PENERAPAN NEURAL NETWORK UNTUK NPC BRAKING DECISION PADA RACING GAME

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Herlambang Yudha Prasetya
NIM: 125150200111068

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2019

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

PENGESAHAN

PENERAPAN *NEURAL NETWORK* UNTUK NPC BRAKING DECISION PADA RACING GAME

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Herlambang Yudha Prasetya
NIM: 125150200111068

Skripsi ini telah diuji dan dinyatakan lulus pada
3 Januari 2019
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Muhammad Aminul Akbar, S.Kom., M.T.	Issa Arwani, S.Kom, M.Sc
NIK: 2016078910131001	NIP: 19830922 201212 1 003

Mengetahui
Ketua Jurusan Teknik Informatika

Tri Astoto Kurniawan, S.T., M.T., Ph.D
NIP: 19710518 200312 1 001

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar pustaka.

Apabila ternyata didalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 3 Januari 2019

Herlambang Yudha Prasetya

NIM: 125150200111068

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

KATA PENGANTAR

Puji Syukur penulis panjatkan kehadirat Tuhan Yang Maha Esa, yang telah melimpahkan rahmat-Nya, sehingga penulis dapat menyelesaikan skripsi dengan judul mengenai "Penerapan *Neural Network* untuk NPC Braking Decision pada Racing Game".

Skripsi merupakan tugas akhir yang diajukan untuk memenuhi syarat dalam memperoleh gelar Sarjana Komputer (S.Kom) pada Fakultas Ilmu Komputer (FILKOM) Universitas Brawijaya Malang.

Untuk kesempatan ini penulis juga menyampaikan rasa terima kasih kepada pihak – pihak yang telah membantu penulis selama penyusunan laporan praktik kerja lapangan, diantaranya:

1. Bapak Wayan Firdaus Mahmudy, S.Si., M.T., Ph.D. selaku Dekan Fakultas Ilmu Komputer Universitas Brawijaya Malang.
2. Bapak Tri Astoto Kurniawan, S.T., M.T., Ph.D. selaku Ketua Jurusan Teknik Informatika, Fakultas Ilmu Komputer Universitas Brawijaya Malang.
3. Bapak Agus Wahyu Widodo, S.T, M.Cs. selaku Kepala Program Studi Teknik Informatika, Fakultas Ilmu Komputer Universitas Brawijaya Malang.
4. Bapak Muhammad Aminul Akbar, S.Kom., M.T. selaku Dosen Pembimbing I yang telah meluangkan waktu untuk membimbing dan mengarahkan penulis dalam penyusunan skripsi.
5. Bapak Issa Arwani, S.Kom, M.Sc. selaku Dosen Pembimbing II yang telah meluangkan waktu untuk membimbing dan mengarahkan penulis dalam penyusunan skripsi.

Penulis menyadari masih banyak kekurangan dalam skripsi ini baik dalam teknik penyajian materi maupun pembahasan. Demi kesempurnaan skripsi ini, diharapkan saran dan kritik yang bersifat membangun. Semoga karya tulis ini bermanfaat dan dapat memberikan sumbangan yang berarti bagi pihak yang membutuhkan.

Malang, 3 Januari 2019

Penulis

h_yudha_p@yahoo.com

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

repository.ub.ac.id

ABSTRAK

Genre game balap merupakan genre yang cukup populer. Alasan utamanya adalah game balap mampu untuk menawarkan bukan hanya sensasi untuk melaju dengan cepat, namun juga grafik yang semakin menarik, *track* yang bervariasi, serta kecerdasan buatan yang cukup menantang untuk dikalahkan pemain. Dari berbagai hal tersebut yang cukup penting untuk dikembangkan adalah kecerdasan buatan, karena inti *game* balap adalah tantangan yang menyenangkan, dan hal itu akan sulit dicapai jika kecerdasan buatan sebagai lawan main berperilaku monoton atau bahkan bermain sangat buruk saat balapan. Maka dari itu diperlukan kecerdasan buatan yang mampu mengenali pola pikir manusia dan menirukannya dalam mengambil keputusan. Salah satu keputusan yang ada pada mobil adalah penentuan gas dan rem atau *braking decision*. Dari hal tersebut menginspirasi penulis untuk merancang agen pada Racing Game Starter Kit(RGSK) dengan algoritma *Neural network*. Dengan meniru cara kerja neuron otak yang berjumlah milyaran membuat *Neural network* mampu menyederhanakan pengkodean mesin yang rumit dengan merelokasikan proses pengambilan keputusan ke satu atau lebih jaringan saraf terlatih. Serta algoritma ini memungkinkan untuk beradaptasi saat game dimainkan menjadi nilai tambah tersendiri terutama bagi peneliti di bidang *Artificial Intelligent*. Berdasarkan hasil akhir proses testing menggunakan *Cross-Validation* menunjukkan algoritma *Neural network* memiliki akurasi belajar sebesar 0.76 yang berarti tingkat kemiripan algoritma dengan perilaku manusia sebesar 76%. Dengan rata-rata fps yang didapat agen adalah 59 fps dan waktu tempuh 10 putaran total, memiliki waktu tempuh 72 detik atau 12% lebih cepat dibanding AI *default kit*.

Kata kunci: *kecerdasan buatan, neural network, kontroler, cross-validation.*

UNIVERSITAS BRAWIJAYA

repository.ub.ac.id

ABSTRACT

The genre of racing games is quite popular. The main reason is racing games are able to offer not only the sensation of speeding up, but also the graphics, varied tracks, and artificial intelligence that is quite challenging for players to beat. Of these things the most important thing to develop is artificial intelligence, because the core of racing games is a fun challenge, and it will be difficult to achieve if artificial intelligence as opponent to playing monotonous behavior or even playing very badly during the race. Therefore, artificial intelligence is needed to be able to recognize the human mindset and imitate it in making decisions. One decision that can be observed is the determination of accelerate and brakes or braking decision. From the problem above, the author was inspired to design an agent in the Racing Game Starter Kit (RGSK) with the Neural network algorithm. By imitating the workings of brain neurons, which amount to billions, the Neural network is able to simplify complex machine coding by relocating the decision-making process to one or more trained neural networks. And the ability of this algorithm to adapt when games are played is an added value, especially for researchers in the field of Artificial Intelligent. Based on the final results of the testing process using Cross-Validation, the Neural network algorithm has a learning accuracy of 0.76, which means the algorithm's similarity to human behavior is 76%. With the average fps obtained by the agent is 59 fps and the travel time of 10 total turns, has a travel time of 72 seconds or 12% faster than the default AI kit.

Keywords : *artificial intelligence, neural network, controller, cross-validation*

UNIVERSITAS BRAWIJAYA

DAFTAR ISI

PENGESAHAN ii
PERNYATAAN ORISINALITAS iii
KATA PENGANTAR iv
ABSTRAK v
ABSTRACT vi
DAFTAR ISI vii
DAFTAR TABEL ix
DAFTAR GAMBAR x
BAB 1 PENDAHULUAN 1
1.1 Latar belakang 1
1.2 Rumusan masalah 2
1.3 Tujuan 2
1.4 Manfaat 2
1.5 Batasan masalah 3
1.6 Sistematika pembahasan 3
BAB 2 LANDASAN KEPUSTAKAAN 4
2.1 Tinjauan pustaka 4
2.2 Artificial Intelligence dan Racing Games 5
2.3 *Neural network* 6
2.4 Unity Engine 7
2.5 Racing Game Starter Kit 8
BAB 3 METODOLOGI 10
3.1 Studi Pustaka 10
3.2 Perancangan 10
3.3 Implementasi 11
3.4 Pengujian dan Analisis 11
BAB 4 PERANCANGAN 12
4.1 Data Training 12
4.2 Logika Agen 14
4.3 K-Fold Cross-Validation 26

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

BAB 5 IMPLEMENTASI 27
5.1 Data Training 27
5.2 Logika Agen 29
BAB 6 PENGUJIAN DAN ANALISIS 39
6.1 Skenario Pengujian 39
6.2 Hasil Pengujian 39
6.3 Analisis Hasil Pengujian 42
BAB 7 PENUTUP 44
7.1 Kesimpulan 44
7.2 Saran 44
DAFTAR PUSTAKA 45

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

DAFTAR TABEL

Tabel 5.1 Fungsi Data Capturing 27
Tabel 5.2 Fungsi Data Training 29
Tabel 5.3 Fungsi *Neural network* 30
Tabel 5.4 Fungsi *Neural network* Lanjutan 30
Tabel 5.5 Fungsi Neuron 30
Tabel 5.6 Fungsi Data Testing 30
Tabel 5.7 Fungsi Navigasi 30
Tabel 6.1 Hasil Pengujian FPS Agen 40
Tabel 6.2 Hasil Pengujian Waktu Agen 30

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

DAFTAR GAMBAR

Gambar 2.1 Struktur Neuron pada manusia(kiri) dan artificial(kanan) 7
Gambar 2.2 Alur algoritma backpropagation. 8
Gambar 2.3 Unity Interface..... 9
Gambar 2.4 *Waypoint* pada lintasan beserta script pada inspector 10
Gambar 4.1 Ilustrasi Sudut dan Jarak Tikungan..... 14
Gambar 4.2 Lintasan Player 14
Gambar 4.3 *Flowchart* Logika Agen RGSK..... 16
Gambar 4.4 Detail proses OnTrigger pada *flowchart* logika agen RGSK 17
Gambar 4.5. Detail proses Start pada *flowchart* logika agen RGSK..... 18
Gambar 4.6. Detail proses FixedUpdate pada *flowchart* logika agen RGSK..... 19
Gambar 4.7. *Flowchart* logika agen *Neural network* 20
Gambar 4.8. Detail *Flowchart* Normalisasi Data *Neural network*..... 21
Gambar 4.9. Detail *Flowchart* Pelatihan Data *Neural network* 21
Gambar 4.10. Detail *Flowchart* Pengujian *Output Neural network*..... 22
Gambar 4.11 Struktur *Neural network* 23
Gambar 4.12 Contoh Struktur *Neural network*..... 25
Gambar 4.13 Contoh Struktur *Neural network*..... 26
Gambar 4.14 Skema K-Fold Cross-Validation 27
Gambar 6.1 Hasil akhir cross-validation 41
Gambar 6.2 Grafik FPS Agen 42
Gambar 6.3 Penalty Surface..... 43
Gambar 6.4 Track Line Agen 44
Gambar 6.5 Posisi Start Track Line Agen 44

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

BAB 1 PENDAHULUAN

1.1 Latar belakang

Genre game balap merupakan genre yang cukup tua, namun tidak menunjukkan tanda-tanda kehilangan popularitas. Hal itu dikarenakan game balap mampu menawarkan berbagai hal menarik antara lain, sensasi dari kecepatan saat melaju, liku-liku *track* yang sangat menantang, dan juga drifting yang membuat adrenalin terpacu. Dan setiap game balap baru hadir dengan fisika permainan yang lebih kompleks, grafik yang terlihat lebih nyata, dan bahkan juga ditambah konektifitas jaringan internet sehingga membuat game balap tetap melekat dihati para penggemar game (Togelius J,2006). Selain hal tersebut diatas, pengembangan game balap juga bergantung pada kecerdasan buatan yang bertindak sebagai lawan bermain saat balapan berlangsung.

Kecerdasan buatan atau Artificial Intelligent(AI) menurut Muhammad Fauzy(2011), yaitu salah satu cabang ilmu yang mengaplikasikan proses berfikir manusia kepada mesin sehingga dapat menirukan tindakan dan perilaku manusia. Hubungan AI dan games sudah terbentuk sejak masa-masa awal terciptanya komputer (Karavolos, 2013), yang pada saat itu banyak peneliti di bidang AI yang berusaha mengembangkan algoritma apa yang digunakan untuk memainkan sebuah game. Dengan menciptakan AI semirip mungkin dengan pola pikir manusia, maka AI tersebut dapat memberikan tantangan menyenangkan sekaligus akan memberikan kesan bahwa pemain tersebut sedang bermain dengan manusia lain. Demikian pula pada pengembangan AI game balap saat ini yang menuntut AI semirip mungkin dengan perilaku manusia. Salah satu hal yang dapat diamati secara langsung apakah AI telah mirip dengan pola pikir manusia adalah pemilihan keputusan kapan AI tersebut menambah atau mengurangi kecepatan kendaraan yang selanjutnya kita sebut dengan *braking decision*.

Braking decision adalah salah satu masalah utama dalam balap mobil. Pengemudi balap Formula 1 misalnya, membutuhkan latihan dan belajar ekstra hanya untuk menentukan di zona mana dia akan melakukan pengereman atau yang disebut *brake zone*. Walaupun di dalam trek balap terdapat penanda atau *braking marker* pada 150 , 100, dan 50 meter sebelum track corner, namun pembalap juga harus mempertimbangkan performa kendaraan, cuaca, dan pembalap lain di depannya sebelum melakukan pengereman. Sehingga pengalaman melaju pada trek balap dan simulator sangat berperan penting dalam hal ini. Maka dari itu, dibutuhkan kecerdasan buatan yang mampu bergerak semirip mungkin dengan manusia. Salah satu algoritma yang digunakan agar mesin dapat belajar atau *machine learning* adalah algoritma *Neural network*.

Algoritma *Neural network* merupakan algoritma yang mampu untuk belajar dan meningkatkan performa dengan pengalaman sebelumnya. *Neural network* yang digunakan dalam permainan cukup sederhana jika dibandingkan dengan otak manusia. Untuk sebagian besar pengaplikasiannya, *neural network* terdiri

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

dari hanya beberapa neuron, sedangkan otak manusia mengandung $\sim 10^{11}$ neuron, elemen dasar sirkuit otak yang dihubungkan oleh $\sim 10^{14}$ koneksi sinaptik (Huang Z.J, 2015). Namun dalam game, *neural network* menawarkan beberapa keunggulan. Pertama, *neural network* dapat menyederhanakan pengkodean mesin yang rumit dengan merelokasikan proses pengambilan keputusan ke satu atau lebih jaringan saraf terlatih. Kedua, *neural network* menawarkan potensi game AI untuk beradaptasi saat game dimainkan. Ini kemungkinan yang sangat unik dan merupakan subjek yang sangat populer di komunitas AI game.

Dari hal tersebut, penulis memiliki target untuk mengeksplorasi bagaimana algoritma *neural network* dapat digunakan untuk melatih sebuah mesin yang bertindak sebagai pengemudi di dalam game balap. Dan dari hasil belajar tersebut mampu didapatkan perilaku cerdas dalam melintasi *track* sesuai dengan apa yang telah mesin tersebut pelajari sebelumnya.

1.2 Rumusan masalah

Berdasarkan uraian latar belakang di atas maka permasalahan yang akan diambil adalah:

1. Bagaimana mengimplementasikan algoritma *neural network* yang digunakan sebagai sistem pengereman otomatis ke dalam game balap?
2. Bagaimana perbandingan waktu lap agen algoritma *neural network* dengan agen AI *default RGSK*?
3. Bagaimana perbandingan rata-rata fps sistem ketika agen algoritma *neural network* dengan agen AI *default RGSK* dijalankan?
4. Bagaimana akurasi agen algoritma *neural network* setelah dilakukan tahap pembelajaran?

1.3 Tujuan

Berdasarkan uraian rumusan masalah di atas maka tujuan yang hendak dipakai dalam penelitian adalah:

1. Mengimplementasi algoritma *neural network* untuk sistem pengereman otomatis ke dalam game balap.
2. Membandingkan waktu lap agen algoritma *neural network* dengan agen *default RGSK*.
3. Membandingkan rata-rata fps scene ketika agen algoritma *neural network* dengan agen AI *default RGSK* dijalankan.
4. Menentukan akurasi agen algoritma *neural network* setelah dilakukan tahap pembelajaran.

1.4 Manfaat

1. Dengan menemukan cara yang baru untuk implementasi kecerdasan buatan pada agen, dapat memperluas pengetahuan mengenai kecerdasan buatan dalam permainan game balap.

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

2. Metode yang baru dapat diimplementasikan pada permainan lain untuk mendapatkan hasil yang baru.

1.5 Batasan masalah

Agar permasalahan yang telah dirumuskan dapat lebih fokus, skripsi ini dibatasi dalam lingkup beberapa hal yaitu sebagai berikut:

1. Implementasi game menggunakan perangkat lunak Game Engine Unity.
2. Untuk implementasi metode agen dilakukan pada bahasa pemrograman C#.
3. Pembahasan difokuskan pada hasil uji pembentukan perilaku agent AI dalam game balap yang valid.

1.6 Sistematika pembahasan

Sistematika penulisan laporan skripsi dijelaskan sebagai berikut:

BAB I Pendahuluan

Dalam bab pendahuluan, akan dijelaskan mengenai latar belakang, identifikasi masalah, batasan masalah, tujuan penelitian, manfaat penelitian, batasan penelitian, sistematika penulisan, dan jadwal pelaksanaan penelitian.

BAB II Landasan Kepustakaan

Pada bab ini dibahas mengenai dasar teori dan kajian pustaka yang digunakan pada penyusunan tugas akhir "PENERAPAN *NEURAL NETWORK* UNTUK NPC BRAKING DECISION PADA RACING GAME", yang meliputi Landasan Teori, Kerangka Berpikir dan Hipotesis Penelitian.

BAB III Metodologi

Berisi metode yang digunakan dalam melakukan penelitian yang diuraikan dalam kajian pustaka, analisis kebutuhan, pengumpulan dan pengolahan data, desain sistem, implementasi, pengujian, analisis hasil, dan penarikan kesimpulan dan saran.

BAB IV Perancangan

Berisi perancangan dari program yang akan diimplementasikan pada penelitian ini.

BAB V Implementasi

Berisi implementasi koding dari perancangan program.

BAB VI Pengujian dan Analisis

Berisi skenario pengujian, hasil, dan analisis pengujian agen.

BAB VII Penutup

Berisi kesimpulan dan saran dari hasil penelitian.

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

BAB 2 LANDASAN KEPUSTAKAAN

2.1 Tinjauan pustaka

Tinjauan pustaka adalah peninjauan tentang pustaka-pustaka yang terkait dalam penelitian ini. Tinjauan pustaka ini mengacu pada beberapa penelitian sebelumnya yang memiliki keterkaitan dengan permasalahan pada penelitian ini khususnya dalam metode yang digunakan. Penelitian ini membahas tentang penggabungan teknik kecerdasan buatan (AI), yang memungkinkan mobil dikontrol dengan cara meniru gerakan pengendara yang natural.

No.	Judul	Objek	Metode	Hasil
1	Aplikasi Jaringan Saraf Tiruan Backpropagation untuk Memprediksi Prestasi Siswa SMA (Studi kasus: Prediksi Prestasi Siswa SMAN 4 Ambon)	Prediksi prestasi siswa	*Neural network* BackPropagation	Pada 65 data latih yang dimasukkan, akurasi yang didapatkan sebesar 60%
2	Penerapan Jaringan Syaraf Tiruan Untuk Memprediksi Jumlah Pengangguran di Provinsi Kalimantan Timur Dengan Penerapan Jaringan Syaraf Tiruan Untuk Memprediksi Jumlah Pengangguran di Provinsi Kalimantan Timur Dengan Menggunakan Algoritma Pembelajaran Backpropagation	Prediksi jumlah pengangguran ditahun berikutnya	*Neural network* BackPropagation	Rata – rata error jaringan sebesar 13.7001521. Hasil prediksi jumlah pengangguran pada tahun 2009 adalah 133.104. Hasil prediksi oleh BPS Provinsi Kalimantan Timur adalah 139.830.

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

3	PENERAPAN ALGORITMA BACKPROPAGATION UNTUK MENENTUKAN LEVEL BONUS DAN SCORE BONUS PADA GAME EDUKASI NAHWU MENGGUNAKAN KARTU BERBASIS ANDROID	Level bonus & Score bonus	*Neural network* BackPropagation	Teknik pembelajaran terawasi Backpropagation berhasil diimplementasikan dengan baik pada game ini, melalui hasil pengujian jaringan syaraf tiruan terhadap 3 model arsitektur diperoleh bahwa arsitektur 3 lapisan tersembunyi memiliki tingkat akurasi 91.71%

2.2 Artificial Intelligence dan Racing Games

Sejak masa-masa awal komputer, pengembangan algoritma dalam bermain game sudah dilakukan oleh berbagai peneliti yang ahli di dalam Artificial Intelligent (AI). Setelah beberapa tahun, game pada komputer terbukti mampu menjadi *testbed* yang ideal bagi algoritma AI. Walaupun game lebih simpel jika dibandingkan dengan permasalahan yang ada di dunia nyata, namun juga cukup rumit untuk dijadikan sebagai tantangan. Dan semenjak game catur buatan Arthur Samuel pada 1952, pengembangan AI menjadi cukup pesat. Sebagai langkah awal, para peneliti hanya berfokus pada game tradisional seperti othello dan catur. Game tersebut merupakan Turn-Based sehingga setiap agent AI mendapatkan informasi keadaan game yang lengkap dan tetap setiap saat. Dalam beberapa waktu selanjutnya, fokus peneliti diperluas kepada game dengan informasi yang tidak lengkap dan acak, seperti poker dan backgammon (Karavolos, 2013).

Pada 1990-an terjadi keberhasilan yang cukup banyak dalam sejarah bagi level kecerdasan AI program. Banyak kompetisi kelas dunia seperti catur pada 1994 dan 1997, serta backgammon dengan program TD-Gammon. Pada periode yang sama, industri video game berkembang sangat pesat dan menjadi lingkungan pengembangan AI yang populer

Dalam konteks game racing, khususnya dalam game-game komersial saat ini, AI telah diset secara tetap, menjadi blok-blok difficulty yang dapat dipilih player sebelum memasuki track. Sehingga AI juga sangat berperan penting dalam menentukan game experience yang dialami oleh player. Selain itu, AI juga dapat

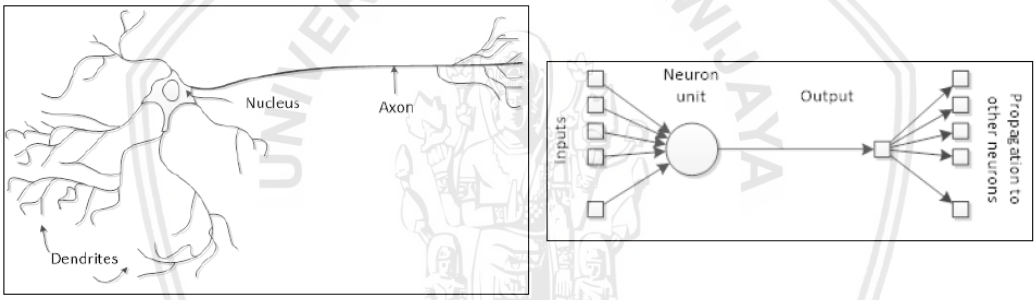
repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

menekan biaya pembuatan aset misalnya dalam pembuatan track, dan juga menambah potensi untuk mengeksplorasi permainan semaksimal mungkin

2.3 *Neural network*

Secara sadar maupun tidak sadar, hal pertama yang terlintas saat kita mendengar kata *neural network* yaitu susunan saraf yang berjumlah sangat besar dalam otak kita. Namun dalam istilah pemrograman, *neural network* yang dimaksud adalah artificial *neural network* atau jaringan saraf tiruan yang berhubungan dengan penciptaan struktur dalam komputer yang mirip dengan otak manusia. Dengan kata lain kecerdasan buatan.

Elemen yang paling dasar dalam sebuah *neural network* adalah neuron. Di tubuh kita, neuron berfungsi sebagai pengolah sinyal yang masuk melalui dendrit dan akan memicu sinyal di akson tergantung dari kekuatan atau besarnya bobot sinyal yang diterima. Neuron kemudian kita terjemahkan ke dalam bahasa komputer sebagai pengumpul sinyal dalam *input* dan unit aktivasi dalam *output* yang dapat memicu sinyal yang akan diteruskan ke neuron lain.



Gambar 2.1 Struktur Neuron pada manusia(kiri) dan artificial(kanan)

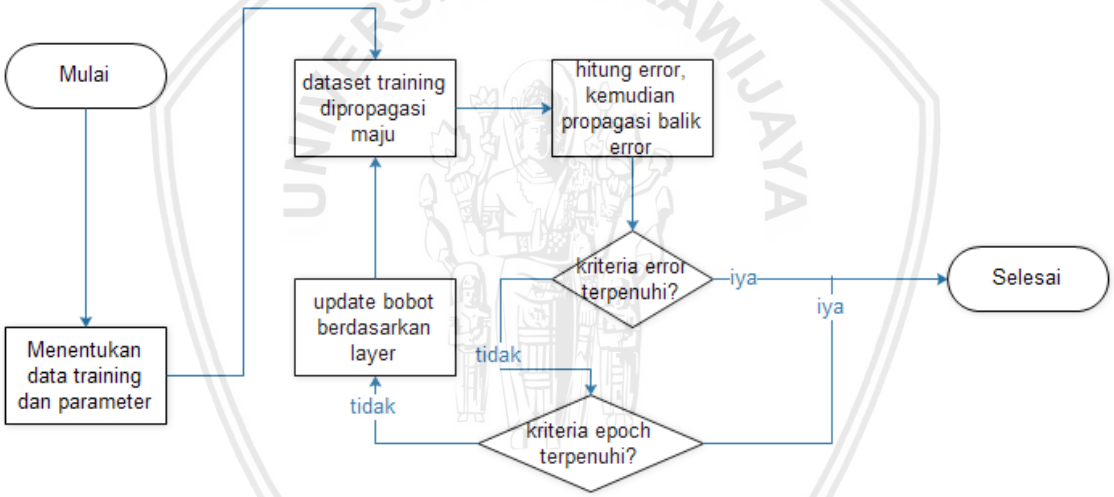
Setiap pola-pola informasi *input* dan *output* yang diberikan ke dalam *neural network* diproses dalam neuron. Neuron-neuron tersebut terkumpul di dalam lapisan-lapisan yang disebut neuron layers. *Neural network* dapat terdiri dari beberapa lapisan yang terhubung dan membentuk jaringan multilayer. Lapisan saraf dalam multilayer kemudian dapat diklasifikasikan ke dalam 3 kategori. Lapisan pertama yaitu *input* layer, dimana lapisan ini menerima *input* dari luar. Lapisan kedua yaitu *hidden layer*, lapisan ini hanya ada pada jaringan multilayer dan terletak diantara *input* dan *output* layer. Lapisan ketiga yaitu *output* layer, hasil dari lapisan ini adalah solusi terhadap masalah yang dihadapi *neural network*.

Manfaat memiliki jaringan multilayer adalah untuk mengabstraksi lebih banyak data dari lingkungan sehingga mampu mencakup masalah yang lebih kompleks. Hal ini kemudian memotivasi pengembangan suatu algoritme pelatihan yang dapat menyesuaikan bobot *hidden layer* dengan tepat. Dan pada tahun 1980 Rummelhart dan beberapa peneliti lain mengusulkan algoritma baru yang disebut backpropagation (Soarez F.M, 2016). Algoritma ini mampu

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

memperbaiki bobot berdasarkan tingkat error antara target *output* dan *neural network output*.

Secara umum, algoritma backpropagation ini memiliki alur yang cukup sederhana sehingga banyak dipakai pada aplikasi di berbagai bidang. Alur tersebut yaitu ketika *neural network* diberikan *input* yang kemudian digunakan sebagai data training, maka data tersebut akan melakukan forward propagation, yaitu menuju ke *hidden layer* dan kemudian diteruskan ke *output* layer. *Output* layer memberikan data *output* yang digunakan sebagai hasil akhir keluaran *neural network*. Namun jika hasil akhir tersebut berbeda dengan target yang diharapkan, maka bobot akan dikoreksi agar error dapat diperkecil dan respon *neural network* selanjutnya akan dilebih mendekati dengan target yang diharapkan. Kemampuan algoritma ini juga termasuk penyesuaian bobot pada *hidden layer* dan *input* layer, dengan kata lain hasil pada *output* layer akan menyebar balik (backward) ke *hidden layer* dan diteruskan ke *input* layer sehingga mekanisme ini dinamakan backpropagation / propagasi balik. Seperti pada gambar berikut ini:



Gambar 2.2 Alur algoritma backpropagation.

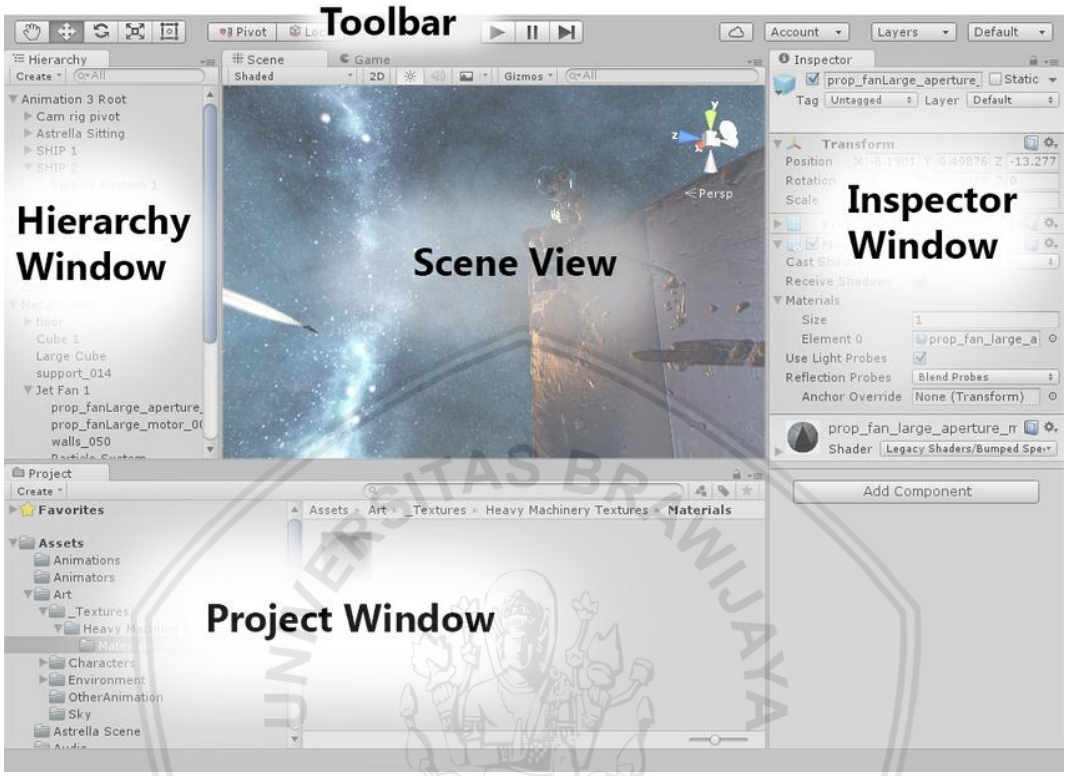
2.4 Unity Engine

Unity merupakan salah satu creation engine yang paling populer di dunia. Dengan adanya fakta bahwa game juga memiliki peran dalam pengembangan AI, membuat Unity berpikir bahwa ini adalah tanggung jawab mereka untuk menyediakan powerful training scenarios yang sangat dibutuhkan peneliti Machine Learning saat ini. Sekaligus juga menjembatani para komunitas gaming dengan teknologi Machine Learning terkini yang berhasil dikembangkan.

Unity mampu melakukan rendering grafis dan perhitungan fisika pada model 3D, sehingga memudahkan penelitian ini untuk fokus pada pengembangan dan pembuatan game software. Dengan Unity, pengembang dapat memposisikan, skala, dan menambahkan fisika realistis ke objek dalam sistem permainan. Gambar 3 menunjukkan antarmuka *editor* Unity. Panel kiri menampilkan daftar

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

objek permainan yang ditentukan, visualisasi tengah menunjukkan objek permainan di ruang 3D dari arena pacuan kuda, panel kanan menunjukkan atribut dari objek permainan yang dipilih, dan panel bawah terdiri dari ikon skrip, masing-masing yang mendefinisikan perilaku objek game



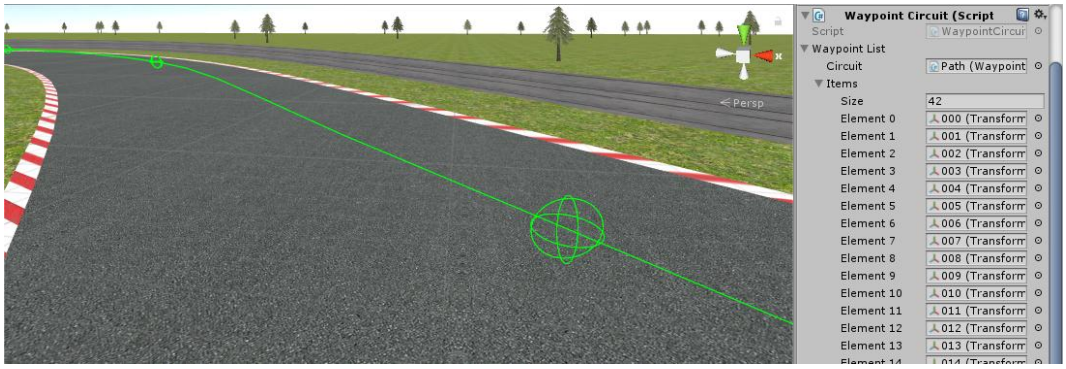
Gambar 2.3 Unity Interface

2.5 Racing Game Starter Kit

The Racing Game Starter Kit (RGSK) merupakan aset yang dirancang untuk memudahkan dalam pembuatan game balap. Ini termasuk fisika mobil / sepeda motor dan sistem balap lengkap dengan 7 jenis mode yang berbeda, diantaranya Sirkuit, Lap Knockout, Time Trial, Speed Trap, Checkpoint, Eliminasi & Drift. Selain itu juga terdapat fitur yang memudahkan penelitian, antara lain replay system, track lap time, vehicle customization system, customizable AI behaviour, dan skidmarks.

AI yang disediakan kit memiliki sistem dengan tingkat difficulty 4 macam. 4 macam tersebut adalah *Easy*, *Medium*, *Hard*, dan *Custom.* Perbedaan antara *difficulty* satu dengan yang lainnya terletak pada nilai variabel `accelerationSensitivity`,`brakeSensitivity`,`accelerationWander`, dan `nitroProbability`. Selain itu AI juga memiliki metode pencarian jalur(*pathfinding)* yang telah disediakan yaitu dengan *waypoint*. *Waypoint* pada RGSK ini berupa gameObject yang dapat diatur letak maupun jumlahnya melalui script yang telah disediakan.

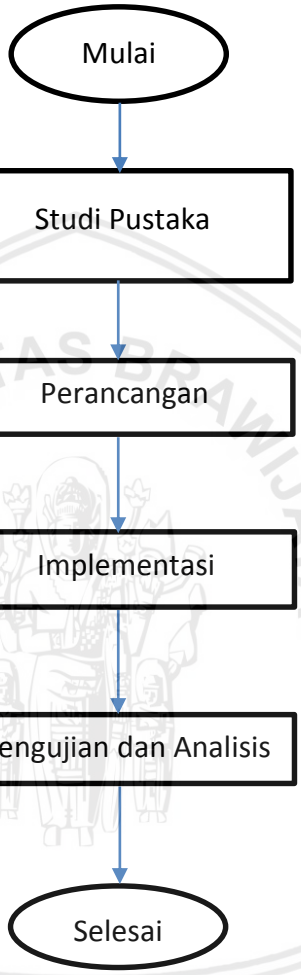
repository.ub.ac.id
UNIVERSITAS BRAWIJAYA



Gambar 2.4 *Waypoint* pada lintasan beserta script pada inspector

BAB 3 METODOLOGI

Pada bab ini akan menjelaskan langkah-langkah yang dilakukan dalam penyusunan skripsi. Penelitian ini bersifat implementatif, dengan mengimplementasi algoritma *Neural network* sebagai *braking decision making* untuk agen. Metode penelitian yang digunakan adalah:



3.1 Studi Pustaka

Studi pustaka merupakan tahap penelusuran pengetahuan dalam rangka menyusun dasar teori yang digunakan sebagai dasar pengetahuan dalam penelitian ini. Penelusuran pengetahuan ini bersumber dari buku, jurnal, dan internet (website resmi) yang berkaitan dengan informasi yang dibutuhkan dalam mengerjakan penelitian ini. Informasi pendukung dalam penelitian ini meliputi Unity, Algoritma *Neural network*, Racing Game Starter Kit.

3.2 Perancangan

Pada tahap perancangan, peneliti memberikan dasar logika untuk pembuatan program yang menerapkan algoritme *Neural network*, mulai dari pengambilan data training dari AI default kit, normalisasi data menggunakan metode *min-max*,

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

algoritma *Neural network* untuk agen, dan pengukuran akurasi menggunakan metode *K-Fold Cross-Validation*. Peneliti menggunakan contoh software kit pada Unity Asset Store sebagai dasar untuk mengembangkan agen yang diinginkan. Software kit yang disediakan menggunakan bahasa C#, yang menggunakan teknik *trigger detection* dari Unity untuk menerima informasi dari permainan game balap yang nantinya digunakan agen untuk menentukan keputusan yang diambil agen.

3.3 Implementasi

Pada tahap implementasi, diberikan script yang menerapkan perancangan pada tahap sebelumnya. Script akan disajikan sesuai dengan fungsinya. Script C# akan digunakan untuk implementasi game Racing Game Starter Kit yang dilakukan di dalam game engine Unity. Untuk hasil akan disertakan dengan gambar screenshot jalannya program tersebut.

3.4 Pengujian dan Analisis

Untuk tahap pengujian, peneliti menguji agen dengan metode blackbox testing. Teknik pengujian respon dari suatu masukan terhadap sistem untuk memastikan tidak ada kesalahan implementasi maupun fungsi yang berjalan tidak sesuai harapan. Peneliti akan menguji agen dengan cara membuat agen memainkan Racing Game Starter Kit sebanyak sepuluh putaran. Hasil yang diambil adalah hasil rata-rata dari setiap lap.

Dari data pengujian sepuluh kali, akan dilakukan analisis untuk performa agen, dengan cara melihat waktu tiap lap, rata-rata fps, dan tingkat akurasi yang dapat dicapai agen. Agen yang mempunyai skor lebih tinggi dianggap merupakan agen yang lebih baik performa permainannya. Penulis juga akan memberikan poin-poin penting yang dapat mempengaruhi performa agen.

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

BAB 4 PERANCANGAN

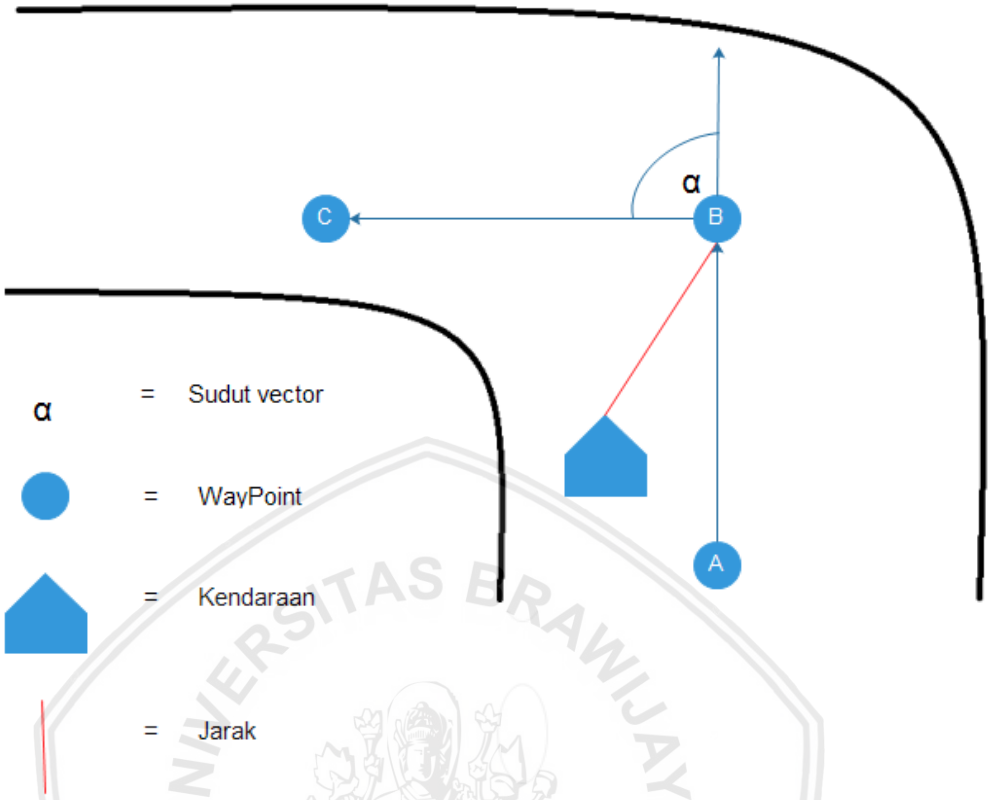
Pada bab perancangan akan dibahas mengenai struktur program yang akan diimplementasikan. Terdapat tiga bagian yang perlu dirancang, yaitu metode pengumpulan data training, penghitungan logika agen dengan algoritma *Neural network*, dan penghitungan tingkat akurasi dengan *K-Fold Cross-Validation*.

4.1 Data Training

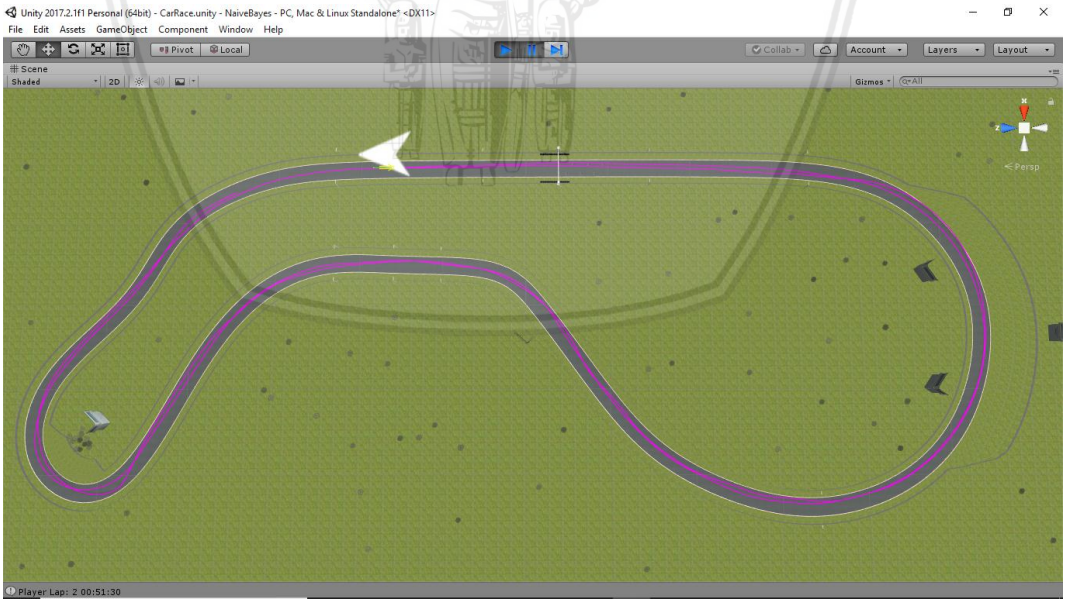
Algoritma *Neural network* yang dirancang dalam penelitian ini yaitu menggunakan metode *backpropagation* sebagai proses *learning*. Sehingga dibutuhkan data training berupa *input* dan *output* yang dituju. Dengan kata lain, proses pada penelitian ini merupakan supervised learning. Namun *neural network* juga bisa belajar dengan *input* data tanpa patokan *output* seperti *self-organizing maps* yang mampu diterapkan pada kasus tertentu seperti *clustering, data compression, statistical modeling,* dan *language modeling* (Soarez F.M, 2016).

Pada gambar 4.1 terdapat ilustrasi yang lebih detail mengenai pengambilan data training besar sudut tikungan, dan jarak mobil ke sudut tikungan. Sudut tikungan digambarkan dengan sudut vektor α(alpha) yang merupakan sudut vektor terbentuk dari perpanjangan vektor AB dan vektor BC. Lalu jarak mobil ke sudut lintasan digambarkan dengan garis merah pada gambar. Sedangkan kecepatan mobil serta nilai *throttle* didapat dari variabel *currentSpeed* dan *throttle* pada *gameobject* mobil saat mobil dicatat datanya. Data inilah yang akan digunakan selanjutnya untuk proses belajar agen *Neural network*. Gambar 4.2 adalah jalur lintasan yang dilalui player saat mengambilan data selama 2 putaran.

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA



Gambar 4.1 Ilustrasi Sudut dan Jarak Tikungan



Gambar 4.2 Lintasan Player

Pada saat mobil melakukan *data capturing* yaitu setiap 0.1 detik sejak dimulainya lap, data latih yang diperoleh pada proses ini disimpan satu persatu pada file berupa txt. Setiap data dipisahkan oleh enter dan setiap fitur dipisahkan

oleh ‘,’ (koma). Sampel data yang diambil pada proses selama 2 putaran ini dapat dilihat pada Tabel 4.1 berikut.

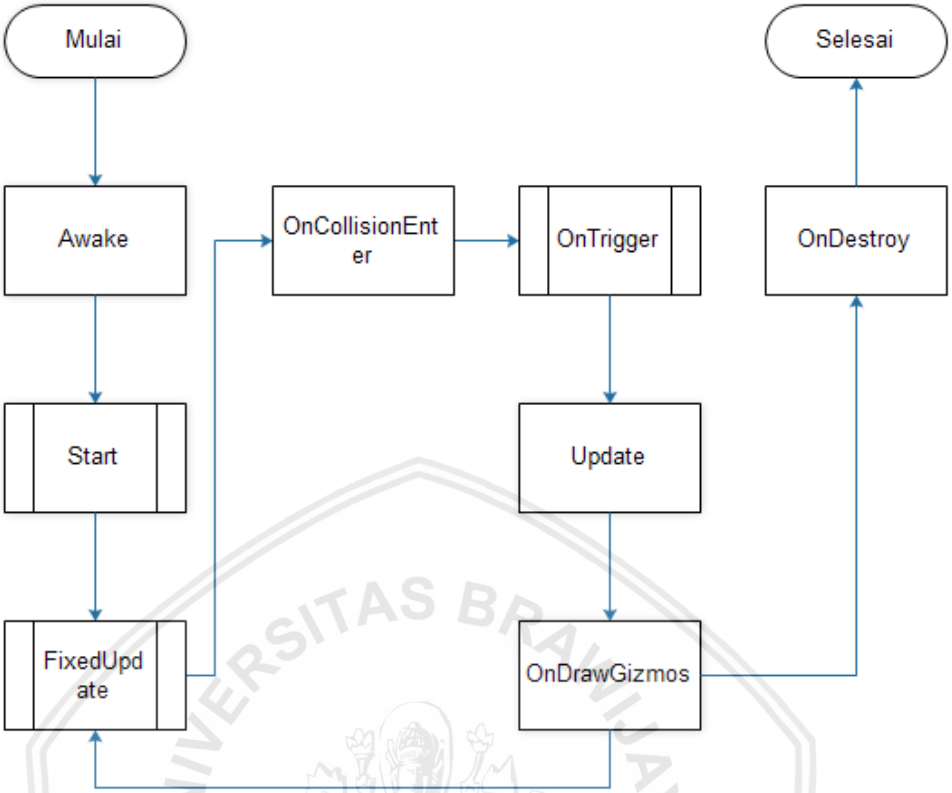
Tabel 4.1 Data Latih

No.	Kecepatan Mobil	Sudut Lintasan	Jarak Mobil ke Sudut	Target *Output*
1	0	1.206259	26.73554	0
2	0	1.206259	26.73554	0
3	0	1.206259	26.73356	0
4	1	1.206259	26.73356	0
5	1	1.206259	26.71585	0
6	1	1.206259	26.70098	0
7	2	1.206259	26.68078	0
8	2	1.206259	26.65574	0
9	2	1.206259	26.6274	0
10	2	1.206259	26.59302	0
11	3	1.206259	26.55452	0
12	3	1.206259	26.51166	0
13	4	1.206259	26.4636	0
14	4	1.206259	26.41106	0
15	4	1.206259	26.35411	0
16	4	1.206259	26.29233	0
17	5	1.206259	26.22526	0
18	5	1.206259	26.15505	0
19	5	1.206259	26.07987	0
20	6	1.206259	26.00054	0
	...			
1958	150	1.565106	8.662871	0

4.2 Logika Agen

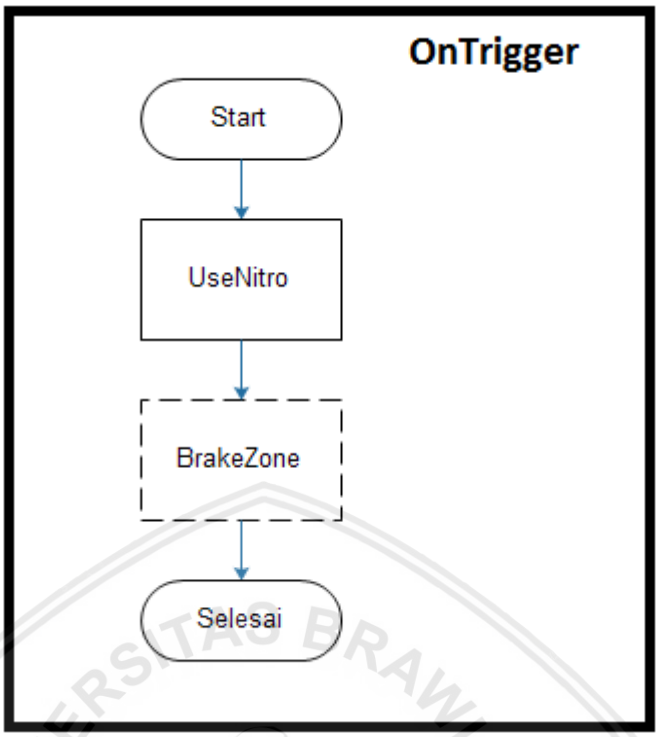
Seperti yang telah dijelaskan sebelumnya, penulis menggunakan *Neural network* dengan BackPropagation sebagai algoritma *learning*-nya untuk *braking decision* agen. Namun pada RGSK sudah tersedia sistem navigasi yang didalamnya terdapat *Braking Decision* dengan sistem *default kit*, sehingga diperlukan pemetaan dan perancangan logika agen yang akan kita gantikan dengan algoritma *Neural network*. Secara garis besar, Diagram *NPC* yang terdapat dalam RGSK tersebut ditunjukkan oleh Gambar 4.3.

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA



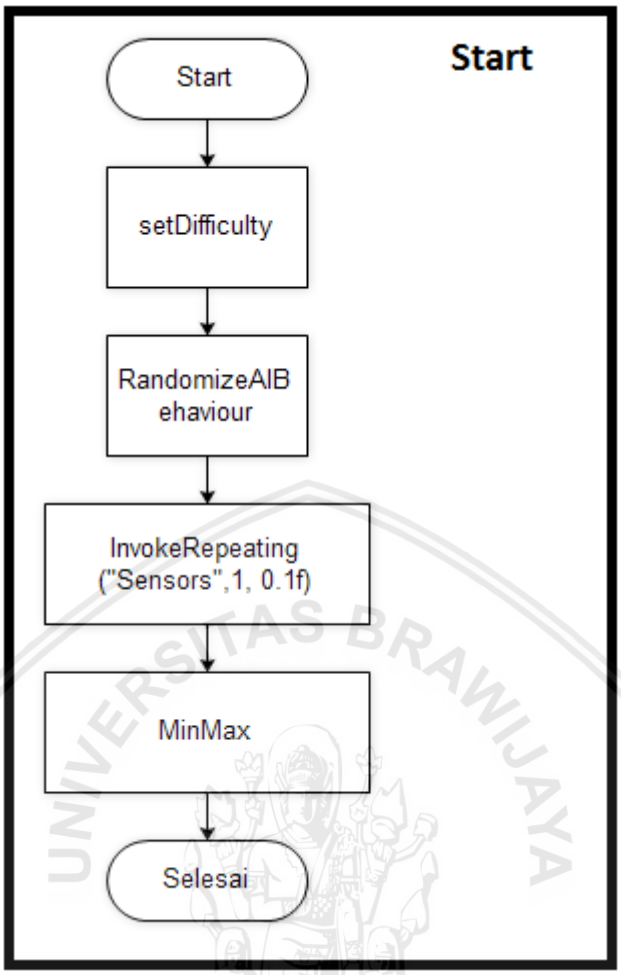
Gambar 4.3 *Flowchart* Logika Agen RGSK

Berikut penjelasan singkat mengenai *flowchart* diatas. Ketika Unity Engine dinyalakan maka proses awake akan langsung melakukan tugasnya yaitu inisialisasi. Kemudian apabila objek yang dipasang script tidak dalam mode disabled maka Proses Start akan dipanggil, dan akan melakukan tugasnya yaitu inisialisasi sesuai kode, lebih detail pada gambar 4.5. Setelah itu proses FixedUpdate dijalankan secara terus menerus selama objek aktif atau enabled, yang secara garis besar berisi hanya untuk physics dari game karena proses ini dijalankan dengan framerate yang tetap sehingga sinkron dengan physics engine. FixedUpdate berisi navigasi AI yang akan dibahas lebih detail pada sub *flowchart* gambar 4.6. Disusul setelahnya yaitu *OnCollisionEnter* untuk mengetahui apakah ada objek yang akan menabrak *collider* (*collision* membutuhkan *rigidbody* pada kedua objek), dalam hal ini digunakan untuk mengetahui apakah ada kendaraan lain di depan dan kemudian mengganti mode mobil menjadi mode menyalip atau *Overtaking*.



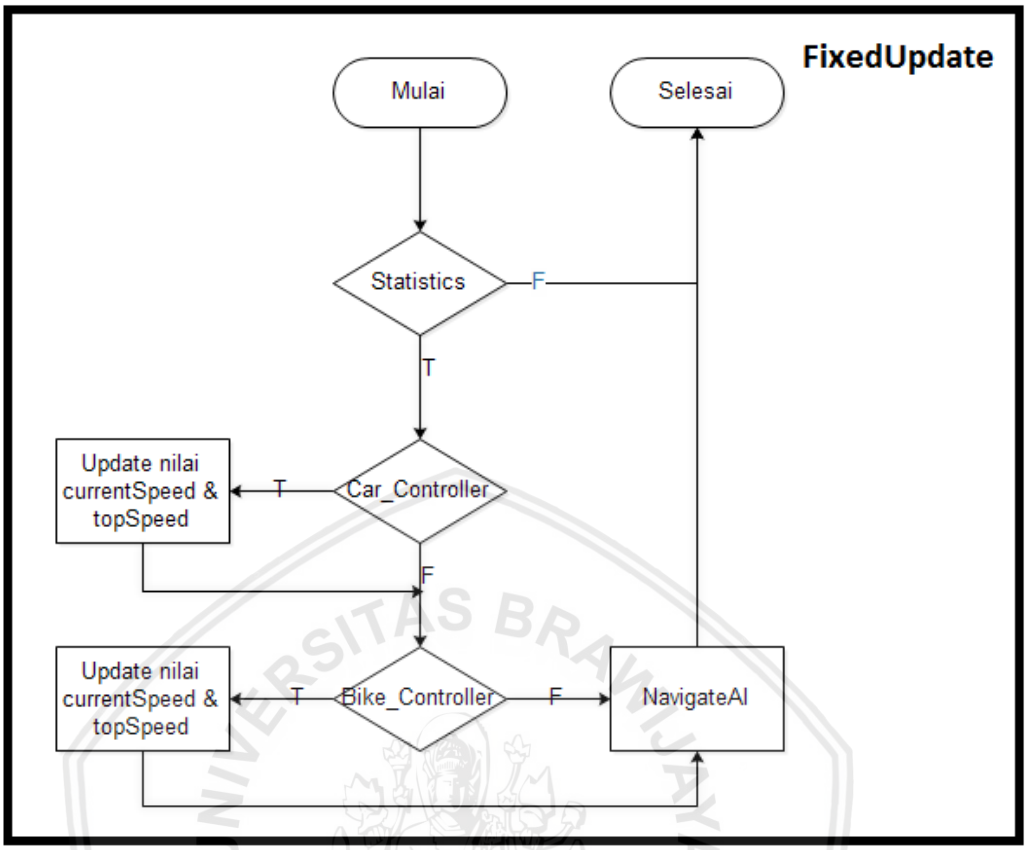
Gambar 4.4 . Detail proses OnTrigger pada *flowchart* logika agen RGSK

Gambar 4.4 berisi skema proses *OnTrigger,* yang digunakan untuk mengetahui apakah ada benda lain yang menyentuh *collider* namun tidak memiliki rigidbody sehingga tidak terjadi tabrakan atau *collision*. Karena pada versi 1.0.2 dilakukan perubahan proses navigasi pada bagian *braking decision*. Maka proses BrakeZone seperti pada gambar 4.4 dihapuskan. Kemudian proses update yang mirip dengan fixedupdate namun dijalankan setiap frame, dalam kasus ini digunakan untuk mengecek kondisi mobil apakah kendaraan tersangkut atau tidak, jika tersangkut maka AI akan memundurkan kendaraan, apabila kecepatan masih tetap dalam *range* tersangkut maka AI akan melakukan *respawn* kendaraan. Skema yang terakhir pada gambar 4.3 yaitu OnDrawGizmos, digunakan untuk memanggil gizmos atau peralatan yang dalam hal ini yaitu memanggil garis untuk memvisualisasikan sudut *waypoint* dan jarak mobil ke sudut seperti pada gambar 4.1 di awal bab.



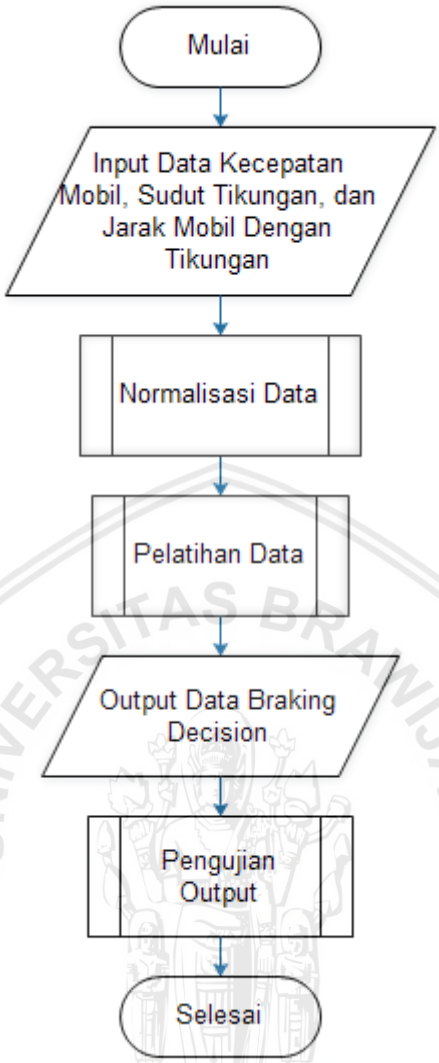
Gambar 4.5. Detail proses Start pada *flowchart* logika agen RGSK

Gambar 4.5 menunjukkan diagram dalam bagian Start, terdapat 3 proses yang dikerjakan yaitu SetDifficulty, InvokeRepeating proses Sensors dan RandomizeAIBehaviour. SetDifficulty digunakan untuk menentukan tingkat kesulitan dari AI, RandomizeAIBehaviour digunakan untuk inisiasi variabel-variabel untuk memberikan nilai acak untuk perilaku AI¸ dan Sensors akan dipanggil 1 detik setelah baris InvokeRepeating tersebut dijalankan dan kemudian dipanggil lagi setiap 0,1 detik. Sensors berfungsi sebagai deteksi bagi AI untuk mengetahui adanya kendaraan atau objek lain di sekitar kendaraan AI.

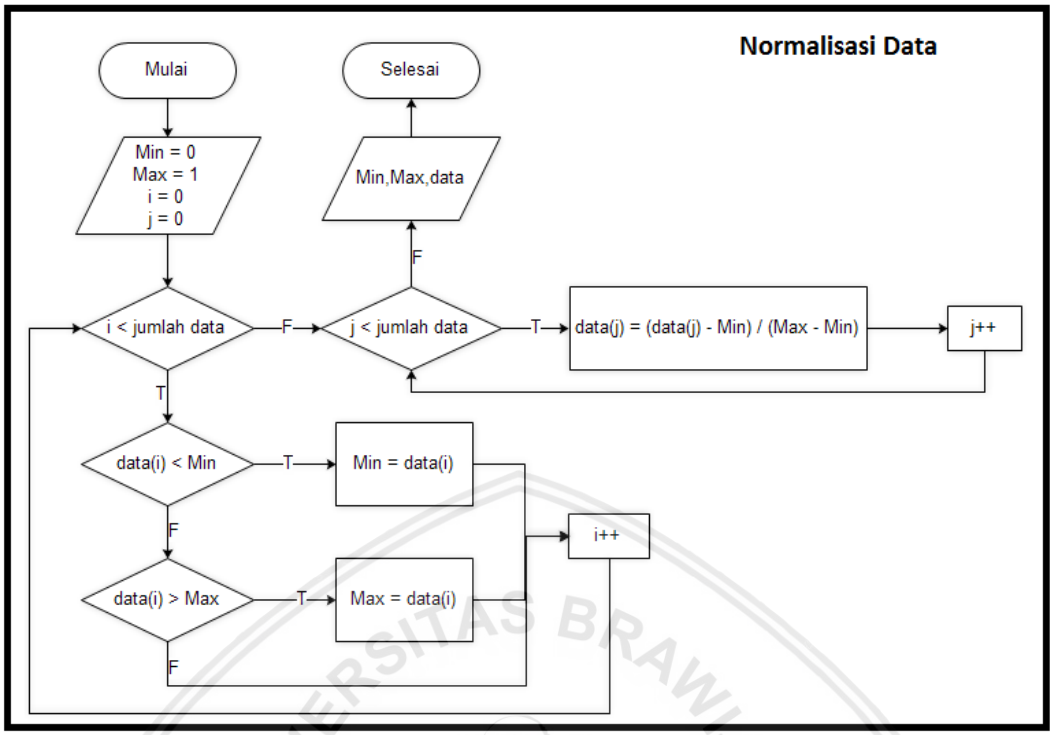


Gambar 4.6. Detail proses FixedUpdate pada *flowchart* logika agen RGSK

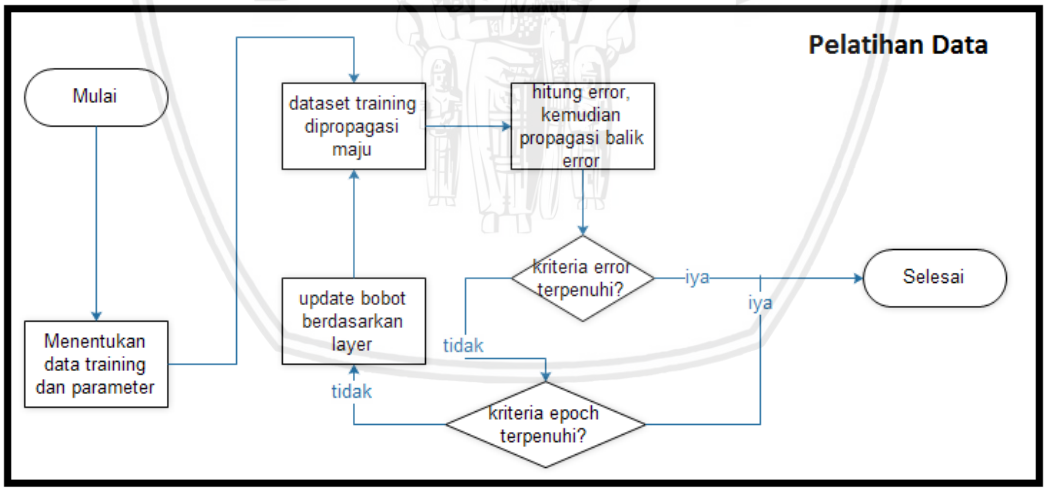
Gambar 4.6 menunjukkan *flowchart* proses FixedUpdate. Pada FixedUpdate dilakukan proses update nilai currentspeed dan topSpeed. Dan juga NavigateAI untuk menjalankan proses navigasi, yang berupa kemudi atau *steer* dan juga gas/rem atau throttle. Dalam penelitian ini difokuskan pada Braking Decision sehingga algoritma *Neural network* akan dirancang dan diimplementasikan pada proses NavigateAI seperti pada *flowchart* berikut.



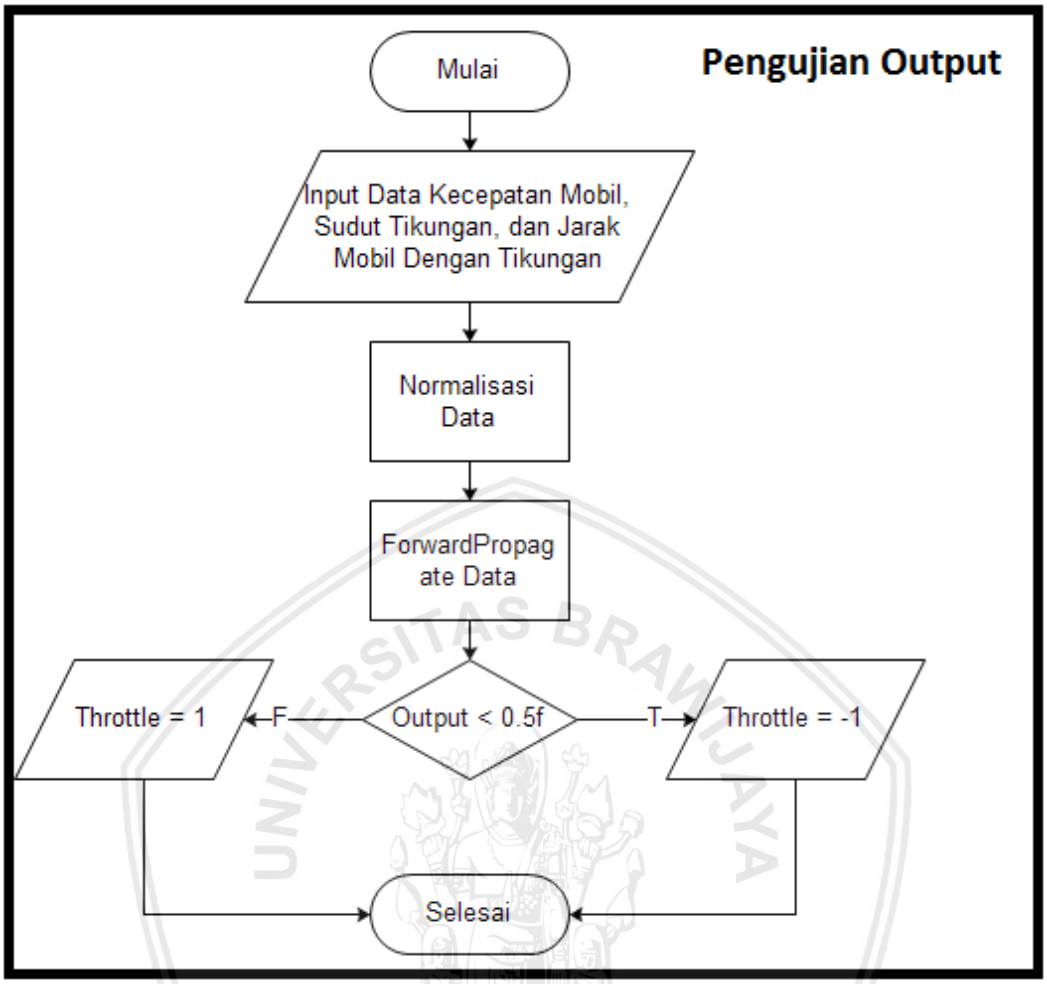
Gambar 4.7. *Flowchart* logika agen *Neural network*



Gambar 4.8. Detail *Flowchart* Normalisasi Data *Neural network*



Gambar 4.9. Detail *Flowchart* Pelatihan Data *Neural network*

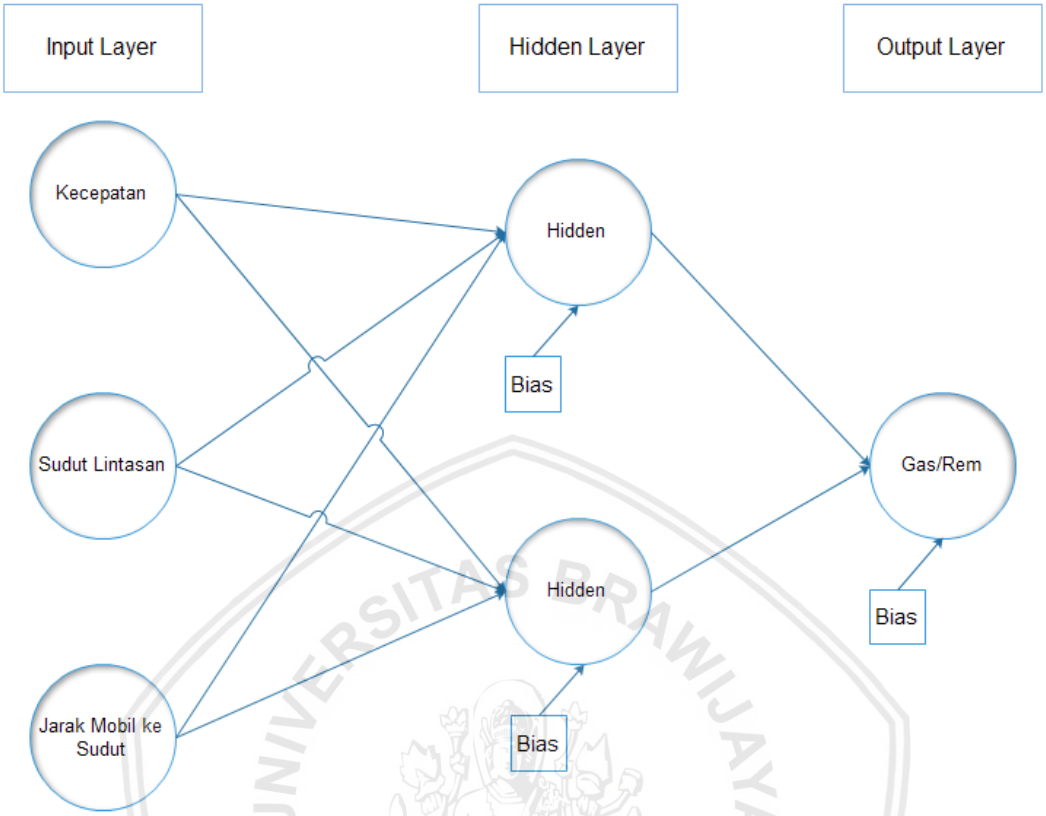


Gambar 4.10. Detail *Flowchart* Pengujian *Output Neural network*

Algoritma *Neural network* Backpropagation merupakan algoritma learning yang populer untuk memecahkan permasalahan yang rumit. Gambaran sederhana alur logika ini dapat dilihat pada gambar 4.7. Pada gambar 4.8 dilakukan normalisasi data guna mempermudah proses perhitungan sekaligus mengurangi waktu proses. Algoritma ini melakukan dua tahap perhitungan, yaitu: perhitungan propagasi maju untuk menghitung error antara keluaran dan target. Dan perhitungan propagasi mundur yang mempropagasikan balik error tersebut untuk memperbaiki bobot-bobot pada semua neuron yang ada, gambaran lebih detail dapat dilihat seperti pada gambar 4.9. sedangkan gambar 4.10 menjelaskan alur pengujian *output* setelah dilakukan pelatihan agen.

Sebelum melakukan perhitungan tahap pertama, terlebih dahulu kita tentukan elemen *neural network* yang paling dasar yaitu neuron buatan beserta strukturnya seperti pada gambar 4.11. Struktur *neural network* dengan 1 *hidden layer* dipilih karena semakin banyak lapisan jaringan saraf, maka proses pelatihan dan proses running program akan semakin lambat, dan menurut pondasi matematika, *neural network* dengan satu atau dua *hidden layer* dapat belajar sebaik deep *neural network* dengan lusinan *hidden layer* (Soarez F.M, 2016).

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA



Gambar 4.11 *Struktur Neural network*

Kita lakukan inisialisasi jumlah neuron beserta bobot tiap-tiap synapse (sambungan yang menghubungkan neuron satu dengan yang lain). Tahap selanjutnya yaitu perhitungan maju atau forward propagate. Untuk propagasi maju pertama, kumpulan bobot(*w*) dipilih secara acak. Kemudian kita jumlahkan produk *input* dengan set bobot yang sesuai untuk mendapatkan nilai pertama untuk *hidden layer*. Bobot merupakan ukuran pengaruh yang dimiliki node *input* pada *output*. Nilai pertama *hidden layer* yang baru saja didapatkan merupakan nilai sementara. Untuk mendapatkan nilai akhir, ditambahkan bias (*b*) lalu diterapkan fungsi aktivasi (*f*) ke jumlah dari *Hidden layer*. Tujuan dari fungsi aktivasi adalah untuk mengubah sinyal *input* menjadi sinyal *output* dan diperlukan untuk jaringan syaraf tiruan untuk memodelkan pola non-linear kompleks yang mungkin terlewatkan oleh model yang lebih sederhana. Bias dapat membantu dalam representasi pengetahuan jaringan saraf sebagai sistem nonlinear yang lebih murni, dengan ketentuan bahwa ketika semua *input* adalah nol, neuron itu tidak akan selalu menghasilkan nol pada *output*, melainkan dapat menembakkan nilai yang berbeda sesuai dengan bobot yang terkait dengan bias. Perhitungan seperti gambar berikut ini dilakukan lagi untuk menentukan nilai akhir pada *output* layer.

$$y_o = f_o\left(\sum_{i=1}^{n_{h_l}} w_i f_i\left(\sum_{j=1}^{n_{h_{l-1}}} w_{ij} f_j\left(\sum_{k=1}^{n_{h_{l-2}}} w_{jk} f_k(\cdots) + b_j\right) + b_i\right) + b_o\right) \quad (4.1)$$

Tahap selanjutnya yaitu perhitungan *backpropagation* atau propagasi balik. Serupa dengan propagasi maju, kalkulasi *backpropagate* terjadi di setiap layer. Kita mulai dengan mengubah bobot antara *hidden layer* dan *output* layer. Menghitung perubahan tambahan untuk bobot ini terjadi dalam dua langkah. Langkah pertama kita menemukan margin kesalahan hasil *output* untuk mundur dan merubah apa yang diperlukan dalam jumlah *output* (*output* delta). Langkah kedua, kita mengekstrak perubahan dalam bobot dengan mengalikan jumlah *output* delta dengan hasil *hidden layer*. Jumlah margin *output* kesalahan adalah hasil *output* target(*o*) dikurangi hasil *output* yang dihitung(*t*). Untuk menghitung perubahan yang diperlukan dalam jumlah *output*, atau jumlah *output* delta, kita mengambil turunan dari fungsi aktivasi(*f'*) dan menerapkannya ke jumlah *output*. Jadi turunan dari sigmoid, juga dikenal sebagai sigmoid prime, akan memberi kita tingkat perubahan (atau "kemiringan gradient") dari fungsi aktivasi pada jumlah keluaran.

$$\delta_i = \left(o_i - t_i\right) f'\left(h_i\right) \quad (4.2)$$

Setelah kita mendapat Gradient pada *output* layer, kita dapat memulai perhitungan gradient pada hiddenlayer.

$$\delta_i = \sum_l \delta_l w_{il} f'\left(h_i\right) \quad (4.3)$$

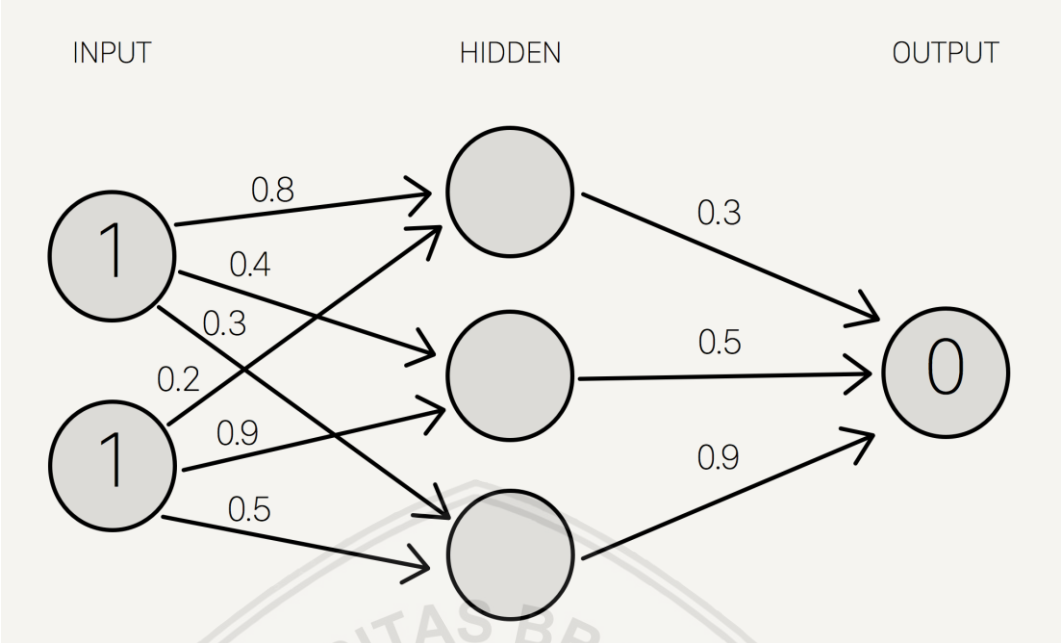
Di penelitian ini, kami mengukur perubahan dengan besaran learningRate, yaitu 0.4 . Tingkat pembelajaran menerapkan porsi yang lebih besar atau lebih kecil dari penyesuaian masing-masing dengan berat yang lama. Jika ada variabilitas besar dalam *input* (ada sedikit hubungan di antara data pelatihan) dan tingkat learningRate tinggi, maka jaringan mungkin tidak belajar dengan baik atau malah tidak belajar sama sekali. Menyetel learningRate terlalu tinggi juga akan mendapatkan risiko *overfitting*. Perhitungan berikutnya adalah *update weight* atau penyesuaian bobot. Operasi penyesuaian bobot dilakukan dalam mode batch. Penyesuaian batch berarti bahwa pertama semua catatan dari dataset disajikan ke *neural network* sebelum mulai memperbarui bobotnya.

$$w_{ji}\left(k+1\right) = w_{ji}\left(k\right) + \Delta w_{ji}\left(k\right) + \mu \Delta w_{ji}\left(k-1\right) \quad (4.4)$$

Jika *output* bobot tidak sesuai kriteria maka akan kembali lagi ke tahap propagasi maju dan terus diulangi hingga memenuhi kriteria yang ditetapkan yaitu minimal error 10% atau maksimum epoch (jumlah iterasi pembelajaran) 20.000.

Berikut merupakan contoh perhitungan sederhana algoritma *Neural network* BackPropagation.

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

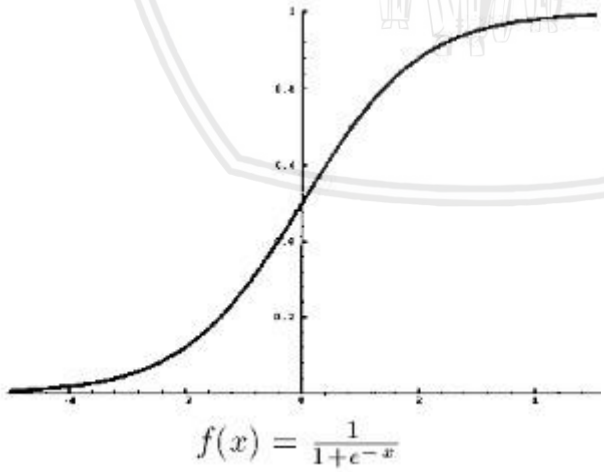


Gambar 4.12 Contoh Struktur *Neural network*

Tahap *Forward Propagate* 1 : jumlahkan produk *input* dengan set bobot yang sesuai untuk mendapatkan nilai pertama untuk *hidden layer*

```
1 * 0.8 + 1 * 0.2 = 1
1 * 0.4 + 1 * 0.9 = 1.3
1 * 0.3 + 1 * 0.5 = 0.8
```

Tahap *Forward Propagate* 2 : diterapkan fungsi aktivasi (*f*) sigmoid ke jumlah dari *Hidden layer*




```
S(1.0) = 0.73
S(1.3) = 0.79
S(0.8) = 0.69
```

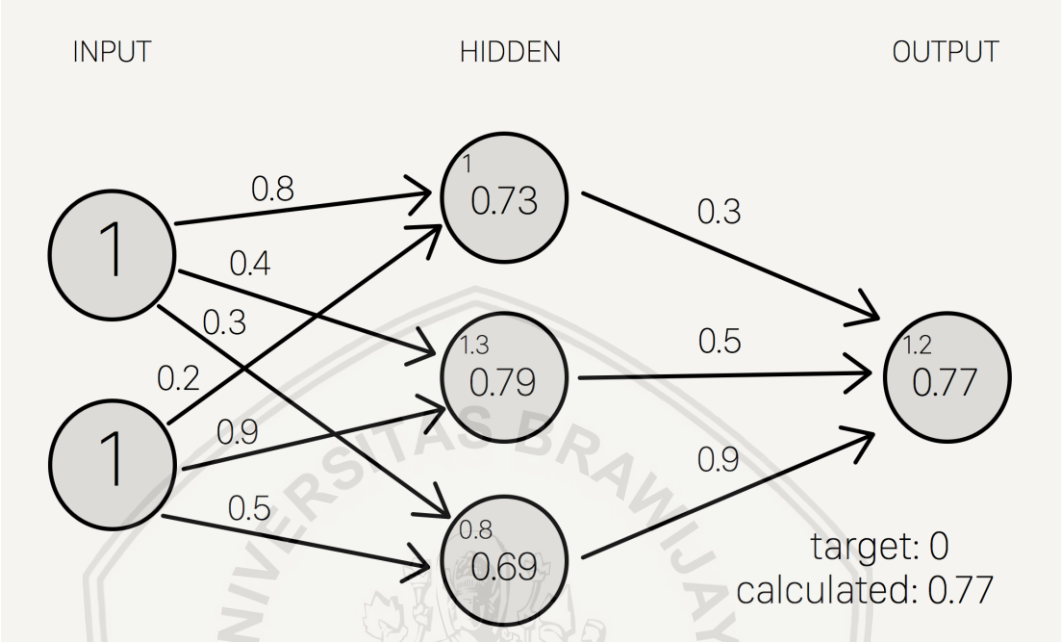
Tahap *Forward Propagate* 3 : jumlahkan produk *input* dengan set bobot yang sesuai untuk mendapatkan nilai pertama untuk *Output* Layer

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

```
0.73 * 0.3 + 0.79 * 0.5 + 0.69 * 0.9 = 1.235
```

Tahap *Forward Propagate* 4 : diterapkan fungsi aktivasi (*f*) sigmoid ke jumlah dari *Output* Layer

```
S(1.235) = 0.77
```



Gambar 4.13 Contoh Struktur *Neural network*

Tahap *BackPropagation* 1 : menemukan margin kesalahan hasil *output* untuk mundur dan merubah apa yang diperlukan dalam jumlah *output* (*output* delta)

```
Target = 0
Calculated = 0.77
Target - calculated = -0.77
```

Tahap *BackPropagation* 2 : kita mengekstrak perubahan dalam bobot dengan mengalikan jumlah *output* delta dengan hasil *hidden layer*. Untuk menghitung perubahan yang diperlukan dalam jumlah *output*, atau jumlah *output* delta, kita mengambil turunan dari fungsi aktivasi(*f'*) dan menerapkannya ke jumlah *output*. *F'(x)=f(x)(1−f(x)).*

```
Delta output sum = S'(sum) * (output sum margin of error)
Delta output sum = S'(1.235) * (-0.77)
Delta output sum = -0.1344
```

Tahap *BackPropagation* 3 : Perhitungan berikutnya adalah *update weight* atau penyesuaian bobot.

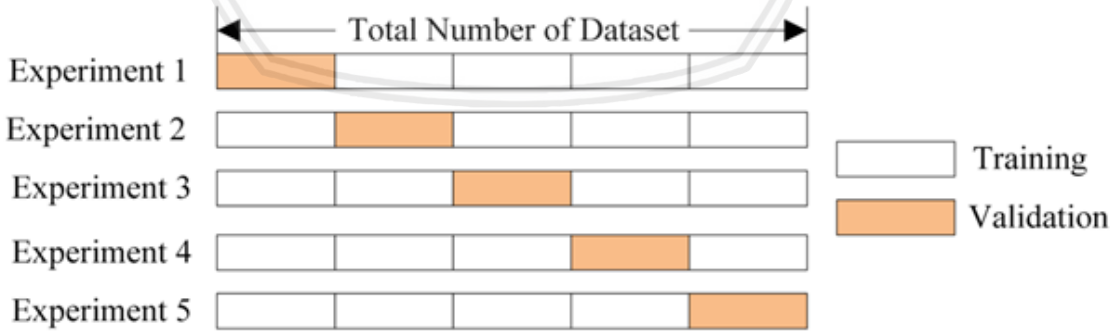
```
hidden result 1 = 0.73
hidden result 2 = 0.79
hidden result 3 = 0.69

Delta weights = delta output sum / hidden layer results
Delta weights = -0.1344 / [0.73, 0.79, 0.69]
Delta weights = [-0.1838, -0.1710, -0.1920]
```


repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

```
old w7 = 0.3
old w8 = 0.5
old w9 = 0.9

new w7 = 0.1162
new w8 = 0.329
new w9 = 0.708

Delta hidden sum = delta output sum / hidden-to-outer weights *
S'(hidden sum)
Delta hidden sum = -0.1344 / [0.3, 0.5, 0.9] * S'([1, 1.3, 0.8])
Delta hidden sum = [-0.448, -0.2688, -0.1493] * [0.1966, 0.1683,
0.2139]
Delta hidden sum = [-0.088, -0.0452, -0.0319]

old w1 = 0.8
old w2 = 0.4
old w3 = 0.3
old w4 = 0.2
old w5 = 0.9
old w6 = 0.5

new w1 = 0.712
new w2 = 0.3548
new w3 = 0.2681
new w4 = 0.112
new w5 = 0.8548
new w6 = 0.4681
```

4.3 K-Fold Cross-Validation

Diantara sejumlah strategi untuk memvalidasi *neural network*, yang sangat penting adalah metode *cross-validation*. Strategi ini memastikan bahwa semua data telah disajikan ke *neural network* sebagai pelatihan dan data uji. Dataset dipartisi ke dalam grup sejumlah K, yang satu dipisahkan untuk pengujian sementara yang lain untuk proses *learning*.



Gambar 4.14 Skema K-Fold Cross-Validation

BAB 5 IMPLEMENTASI

Bab ini menjelaskan hasil implementasi program untuk menerapkan *neural network* pada agen. Implementasi script dilakukan dengan integrated development environment (IDE) MonoDevelop yang tersedia dalam Game engine Unity. Script dieksekusi ketika permainan berjalan. Script akan dibagi berdasarkan perancangan pada bab sebelumnya. Kode yang disertakan pada setiap tabel merupakan kode yang ditambahkan oleh penulis.

5.1 Data Training

Pada Tabel 5.1 merupakan kode untuk class PlayerControl. Class PlayerControl merupakan class yang mengatur *input* dari player pada program ini. Berikut akan dijelaskan potongan kode pada tabel secara baris per baris.

- Pada baris 1 hingga 15 adalah proses yang terdapat pada fixed Update, yang bertugas untuk memanggil proses CaptureData() setiap 0.1 detik.
- Pada baris 14 hingga 28. Dilakukan pencatatan 4 data yang masuk ke fungsi selanjutnya yaitu, `currentSpeed`, `targetAngle`, `distance`, dan `datatest`. Yang akan dilakukan proses train pada baris 26. Proses train akan dibahas pada tabel 5.2.
- Selajutnya pada baris ke 29 hingga 53, merupakan proses Send*Inputs* yang bertugas untuk mengirim *input* yang dihasilkan *controller* milik *player* untuk proses selanjutnya. Dilakukan pencatatan data kembali setiap player melakukan pengereman, seperti pada baris 33 hingga 37.

Tabel 5.1 Fungsi Data Capturing

No	Kode Program
1	`void FixedUpdate()`
2	`{`
3	`if (!stats)`
4	`return;`
5	
6	`if (car_controller) {`
7	`if (captureTimer > 0.1f) {`
8	`captureTimer = 0;`
9	`CaptureData ();`
10	`} else {`
11	`captureTimer += Time.deltaTime;`
12	`}`
13	`currentSpeed = car_controller.currentSpeed;`
14	`}`
15	`}`
16	`void CaptureData(){`
17	`var DataTest = (car_controller.motorInput > 0) ? 0 : 1;`
18	`var distancee = Vector3.Distance (transform.position, stats.PassedNodeB.position);`
19	`//Debug.Log ("ok");`

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

```
20              var directionA = (stats.PassedNodeB.position - stats.PassedNodeA.position).normalized;
21              var directionB = (stats.PassedNodeC.position - stats.PassedNodeB.position).normalized;
22              var directionC = (stats.PassedNodeC.position - stats.PassedNodeB.position).normalized;
23              var directionD = (stats.PassedNodeA.position - stats.PassedNodeB.position).normalized;
24              var targetAngleAB = Vector3.Angle (directionA,directionB);
25              if(ANNlearn){
26                  ANN.Train (currentSpeed,targetAngleAB,distancee,DataTest);
27              }
28          }
29  void SendInputs(float accel, float brake, float steer, float handbrake, bool nitro)
30          {
31              if (car_controller)
32              {
33                  car_controller.motorInput = (brake <= 0) ? accel : 0;
34                  if (car_controller.motorInput == 0 && pushButtonTimer >= 1.0f) {
35                      CaptureData ();
36                      pushButtonTimer = 0;
37                  } else {
38                      pushButtonTimer += Time.deltaTime;
39                  }
40                  car_controller.brakeInput = brake;
41                  car_controller.steerInput = steer;
42                  car_controller.handbrakeInput = handbrake;
43                  car_controller.usingNitro = nitro;
44              }
45
46              if (bike_controller)
47              {
48                  bike_controller.motorInput = (brake <= 0) ? accel : 0;
49                  bike_controller.brakeInput = brake;
50                  bike_controller.steerInput = steer;
51                  bike_controller.usingNitro = nitro;
52              }
53          }
```

Pada Tabel 5.2 adalah potongan kode dari class CarANN, Script ini merupakan lanjutan dari proses data training yaitu pada tabel 5.1. Berikut adalah penjelasan lebih lanjut:

- Pada baris 1 terdapat 4 data yang masuk ke fungsi Train yaitu, `currentSpeed, targetAngle, distance, dan val.`
- Baris ke 5 merupakan deklarasi variabel path yang berguna sebagai alamat untuk menyimpan file text yang bisa dimanfaatkan untuk kedepannya.

- Pada baris 7 hingga 9 kita menyimpan 4 data tersebut ke dalam file text dengan metode StreamWriter.
- Pada baris 10 kita menyimpan data ke dalam array allData.
- Kemudian pada baris 12, kita kemudian mengecek apakah data yang masuk telah terkumpul hingga sejumlah captureData dan mobil dalam keadaan belum trained.
- Pada baris 13, Apabila bernilai true, maka semua adata akan di normalisasi menggunakan metode min-max
- Dan dilanjutkan dengan fungsi Train pada baris ke 14 yang akan dibahas pada sub bab selanjutnya yaitu logika agen.
- Sedangkan baris 15 bertugas untuk menyimpan class NeuralNet yang juga akan dibahas pada sub bab logika agen ke dalam file newbi.sa dan akan berguna apabila kita ingin me-load kecerdasan yang dihasilkan untuk sesi-sesi yang lain.

Tabel 5.2 Fungsi Data Training

No	Kode Program
1	`public void Train(float currentSpeed, float targetAngle, float distance, float val)`
2	`{ double currentSpeed2 = currentSpeed;`
3	`double targetAngle2 = targetAngle;`
4	`double distance2 = distance;`
5	`string path = @"D:\Unity-Neural-Network-master\Assets\Datatest v0.2.txt";`
6	`// Create a file to write to.`
7	`using (StreamWriter sw = new StreamWriter(path, true))`
8	`{`
9	`sw.WriteLine(currentSpeed2+","+targetAngle2+","+distance2+","+val);`
10	`allData[i] = new double[] {currentSpeed2, targetAngle2,distance2,val};`
11	`}`
12	`if (!trained && i == captureData) {`
13	`normalize (allData);`
14	`Train ();`
15	`Serializer.Save<NeuralNet>(Application.persistentDataPath + "/newbi.sa", net);`
16	`}`
17	`i++;`
18	`}`

5.2 Logika Agen

Pada Tabel 5.3, class NeuralNet adalah class untuk memproses perhitungan *Neural network* mulai dari inisialisasi neuron hingga proses learning neuron selesai.

- Pada baris 1 hingga 9 yaitu inisialisasi bagian-bagian *neural network* mulai dari jumlah *input*, hidden, dan *output* layer, kemudian nilai learnrate, dan juga nilai momentum

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

- Kemudian pada baris 10 hingga 12 proses inisialisasi jumlah neuron untuk *input* layer
- Baris 14 hingga 20 proses inisialisasi jumlah neuron untuk *hidden layer*
- Dan baris 22 hingga 25 adalah proses inisialisasi jumlah neuron untuk *output* layer.

Tabel 5.3 Fungsi *Neural network*

No	Kode Program
1	`public NeuralNet(int inputSize, int hiddenSize, int outputSize,`
2	`int numHiddenLayers = 1, double? learnRate = null, double?`
3	`momentum = null)`
4	`{ Instance = this;`
5	`LearnRate = learnRate ?? .4;`
6	`Momentum = momentum ?? .9;`
7	`InputLayer = new List<Neuron>();`
8	`HiddenLayers = new List<List<Neuron>>();`
9	`OutputLayer = new List<Neuron>();`
10	`for (var i = 0; i < inputSize; i++) {`
11	`InputLayer.Add (new Neuron ());`
12	`}`
13	
14	`for (int i = 0; i < numHiddenLayers; i++)`
15	`{`
16	`HiddenLayers.Add(new List<Neuron>());`
17	`for (var j = 0; j < hiddenSize; j++) {`
18	`HiddenLayers [i].Add (new Neuron (i == 0 ? InputLayer : HiddenLayers [i - 1]));`
19	`}`
20	`}`
21	
22	`for (var i = 0; i < outputSize; i++) {`
23	`OutputLayer.Add (new Neuron (HiddenLayers [numHiddenLayers - 1]));`
24	`}`
25	`}`

Tabel 5.4 adalah class NeuralNet lanjutan, disinilah proses-proses perhitungan neuron pada tabel 5.5 akan digunakan. Disini terdapat fungsi utama yang akan menunjukkan alur-alur perhitungan mulai data masuk hingga proses belajar selesai.

- Dimulai dari baris 10 yaitu memasukkan 3 jenis *input* dataSet[0] yang merupakan *input* kecepatan mobil, dataSet[1] sudut lintasan, dan dataSet[2] jarak mobil ke sudut lintasan, ke dalam variabel dataSetTrain.
- Kemudian baris 11 akan memanggil fungsi ForwardPropagate
  - Pada baris 20 hingga 28 merupakan fungsi ForwardPropagate dengan parameter dataSetTrain diatas.
  - Pada baris 23 yaitu alur ForwardPropagate pertama yang akan memasukkan nilai dataSet[0], dataSet[1], dataSet[2] ke dalam masing-masing neuron yang berada pada *Input*Layer sebagai Value awal.

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

- Pada baris 24 hingga 26 akan menghitung Value untuk masing-masing *hidden layer*.
- Baris 27 akan menghitung Value untuk *output* layer yang digunakan sebagai nilai akhir tahap forwardpropagate.

- Pada baris 12 merupakan tahap backpropagate dengan memanggil fungsi BackPropagate.
  - Pada baris 30 hingga 39 adalah fungsi BackPropagate dengan parameter *input* dataSet[3].
  - Pada baris 33 adalah awal fungsi backpropagate yang akan melakukan perhitungan gradient pada *output* layer yaitu dengan cara menghitung besar error dengan mengurangi nilai target dengan nilai akhir yang didapat dari forwardpropagate dan mengalikannya dengan turunan dari fungsi aktivasi sigmoid dengan parameter *input* nilai akhir dari tahap forwardpropagate.
  - Pada baris ke 34 hingga 37, secara terbalik list neuron di *hidden layer* akan dihitung besar gradient dan kemudian dilakukan update weight berdasarkan learnRate, BiasDelta, momentum, gradient dan nilai masing-masing synapse.
  - Pada baris 38 adalah proses penyesuaian bobot dengan fungsi UpdateWeight pada semua neuron di *Output*Layer.
- Baris 41 hingga 45 hanya terdiri dari forward propagate dan berguna untuk proses testing.
- Baris 47 hingga 51 untuk menghitung error pada fungsi utama Train diatas.
- Baris 53 hingga 58 untuk mendapatkan nilai random dengan seed, Ini akan memastikan bahwa pola acak yang sama dihasilkan setiap kali game dimainkan. Kita dapat menggunakan elemen yang dipilih secara acak secara prosedural namun tetap terlihat alami.

Tabel 5.4 Fungsi *Neural network* Lanjutan

No	Kode Program
1	`public void Train(double[][] allData,  int numEpochs, double minimumError){`
2	`  var error = 1.0;`
3	`  var epoch = 0;`
4	
5	`  while (error > minimumError && epoch < numEpochs)`
6	`  {`
7	`  var errors = new List<double>();`
8	`  foreach (var dataSet in allData)`
9	`  {`
10	`  double[] dataSetTrain = {dataSet[0],dataSet[1],dataSet[2]};`
11	`  ForwardPropagate(dataSetTrain);`
12	`  BackPropagate(dataSet[3]);`
13	`  errors.Add(CalculateError(dataSet[3]));`
14	`  }`
15	`  error = errors.Average();`
16	`  epoch++;`
17	`  }`

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

```
18  }
19
20  private void ForwardPropagate(params double[] inputs)
21  {
22        var i = 0;
23        InputLayer.ForEach(a => a.Value = inputs[i++]);
24        foreach (var layer in HiddenLayers) {
25              layer.ForEach (a => a.CalculateValue ());
26        }
27        OutputLayer.ForEach(a => a.CalculateValue());
28  }
29
30  private void BackPropagate(params double[] targets)
31  {
32        var i = 0;
33        OutputLayer.ForEach(a => a.CalculateGradient(targets[i++]));
34        foreach(var layer in
    HiddenLayers.AsEnumerable<List<Neuron>>().Reverse() )
35        {     layer.ForEach(a => a.CalculateGradient());
36              layer.ForEach(a => a.UpdateWeights(LearnRate,
    Momentum));
37        }
38                    OutputLayer.ForEach(a =>
    a.UpdateWeights(LearnRate, Momentum));
39  }
40
41  public double[] Compute(params double[] inputs)
42  {
43        ForwardPropagate(inputs);
44        return OutputLayer.Select(a => a.Value).ToArray();
45  }
46
47  private double CalculateError(params double[] targets)
48  {
49        var i = 0;
50        return OutputLayer.Sum(a =>
    Mathf.Abs((float)a.CalculateError(targets[i++])));
51  }
52
53  public double GetRandom()
54  {
55        Random1 = new System.Random (Seed);
56        Seed++;
57        return (2 * Random1.NextDouble () - 1);
58  }
```

Tabel 5.5 adalah class Neuron, classini berisi inisialisasi dan beberapa perhitungan neuron.

- Baris 1 hingga 6 merupakan inisialisasi neuron tanpa parameter, fungsi ini digunakan pada bagian awal yaitu *input* layer.
- Baris 8 hingga 16 adalah inisialisasi neuron dengan parameter `IEnumerable<Neuron>` yang digunakan pada bagian *hidden layer* dan *output* layer. Pada *hidden layer* parameter diisi dengan neuron `InputLayer`.

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

Sedangkan pada *output* layer parameter diisi dengan neuron `HiddenLayers[0]`.

- Pada baris 56 hingga 62 adalah fungsi inisialisasi neuron yang kedua, dan akan memanggil fungsi inisialisasi synapse.
- Selanjutnya pada baris 18 hingga 21 terdapat fungsi perhitungan neuron pada class ini. Perhitungan nilai neuron `CalculateValue()` pada baris ini digunakan pada tahap *forward propagate*. Seperti yang telah dijelaskan sebelumnya, fungsi ini mengalikan masing-masing bobot synapse dengan nilai *input* yang masuk melalui *input* layer, kemudian menjumlahkan semua hasil perkalian tadi.
- Setelah itu pada baris 64 hingga 69 ditambahkan bias lalu diterapkan fungsi aktivasi sigmoid.
- Pada baris 28 hingga 34 adalah perhitungan neuron pada tahap *back propagate* yaitu `CalculateGradient`
- Baris 36 hingga 46 untuk menghitung bobot synapse yang baru yaitu `UpdateWeights`.

.

Tabel 5.5 Fungsi Neuron

No	Kode Program
1	`public Neuron()`
2	`{`
3	`InputSynapses = new List<Synapse>();`
4	`OutputSynapses = new List<Synapse>();`
5	`Bias = NeuralNet.Instance.GetRandom();`
6	`}`
7	
8	`public Neuron(IEnumerable<Neuron> inputNeurons) : this()`
9	`{`
10	`foreach (var inputNeuron in inputNeurons)`
11	`{`
12	`var synapse = new Synapse(inputNeuron, this);`
13	`inputNeuron.OutputSynapses.Add(synapse);`
14	`InputSynapses.Add(synapse);`
15	`}`
16	`}`
17	
18	`public virtual double CalculateValue()`
19	`{ Value = Sigmoid.Output(InputSynapses.Sum(a => a.Weight * a.InputNeuron.Value) + Bias);`
20	`return Value;`
21	`}`
22	
23	`public double CalculateError(double target)`
24	`{`
25	`return target - Value;`
26	`}`
27	
28	`public double CalculateGradient(double? target = null)`
29	`{`

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

```
30                  if (target == null) {
31                  return Gradient = OutputSynapses.Sum (a =>
    a.OutputNeuron.Gradient * a.Weight) * Sigmoid.Derivative (Value);
32                  }
33                  return Gradient = CalculateError(target.Value) *
    Sigmoid.Derivative(Value);
34          }
35
36  public void UpdateWeights(double learnRate, double momentum)
37          {
38                  var prevDelta = BiasDelta;
39                  BiasDelta = learnRate * Gradient;
40                  Bias += BiasDelta + momentum * prevDelta;
41                  foreach (var synapse in InputSynapses)
42                  {       prevDelta = synapse.WeightDelta;
43                          synapse.WeightDelta = learnRate * Gradient *
    synapse.InputNeuron.Value;
44                          synapse.Weight += synapse.WeightDelta + momentum
    * prevDelta;
45                  }
46          }
47
48  [System.Serializable]
49  public class Synapse
50  {               public Neuron InputNeuron { get; set; }
51                  public Neuron OutputNeuron { get; set; }
52                  public double Weight { get; set; }
53                  public double WeightDelta { get; set; }
54                  public static NeuralNet Neuron;
55
56  public Synapse(Neuron inputNeuron, Neuron outputNeuron)
57          {
58                  InputNeuron = inputNeuron;
59                  OutputNeuron = outputNeuron;
60                  Weight = NeuralNet.Instance.GetRandom();
61                  }
62  }
63
64  public static class Sigmoid
65  {
66          public static double Output(double x)
67          {
68                  return x < -45.0 ? 0.0 : x > 45.0 ? 1.0 : 1.0 / (1.0 +
    Mathf.Exp((float)-x));
69          }
70
71          public static double Derivative(double x)
72          {
73                  return x * (1 - x);
74          }
75  }
```

Berikut ini adalah cuplikan class CarANN.cs dengan tujuan utama sebagai jembatan antara algoritma *neural network* dengan script kendaraan.

- Pada baris 1 hingga 17 merupakan fungsi untuk proses data testing dengan kecerdasan yang telah didapatkan.

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

- Baris 18 hingga 49 merupakan proses normalisasi dengan metode min-max.
- baris 50 hingga 73 merupakan metode untuk mengurutkan data mulai dari besar ke kecil dan kecil ke besar.
- Baris 75 hingga 85 merupakan fungsi utama untuk data training.
- Baris 87 hingga 95 adalah sub bagian dari proses data testing.
- Baris 96 hingga 106 digunakan untuk mengacak data untuk test akurasi.

Tabel 5.6 Fungsi Data Testing

No	Kode Program
1	`public float Next(float currentSpeed, float targetAngle, float distance)`
2	`{`
3	`float throttle = 1;`
4	`currentSpeed = (float)Math.Round`
5	`(currentSpeed, 1);`
	`targetAngle = (float)Math.Round (targetAngle,`
6	`1);`
	`distance = (float)Math.Round (distance, 1);`
7	`double[] C = {(double)currentSpeed,`
8	`(double)targetAngle, (double)distance};`
	`if (trained) {`
9	`double d = tryValues (C);`
10	`if (d < 0.5f)`
11	`throttle = -1;`
12	`else`
13	`throttle = 1;`
14	`}`
15	`return throttle;`
16	
17	`}`
18	`private void normalize(double[][] inputs){`
19	`int ymax = 1;`
20	`int ymin = 0;`
21	`double[] currentSpeeds = new`
22	`double[captureData];`
	`double[] targetAngles = new`
23	`double[captureData];`
	`double[] distances = new double[captureData];`
24	`for (int i = 0; i < captureData; i++) {`
25	`currentSpeeds[i] = inputs[i][0];`
26	`targetAngles [i] = inputs[i][1];`
27	`distances [i] = inputs[i][2];`
28	`}`
29	`double maxSpeed = MaxIndex (currentSpeeds);`
30	`double maxAngle = MaxIndex (targetAngles);`
31	`double maxDistance = MaxIndex (distances);`
32	`double minSpeed = MinIndex(currentSpeeds);`
33	`double minAngle = MinIndex(targetAngles);`
34	`double minDistance = MinIndex(distances);`
35	
36	`string path = @"D:\Unity-Neural-Network-`
37	`master\Assets\MinMax.txt";`
	`// Create a file to write to.`
38	`using (StreamWriter sw = new`
39	`StreamWriter(path, false))`
	`{`

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

```
40                          sw.Write(maxSpeed+", "+maxAngle+", "+
41  maxDistance+", "+ minSpeed+", "+minAngle+", "+minDistance);
                    }
42                  for(int i = 0; i < captureData; i++){
43                          var normalizedCurrentSpeed =
44  Math.Round(((ymax-ymin) * (inputs[i][0]-minSpeed) / (maxSpeed-
    minSpeed) + ymin),1);
                            var normalizedTargetAngle =
45  Math.Round(((ymax-ymin) * (inputs[i][1]-minAngle) / (maxAngle-
    minAngle) + ymin),1);
                            var normalizedDistance =
46  Math.Round(((ymax-ymin) * (inputs[i][2]-minDistance) /
    (maxDistance-minDistance) + ymin),1);

47                          allData[i] = new double[]
48  {normalizedCurrentSpeed, normalizedTargetAngle,
    normalizedDistance,inputs[i][3]};
                    }
49              }
50          private static double MaxIndex(double[] vector) // helper
51  for Accuracy()
                {
52                  double biggestVal = vector[0];
53                  for (int i = 0; i < vector.Length; ++i)
54                  {       if (vector[i] > biggestVal)
55                          {
56                                  biggestVal = vector[i];
57                          }
58                  }
59                  return biggestVal;
60              }
61
62          private static double MinIndex(double[] vector) // helper
63  for Accuracy()
                {
64                  double smallestVal = 1000;
65                  for (int i = 0; i < vector.Length; ++i)
66                  {
67                          if (vector[i] < smallestVal)
68                          {
69                                  smallestVal = vector[i];
70                          }
71                  }
72                  return smallestVal;
73              }
74
75          private void Train()
76          {
77                  RandomizeOrder(allData);
78                  double mce = CrossValidate (allData,
79  numFolds);
                    double mca = 1.0 - mce;
80                  net = new NeuralNet(3, 2, 1);
81                  net.Train4 (allData,70000, MinimumError);
82                  OC.ANNlearn = false;
83                  trained = true;
84                  OC.MinMax ();
85          }
86
```

```
87          double tryValues(double[] vals)
88          {       double[] result;
89                      if (load != null) {
90                              result = load.Compute (vals);
91                      } else {
92                              result = net.Compute (vals);
93                      }
94              return result[0];
95          }
96          static void RandomizeOrder(double[][] allData)
97              {
98                      Random rnd = new Random(0);
99                      for (int i = 0; i < allData.Length; ++i)
100                     {
101                             int r = rnd.Next(i, allData.Length);
102                             double[] tmp = allData[r];
103                             allData[r] = allData[i];
104                             allData[i] = tmp;
105                     }
106                 }
```

Pada Tabel 5.7 adalah class OpponentControl lanjutan dari tabel 5.1.

- Baris 1 hingga 20 merupakan control yang telah disediakan oleh kit. Penelitian ini difokuskan untuk braking decision sehingga hanya kontrol throttle atau gas kendaraan saja yang disesuaikan dengan algoritma *Neural network*.
- Pada baris 21 hingga 28 adalah script untuk mendapatkan 3 nilai *input* yaitu kecepatan mobil, sudut lintasan, dan jarak kendaraan ke sudut beserta nilai normalisasinya dengan metode min-max.
- Pada baris 29 dilakukan proses penghitungan nilai throttle dengan fungsi next.
- Setelah itu pada baris 31 nilai throttle dari algoritma *neural network* dan nilai newsteer dari kit akan dikirim ke fungsi Feed*Input* dan menjadi *input*an kendaraan.

Tabel 5.7 Fungsi Navigasi

No	Kode Program
1	`void NavigateAI()`
2	`{`
3	`...`
21	`if (!ANNlearn) {`
22	`var distancee = Vector3.Distance (transform.position, stats.PassedNodeB.position);`
23	`var directionA = (stats.PassedNodeB.position - stats.PassedNodeA.position).normalized;`
24	`var directionB = (stats.PassedNodeC.position - stats.PassedNodeB.position).normalized;`
25	`var targetAngleAB = Vector3.Angle (directionA, directionB);`
26	`var normalizedCurrentSpeed = (1 - 0) * (currentSpeed - (float)floatArray [3]) / ((float)floatArray [0] - (float)floatArray [3]) + 0;`

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

```
27                          var normalizedTargetAngle = (1 - 0) *
(targetAngleAB - (float)floatArray [4]) / ((float)floatArray [1]
- (float)floatArray [4]) + 0;
28                          var normalizedDistance = (1 - 0) *
(distancee - (float)floatArray [5]) / ((float)floatArray [2] -
(float)floatArray [5]) + 0;
29                          throttle =
ANN.Next(normalizedCurrentSpeed, normalizedTargetAngle,
normalizedDistance);
30                      }
31                      FeedInput(throttle, throttle, newSteer);
32          }
33          void FeedInput(float motor, float brake, float steer)
34          {
35              float m = (!reverse) ? motor = Mathf.Clamp(motor, 0,
1) : 0;
36              float b = (!reverse) ? brake = -1 *
Mathf.Clamp(brake, -1, 0) : 1;
37              float s = (!reverse) ? steer : -steer;
38              s += avoidanceSteer;
39              //Car Input
40              if (car_controller)
41              {
42                  //motor value
43                  car_controller.motorInput = m;
44                  //brake value
45                  car_controller.brakeInput = b;
46                  //handbrake value (always 0 for AI)
47                  car_controller.handbrakeInput = 0;
48                  //steer value
49                  car_controller.steerInput = s + avoidanceSteer;
50              }
51              //Bike Input
52              if (bike_controller)
53              {
54                  //motor value
55                  bike_controller.motorInput = m;
56                  //brake value
57                  bike_controller.brakeInput = b;
58                  //steer value
59                  bike_controller.steerInput = s + avoidanceSteer;
60              }
61          }
```

BAB 6 PENGUJIAN DAN ANALISIS

Bagian ini memuat hasil pengujian agen dalam memainkan track atau lintasan dalam Racing Game Starter Kit dan analisis dari hasil yang didapatkan dari pengujian.

6.1 Skenario Pengujian

Untuk menguji performa agen, penulis membandingkan skor dari kedua agen. Dimana agen dijalankan pada lintasan yang sama sebanyak sepuluh kali putaran, lalu diambil waktu tiap lintasan dan frame per second rata-rata agen. Dari hasil tersebut dapat disimpulkan agen mana yang mempunyai performa yang lebih bagus. Untuk aturan kendaraan, kendaraan yang digunakan memiliki jenis dan spesifikasi yang sama. Agen penulis akan dibandingkan dengan agen default dari kit dimana performa agen tersebut diambil dari komputer penulis yang digunakan saat melakukan penelitian.

6.2 Hasil Pengujian

Berdasarkan hasil akhir proses testing menggunakan *Cross-Validation* menunjukkan algoritma *Neural network* memiliki akurasi belajar sebesar 76%. Hasil ini didapatkan dengan mengacak data, kemudian membagi dataset yang berjumlah 1958 tadi ke dalam 10 grup. Salah satu bagian dipisahkan untuk pengujian sementara yang lain untuk proses *learning*.

Dan hasil yang didapatkan adalah maksimum error adalah 0.33 yaitu pada iterasi ke 10 dengan jumlah benar 136 dan jumlah salah 67. Dan minimum error adalah 0.2 yaitu pada iterasi ke 2 dan ke 4 dengan jumlah benar 156 dan jumlah salah 39. Sedangkan rata-rata dari iterasi ke 1 hingga 10 menunjukkan akurasi algoritma *neural network* sebesar 0.7620 atau 76%. Untuk lebih lengkap dapat dilihat pada gambar 6.1.

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

```
Fold = 0: wrong = 53 correct = 142    error = 0.2718
UnityEngine.Debug:Log(Object)
Fold = 1: wrong = 39 correct = 156    error = 0.2000
UnityEngine.Debug:Log(Object)
Fold = 2: wrong = 50 correct = 145    error = 0.2564
UnityEngine.Debug:Log(Object)
Fold = 3: wrong = 39 correct = 156    error = 0.2000
UnityEngine.Debug:Log(Object)
Fold = 4: wrong = 41 correct = 154    error = 0.2103
UnityEngine.Debug:Log(Object)
Fold = 5: wrong = 46 correct = 149    error = 0.2359
UnityEngine.Debug:Log(Object)
Fold = 6: wrong = 47 correct = 148    error = 0.2410
UnityEngine.Debug:Log(Object)
Fold = 7: wrong = 41 correct = 154    error = 0.2103
UnityEngine.Debug:Log(Object)
Fold = 8: wrong = 43 correct = 152    error = 0.2205
UnityEngine.Debug:Log(Object)
Fold = 9: wrong = 67 correct = 136    error = 0.3300
UnityEngine.Debug:Log(Object)
Mean cross-validation classification error = 0.2380
UnityEngine.Debug:Log(Object)
Mean cross-validation classification accuracy = 0.7620
UnityEngine.Debug:Log(Object)
```

Gambar 6.1 Hasil akhir cross-validation

Hal ini menunjukkan bahwa agen *neural network* mampu belajar dan menjawab dengan sesuai apa yang ia pelajari sebanyak 76% dan meleset dari hasil belajarnya sebanyak 24%.

Tabel 6.1 merupakan hasil pengujian FPS agen berdasarkan skenario pengujian. Tabel 6.2 merupakan hasil pengujian waktu agen berdasarkan skenario. Terdapat juga gambar 6.4 sebagai grafik dari performa kedua agen.

Tabel 6.1 Hasil Pengujian FPS Agen

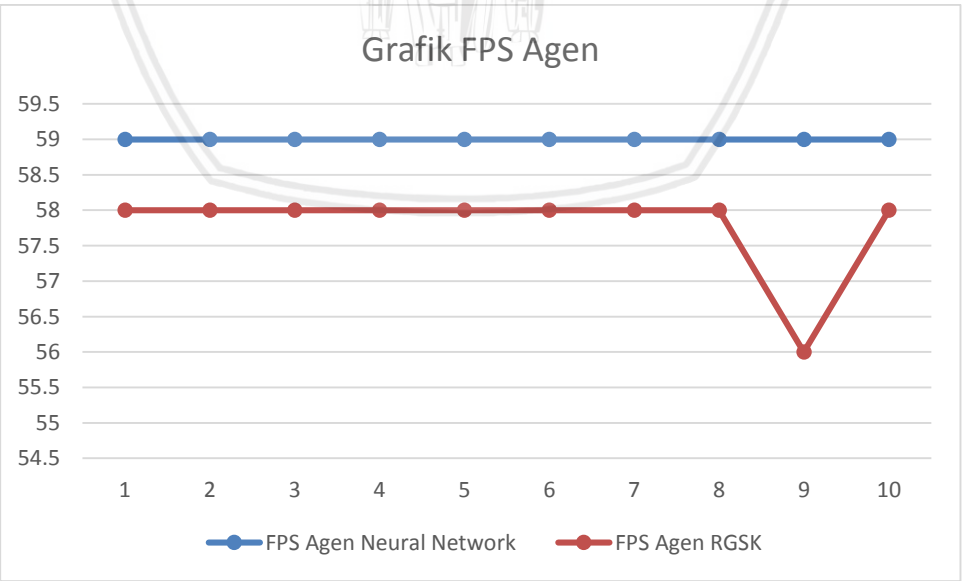
Putaran ke-	FPS Agen *Neural network*	FPS Agen RGSK
1	59	58
2	59	58
3	59	58
4	59	58
5	59	58
6	59	58
7	59	58
8	59	58
9	59	56

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

10	59	58
Rata-Rata	59	57.8

Tabel 6.2 Hasil Pengujian Waktu Agen

Putaran ke-	Waktu Tempuh Agen *Neural network*	Waktu Tempuh Agen RGSK
1	00:58:76	01:04:79
2	00:49:94	00:56:92
3	00:49:82	00:56:80
4	00:49:79	00:57:32
5	00:49:72	00:56:94
6	00:49:69	00:57:27
7	00:49:80	00:57:10
8	00:49:76	00:57:35
9	00:49:92	00:57:23
10	00:49:88	00:57:29
Total	08:27:21	9:39:03



Gambar 6.2 Grafik FPS Agen

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

6.3 Analisis Hasil Pengujian

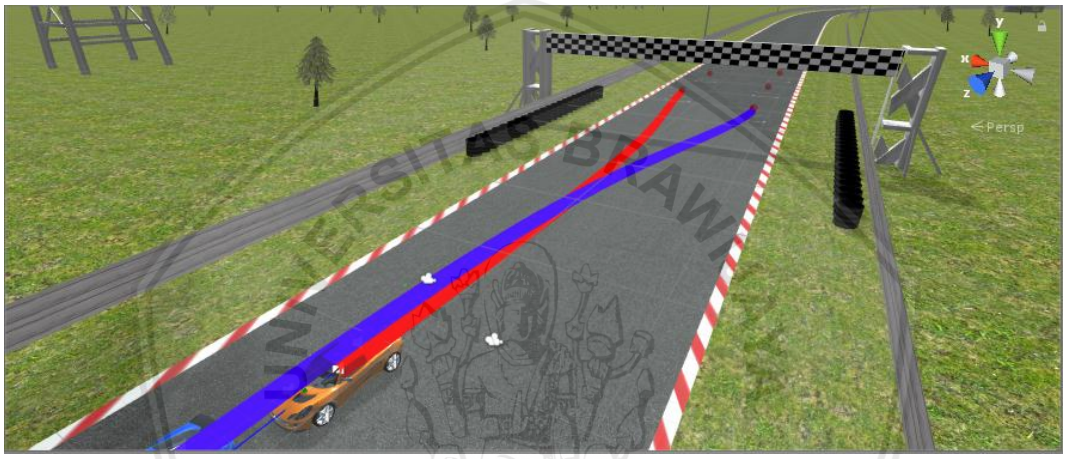
Dari sepuluh kali putaran, rata-rata fps yang didapat agen adalah 59 fps, sedangkan agen default kit memiliki rata-rata 57.8 fps. Dan performa yang dicapai agen memiliki keunggulan jika dibandingkan dengan agen milik RGSK. Hal ini dapat dilihat pada waktu tempuh tiap lap saat percobaan masing-masing agen. Agen *neural network* memiliki waktu tempuh selama 8 menit 27 detik atau setara dengan 507 detik. Sedangkan agen *default* kit memiliki waktu tempuh 9 menit 39 detik atau setara dengan 579 detik. Dari fakta tersebut dapat kita tarik kesimpulan bahwa agen *neural network* lebih unggul 72 detik atau perhitungan lebih lanjut yaitu 12% lebih cepat, hasil ini didapat dari selisih waktu tempuh dibagi dengan waktu tempuh terlama. Hal itu dikarenakan jumlah keputusan rem agen RGSK jauh lebih banyak dan mobil menjadi lebih lambat terutama saat tikungan. Namun, agen *Neural network* lebih sedikit melakukan pengereman sehingga terlihat hanya mengerem agar tidak menabrak pembatas luar jalan dan mengakibatkan agen *Neural network* cukup sering menyentuh rumput apabila melakukan tikungan.

Gambar 6.3 Penalty Surface

Dalam percobaan 10 kali putaran, Agen *Neural network* menyentuh permukaan penalti selama 40 detik lebih 88 milidetik yang ditunjukkan dengan garis merah dan ditandai dengan panah pada gambar 6.4. Sedangkan agen RGSK adalah 0 detik atau dengan kata lain tidak menyentuh rumput sama sekali yang ditunjukkan dengan garis warna biru pada gambar 6.4. Dan yang perlu diketahui pada kit ini menyentuh permukaan penalti tidak mempengaruhi apapun kecuali ban mobil yang akan mengeluarkan partikel emisi dan juga suara tambahan agar terlihat seperti menginjak rumput yang sesungguhnya.

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

Gambar 6.4 Track Line Agen



Gambar 6.5 Posisi Start Track Line Agen

Penulis menilai hasil tersebut cukup baik, mengingat ada beberapa faktor yang ditemukan penulis yang dapat mempengaruhi performa agen yang masih belum dieksplorasi secara keseluruhan, seperti jumlah layer dan neuron pada *hidden layer* yang tidak memiliki patokan yang tetap dan sangat tergantung pada kasus yang dihadapi, fungsi aktivasi yang bermacam-macam dengan kelebihan dan kekurangan masing-masing, fungsi normalisasi data yang memiliki metode yang bervariasi, teknik pengambilan data yang efisien, serta kemampuan perangkat keras peneliti sendiri dalam menghitung dan memproses informasi.

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

BAB 7 PENUTUP

7.1 Kesimpulan

Berdasarkan rumusan masalah di atas maka dapat diambil kesimpulan seperti berikut:

1. Implementasi *neural network* berhasil dilakukan dengan baik ke dalam RGSK
2. Melalui hasil pengujian jaringan syaraf tiruan terhadap model arsitektur diperoleh bahwa perbandingan waktu agen algoritma *neural network* lebih cepat dibandingkan dengan agen RGSK dengan nilai rata-rata FPS yang hampir sama saat game dijalankan.
3. Dan yang terakhir yaitu tingkat akurasi yang didapat dengan metode *k-fold cross-validation*, arsitektur dengan 2 neuron pada lapisan tersembunyi adalah sebesar 76%.

7.2 Saran

Berikut ini beberapa saran yang dapat dimanfaatkan untuk penelitian selanjutnya:

1. Penerapan Algoritma *Neural network* dengan metode pambelajaran selain Backpropagation
2. Perubahan teknik pengambilan data *input* sudut lintasan menggunakan *vector* posisi dari mobil
3. Metode evaluasi dengan metode selain *k-fold cross-validation*.

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

DAFTAR PUSTAKA

Chan M.T, Chan C.W, Gelowitz C. 2015. Development of a Car Racing Simulator Game Using Artificial Intelligence Techniques. Hindawi Publishing Corporation International Journal of Computer Games Technology ,Volume 2015, Article ID 839721

Lesnussa Y.A, Latuconsina S, Persulessy E.R. Aplikasi Jaringan Saraf Tiruan Backpropagation untuk Memprediksi Prestasi Siswa SMA (Studi kasus: Prediksi Prestasi Siswa SMAN 4 Ambon). Jurnal Matematika Integratif ISSN 1412-6184 Volume 11 No 2, Oktober 2015 , pp 149 –160

Syarif M.S. 2016. Penerapan Algoritma Backpropagation Untuk Menentukan Level Bonus dan Score Bonus Pada Game Edukasi Nahwu Menggunakan Kartu Berbasis Android. Published Skripsi, Malang: UIN Maulana Malik Ibrahim

Karavolos, Daniel. 2013. Q-learning with heuristic exploration in Simulated Car Racing. Published Thesis, University of Amsterdam

Fauzy, Muhammad. 2011. Game Catur Jawa with Reinforcement Learning. Published Tugas Akhir, Politeknik Elektronika Negeri Surabaya – Eepis.

Togelius, J, Nardi, R. D, Lucas, S. M. Making racing fun through player modeling and track evolution.In Proc. of the Workshop on Adaptive Approaches for Optimizing Player Satisfaction in Computer and Physical Games, 2006.

Huang Z.J, Luo L. 2015. It Takes the World to Understand the Brain. Published in final edited form as: Science. 2015 October 2; 350(6256): 42–44. doi:10.1126/science.aad4120.

Soarez F.M, Souza A.M.F, 2016. *Neural network Programming with Java second edition*. Birmingham : Packt Publishing Ltd.

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA