BAB II

LANDASAN TEORI

2.1 Game dan Simulasi

Pada umumnya, game dan simulasi mengambil bentuk dunia tiruan dengan lingkungan tertutup dan terkontrol yang memungkinkan dilakukannya pengujian suatu teknik yang hasilnya dapat diterapkan di dunia nyata (Salem, dkk, 2019). Game dan simulasi dibutuhkan untuk mengembangkan pemahaman mengenai sebuah sistem dengan mengamati bagaimana sistem tiruan tersebut beroperasi. Simulasi sudah menunjukkan kegunaannya dalam berbagai penerapan dan sudah menjadi disiplin ilmu yang memiliki teori dan metodologi penelitiannya sendiri (Sokolowski dan Bank, 2009). Dalam simulasi, model diartikan sebagai perkiraan suatu objek di dunia nyata dan definsi dari simulasi mencakup.

metode penerapan model dalam suatu periode waktu

1. teknik untuk pengujian, analisis, atau pelatihan dengan menggunakan sistem tiruan yang menyerupai dunia nyata dari sebuah proses yang dihasilkan kembali oleh sebuah model.
2. metodologi untuk mendapatkan informasi dari sebuah model dengan mengamati perilaku ketika model tersebut dijalankan.
3. dalam artian non-teknis berarti tiruan atau imitasi.

Tujuan simulasi yang dapat dikategorikan menjadi.

1. *Training* – menyediakan tiruan dunia nyata untuk mendapatkan lingkungan yang dapat dikontrol.

2. *Decision Support* – menyediakan alat yang dapat mendeskripsikan, menjelaskan, memprediksi, dan mengevaluasi.
3. *Understanding* – memfasilitasi pengujian hipotesis untuk sistem yang kompleks.
4. *Education and Learning* – memfasilitasi pengajaran dan pembelajaran dalam sistem *game-based learning.*
5. *Entertainment* – menyediakan representasi yang realistis untuk elemen yang memiliki perilaku dinamis.

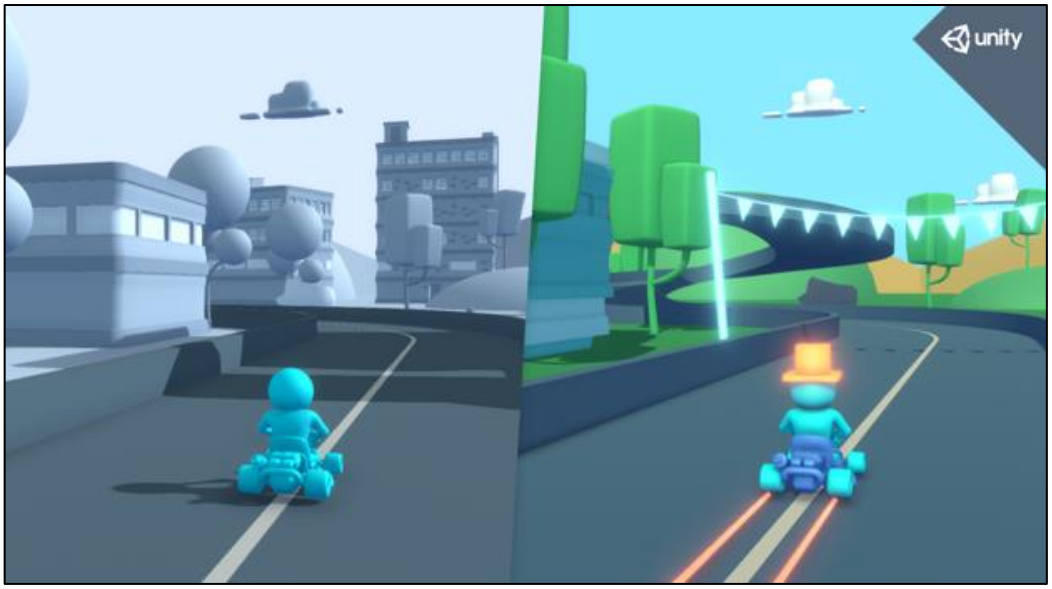
2.2 Unity Karting Microgame dan Self-Driving Kart

Dalam kamus Merriam-Webster, kart atau go-kart memiliki arti sebagai kendaraan bermotor berukuran kecil yang digunakan khususnya untuk balapan. Menurut David (2013), kart atau go-kart dirancang untuk balapan pada lintasan dengan ciri utama mudah dioperasikan, bahkan untuk pengguna pemula, karena baik mesin maupun lintasannya berukuran lebih kecil. Kart pertama kali dibangun oleh Art Ingels di California Selatan pada tahun 1956 (Nilawar, dkk, 2015).

Karting atau *kart racing* adalah balapan menggunakan kart. *Karting* dilihat sebagai cara untuk mengembangkan keahlian balapan sebelum menggunakan mobil-mobil yang lebih besar dan lebih cepat. Wajar apabila *karting* terpilih menjadi salah satu olahraga yang populer di dunia. (David, 2013).

Unity Karting Microgame adalah sebuah *3D kart racing game* yang dapat dimodifikasi dan dikustomisasi oleh pengguna Unity, mesin pengembang game (Unity Learn, 2020). Unity Karting Microgame, ditunjukkan pada Gambar 2.1,

mengambil konsep simulasi *racing game* dimana *kart* dikendalikan untuk menyelesaikan lintasan. Menurut Sazaki, dkk (2017), *racing game* adalah game berbentuk simulasi yang membuat tiruan balapan untuk mendapatkan pemenang yang dapat mencapai garis akhir dengan waktu terbaik.

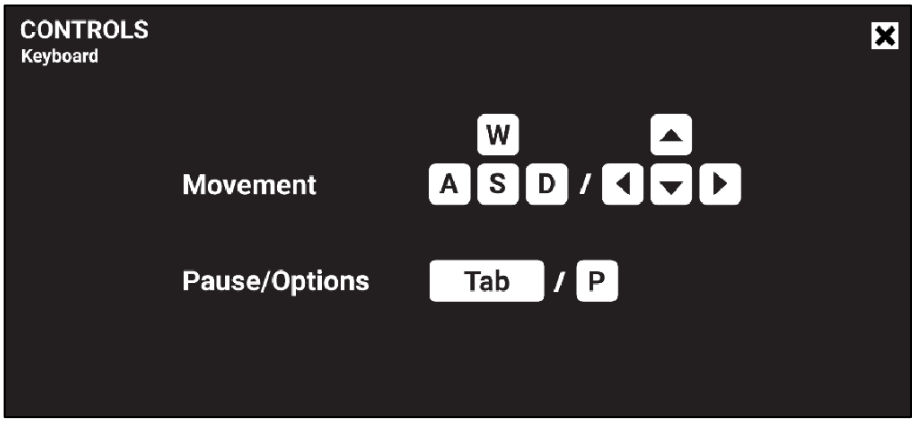


Gambar 2.1 Ilustrasi Unity Karting Microgame
(Unity Learn, 2020)

Pada Unity Karting Microgame, *kart* dikendalikan oleh pengguna menggunakan *arrow keys* dengan dua jenis pergerakan.

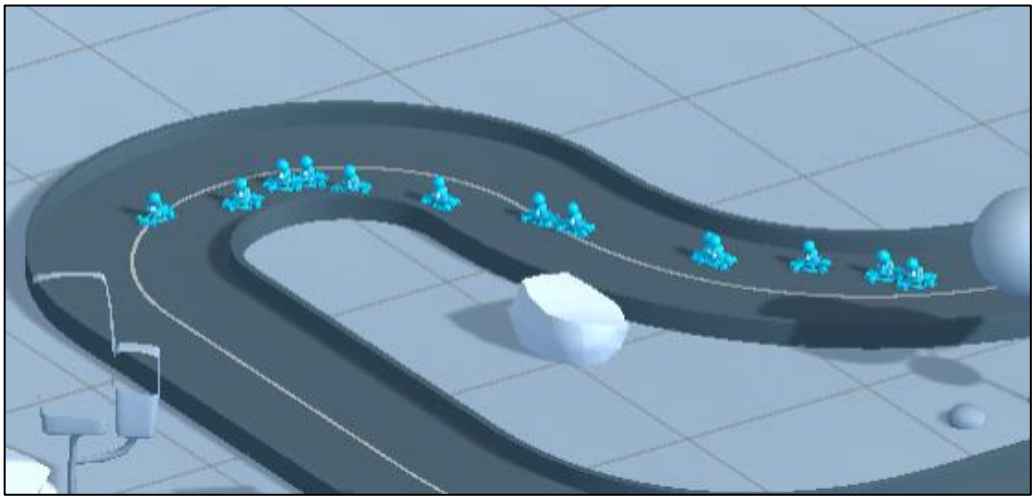
1. *Steering* : kiri (tombol 'A' atau '←') dan kanan (tombol 'D' atau '→')
2. *Acceleration*: gas (tombol 'W' atau '↑') dan rem (tombol 'S' atau '↓')

Steering dikendalikan dengan tombol 'A' dan 'D' atau tombol '←' dan '→', untuk mengatur pergerakan horizontal *kart*, sedangkan *acceleration* dikendalikan dengan tombol 'W' dan 'S' atau tombol '↑' dan '↓' untuk mengatur pergerakan vertikal *kart*. Kontrol pada Unity Karting Microgame ditunjukkan oleh Gambar 2.2.



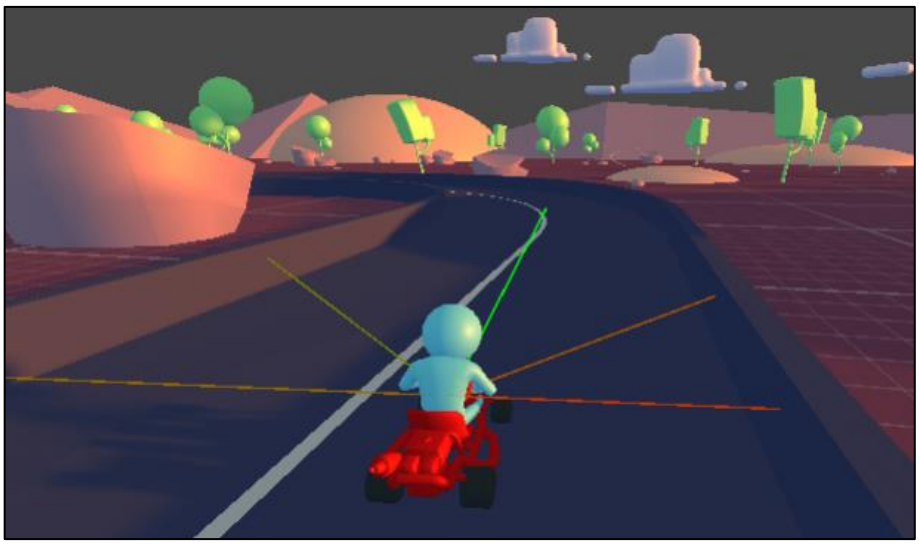
Gambar 2.2 Kontrol pada Unity Karting Microgame

Pada tahun 2017, Unity memperkenalkan Unity Machine Learning Agents Toolkit (Unity ML-Agents Toolkit), proyek *open-source* yang menjadikan game dan simulasi sebagai lingkungan untuk melatih kecerdasan agen. Pada tahun 2020, Unity ML-Agents Toolkit v1.0 dirilis bersamaan dengan beberapa implementasinya pada Unity Karting Microgame (Mattar, dkk, 2020). Implementasi Unity ML-Agents Toolkit pada Unity Karting Microgame bertujuan untuk menghasilkan *self-driving kart*. Simulasi *self-driving kart* dengan Unity ML-Agents Toolkit ditunjukkan pada Gambar 2.3.



Gambar 2.3 Simulasi *Self-Driving Kart* dengan Unity ML-Agents Toolkit

Self-driving adalah kemampuan suatu entitas untuk mengemudikan dirinya sendiri tanpa bantuan manusia. *Self-driving* merupakan contoh penerapan kecerdasan pada agen, dimana agen diartikan sebagai apapun yang dapat mengamati lingkungan melalui sensor dan merespon lingkungan melalui aktuator. Agen diharapkan mengambil tindakan terbaik berdasarkan situasi yang ada (Russell dan Norvig, 2010). Dalam implementasi NEAT pada Unity Karting Microgame, *kart* digunakan sebagai agen cerdas untuk menghasilkan *self-driving kart.*



Gambar 2.4 Contoh *Self-Driving Kart* dengan 5 Input Sensor

Self-driving kart adalah agen yang telah dilatih oleh pembelajaran mesin agar dapat menyelesaikan lintasan dengan otaknya sendiri dan tanpa bantuan manusia. Hal ini dimungkinkan ketika *kart* berlaku sebagai agen cerdas mengamati lingkungan melalui *input sensor* dan merespon lingkungan dengan pergerakan yang paling rasional. Dalam penelitian oleh Jallov (2019) dan Immanuel (2019), input sensor pada *self-driving kart* akan membaca jarak dari *kart* ke tepi lintasan, ditunjukkan pada Gambar 2.4. Respon pergerakan yang paling rasional yang

diekspektasikan dari *self-driving kart* adalah *kart* dapat menyelesaikan lintasan dengan bebas tabrakan.

Reward system digunakan sebagai suatu cara untuk menentukan nilai *fitness* suatu *self-driving kart.* Berdasarkan *reward system* pada penelitian oleh Jallov (2019), jika suatu *kart* berada pada lintasan yang benar dengan v_i sebagai kecepatan saat melewati *checkpoint i* dan memiliki nilai *fitness F* dengan *c* sebagai *reward* konstan tiap *checkpoint* serta *b* sebagai *reward* bias untuk kecepatan, maka diperoleh persamaan *fitness* sebagai berikut.

$$F = \sum_{n=1}^{i} c + (v_i . b) \quad \ldots (2.1)$$

Berdasarkan implementasi Unity ML-Agents Toolkit pada Unity Karting Microgame dan penelitian oleh Jallov (2019), *reward system* akan menambah nilai *fitness* apabila *kart* melewati *checkpoint* dengan urutan yang sesuai dengan memperhitungkan kecepatan kart saat melewati *checkpoint* tersebut.

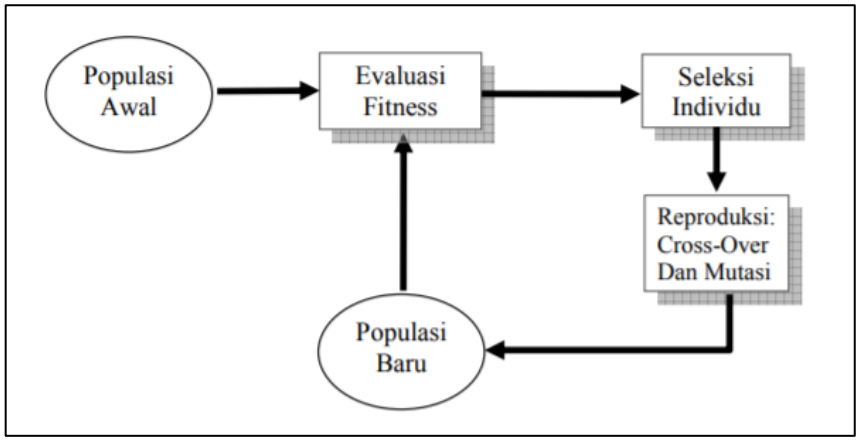
2.3 Algoritma Genetik

Algoritma genetik adalah algoritma pencarian dengan mekanisme yang menyerupai seleksi alam dan genetika alam (Martiana, dkk, 2014). Algoritma genetik dikembangkan oleh John Holland (1975) untuk menggambarkan proses adaptasi pada alam dan merancang sistem tiruan dengan mempertahankan mekanisme penting dari proses alamiah tersebut dengan mengikuti prinsip dasar *"survival of the fittest"* yang berarti kelangsungan hidup dari yang paling dapat

bertahan. Martiana, dkk, (2014) menjelaskan dalam algoritma genetik dikenal beberapa terminologi yang menyerupai terminologi pada evolusi alam seperti.

1. Individu, solusi dari permasalahan yang dicari oleh algoritma genetik.
2. Populasi, himpunan individu yang akan ditempatkan dalam suatu lingkungan dan diuji bersama dalam satu siklus evolusi.
3. Generasi, penanda siklus evolusi atau iterasi dalam algoritma genetik.
4. Fitness, ukuran kemampuan bertahan hidup suatu individu dalam satu siklus evolusi yang menyatakan baik atau tidaknya individu sebagai solusi.
5. Champion, individu dengan nilai *fitness* terbaik atau optimal.

Konsep algoritma genetik adalah mencari individu hasil dari seleksi dan genetika alam yang memiliki nilai *fitness* paling tinggi. Dalam setiap generasi, populasi baru dibuat menggunakan potongan informasi dari generasi sebelumnya untuk menghasilkan individu yang dapat bertahan hidup sebaik mungkin dalam kondisi yang diberikan dengan performa yang lebih baik.



Gambar 2.5 Siklus Algoritma Genetika oleh David Goldberg (Martiana, dkk, 2014)

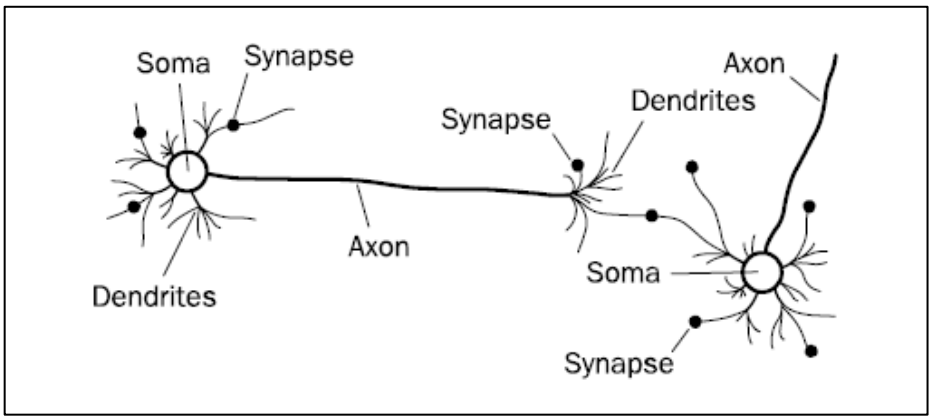
Gambar 2.5 menunjukkan siklus algoritma genetika oleh David Goldberg. Hal utama yang harus dilakukan dalam algoritma genetika adalah sebagai berikut.

1. Mendefinisikan individu, dimana individu mengacu kepada solusi dari sebuah permasalahan.
2. Mendefinisikan nilai *fitness,* dimana nilai *fitness* mengacu kepada ukuran baik atau tidaknya sebuah individu dalam menyelesaikan masalah (bertahan hidup).
3. Menginisialisasi sekumpulan individu baru (populasi awal) sebagai solusi awal dari permasalahan yang akan dilihat nilai *fitness*-nya.
4. Mengevaluasi *fitness,* dimana setiap individu diukur tingkat keberhasilannya untuk bertahan hidup.
5. Melakukan operasi evolusi yang melibatkan proses seleksi (*selection*) untuk selanjutnya digunakan dalam operasi genetika.
6. Melakukan operasi genetika yang melibatkan proses pindah silang (*crossover*) dan mutasi (*mutation*) untuk menciptakan keturunan (populasi baru).

2.4 Jaringan Saraf Tiruan

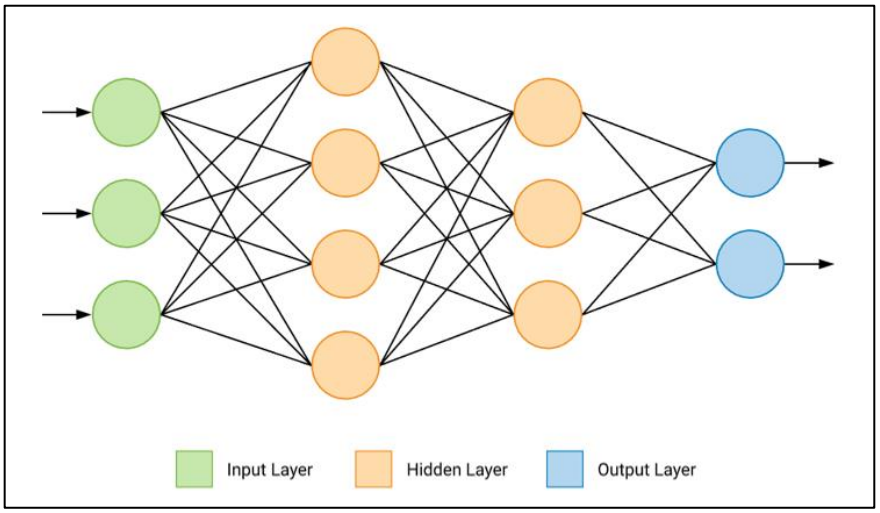
Jaringan saraf atau *neural network* dapat didefinisikan sebagai model logika dalam otak manusia yang tersusun atas unit-unit pemrosesan informasi yang disebut *neuron* (Sher, 2012). *Neuron,* ditunjukkan pada Gambar 2.6, dalam artian biologis terdiri atas badan sel (soma), dendrit, dan akson dan digunakan untuk menerima dan meneruskan informasi. Menurut Negnevitsky (2011) pada dasarnya jaringan saraf memiliki kemampuan yang untuk belajar, sehingga kemampuan untuk belajar

inilah yang diadopsi dan dimiliki oleh jaringan saraf tiruan *(artificial neural network)*.



Gambar 2.6 Jaringan Saraf Biologis
(Negnevitsky, 2011)

Jaringan saraf tiruan adalah simulasi biologis dari jaringan saraf yang memiliki tingkatan presisi yang beragam karena jaringan saraf tiruan tidak sepenuhnya sama dengan jaringan saraf biologis (Sher, 2012). Sher juga berpendapat bahwa jaringan saraf tiruan hanya merepresentasikan esensi biologis dari jaringan saraf, yaitu kemampuan untuk belajar.



Gambar 2.7 Topologi Umum Jaringan Saraf Tiruan

Topologi adalah arsitektur dari sebuah jaringan saraf tiruan yang mempengaruhi bagaimana sinyal diterima, diproses, dan dikembalikan Iba (2018). Topologi jaringan saraf tiruan pada umumnya ditunjukkan oleh Gambar 2.7. Menurut Iba (2018), topologi jaringan saraf tiruan disusun oleh.

1. *input layer,* disusun oleh *input nodes* sebagai penerima sinyal masuk.
2. *middle layer,* disusun oleh *hidden nodes* sebagai tempat sinyal diproses.
3. *output layer,* disusun oleh *output nodes* sebagai pemberi sinyal keluar.
4. *connections* sebagai penghubung antar *node* yang memiliki bobot.

Negnevitsky (2011) mengatakan pembobotan pada *connections* bertujuan sebagai penanda ingatan jangka panjang pada jaringan saraf tiruan yang menunjukkan kekuatan (kepentingan) sebuah sinyal masuk. Proses pembelajaran pada jaringan saraf tiruan terjadi apabila pengaturan bobot ini dilakukan terus-menerus. Jadi, sebelum jaringan saraf tiruan dapat melakukan komputasi dan diterapkan untuk menyelesaikan suatu masalah, topologi dan pembobotannya perlu diatur melalui proses evolusi agar dapat berfungsi dengan opitmal (Sher, 2012).

Berdasarkan penelitian oleh Jallov (2019), kompleksitas adalah jumlah *connections* dalam suatu jaringan saraf tiruan. Jika dalam suatu jaringan saraf tiruan terdapat suatu himpunan E yang beranggotakan *connections* pada jaringan saraf tiruan tersebut dan memiliki kompleksitas C, maka persamaan untuk mendapatkan kompleksitas adalah sebagai berikut.

$$C = n(E) \quad \dots (2.2)$$

Semakin banyak jumlah *nodes* dan *connections* pada topologi maka jaringan saraf tiruan menjadi semakin kompleks dan komputasi yang dibutuhkan saat

jaringan saraf tiruan dijalankan semakin tinggi (Heidenreich, 2019). Menurut Heidenreich (2019), untuk menyelesaikan masalah kompleksitas dapat diterapkan algoritma optimisasi, yaitu penyusutan kompleksitas dengan mengurangi jumlah *nodes* dan *connections* agar ditemukan topologi jaringan saraf tiruan yang optimal.

2.5 NeuroEvolution of Augmenting Topologies (NEAT)

Neuroevolution adalah integrasi algoritma genetik dengan jaringan saraf tiruan yang dibentuk dari kata *neuron* pada jaringan saraf dan *evolution* pada algoritma genetik. Menurut Iba (2018), seperti dalam algoritma genetik, implementasi *neuroevolution* perlu mendefinisikan individu sebagai solusi dari permasalahan yang akan diselesaikan oleh *neuroevolution*. Individu dalam *neuroevolution* adalah jaringan saraf tiruan. Proses evolusi pada *neuroevolution* akan berlangsung hingga didapatkan jaringan saraf tiruan dengan topologi serta pembobotan terbaik didapatkan sebagai solusi optimal.

NeuroEvolution of Augmenting Topologies (NEAT) adalah metode *neuroevolution* yang diajukan oleh Stanley dan Miikkulainen (2002) dalam penelitian mereka dengan judul *Evolving Neural Networks through Augmenting Topologies*. Menurut Heidenreich (2019), pada umumnya, jaringan saraf tiruan dibangun dari kompleksitas tinggi lalu disederhanakan (optimisasi), sedangkan Stanley dan Miikkulainen mengajukan teknik *augmenting topologies* atau topologi jaringan saraf tiruan yang arsitekturnya bertambah seiring evolusi terjadi. Ide dasar NEAT adalah membangun jaringan saraf tiruan yang dimulai dari jumlah *nodes* dan

connection yang paling kecil atau sederhana lalu mengevolusikan arsitekturnya seiring berjalan waktu jika dan hanya jika evolusi tersebut berguna dan diperlukan.

Menurut Iba (2018), dalam *neuroevolution*, *genotype* dan *phenotype* digunakan untuk merepresentasikan suatu jaringan saraf tiruan.

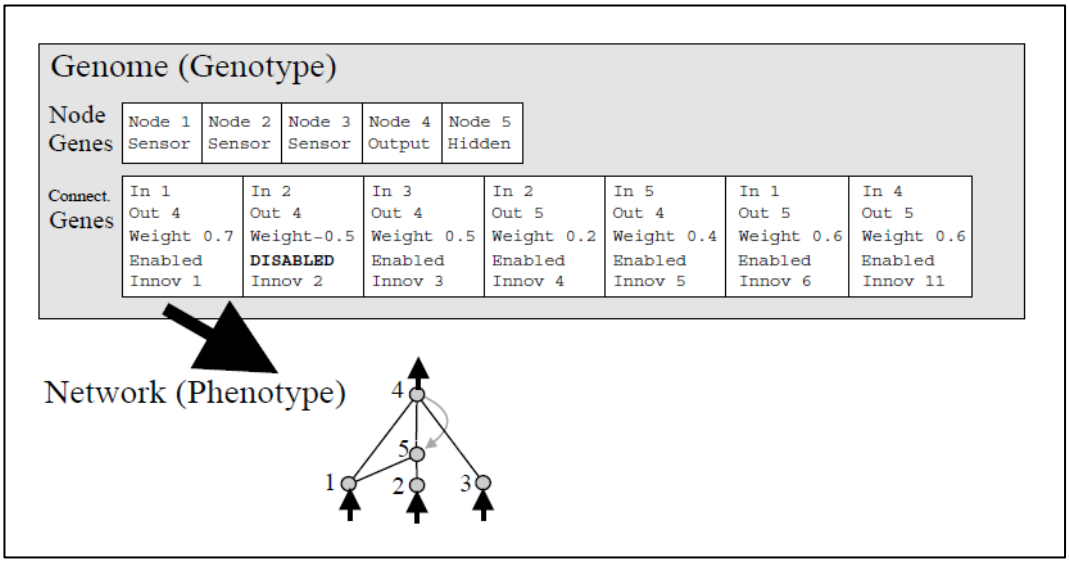
1. *Genotype* adalah representasi genetik dari jaringan saraf tiruan yang tersusun dari rangkaian kode genetik hasil *genetic encoding*.
2. *Phenotype* adalah representasi fisik dari jaringan saraf tiruan yang dibangun berdasarkan susunan kode genetik pada *genotype.*untuk membentuk topologi.

Penulisan kode genetik *(genetic encoding)* diperlukan sebagai cara untuk menghasilkan *genotype* yang dapat merepresentasikan topologi jaringan saraf tiruan dalam *neuroevolution*. Heidenreich (2019) mengatakan bahwa terdapat dua jenis penulisan kode genetik, yaitu.

1. *Direct Encoding,* penulisan kode secara langsung terhadap gen
2. *Indirect Encoding,* penulisan kode secara tidak langsung terhadap aturan

NEAT menggunakan *direct encoding* sebagai cara penulisan kode genetik. Walaupun penulisan kode genetiknya lebih rumit daripada *indirect encoding,* namun penulisan kode genetik dengan cara *direct encoding* lebih mudah dipahami (Heidenreich, 2019). *Direct encoding* pada NEAT menghasilkan *genotype* yang terdiri dari dua jenis gen, yaitu *node genes* dan *connection genes*.

1. *Node Genes*, rangkaian *input nodes, hidden nodes,* dan *output nodes*.
2. *Connection Genes*, rangkaian *connections* yang menghubungkan dua *node*.



Gambar 2.8 Contoh Pemetaan dari Genotype menjadi Phenotype
(Stanley dan Miikkulainen, 2002)

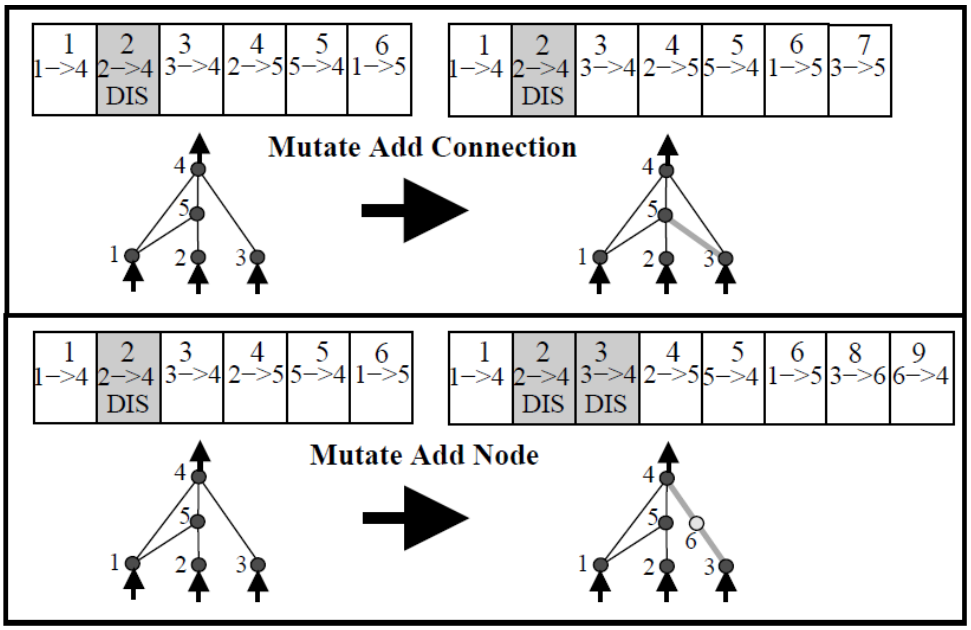
Gambar 2.8 menunjukkan bagaimana NEAT merepresentasikan *genotype* dan *phenotype. Genotype* yang terdiri dari dua jenis gen, yaitu gen *node genes* dan *connection genes*.

Node genes menyimpan nomor identifikasi *node* dan tipe *node*.

1. Nomor identifikasi *node* bertujuan untuk menandakan dan mengindentifikasi suatu *node.*
2. Tipe node bertujuan untuk menentukan jenis dan fungsi *node. Sensor* menandakan *input nodes* sebagai penerima sinyal, *output* menandakan *output nodes* sebagai pemberi sinyal keluar, dan *hidden* menandakan *hidden nodes* sebagai tempat sinyal diproses.

Connection Genes menyimpan nomor identifikasi *node* asal dan *node* tujuan, bobot koneksi, penanda koneksi dipakai (*enabled*/disabled), dan nomor inovasi.

1. Nomor identifikasi *node* asal dan *node* tujuan digunakan untuk mengidentifikasikan asal dan tujuan koneksi.
2. Bobot koneksi digunakan untuk menunjukkan kepentingan sebuah sinyal yang diteruskan melalui *connection genes*.
3. Penanda koneksi dipakai untuk menandakan apakah suatu *connection gen* digunakan pada topologi.
4. Nomor inovasi digunakan pada saat *seleksi* dan *crossover* sebagai penanda historis sebuah *connection gen.*



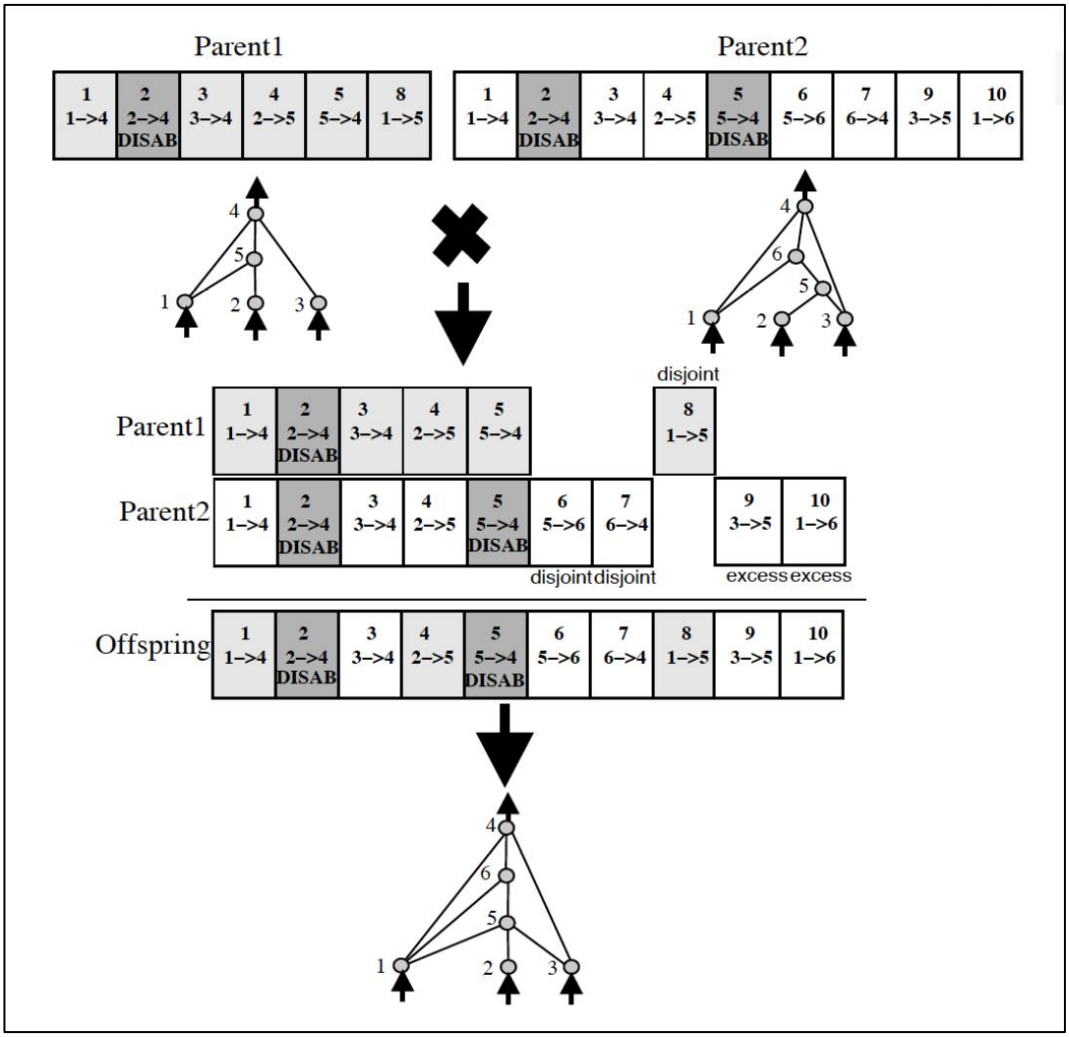
Gambar 2.9 Contoh Mutasi Topologi pada NEAT
(Stanley dan Miikkulainen, 2002)

Mutasi topologi pada NEAT ditunjukkan pada Gambar 2.9. Dalam NEAT terdapat dua jenis mutasi topologi, yaitu *add connection* dan *add node.*

1. *Add Connection*, sebuah *connection gen* baru akan ditambahkan pada rangkaian *connection genes* dengan nomor inovasi yang terbaru.

2. *Add Node*, sebuah *node gen* baru akan ditambahkan diantara dua *node* yang sudah terkoneksi. *Connection genes* yang menghubungkan dua *node* sebelumnya akan masuk dalam status *disabled,* sedangkan dua *connection genes* baru akan dibuat untuk menghubungkan *node* asal ke *node gen* baru dan dari *node gen* baru ke *node* tujuan.

Nomor inovasi digunakan sebagai penanda historis yang melanjutkan penomoran secara umum sehingga dimungkinkan penelusuran *node* pendahulu.



Gambar 2.10 Contoh Crossover dari Dua Topologi yang berbeda pada NEAT (Stanley dan Miikkulainen, 2002)

Gambar 2.10 menunjukkan proses *crossover* dalam NEAT. Proses *crossover* dalam NEAT mengadopsi homologi, yaitu penyelerasan kromosom berdasarkan gen yang cocok. Dalam NEAT penyelarasan dilakukan pada *connection genes* berdasarkan nomor inovasi (Heidenreich, 2019).. Pada saat *crossover*, *connection genes* Induk 1 dan Induk 2 diselaraskan berdasarkan nomor inovasi. Apabila kedua induk memiliki nomor inovasi yang sama, maka keturunannya akan mewarisi *connection gen* dari induk secara acak. Apabila dalam terdapat ketidakcocokan nomor inovasi, maka keturunannya akan mewarisi *connection gen* dari induk berdasarkan nilai *fitness* induk yang lebih tinggi.

Spesies dalam metode NEAT mengacu pada kelompok jaringan saraf tiruan dengan topologi yang memiliki kemiripan (Heidenreich, 2019). Menurut Heidenreich (2019), pada implementasi NEAT, kebanyakan evolusi topologi yang terbaru bukanlah yang terbaik karena menambahkan koneksi atau node baru sebelum optimasi bobot sering membuat nilai *fitness* individu menjadi rendah. Hal ini menyebabkan topologi baru pada posisi yang kurang menguntungkan, sehingga dibutuhkan cara untuk melindungi topologi baru dan memberi kesempatan pada individu baru untuk dioptimalkan sebelum dihilangkan dari populasi. Cara yang diberikan pada NEAT adalah melalui spesiasi. Spesiasi adalah proses membagi populasi menjadi beberapa spesies berdasarkan kesamaan topologi dan koneksi. NEAT menggunakan nomor inovasi sebagai penanda historis dalam penulisan kode genetiknya, sehingga pengelompokkan lebih mudah untuk dilakukan. Menurut Heidenreich (2019), spesiasi memungkinkan individu dalam suatu populasi hanya harus bersaing dengan individu lain dalam spesies yang sama.