantaranya :

  - Control

    Berisi Kontrol pada game

  - Credit

    Berisi tentang pembuat game

  - Back

    Kembali ke menu utama

❖ Exit

Menu untuk keluar dari game

Gambar 3. 2 Menu Utama

c. Game

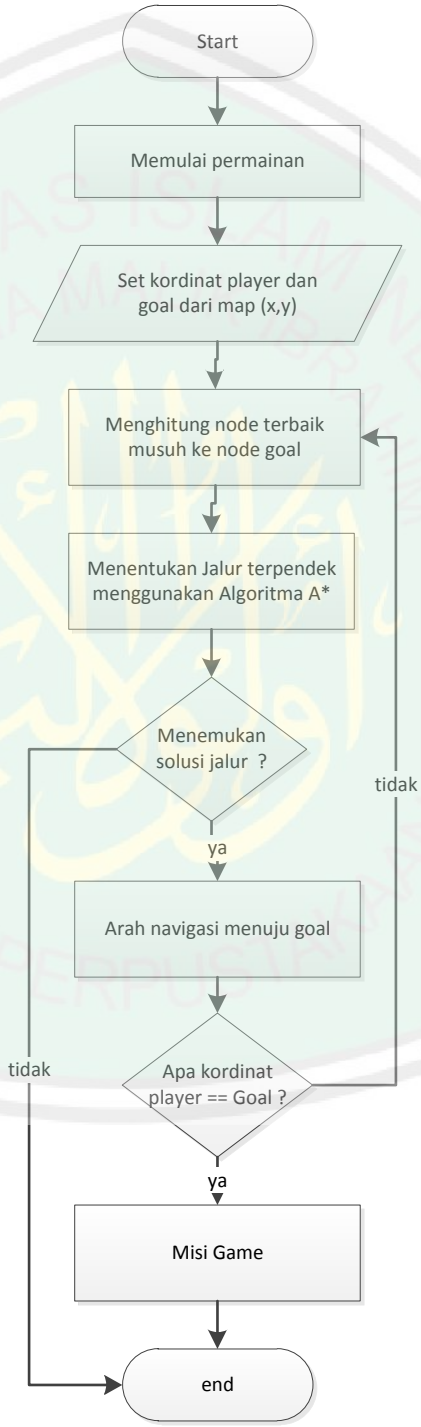
Pada tampilan game ini menampilkan beberapa item diantaranya :

- ❖ Spedo meter : Menampilkan kecepatan sepeda
- ❖ Point pickup : Berupa kubus yang apabila di koleksi / berbenturan pada sepeda , akan menambah score
- ❖ Timer : Menampilkan waktu yang tersisa menjalankan suatu misi
- ❖ Minimaps : Berisi map kecil , yang menampilkan jalan rute jalan.
- ❖ Score : Menampilkan score yang di dapat

Gambar 3. 3 Ilustrasi Permainan Game

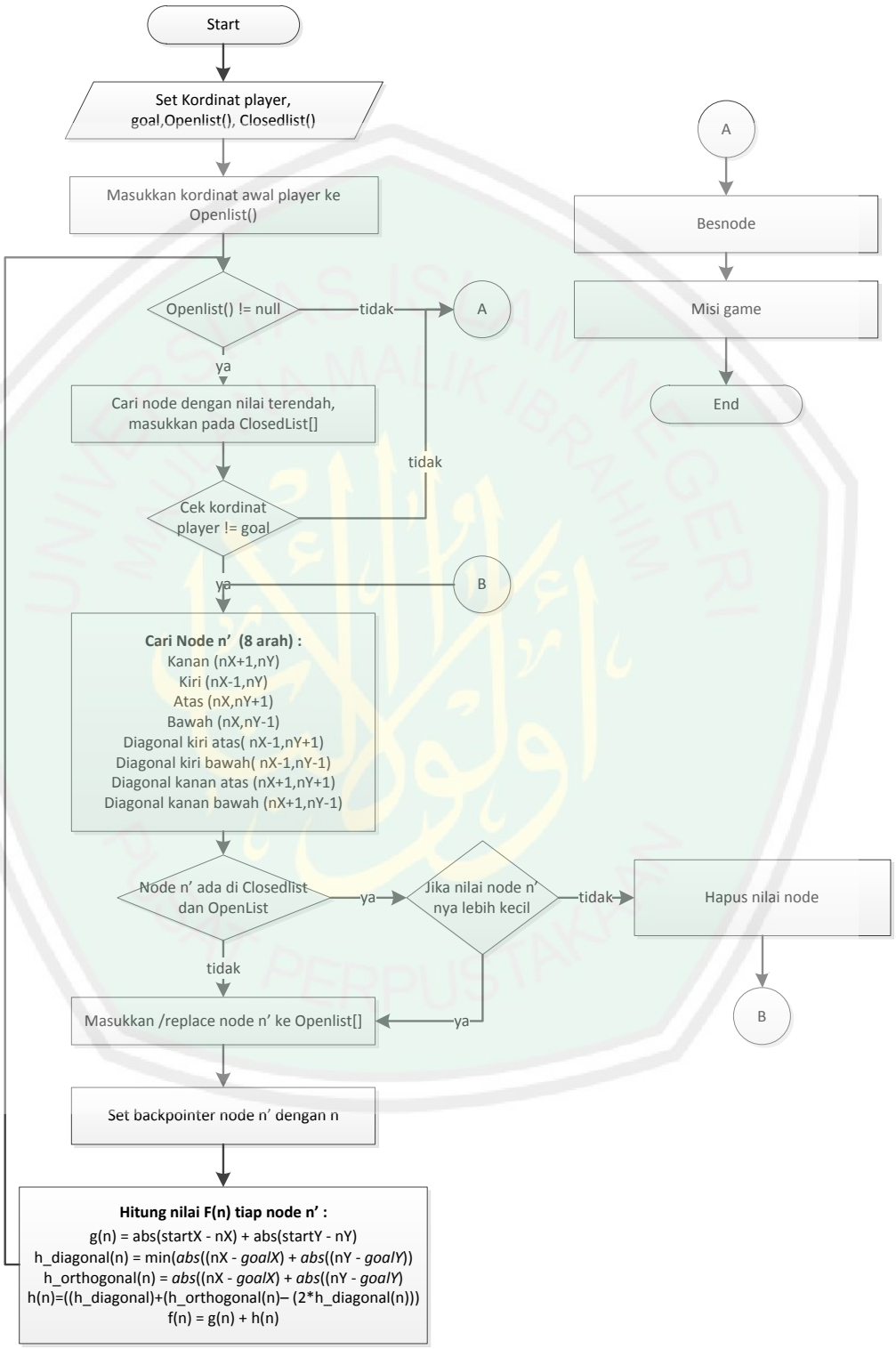
3.2.3 Flowchart Game

Rancangan game menggunakan *Astar* yang akan dibuat pada penelitian ini adalah sebagai berikut:



Gambar 3. 4 Flowchart game Menggunakan *Astar*

3.3 Proses *Astar*

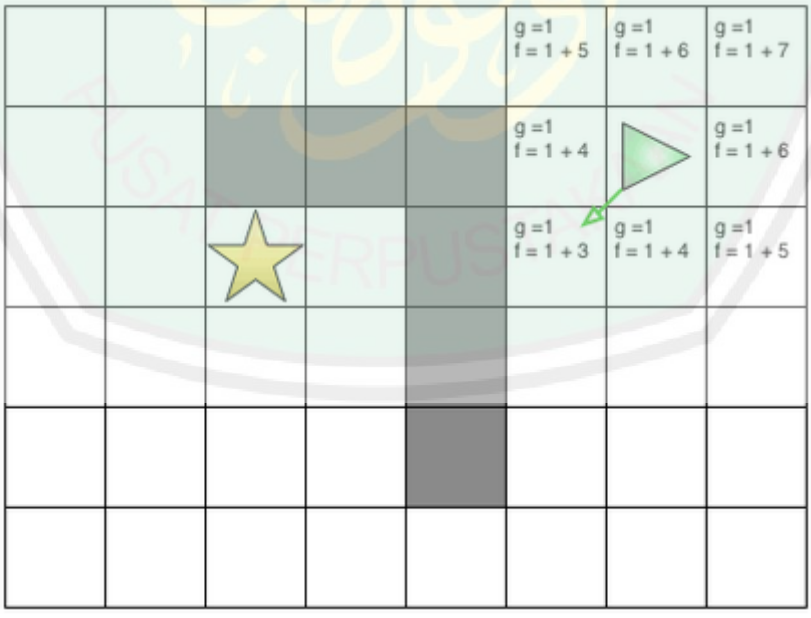


Gambar 3. 5 Flowchart *Astar* Pada game

Cara kerja algoritma ini adalah mencari jalur terpendek dari posisi awal sampai posisi tujuan,algoritma ini memiliki 2 fungsi untuk menentukan solusi yang terbaik.Fungsi yang pertama adalah *g(n)* ,adalah fungsi untuk menghitung total cost yang di butuhkan dari node awal (starting point) sampai ke tujuan (goal). Fungsi yang kedua adalah *h(n)*, adalah fungsi untuk perkiraan total *cost* yang di perkirakan dari node awal sampai akhir. (Patel, 2013), dan A * menyeimbangkan dua fungsi tersebut ketika bergerak dari titik awal ke tujuan. Setiap pergerakan melihat rumus

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

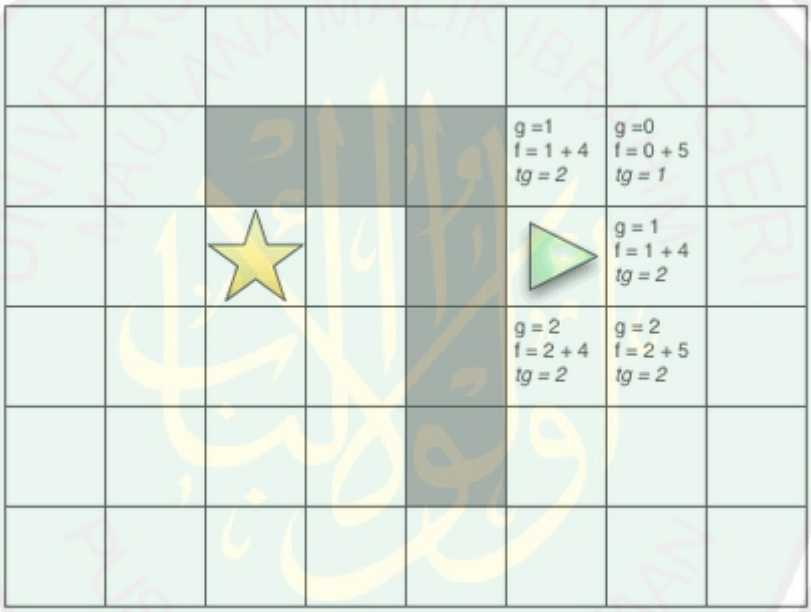
Jadi untuk titik yang terbaik(dengan f terendah) di dalam open set(titik yang bisa di lalu), kemudian cek apakah itu posisi goal atau tidak,jika itu bukan goal dan *walkable(*dapat dilalui*)*, maka dihitung *cost* terbaik (Unitygems, 2012), lihat gambar 3.4:



Gambar 3. 6 Menghitung *cost* Terbaik

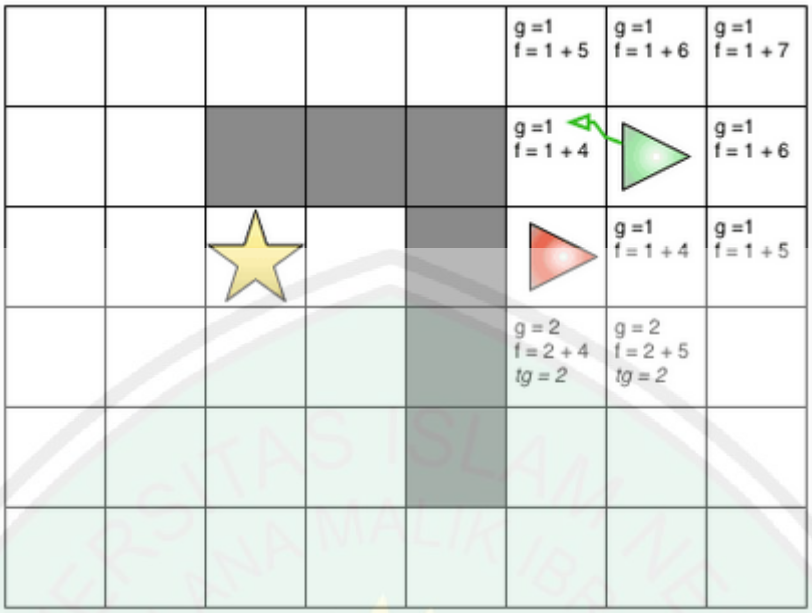
Pada gambar 3.4 di atas , segitiga adalah posisi awal (starting point) dan bintang adalah goal,untuk langkah pertamadari algoritma adalah, menemukan semua titik sekitar yang bisa dilakukan pergerakan,kemudian lakukan perhitungan skor masing g dan f.

Titik yang memiliki nilai f rendah seperti yang di tunjukkan pada arah panah,pada step ini terdabat kandidate yang mana skor g pada saat titik sekarang adalah lebih besar dari pada g skore pada tes pertama, jadi tidak ada perubahan



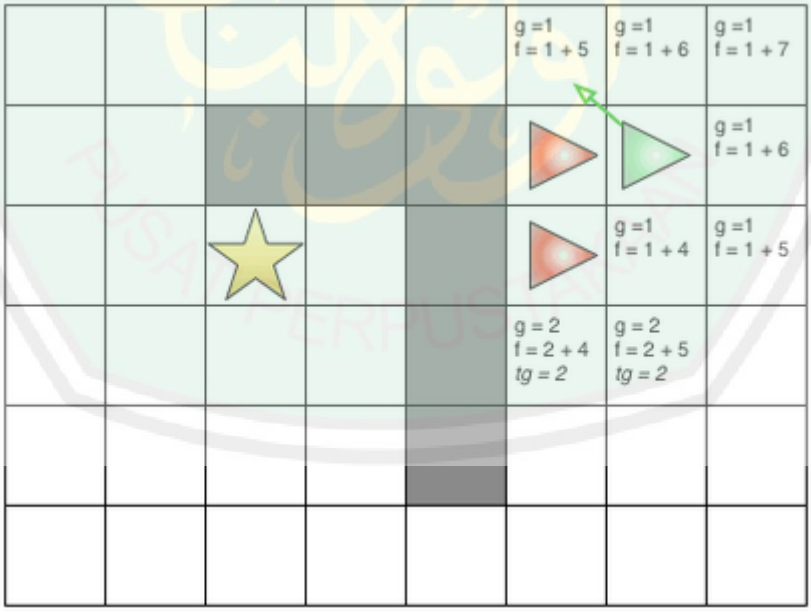
Gambar 3. 7 Langkah Pertama

Langkah selanjutnya adalah mengambil skor terendah pada titik *open set* . lihat gambar 3.6 , segitiga berwarna merah adalah menunjukkan bah wa pada *closed set* saat ini hanya terdapat 1



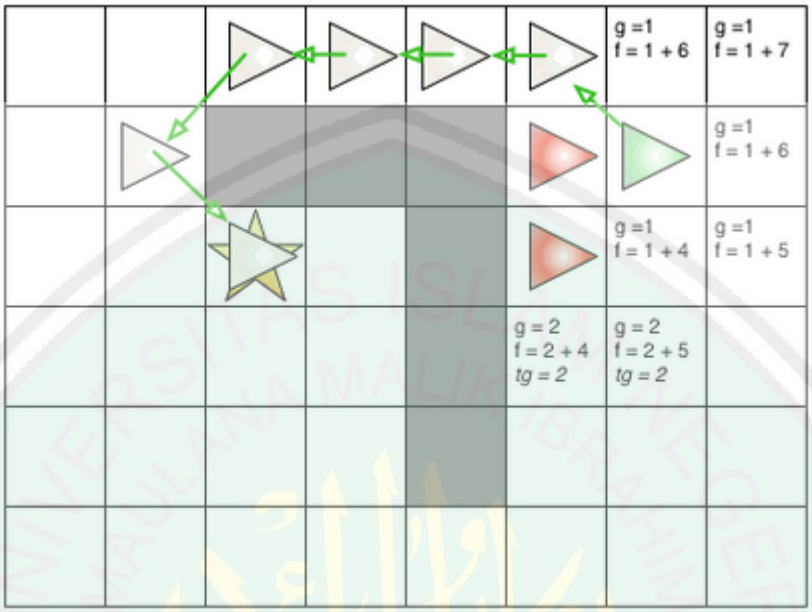
Gambar 3. 8 *Closed list* saat ini

Setelah memprosesnya , lanjut dengan mencari kembali titik dengan skor f terendah.



Gambar 3. 9 Pencarian F Terendah dan Pergerakan Objek

Selanjutnya hanya meneruskan proses yang telah dilakukan secara berulang-ulang hingga sampai pada tujuan.

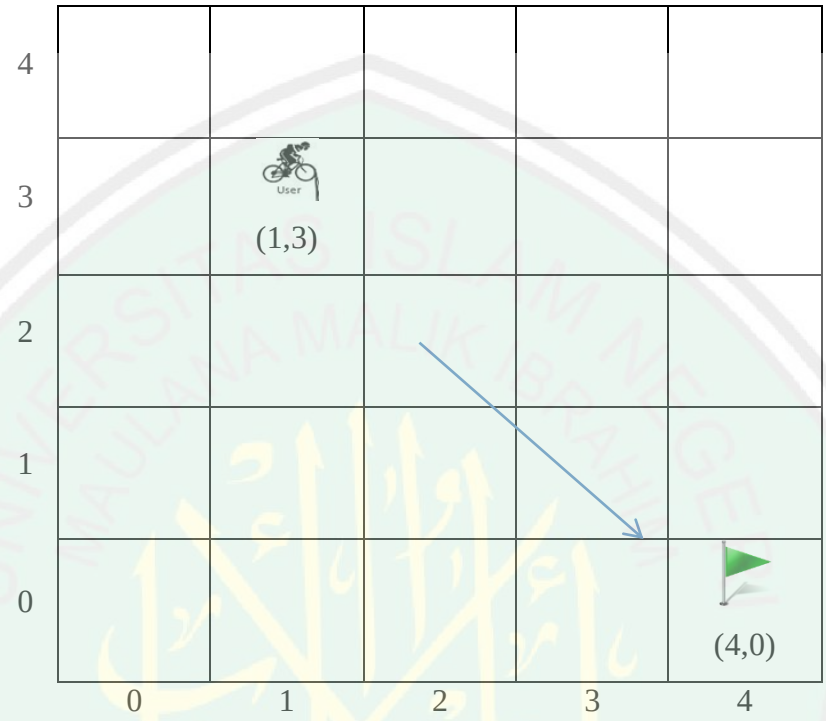


Gambar 3. 10 Finish

- Contoh penerapan pada *player* :

  Terdapat beberapa hal yang perlu didefinisikan terlebih dahulu dalam pathfinding dengan penerapan algoritma A* (A Star).seperti path, open list, closed list, nilai f, g dan n.Algoritma A* menggunakan dua List yaitu OPEN dan CLOSED. OPEN menyimpan simpul-simpul yang pernah dibangkitkan dan nilai heuristiknya telah dihitung tetapi belum terpilih sebagai simpul terbaik . CLOSED menyimpan simpul-simpul yang sudah pernah dibangkitkan dan sudah pernah terpilih sebagai simpul terbaik.pada saat start game pertama kali, posisi awal antara *player* dan obstacle sudah di inisiasikan, sehingga kita mengetahui node awal dan node akhir

Contoh kasus : *player* pada gambar di bawah ini akan menuju ke arah bendera (lihat Gambar 12).



Gambar 3. 11 Ilustrasi *Player* ke bendera

Dapat diketahui posisi start(n) pada node (1,1) dan end(goal) pada (3,3)

- **Input** nX=1 , nY=3 ,goalX = 4, goalY=0
- **Proses :** f(n) = g(n) + h(n)

g(n) = biaya (cost) yang dikeluarkan dari keadaan awal sampai keadaan n

h(n) = estimasi biaya untuk sampai pada suatu tujuan mulai dari n

g(n) = abs(startX - nX) + abs(startY - nY)

xdist = math.abs(curnode.x – goalnode.x)

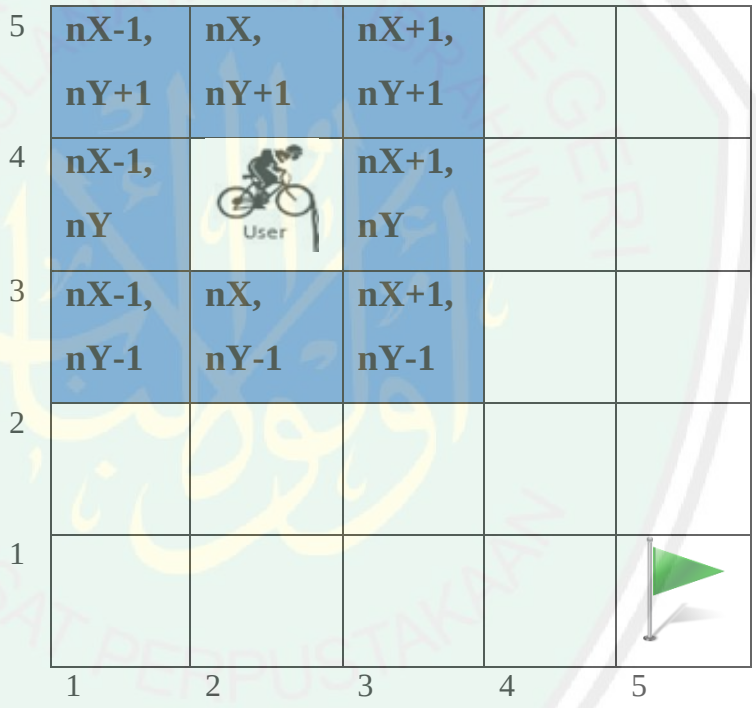
Ydist = math.abs(curnode.z - goalnode.z)

If (xdist > zdist) ,h(n) = 1.4*zdist + (xdist - zdist);

Else , h(n) = 1,4* xdist + (zdist - xdist);

Untuk mengetahui open list

- Kanan (nX+1,nY)
- Kiri (nX-1,nY)
- Atas (nX,nY-1)
- Bawah (nX,nY-1)
- Diagonal kiri atas (nX-1,nY+1)
- Diagonal kiri bawah (nX-1,nY-1)
- Diagonal kanan atas (nX+1,nY+1)
- Diagonal kanan bawah (nX+1,nY-1)



Gambar 3. 12 Open list pada kordinat (1,3)

Perhitungan manual algoritma A*

(1) Kondisi awal : goal (5,1),*player* (2,4)

Masukkan koordinat *player* ke OPEN

(2) Hitung semua *node* terdekat yang mungkin dilewati

Langkah 1 :

Arah kanan (3,4)

```
g(3,4)=1;
xdist = abs(3-5) = 2;
zdist = abs(4-1) = 3;
h(n) =1,4* xdist + (zdist - xdist);
h(n) = 1,4* 2 + (3-2) = 3,8;
f (3,4)  =  g (3,4) + h (3,4) =  1 + 3,8 =  4,8
```

Masukkan nilai f(1) arah kanan(3,4) ke OPEN list

Arah kiri (1,4)

```
g(1,4)=1;
xdist = abs(1-5) = 4;
zdist = abs(4-1) = 3;
h(n) =1,4* zdist + (xdist - zdist);
h(n) = 1,4* 3 + (4-3) = 5,2;
f (1,4)  =  g (1,4) + h (1,4) =  1 + 5,2 =  6,2
```

Masukkan nilai f(1) arah kiri(1,4) ke OPEN list

Arah atas (2,5)

```
g(2,5)=1;
xdist = abs(2-5) = 3;
zdist = abs(5-1) = 4;
h(n) =1,4* xdist + (zdist - xdist);
h(n) = 1,4* 3 + (4-3) = 5,2;
f (2,5)  =  g (2,5) + h (2,5) =  1 + 5,2 =  6,2
```

Masukkan nilai f(1) arah atas (2,5) ke OPEN list

Arah bawah (2,3)

```
g(2,3)=1;
xdist = abs(2-5) = 3;
zdist = abs(3-1) = 2;
```

```
h(n) =1,4* ydist + (xdist - zdist);
h(n) = 1,4* 2 + (3-2) = 3,8;
f (3,4)  =  g (3,4) + h (3,4) =  1 + 3,8 =  4,8
```

Masukkan nilai f(1) arah bawah (2,3) ke OPEN list

Arah kanan atas (3,5)

```
g(3,5)=1;
xdist = abs(3-5) = 2;
zdist = abs(5-1) = 4;
h(n) =1,4* xdist + (zdist - xdist);
h(n) = 1,4* 2 + (4-2) = 4,8;
f (3,5)  =  g (3,5) + h (3,5) =  1 + 4,8 =  5,8
```

Masukkan nilai f(1) arah kanan atas (3,5) ke OPEN list

Arah kanan bawah (3,3)

```
g(3,3)=1;
xdist = abs(3-5) = 2;
zdist = abs(3-1) = 2;
h(n) =1,4* xdist + (zdist - xdist);
h(n) = 1,4* 2 + (2-2) = 2,8;
f (3,3)  =  g (3,3) + h (3,3) =  1 + 2,8 =  2,8
```

Masukkan nilai f(1) arah kanan bawah (3,3) ke OPEN list

Arah kiri atas (1,5)

```
g(1,5)=1;
xdist = abs(1-5) = 4;
zdist = abs(5-1) = 4;
h(n) =1,4* xdist + (zdist - xdist);
h(n) = 1,4* 4 + (4-4) = 5,6;
f (1,5)  =  g (1,5) + h (1,5) =  1 + 5,6 =  6,6
```

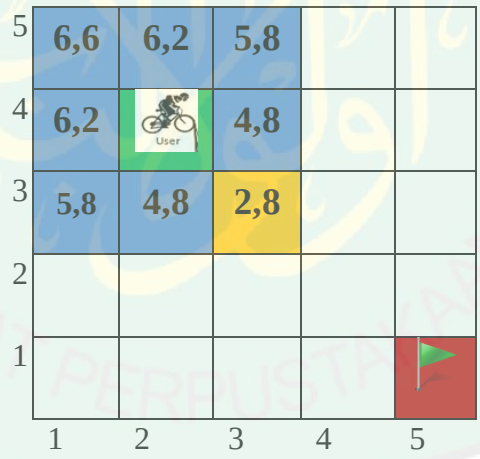
Masukkan nilai f(1) arah kiri atas (1,5) ke OPEN list

Arah kiri bawah (1,3)

```
g(1,3)=1;
xdist = abs(1-5) = 4;
zdist = abs(3-1) = 2;
h(n) =1,4* zdist + (xdist - zdist);
h(n) = 1,4* 2 + (4-2) = 4,8;
f (1,3)  =  g (1,3) + h (1,3) =  1 + 4,8 =  5,8
```

```
Masukkan nilai f(1) arah kiri bawah (3,4) ke OPEN list
```

Setelah menghitung semua node yang bisa dilewati, simpan nilai terkecil pada CLOSE List , dari perhitungan di atas nilai terkecil adalah pada node arah kanan bawah (3,3), pindah kan kordinat sekarang dengan node arah kanan bawah (3,3).



Gambar 3. 13 Nilai tiap Open List pada kordinat (2,4)

Langkah 2 : *player* (3,3) , Goal (5,1)

Arah kanan (4,3)

```
g(4,3)=1;
xdist = abs(4-5) = 1;
zdist = abs(3-1) = 2;
```

```
h(n) =1,4* xdist + (zdist - xdist);
h(n) = 1,4* 1 + (2-1) = 2,4;
f (4,3)  =  g (4,3) + h (4,3) =  1 + 2,4 =  3,4
```

Masukkan nilai f(1) arah kanan(4,3) ke OPEN list

Arah kiri (2,3)

```
g(2,3)=1;
xdist = abs(2-5) = 3;
zdist = abs(3-1) = 2;
h(n) =1,4* zdist + (xdist - zdist);
h(n) = 1,4* 2 + (3-2) = 3,8;
f (2,3)  =  g (2,3) + h (2,3) =  1 + 3,8 =  4,8
```

Masukkan nilai f(1) arah kiri(2,3) ke OPEN list

Arah atas (3,4)

```
g(3,4)=1;
xdist = abs(3-5) = 2;
zdist = abs(4-1) = 3;
h(n) =1,4* xdist + (zdist - xdist);
h(n) = 1,4* 2 + (3-2) = 3,8;
f (3,4)  =  g (3,4) + h (3,4) =  1 + 3,8 =  4,8
```

Masukkan nilai f(1) arah atas (3,4) ke OPEN list

Arah bawah (3,2)

```
g(3,2)=1;
xdist = abs(3-5) = 2;
zdist = abs(2-1) = 1;
h(n) =1,4* ydist + (xdist - zdist);
h(n) = 1,4* 1 + (2-1) = 2,4;
f (3,4)  =  g (3,4) + h (3,4) =  1 + 2,4 =  3,4
```

Masukkan nilai f(1) arah bawah (3,2) ke OPEN list

Arah kanan atas (3,5)

```
g(3,5)=1;
xdist = abs(3-5) = 2;
zdist = abs(5-1) = 4;
h(n) =1,4* xdist + (zdist - xdist);
h(n) = 1,4* 2 + (4-2) = 4,8;
f (3,5)  =  g (3,5) + h (3,5) =  1 + 4,8 =  5,8
```

Masukkan nilai f(1) arah kanan atas (3,5) ke OPEN list

Arah kanan bawah (4,2)

```
g(4,2)=1;
xdist = abs(4-5) = 1;
zdist = abs(2-1) = 1;
h(n) =1,4* xdist + (zdist - xdist);
h(n) = 1,4* 1 + (1-1) = 1,4;
f (4,2)  =  g (4,2) + h (4,2) =  1 + 1,4 =  2,4
```

Masukkan nilai f(1) arah kanan bawah (4,2) ke OPEN list

Arah kiri atas (2,4)

```
g(2,4)=1;
xdist = abs(2-5) = 3;
zdist = abs(4-1) = 3;
h(n) =1,4* xdist + (zdist - xdist);
h(n) = 1,4* 3 + (3-3) = 4,2;
f (2,4)  =  g (2,4) + h (2,4) =  1 + 4,2 =  5,2
```

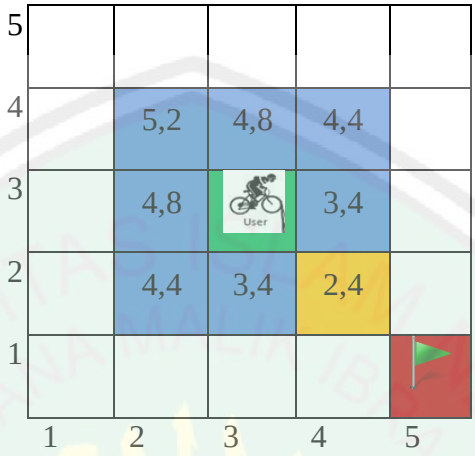
Masukkan nilai f(1) arah kiri atas (2,4) ke OPEN list

Arah kiri bawah (2,2)

```
g(2,2)=1;
xdist = abs(2-5) = 3;
zdist = abs(2-1) = 1;
h(n) =1,4* zdist + (xdist - zdist);
```

```
h(n) = 1,4* 1 + (3-1) = 3,4;
f (1,3)  =  g (1,3) + h (1,3) =  1 + 3,4 =  4,4
```

```
Masukkan nilai f(1) arah kiri bawah (3,4) ke OPEN list
```

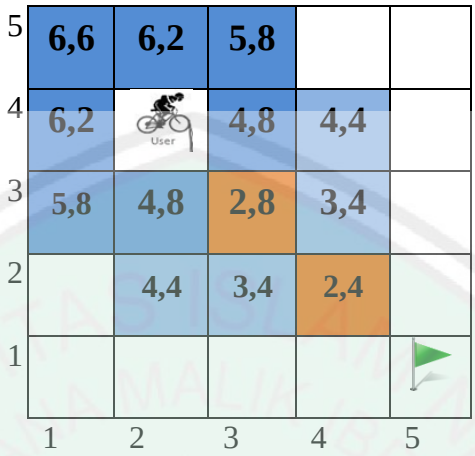


Gambar 3. 14 Nilai tiap Open List pada kordinat (3,3)

- Menambahkan Openlist yang sudah di bangkitkan kedalam list
- Menambahkan Openlist ke dalam Closelist
- Apabila ada list yang sama, maka di bandinngkan nilai heuristiknya, jika lebih kecil dari yang lama , maka list baru dari openlist akan di tambahkan, apabila tidak maka di biarkan.
- Mencari bestnode dari Closelist.
- Menambahkan bestnode dari closelist,kedalam bestnodelist.

- **Output :**

  Bestnode list {kanan bawah (3,3), kanan bawah (4,2)}



Gambar 3. 15 Besnode nerwarna orange dari *player* (1,3) dengan Goal (4,0)

BAB IV
HASIL DAN PEMBAHASAN

Dalam bab ini akan di bahas mengenai implementasi dari hasil perancangan sistem yang telah di buat, berikut adalah penjelasan dari implementasinya.

4.1. Implementasi Antarmuka

Pada sub bab implementasi antarmuka ini akan di jelaskan komponen – komponen yang ada pada Game Gowes.

4.1.1 Splash screen

Adalah tampilan awal ketika pertama kali membuka game, tampilan splash berdurasi dalam hitungan detik, berisi judul game dan logo UIN malang dengan sedikit animasi transisi. Ilustrasi dapat di lihat pada gambar 3.1

Gambar 4. 1 *Splash Screen*

4.1.2 Main Menu

Adalah menu utama sebelum melakukan permainan , terdapat beberapa pilihan diantaranya sebagai berikut:

Gambar 4. 2 Menu Utama

1. Play

   Tombol untuk memulai game

2. Option

   Menu untuk beberapa pilihan yang telah disediakan, diantaranya :

   a. Control

      Tombol ini Berisi halaman Kontrol pada game yang mana *player* dapat mengetahui control yang ada pada game.

   b. Credit

      Tombol ini Berisi tentang hal hal seputar pembuat game ini

   c. Back

      Tombol ini akan membawa Kembali ke menu utama

3. Exit

   Tombol untuk keluar dari game

4.1.3 Game

Pada tampilan game ini menampilkan beberapa item diantaranya :

Gambar 4. 3 Tampilan permainan Game

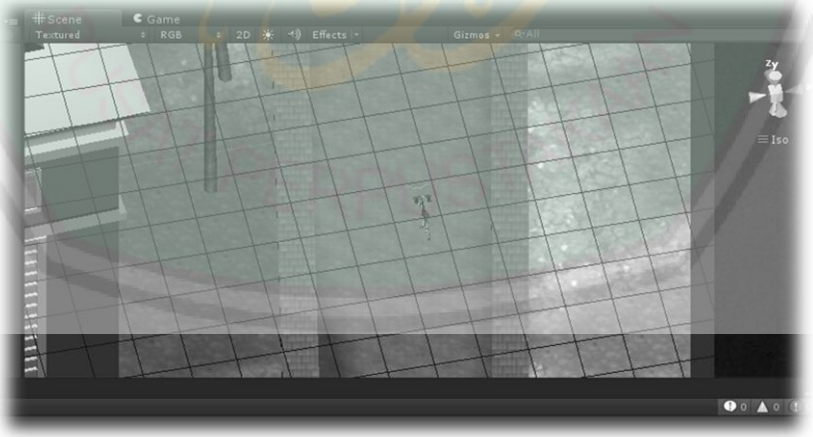
1. Spedo meter : Menampilkan kecepatan sepeda
2. Score : Menampilkan score yang di dapat
3. Timer : Menampilkan waktu yang tersisa menjalankan suatu misi
4. Panah : Menunjukkan Arah , berdasarkan path yang sudah di proses
5. Point pickup : Berupa kubus yang apabila di koleksi / berbenturan pada sepeda , akan menambah score
6. Minimaps : Berisi map kecil , yang menampilkan jalan rute jalan.

4.2. Implementasi Sistem Game

Sistem game yang akan di implementasikan pada game sepeda ini menggunakan bahasa pemrograman C# dan javascript sebagai core dari game, dan menggunakan editor monodevelop. Sedangkan desain visual dan interface mengunakan *Game engine*.

4.2.1 Pengaturan *Grid*

Dalam suatu game simulasi dibutuhkan yang namanya Map, di dalam map kita dapat mengetahui koodrinat dari suatu objek,luas suatu map sangat berpengaruh terhadap kinerja algoritma astar, jika semakin luas map maka semakin berat proses pencariannya, untuk menangani hal itu maka kita perlu membuat sebuah pengaturan yang di sebut *grid*, yaitu dengan mengkotak kotakkan suatu map yang bisa kita atur jumlah dan skalanya, sehingga dapat mempermudah jalanya proses algoritma astar.



Gambar 4. 4 Grid pada area game yang disesuaikan dengan skala

Pada gambar di atas (Gambar 4.4) terlihat grid pada waktu debugging mode.besarnya skala pada grid akan mempengaruhi jumlah dari pada matrix grid

yang ada pada maps,dan jumlah grid akan berpengaruh terhadap proses pencarian, karena semakin banyak grid akan semakun banyak proses pencarianya.

Berikut adalah potongan kode dari kelas *gridmanager.cs*

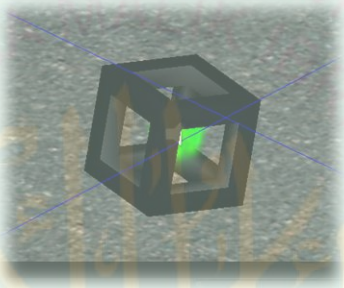
```
void OnDrawGizmos()
  {
      //Draw Grid
      if (showGrid)
      {
           DebugDrawGrid(transform.position, numOfRows,
          numOfColumns, gridCellSize, Color.blue);
      }
      //Grid Start Position
      Gizmos.DrawSphere(transform.position, 0.5f);
      //Draw Obstacle obstruction
      if (showObstacleBlocks)
      {
    Vector3 cellSize = new Vector3(gridCellSize, 1.0f,
    gridCellSize);

        if (obstacleList != null && obstacleList.Length > 0)
          {
              foreach (GameObject data in obstacleList)
                {
                Gizmos.DrawCube(GetGridCellCenter(GetGridInd
                ex(data.transform.position)), cellSize);
              }
          }
      }
  }
```

Dari potongan Source code di atas kita bisa melihat DebugDrawGrid adalah suatu prosedur untuk menampilkan grid pada saat debug, danjuga void *OnDrawgizmos()* adalah method yang di sediakan *Game engine* untuk keperluan

debugging, sehingga grid akan terlihat hanya pada saat kita pada debug mode, dan pada saat proses *running mode/building project* grid ini tidak tampak.

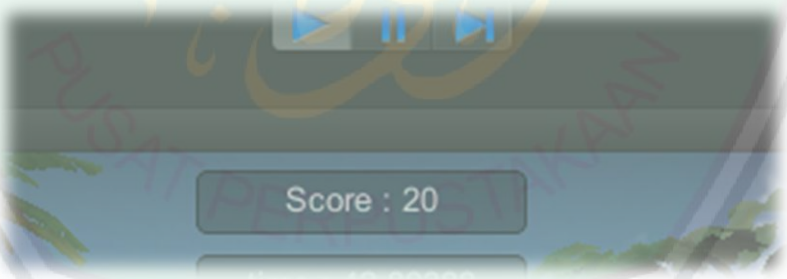
4.2.2 Pengaturan score

Game pada umumnya terdapat scoring, begitu juga pada game ini , proses penambahan *score* ini dilakukan pada saat sepeda berbenturan dengan objek yang bernama pickup, berbentuk kubus (Gambar4.5).



Gambar 4. 5 Pickup untuk point

Berikut adalah GUI dari Score yang ada pada game (Gambar 4.6)



Gambar 4. 6 Gui Score

Untuk implementasinya mengunakan beberapa kelas, diantaranya *pickupcontroller*.cs dan *cbpickup*.cs. berikut adalah baris kodenya :

a. Kelas Cbpickup.cs

```
#pragma strict
function OnTriggerEnter (info : Collider){
	if (info.tag == "Sepeda"){
	updateScore.currentscore += 10;
	//yield  WaitForSeconds(5);
	Destroy(gameObject);
	}
}
```

Kelas di atas menangani proses *scoring* pada game , apabila objek yang menyentuh adalah sepeda maka poin bertambah.

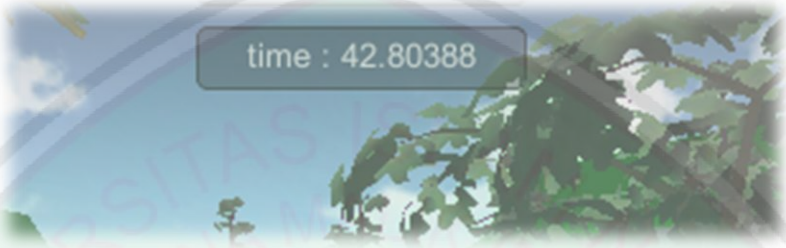
b. Kelas Pickupcontroller.cs

```
function Awake()
	{
	spawnPointList =
	gameObject.FindGameObjectsWithTag("SpawnPoint");
	numberOfSpawnPoints = spawnPointList.length;

		if (numberOfPickups > numberOfSpawnPoints)
	numberOfPickups = numberOfSpawnPoints;
	for (var i:int = 0;  i < numberOfSpawnPoints; i++)
	{
		spawnIndexAvailableList[i] = true;
	}
	for (var j:int = 0;  j < numberOfPickups; j++)
	{
		SpawnPickup();
	}
`
```

Kelas diatas menangani spawning dari kubus,sehingga kubus bisa muncul secara otomatis pada game.

4.2.3 Pengaturan Timer

Pada game ini suatu misi dibatasi oleh waktu, sehingga *player* bermain berdasarka waktu yang di tentukan , wakyu yang di sediakan di hitung mundur dan di tampilkan pada layar seperti (gambar 4.6)



Gambar 4. 7 Gui Timer

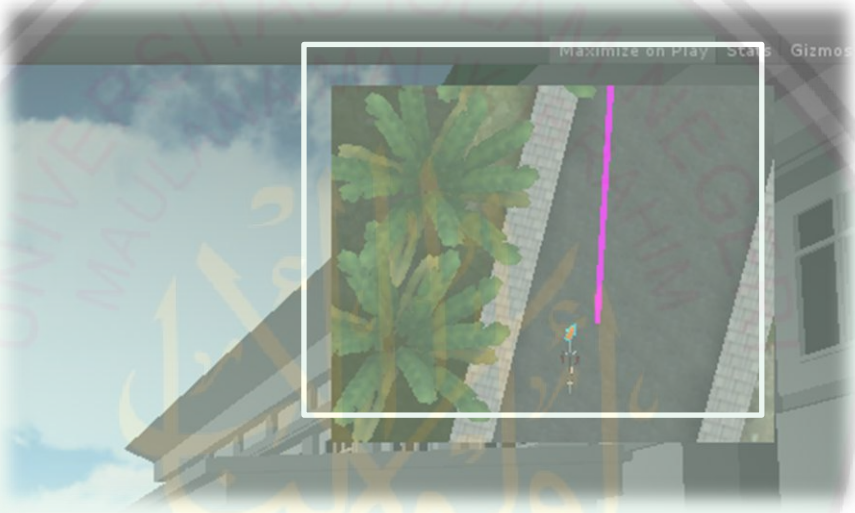
Berikut adalah implementasi kedalam kode C dalam kelas *coundowntimer.cs:*

```
      public float currentTime = 90;
      public float offsety     = 40;
      public float sizex       = 100;
      public float sizey       = 40;
      public string nextLevelToLoad   ;
      // Use this for initialization

void FixedUpdate ()
      {
      if (currentTime <= 0) {
         AutoFade.LoadLevel(nextLevelToLoad,1,1,Color.white);
            }
            currentTime -= Time.deltaTime;
      }
      void OnGUI(){
            GUI.Box(new Rect(Screen.width/2-sizex/2,offsety,
      sizex,sizey),"time : " +currentTime );
      }
}
```

Dari potongan kode di atas , sekilas kita mengetahui dapat mengatur watu yang di tentukan melalui variabel *currentTime.* Apabila kita melebihi waktu yang di tentukan maka secara otomatis game akan meninggalkan levelnya, ke level yang sudah di tentukan , atau permainan akan berakhir.

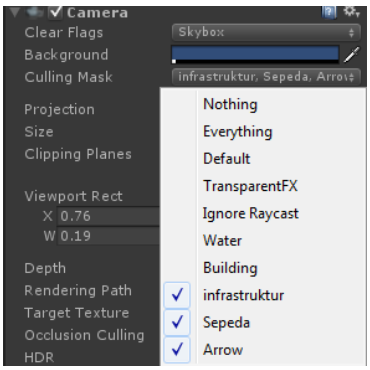
4.2.4 *Mini Map*

Mini map adalah Map kecil yang di tempatkan pada kanan atas layar yang berfunsi sebagai komponen pembatu navigasi , yang nantinya berisi garis garis rute yang akan di lalui *player*. Mini map ini akan mengikuti kemanapun arah pergerakan *player* dan juga memberikan arah setiap pergerakannya secara realtime. Berikut adalah tampilan minimap pada layar game (Gambar 4.7)



Gambar 4. 8 Tampilan Mini Map

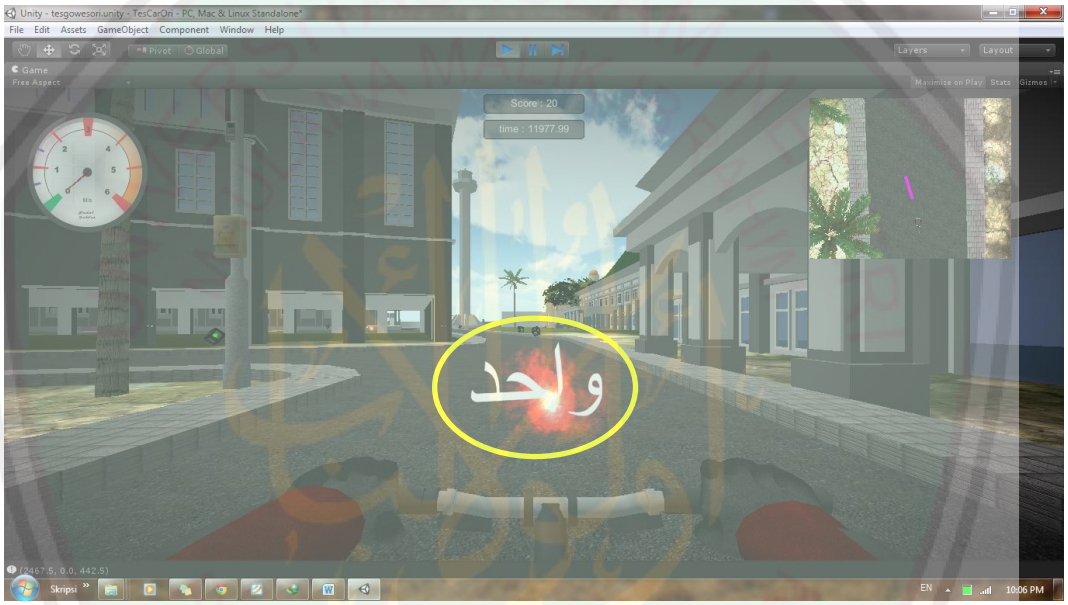
Pembuatan Mini map menggunakan *Second Camera*, yang di tempatkan pada atas *player* sehingga dapat terlihat lingkungan sekitar *player*, untuk pengaturan objek apa saja yang hanya bisa di lihat oleh mini map adalah layer infrastruktur, sepeda, dan arrow yang dapat dilihat pada gambar



Gambar 4. 9 Layer pada kamera

4.2.5 *Obstacle*

Obstacle adalah halangan atau rintangan, yang dalam game ini menjadi misi utama, yakni mencari halangan berupa kata berbahasa arab dengan mengurutkan angka arab mulai 1 – 7, Obstacle dibuat dengan warna mencolok dan berbeda dengan yang lainnya agar mudah dilihat ,seperti pada gambar di bawah ini.



Gambar 4. 10 *Obstacle*

Gambar di atas adalah obstacle pertama yang beruliskan "*wahid*" yang berarti angka pertama(gambar di lingkari), Player akan di arahkan rute menuju *Obstacle* dari angka satu sampai tujuh ,dengan map kecil yang ada pada pojok kanan atas. *Obstacle* ini di susun secara berurutan, sehingga player tidak bisa mengambilnya secara acak, dan misi akan selesai jika player sudah mengumpulkan sebanyak 7 *Obstacle* dalam waktu yang sudah di tentukan.

4.3 Implementasi *Astar*

Pada game sepeda ini implementasi algorima astar digunakan pada sistem navigasi yang ada pada sepeda berupa anak panah , dan juga pada minimap yang ada pada pojok layar menunjukkan rute yang akan di lalui.

4.3.1 Pembangkitan Node

Pada algoritma Astar terdapat proses untuk membangkitkan simpul yang berada di sekitar *player* yang nantinya berfungsi untuk menghitung *cost* jalan yang akan di lalui *player*. Dalam pembangkitan ini terdapat 8 arah yang di ilustrasikan pada (gambar 4.5).

3	**nX-1, nY+1**	**nX, nY+1**	**nX+1, nY+1**
2	**nX-1, nY**		**nX+1, nY**
1	**nX-1, nY-1**	**nX, nY-1**	**nX+1, nY-1**
	1	2	3

Gambar 4. 11 Node sekitar

Untuk implementasinya menggunakan kelas GridManager.cs untuk mempermudah pengorganisasian kelas, untuk baris kodenya adalah sebagai berikut :

```
int row = GetRow(neighborIndex);
int column = GetColumn(neighborIndex);

      //Top
      int leftNodeRow = row + 1;
      int leftNodeColumn = column;
      AssignNeighbour(leftNodeRow, leftNodeColumn, neighbors);

      //diagonal kakan atas
      leftNodeRow = row + 1;
      leftNodeColumn = column + 1;
      AssignNeighbour(leftNodeRow, leftNodeColumn, neighbors);

      //Right
      leftNodeRow = row;
      leftNodeColumn = column + 1;
      AssignNeighbour(leftNodeRow, leftNodeColumn, neighbors);

      //diagonal kanan bawah
      leftNodeRow = row - 1;
      leftNodeColumn = column + 1;
      AssignNeighbour(leftNodeRow, leftNodeColumn, neighbors);

      //Bottom
      leftNodeRow = row - 1;
      leftNodeColumn = column;
      AssignNeighbour(leftNodeRow, leftNodeColumn, neighbors);

      //diagonal kiri bawah
      leftNodeRow = row - 1;
      leftNodeColumn = column - 1;
      AssignNeighbour(leftNodeRow, leftNodeColumn, neighbors);

      //Left
      leftNodeRow = row;
      leftNodeColumn = column - 1;
      AssignNeighbour(leftNodeRow, leftNodeColumn, neighbors);

      //diagonal kiri atas
      leftNodeRow = row + 1;
      leftNodeColumn = column - 1;
      AssignNeighbour(leftNodeRow, leftNodeColumn, neighbors);
```

Potongan kode di atas adalah implementasi dari pembangkitan node dengan 8 arah yaitu kanan, kiri, atas, bawah, diagonal kanan atas, diagonal kiri atas, diagonal kanan bawah, diagonal kiri bawah.

4.3.2 Fungsi Heuristik

Di dalam algoritma astar terdapat fungsi heuristic, dengan funsi ini proses astar dapat memberikan rute yang terbaik dengan cara menghitung tiap node , dan mengambil node dengan nilai terkecil. Fungsi ini di implementasikan pada kelas Astar.cs

```
  private static float HeuristicEstimateCost(Node curNode,
Node goalNode)
    {
      float xdistace = Mathf.Abs(curNode.position.x-
goalNode.position.x);
            float zdistace = Mathf.Abs(curNode.position.z-
goalNode.position.z);
            float h_n;
            if(xdistace>zdistace){
                  h_n=1.4f*zdistace+(xdistace-zdistace);
            }     else{
                  h_n = 1.4f*xdistace+(zdistace-xdistace);
            }
            return h_n;
      }
```

Method di atas adalah berfungsi untuk menghitung fungsi heuristic h(n) dari *player* menuju ke goal , untuk proses secara lngkap ada pada proses findpath di bawah ini :

```
public static ArrayList FindPath(Node start, Node goal){
        openList = new PriorityQueue();
        openList.Push(start);

        start.nodeTotalCost = 0.0f;
        start.estimatedCost = HeuristicEstimateCost(start,
goal);
        closedList = new PriorityQueue();
        Node node = null;
```

```
while (openList.Length != 0)

        {
            node = openList.First();

            if (node.position == goal.position)

            {
                return CalculatePath(node);
            }

            ArrayList neighbours = new ArrayList();

         GridManager.instance.GetNeighbours(node, neighbours);

            for (int i = 0; i < neighbours.Count; i++)

            {
                Node neighbourNode = (Node)neighbours[i];
                if (!closedList.Contains(neighbourNode))

                {
                   float cost = HeuristicEstimateCost(node,
            neighbourNode);
                  float totalCost = node.nodeTotalCost + cost;

      float neighbourNodeEstCost =
      HeuristicEstimateCost(neighbourNode, goal);
       neighbourNode.nodeTotalCost = totalCost;
       neighbourNode.parent = node;
             neighbourNode.estimatedCost = totalCost +
neighbourNodeEstCost;

                    if (!openList.Contains(neighbourNode))
                    {
                        openList.Push(neighbourNode);
                    }
                }
            }
            closedList.Push(node);
            openList.Remove(node);
        }

        //jika proses di atas sudah selesai tetapi tidak ada
goal maka return null
        if (node.position != goal.position)
        {
            Debug.LogError("Goal Not Found");
            return null;
        }

        //bestnode

        return CalculatePath(node);
    }
```

Fungsi algoritma astar ada pada method findpath, pada awal prosedur adalah inisialisasi *arraylist Openlist* dan *closelist,* kemudian memasukkan node *player* ke dalam *Openlist* , karena awal mula open list adalah kosong. Selanjutnya adalah mengecek apakah *openlist* kosong, jika openlist kosong maka ada kemungkinan tidak menemukan jalan , dan itu artinya tidak ada solusi. Apabila *openlist* terdapat data, maka lakukan proses pengecekan apakah *Node player* sama dengan node *goal,* jika ternyata sama / sudah mencapai goal , maka otomatis menjalankan *method calculatepath()* , isi dari *method* adalah membalik *arraylist* yang sudah berisi node pilihan .

```
private static ArrayList CalculatePath(Node node)
    {
        ArrayList list = new ArrayList();
        while (node != null)
        {
            list.Add(node);
            node = node.parent;
        }
        list.Reverse();
        return list;
    }
```

apabila posisi *player* tidak sama dengan goal maka proses perhitungan node akan di mulai dengan pengecekan *node neighbour* atau node yang ada di sekitar *player* untuk di lalui dan juga menghitung *cost* dari node tersebut, jika pada *Openlist* tidak ada node yang di cari tersebut maka node akan di masukkan dalam Openlist. Proses tersebut dilakukan secara terus menerus hingga menemukan goal. Apabila tidak menemukan *goal* maka memproses *null* , apabila menemui node goal maka *Method calculatepath* di jalankan.

4.4 Uji Coba

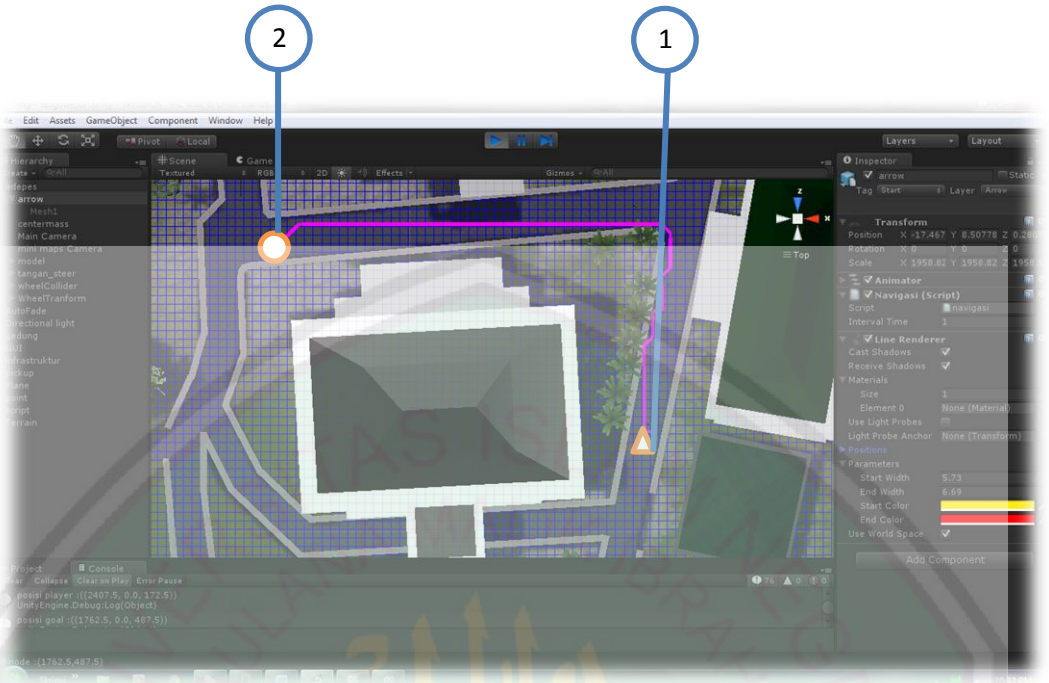
Untuk mengetahui sejauh mana implementasi algoritma terhadap sistem game sepeda ,maka perlu dilakukan pengujian. Pengujian dilakukan beberapa kali dengan memasang *player* dan goal dengan koordiant berbeda pada map. Uji coba dilakukan 3(tiga) kali dengan penempatan koordinat *player* dan goal yang berbeda dan di jalankan pada computer dengan spesifikasi sebagai berikut:

1. Prosesor intel Core2Duo T6570 @2.10 GHz
2. HardDisk 250 GB - 5600 RPM
3. RAM 4GB(2x2GB)
4. VGA IntelHD 45 series 1 Gb Shared memory
5. Keyboard
6. Monitor 14''

4.4.1 Hasil *Path*

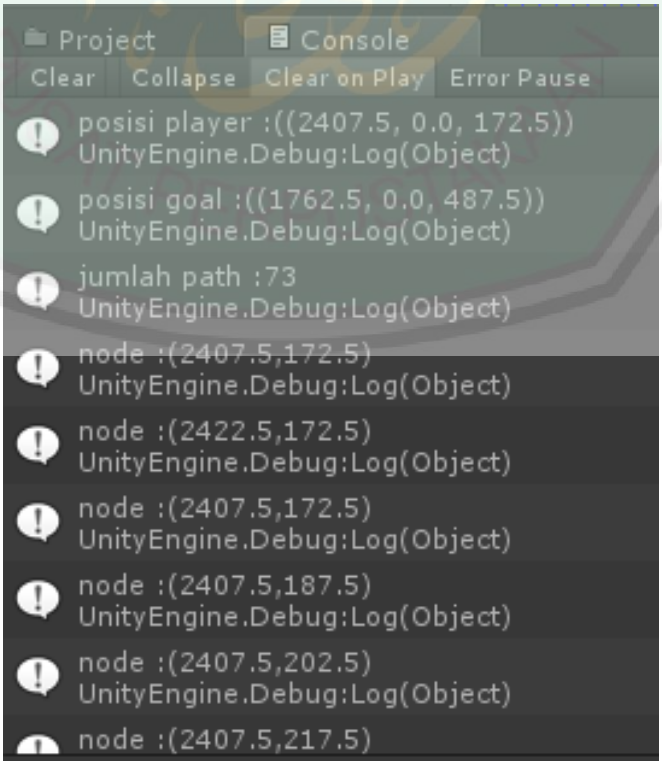
a. Pengujian pertama

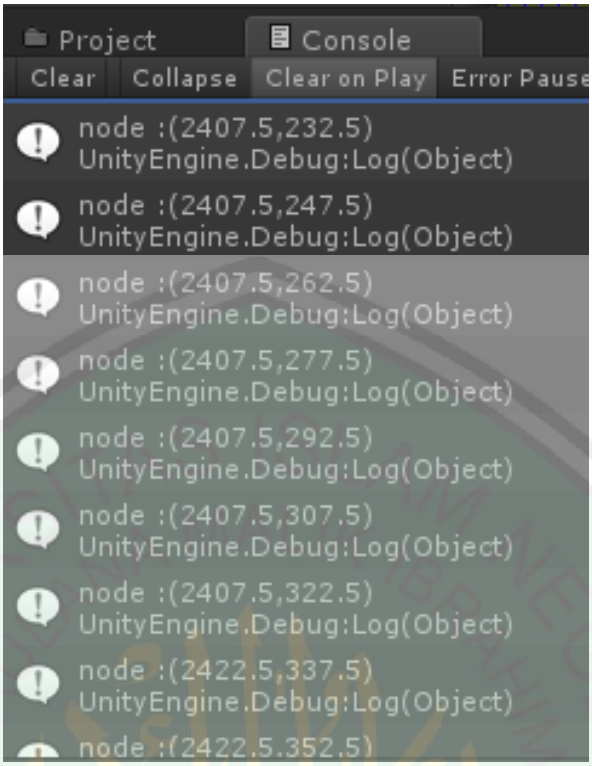
Pada pengujian pertama ini posisi awal *player* (pada gambar 4.11 nomor 1) berada pada koordinat (2407,172) dan goal (pada gambar 4.11 nomor 2) berada pada (1762,487), pada saat di jalankan keadaan *player* dalam keadaan diam / *standby*, hasil yang didapatkan berupa garis yang mangarahkan pada goal. Dan juga hasil path yang di tampilkan pada *console Game engine* yang berisi path yang di lalui oleh *player*. berikut adalah tampilan yang diambil debug mode . Gambar (4.11)

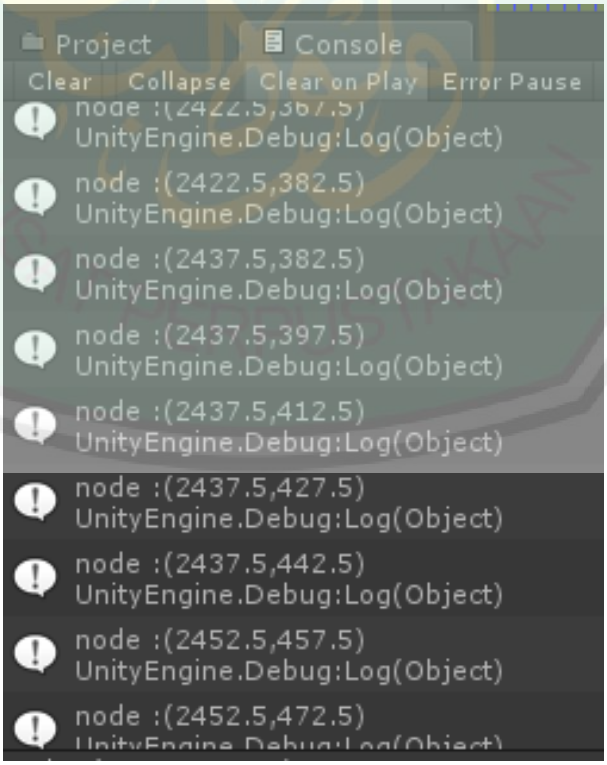


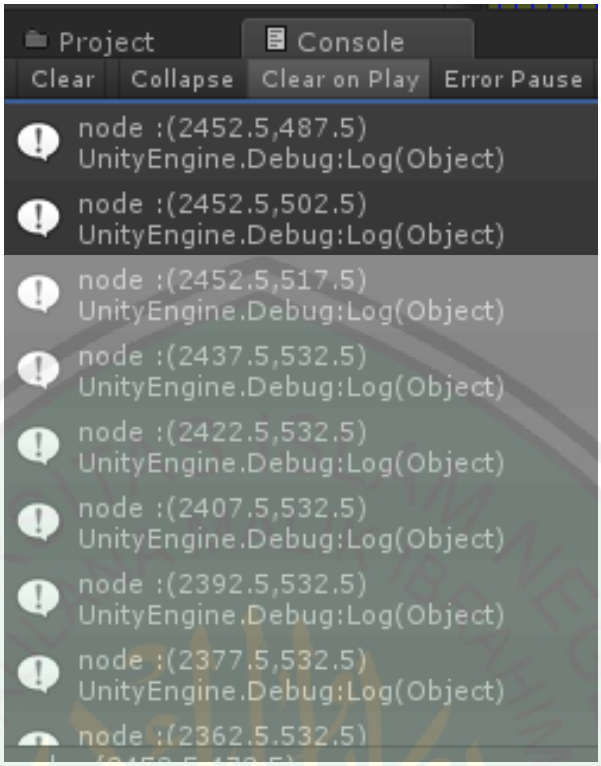
Gambar 4. 12 path pada pengujian pertama

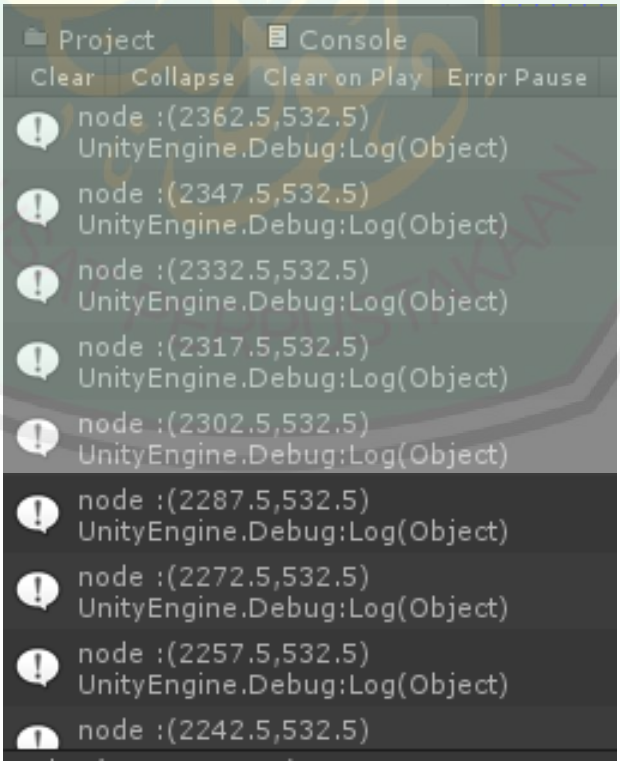
Dari hasil proses tersebut terdapat node terbaik hasil proses algoritma astar yang terdapat pada console, berikut adalah *screenshoot* dari *console:*

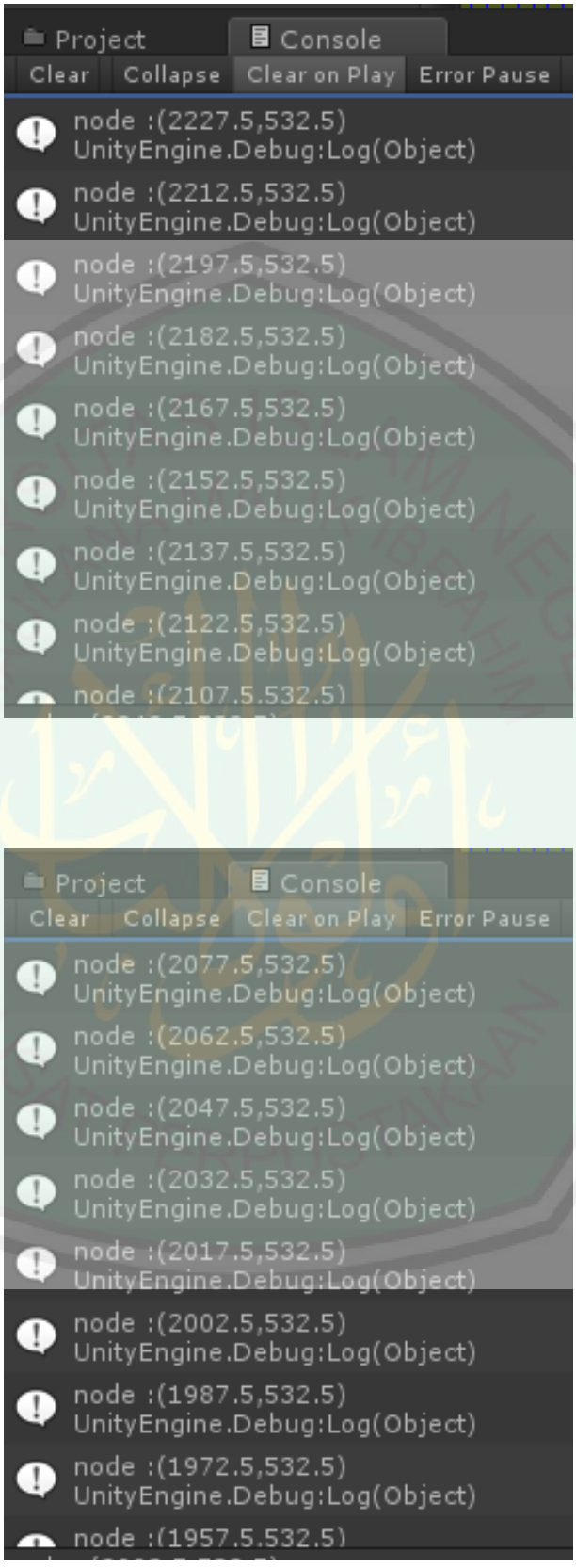


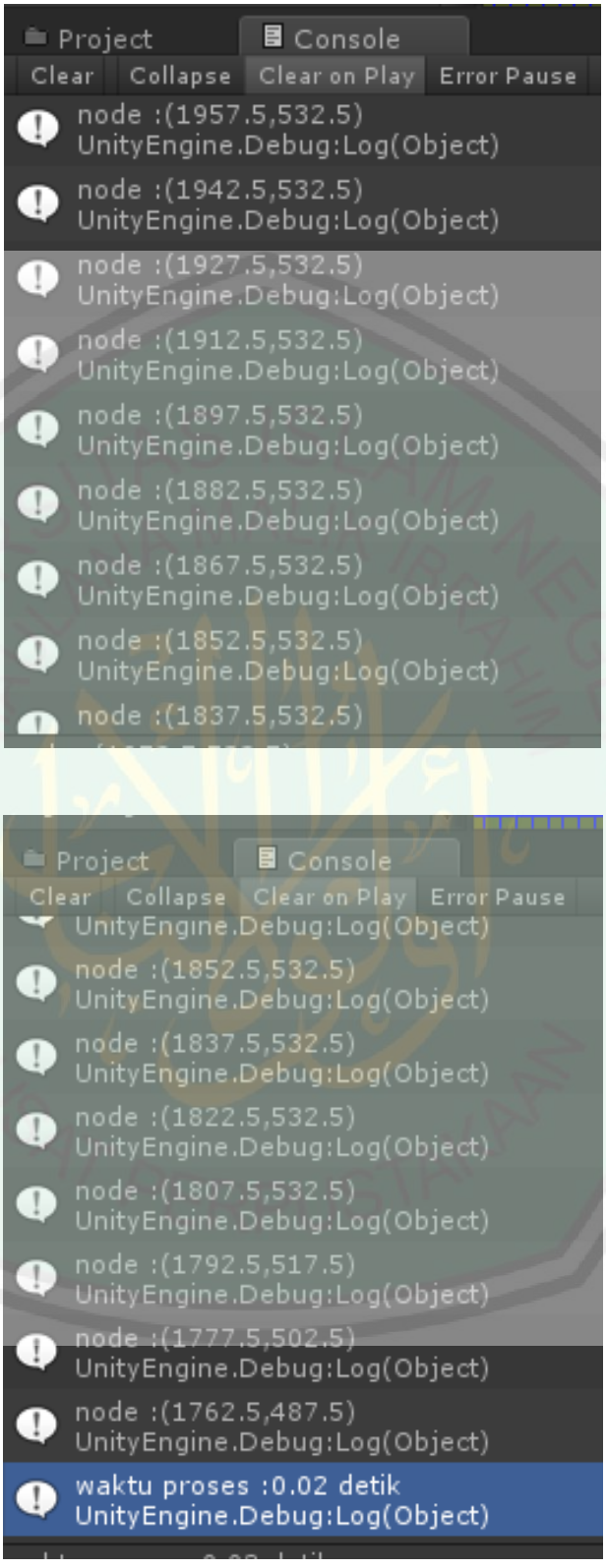












Gambar 4. 13 Path pada Console pada pengujian pertama

Berikut adalah rekapitulasi dari path yang akan di lalui *player* :

No	Nama item	Hasil	Keterangan
1	Posisi *player*	(2407.5, 172.5)	
2	Posisi objek	(1762.5, 487.5)	
3	Jumlah path	73 node	
4	Waktu proses	0.02 detik	
5	Path	node :(2407.5,172.5) node :(2422.5,172.5) node :(2407.5,172.5) node :(2407.5,187.5) node :(2407.5,202.5) node :(2407.5,217.5) node :(2407.5,232.5) node :(2407.5,247.5) node :(2407.5,262.5) node :(2407.5,277.5) node :(2407.5,292.5) node :(2407.5,307.5) node :(2407.5,322.5) node :(2422.5,337.5) node :(2422.5,352.5) node :(2422.5,367.5) node :(2422.5,382.5) node :(2437.5,382.5) node :(2437.5,397.5) node :(2437.5,412.5)	Menemukan solusi

		node :(2437.5,427.5) node :(2437.5,442.5) node :(2452.5,457.5) node :(2452.5,472.5) node :(2452.5,487.5) node :(2452.5,502.5) node :(2452.5,517.5) node :(2437.5,532.5) node :(2422.5,532.5) node :(2407.5,532.5) node :(2392.5,532.5) node :(2377.5,532.5) node :(2362.5,532.5) node :(2347.5,532.5) node :(2332.5,532.5) node :(2317.5,532.5) node :(2302.5,532.5) node :(2287.5,532.5) node :(2272.5,532.5) node :(2257.5,532.5) node :(2242.5,532.5) node :(2227.5,532.5) node :(2212.5,532.5) node :(2197.5,532.5) node :(2182.5,532.5) node :(2167.5,532.5) node :(2152.5,532.5) node :(2137.5,532.5) node :(2122.5,532.5) node :(2107.5,532.5) node :(2092.5,532.5) node :(2077.5,532.5)	

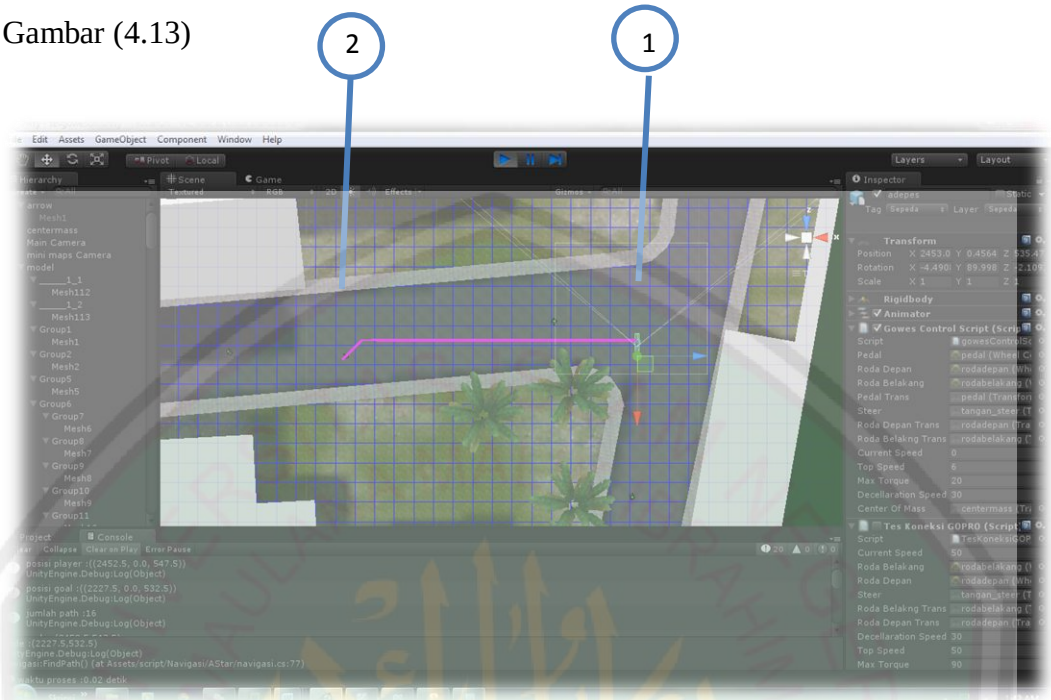
		node :(2062.5,532.5) node :(2047.5,532.5) node :(2032.5,532.5) node :(2017.5,532.5) node :(2002.5,532.5) node :(1987.5,532.5) node :(1972.5,532.5) node :(1957.5,532.5) node :(1942.5,532.5) node :(1927.5,532.5) node :(1912.5,532.5) node :(1897.5,532.5) node :(1882.5,532.5) node :(1867.5,532.5) node :(1852.5,532.5) node :(1837.5,532.5) node :(1822.5,532.5) node :(1807.5,532.5) node :(1792.5,517.5) node :(1777.5,502.5) node :(1762.5,487.5)	

Tabel 4. 1 Hasil Pengujian Pertama

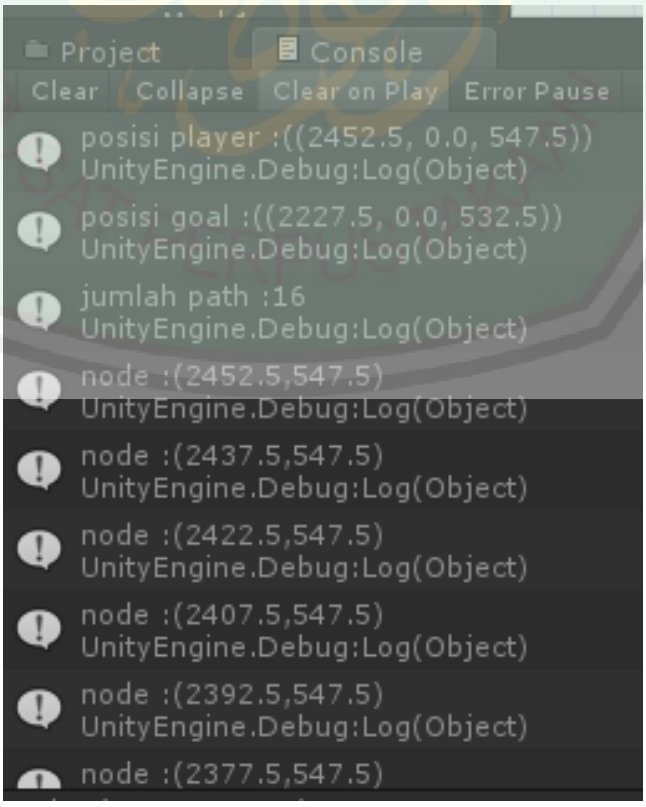
b. Pengujian kedua

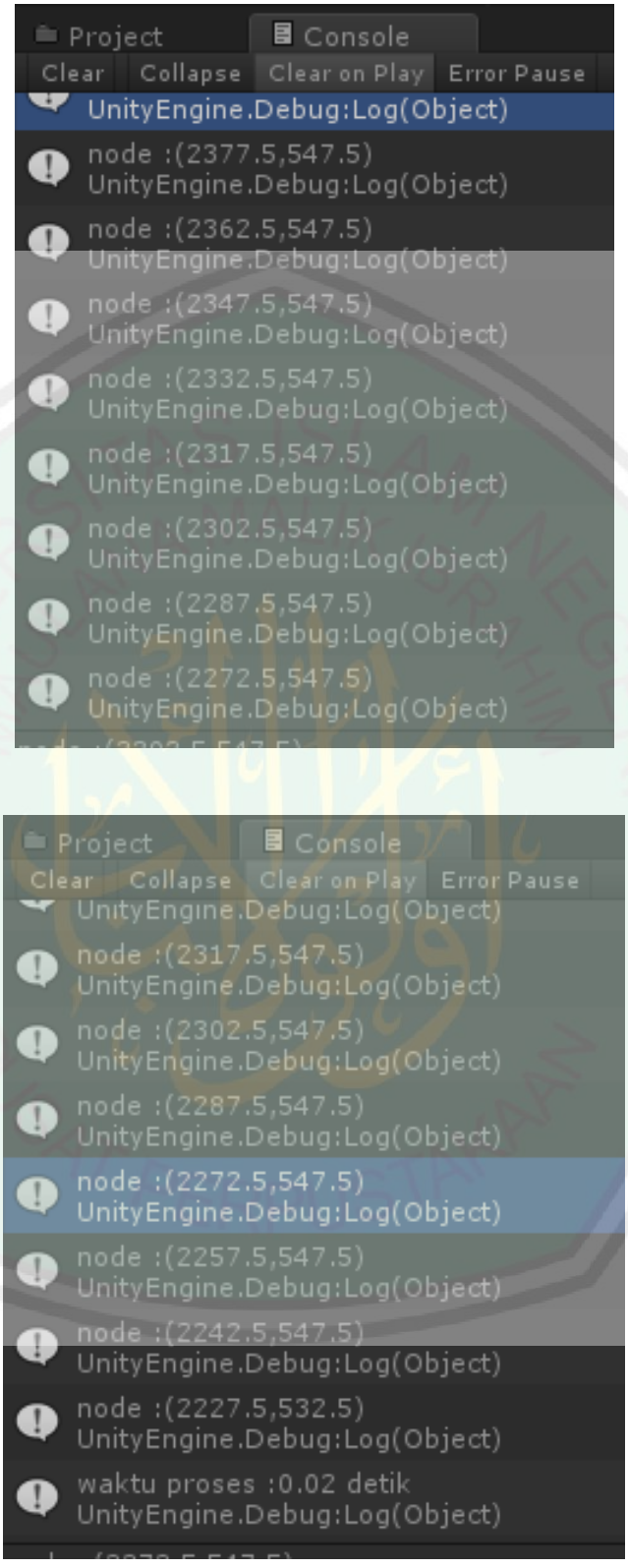
Pada pengujian kedua ini posisi awal *player* (pada gambar 4.13 nomor 1) berada pada koordinat (2452.5, 547.5) dan goal (pada gambar 4.13 nomor 2) berada pada (2227.5, 532.5), pada saat di jalankan keadaan *player* dalam keadaan diam / *standby,* hasil yang didapatkan berupa garis yang mangarahkan pada goal. Dan juga hasil path yang di tampilkan pada *console Game engine* yang berisi path

yang di lalui oleh *player*. berikut adalah tampilan yang diambil debug mode . Gambar (4.13)



Gambar 4. 14 path pada pengujian Kedua





Gambar 4. 15 Path pada Console pada pengujian kedua

Berikut adalah rekapitulasi dari path yang akan di lalui *player* :

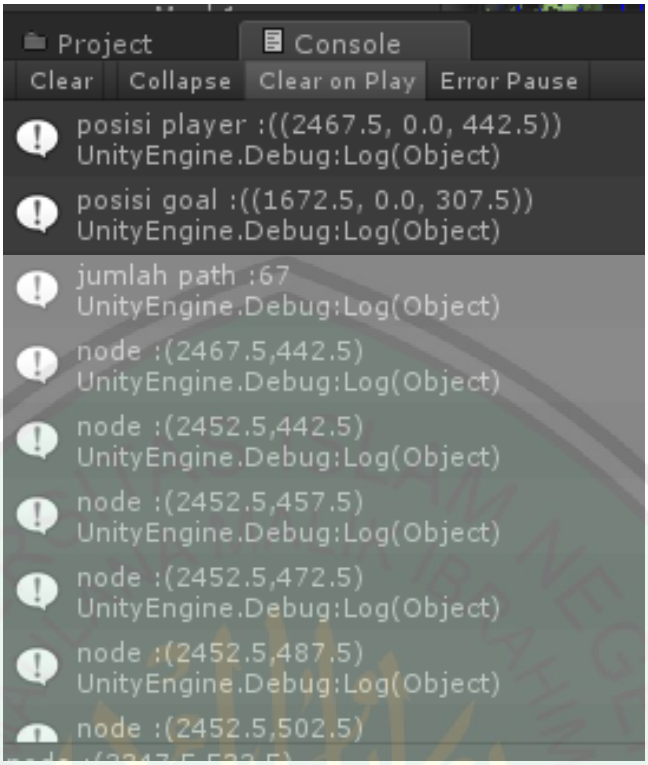
No	Nama item	Hasil	Keterangan
1	Posisi *player*	(2452.5, 547.5)	
2	Posisi objek	(2227.5, 532.5)	
3	Jumlah path	16 node	
4	Waktu proses	0.02 detik	
5	Path	node :(2452.5,547.5) node :(2437.5,547.5) node :(2422.5,547.5) node :(2407.5,547.5) node :(2392.5,547.5) node :(2377.5,547.5) node :(2362.5,547.5) node :(2347.5,547.5) node :(2332.5,547.5) node :(2317.5,547.5) node :(2302.5,547.5) node :(2287.5,547.5) node :(2272.5,547.5) node :(2257.5,547.5) node :(2242.5,547.5) node :(2227.5,532.5)	Menemukan solusi

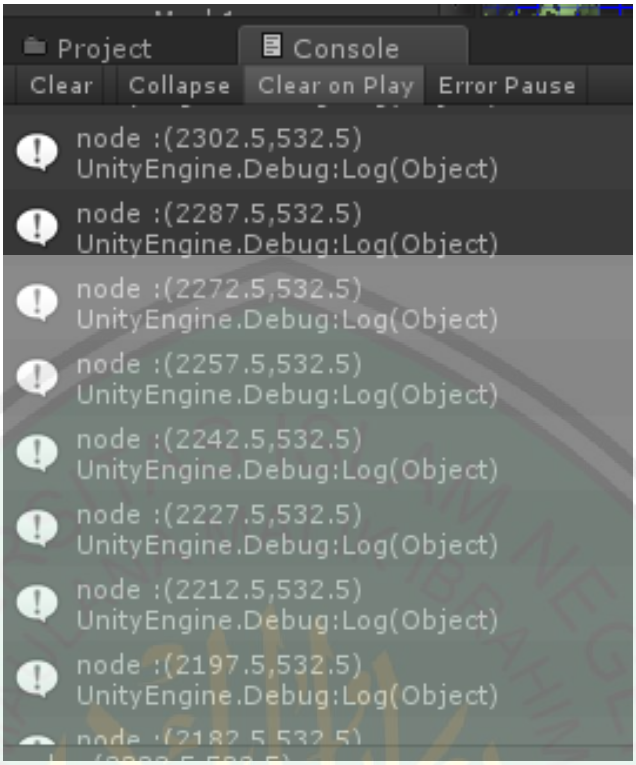
Tabel 4. 2 Hasil Pengujian Kedua

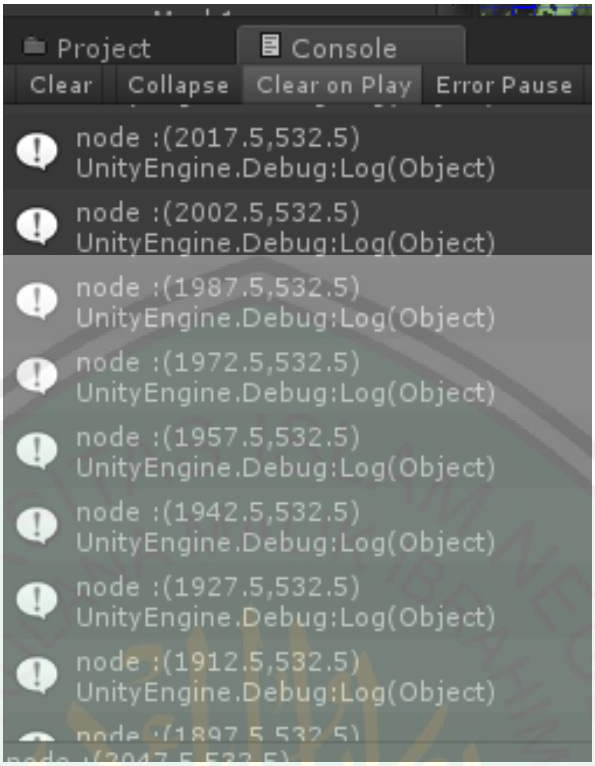
b. Pengujian ketiga

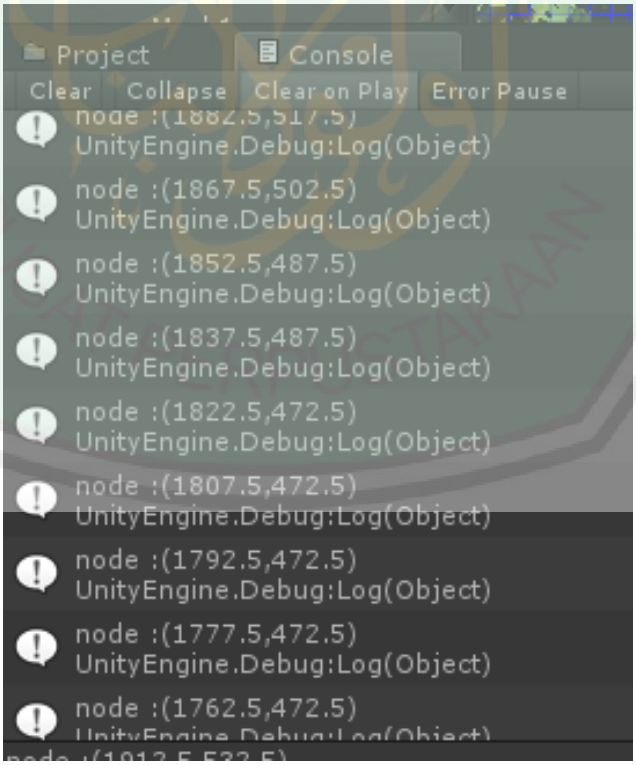
Pada pengujian ketiga ini posisi awal *player* (pada gambar 4.15 nomor 1) berada pada koordinat (2452.5, 547.5) dan goal (pada gambar 4.15 nomor 2) berada pada (2227.5, 532.5), pada saat di jalankan keadaan *player* dalam keadaan diam / *standby*, hasil yang didapatkan berupa garis yang mangarahkan pada goal. Dan juga hasil path yang di tampilkan pada *console Game engine* yang berisi path yang di lalui oleh *player*. berikut adalah tampilan yang diambil debug mode . Gambar (4.13)

Gambar 4. 16 path pada pengujian ketiga









Gambar 4. 17 Path pada Console pada pengujian ketiga

Berikut adalah rekapitulasi dari path yang akan di lalui *player* :

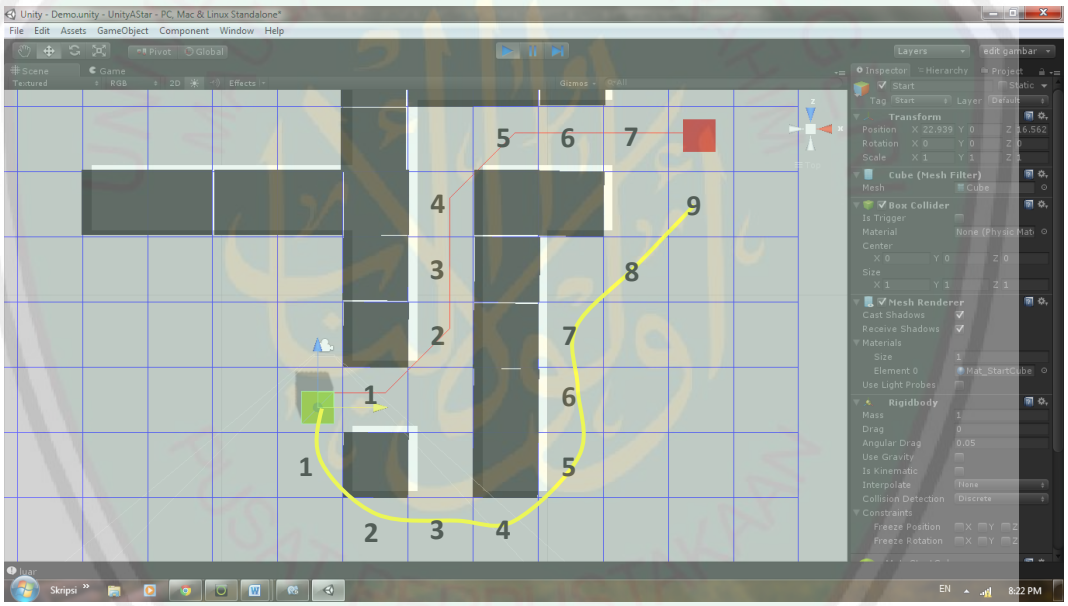
No	Nama item	Hasil	Keterangan
1	Posisi *player*	(2467.5, 442.5)	
2	Posisi objek	(1672.5, 307.5)	
3	Jumlah path	67 node	
4	Waktu proses	0.02 detik	
5	Path	node :(2467.5,442.5) node :(2452.5,442.5) node :(2452.5,457.5) node :(2452.5,472.5) node :(2452.5,487.5) node :(2452.5,502.5) node :(2452.5,517.5) node :(2437.5,532.5) node :(2422.5,532.5) node :(2407.5,532.5) node :(2392.5,532.5) node :(2377.5,532.5) node :(2362.5,532.5) node :(2347.5,532.5) node :(2332.5,532.5) node :(2317.5,532.5) node :(2302.5,532.5) node :(2287.5,532.5) node :(2272.5,532.5) node :(2257.5,532.5) node :(2242.5,532.5) node :(2227.5,532.5) node :(2212.5,532.5) node :(2197.5,532.5) node :(2182.5,532.5) node :(2167.5,532.5) node :(2152.5,532.5) node :(2137.5,532.5) node :(2122.5,532.5) node :(2107.5,532.5) node :(2092.5,532.5)	Menemukan solusi

		node :(2077.5,532.5) node :(2062.5,532.5) node :(2047.5,532.5) node :(2032.5,532.5) node :(2017.5,532.5) node :(2002.5,532.5) node :(1987.5,532.5) node :(1972.5,532.5) node :(1957.5,532.5) node :(1942.5,532.5) node :(1927.5,532.5) node :(1912.5,532.5) node :(1897.5,532.5) node :(1882.5,517.5) node :(1867.5,502.5) node :(1852.5,487.5) node :(1837.5,487.5) node :(1822.5,472.5) node :(1807.5,472.5) node :(1792.5,472.5) node :(1777.5,472.5) node :(1762.5,472.5) node :(1747.5,472.5) node :(1732.5,472.5) node :(1717.5,472.5) node :(1702.5,457.5) node :(1687.5,442.5) node :(1687.5,427.5) node :(1672.5,412.5) node :(1672.5,397.5) node :(1672.5,382.5) node :(1672.5,367.5) node :(1672.5,352.5) node :(1672.5,337.5) node :(1672.5,322.5) node :(1672.5,307.5)	

Tabel 4. 3 Hasil Pengujian Ketiga

4.4.2 Hasil Rute

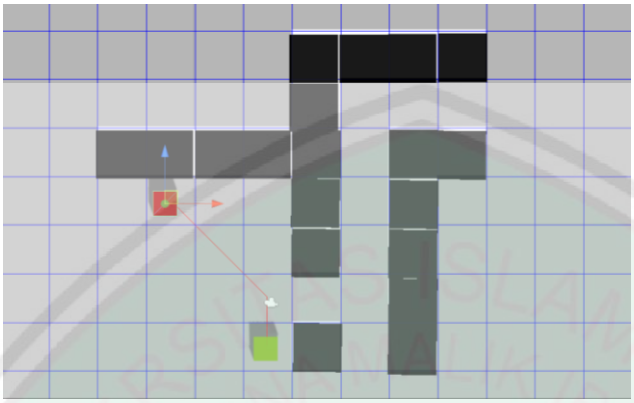
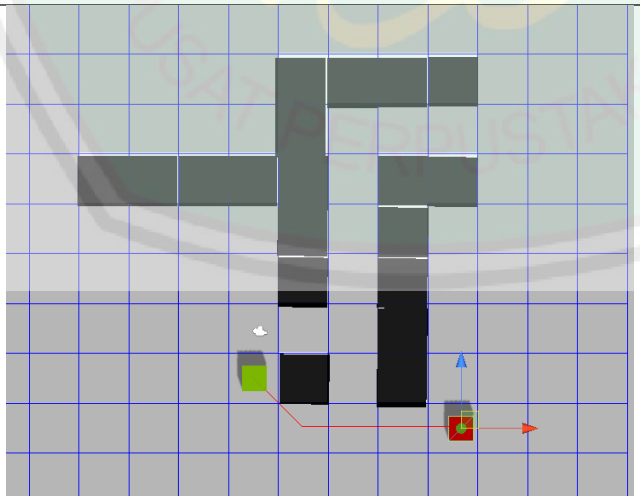
Pada pengujian ini akan menampilkan hasil bahwa rute yang di hasilkan adalah yang terbaik, pengujian dilakukan dengan komponen yang sama tetapi dengan map yang berbeda dengan tujuan untuk memudahkan untuk melihat hasilnya. Map dibuat putih polos dengan dinding penghalang berwarna hitam(area yang tidak bisa di lewati) , start berwarna hijau , dan Goal berwarna merah, sedangkan rute yang di lalui berupa garis berwarna merah, seperti di gambar di bawah ini.

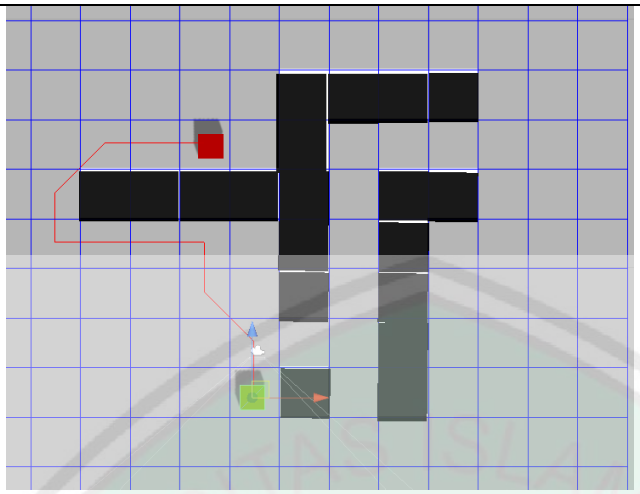
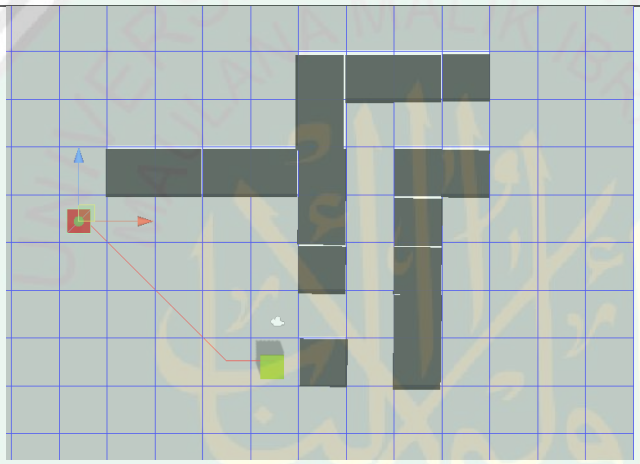


Gambar 4. 18 Jalur terbaik

Dalam gambar di atas path yang di hasilkan algoritma *astar* sebanyak 7 node ,dan dari kemungkinan rute lain (garis berwarna kuning) yang menghasilkan rute sebanyak 9 node , hal ini bisa menjadi salah satu keunggulan dari algoritma *astar* untuk menentukan jalur yang terbaik (lihat tabel). Dan dapat di implementasikan sebagai alat navigasi pada game sepeda.

Berikut adalah ujicoba path untuk pemilihan jalur terbaik:

No	Screenshot	Jml path	ket
1.		2	Sudah memenuhi jalur terbaik dari kemungkinan jalur yang ada
2.		1	Sudah memenuhi jalur terbaik dari kemungkinan jalur yang ada
3.		3	Sudah memenuhi jalur terbaik dari kemungkinan jalur yang ada

4.		9	Belum memenuhi jalur terbaik, di karenakan adanya tembok penghalang.
5.		3	Sudah memenuhi jalur terbaik dari kemungkinan jalur yang ada

Tabel 4. 4 Hasil Pengujian rute terbaik

4.4.3 Hasil *Line Renderer*

Line renderer adalah suatu komponen yang di sediakan oleh *Game engine* yang berfungsi untuk menampilkan garis dengan proses render, komponen ini akan membentuk garis dengan menggabungkan antar titik, untuk membentuk garis yang berkelok kelok maka dibutuhkan titik kordinat yang banyak, untuk itu di butuhkan suatu *path*, *path* ini yang akan meng hubungkan antara player dengan goal, *Path* di tampung dalam bentuk *array*. Yang kemudian di *render* menjadi garis yang saling berkaitan dari data pada *path* pertama sampai terakhir. Berikut adalah hasil dari *line renderer :*

Gambar 4. 19 line renderer pada mini map

BAB V
PENUTUP

5.1 Kesimpulan

Berdasarkan game yang telah di buat dan uji coba yang telah dilakukan , maka dapat ditarik kesimpulan sebagai berikut:

1. Algoritma *Astar* bisa digunakan untuk menyelesaikan pencarian objek pada game sepeda
2. Berdasarkan dari uji coba, implementasi algoritma yang berjalan secara realtime akan membutuhkan komputasi yang lebih juga.

5.2 Saran

Berdasarkan kesimpulan di atas, terdapat beberapa saran untuk pengembangan game ini selanjutnya:

1. Game yang di bangun memiliki tantangan yang kompleks dan menantang, sehingga dapat menarik
2. Penambahan algoritma yang lebih bagus agar game berjalan lebih baik
3. Penggunaan komputasi yang lebih bisa di siasati dengan mengurangi jumlah geometri dalam model 3d dan juga pada Scripting, sehingga pada waktu proses *rendering* tidak memakan banyak *resource*

DAFTAR PUSTAKA

Peters, C., Kyaw, A. S., & Swe, T. N. (2013). *Unity 4.x Game AI Programming.* BIRMINGHAM - MUMBAI: Packt Publishing.

Al-Qazwiniy, A.-H. A. (n.d.). Sunan Ibn Mājah. Dar al-Ihya al-Kutub al-ʻArabiyah.

Ashaolu, P. (2012). Development of an Interactive 3-D Virtual Environment Test Bed for Motorbikes. *Electrical and Information Technology Research Unit, Savonia University of Applied Sciences* , 21.

Barr, A., Feigenbaum, & Edward, A. (2002). *The Hanbook Of Artificial Intelligence.* California: HeurisTech Press.

Galochkin, I. (2013). *Implementation of a cross-platform strategy multiplayer game based on Unity3D.* MÜNCHEN .

Higgins, D. (2002). Pathfinding Design Architecture. In S. Rabin, *AI Game Programming Wisdom* (p. 123). 20 Downer Ave, Minham: Charles River Media.

Kusumadewi, s. (2003). *Artificial Intelegenci (Teknik dan Aplikasinya)/ Sri Kusumadewi.* yogyakarta: Penerbit Graha Ilmu.

Matthews, J. (2002). Basic A* Pathfinding Made Simple. In S. Robin, *AI Game Programming Wisdom* (p. 105). Charles River Media.

Patel, A. (2013, october 12). *Introduction to A**. Retrieved october 28, 2013, from Introduction to A*: http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html

unity3d. (2014, March 1). *unity3d - multiplatform*. Retrieved March March, 2014, from unity3d: https://unity3d.com/unity/multiplatform

Unitygems. (2012, november 27). *Unitygems*. Retrieved october 28, 2013, from Unitygems: http://unitygems.com/astar-1-journeys-start-single-step

Xiao Cui. (2011). Direction Oriented Pathfinding in Video Gamed. *International Journal of Artificial Intelligence & Applications (IJAIA)*, No.4.