**Default:**

**Neural Network:**

24 Input: 3 x 8 -> 8: direction, 3: food, body, wall

16 Hidden Node

1 Hidden Layer

4 Output: move direction (up, down, right, left)

Weight awal dirandom

Bias = 0

Menggunakan Relu, bukan Sigmoid

**Genetic Algorithm:**

Population: 4000 (changeable)

Fitness score: period alive \* (score^2)

Mutation chance: 10% (changeable)

Clone: hanya 1 snake terbaik yang di clone ke generasi selanjutnya

Crossover: inherit weight score dari 2 parent random

Mutation: Mengubah beberapa weight dari sebuah snake hasil crossover

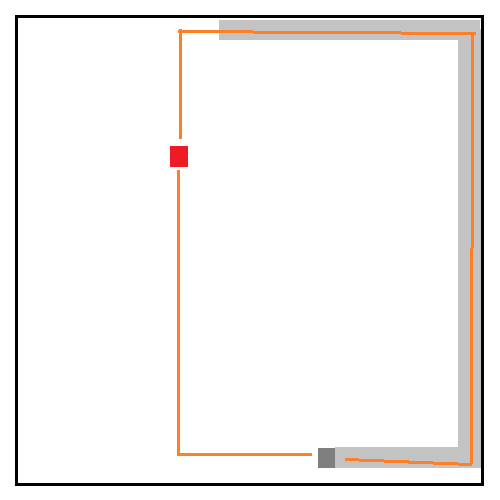
Hasilnya:

Setelah beberapa percobaan, hasil yang didapatkan yaitu:

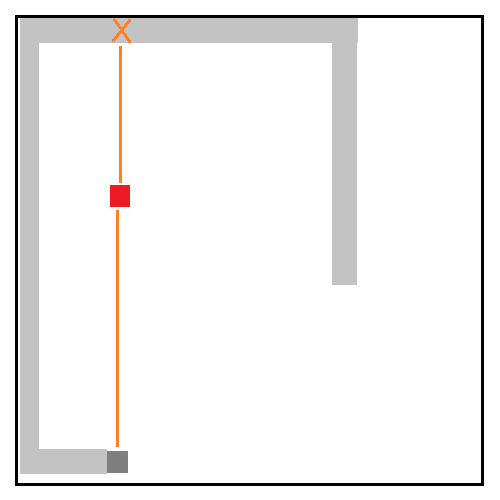
Sekitar generasi 1-3, snake terbaik akan memiliki fokus untuk bertahan hidup. Apabila ia berhasil memakan, setelah memakan ia tidak fokus mencari makan lagi, namun fokus berputar untuk bertahan hidup

Setelah beberapa generasi selanjutnya, Snake akan mulai fokus mencari makanan tetapi mudah mati dengan menabrakan diri ke badan atau tembok.

Setelah puluhan generasi, snake mulai handal dalam mencari makan, dan memiliki kecenderungan untuk berputar di ujung salah satu tembok. Lihat gambar untuk penjelasan lebih lanjut.



Gambar 1.1  
Salah satu contoh kelakuan snake pada sebuah percobaan

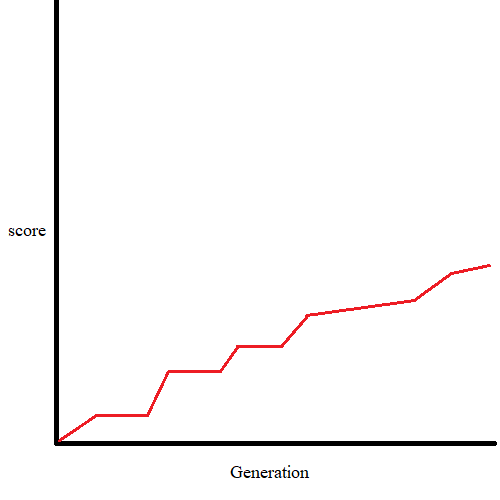


Gambar 1.2

Contoh kelakuan snake pada percobaan yang berbeda

Pada gambar 1.1 dan 1.2 snake membentuk sebuah pola berbentuk kotak dengan kecenderungan untuk menempelkan diri pada satu sisi tembok. Hal ini tidak masalah ketika snake masih tidak terlalu panjang seperti pada gambar 1.1, namun ketika snake tumbuh hingga sangat panjang, kemungkinan besar ia akan menabrak dengan badannya sendiri.

Pada sekitar generasi 150, masalah snake berputar membentuk pola kotak masih terjadi. Pada saat ini disimpulkan bahwa masih perlu diadakan perubahan pada Neural Network dan atau Genetic Algorithm sehingga menghasilkan snake yang lebih efisien. Berikut adalah graf yang menggambarkan keseluruhan dari training AI.

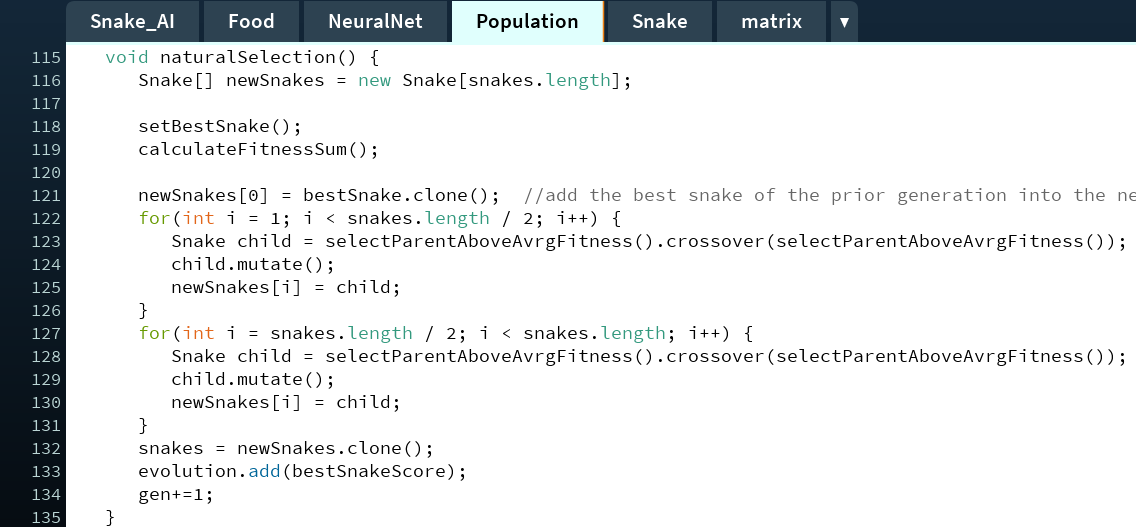


Gambar 1.3  
Graf perbandingan generasi dan skor snake (rata-rata dari beberapa test)

Dari graf di atas, terlihat masih kurang efisien, terutama semakin banyak jumlah generasi, semakin lambat peningkatan dari AI. Maka dari itu, akan diadakan beberapa perubahan dengan tujuan untuk meningkatkan efisiensi dari pelatihan AI.

**Freonanda Purnomo / C14190013:**

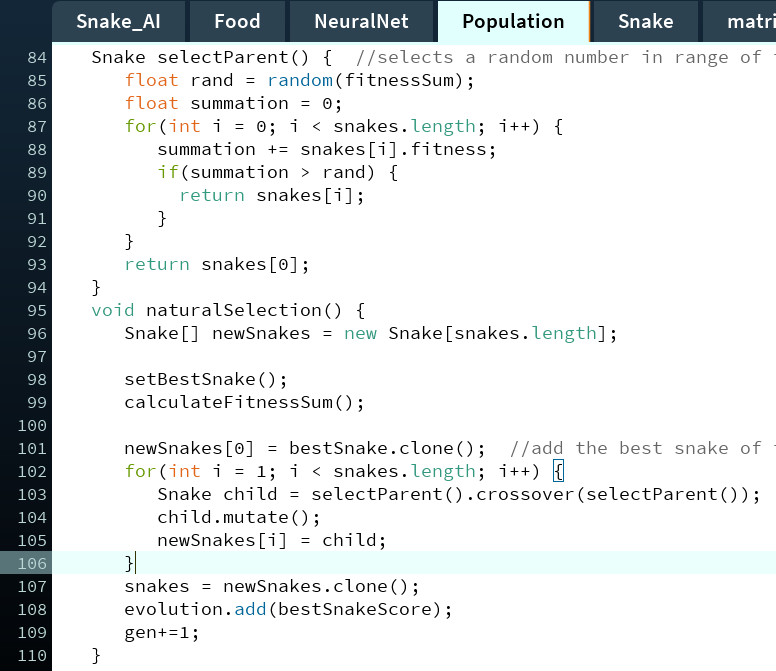
Crossover: setengah populasi inherit weight score dari best snake dan 1 parent random, sedangkan setengah lagi inherit weight score dari 1 parent yang memiliki fitness score diatas rata” yang dihitung dari 0 sampai snakes.length / 2 dan 1 parent random.parent yang memiliki fitness score diatas rata” yang dihitung dari snakes.length / 2 sampai snakes.length.



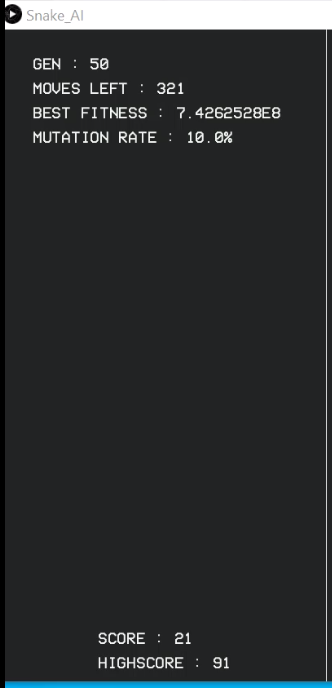
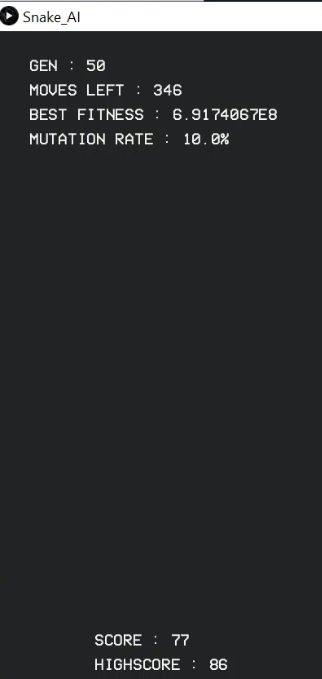
Gambar 2A.1



Gambar 2A.2



Gambar 2B.1

Gambar 2C.1Gambar 2C.2

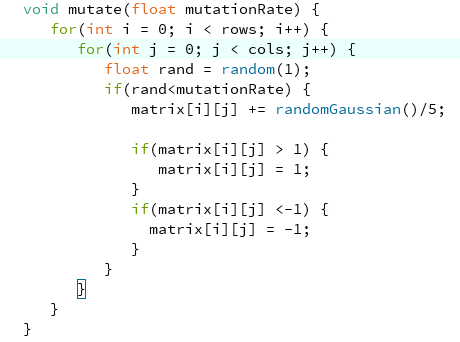
Pada gambar 2A.1 dan 2A.2 itu natural selection dan select parent pada Population.pde adalah coding dari hasil gambar 2C.1, pada gambar 2B.1 adalah coding dari hasil gambar 2C.2 yang di run sampai Gen = 50.

Jadi eksperimen saya mengimprove Fitness rate dengan cara yang baru yaitu pada gambar 2A.2. Setelah summation lebih besar dari random akan masuk ke while loop untuk membandingkan snakes.fitness lebih besar dari average fitness yang langsung mencari parent yang lebih bagus supaya harapannya fitness dan score generasi selanjutnya menjadi lebih baik.

Jadi terbukti dengan gambar 2C.1 dengan 2C.2 program lebih efisien walaupun sedikit. Lebih cepat dalam hasil high score dan best fitness pada gambar 2C.1 dibandingkan gambar 2C.2

**Joseph Clio Limantara - C14190047**

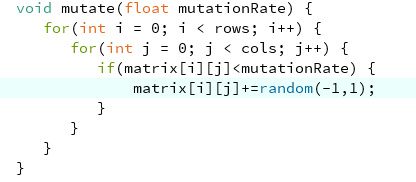
Mengimprove coding dengan mengubah mutasi dari yang awalnya diubah weight nya diubah semua menjadi mengubah yang perlu di ubah (tidak semua)



Gambar 3.1

Pada gambar 3.1, pada function ini akan dirandom angka baru antara 0-1. Jika angka baru tsb lebih kecil dari mutation rate maka akan dirandom hanya pada col dan row matrix tersebut.

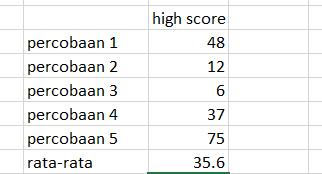
Jadi untuk eksperimen saya akan mengimprove menjadi membandingkan weight dengan mutation ratenya jika lebih kecil maka weight akan ditambah dengan random dari -1 sampai 1



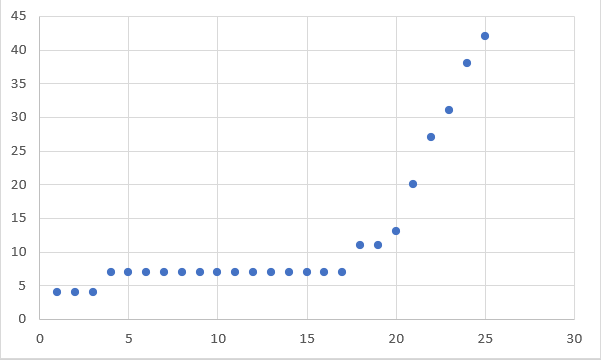
Gambar 3.2

Hasil yang didapat adalah sebagai berikut

High score rata2 saat generasi 20 dari 5 kali percobaan adalah:

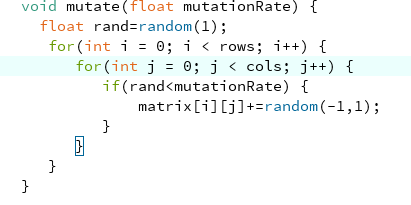


Gambar 3.3



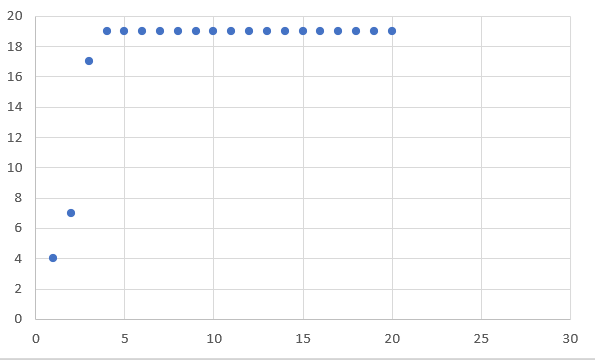
Gambar 3.4

Pada gambar 3.4 menunjukan kebalikan dari gambar 1.3 makin banyak generasi maka makin jauh juga lompatan dari highscorenya.



Gambar 3.5

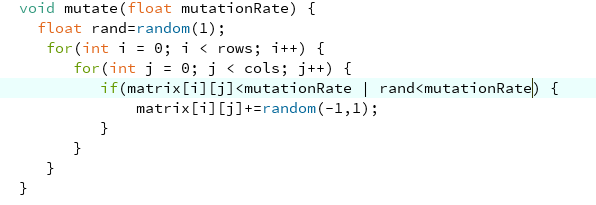
Jadi dalam fungsi ini angka akan dirandom 1 kali. Lalu angka random yang didapat akan dibandingkan dengan mutation ratenya. Jika lebih kecil maka seluruh weight dari snake tersebut akan di random, jika lebih besar maka tidak akan dimutasi.



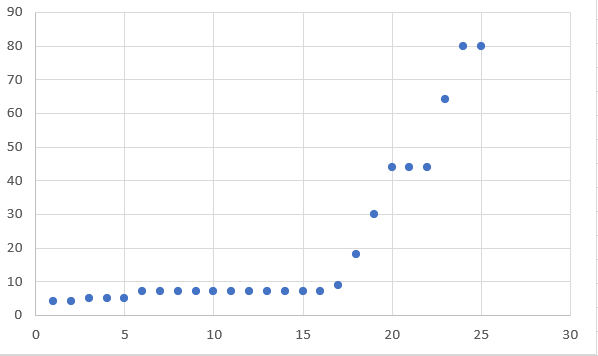
Gambar 3.6

Pada gambar 3.6 menunjukan bahwa dengan perubahan ini maka high score akan dengan cepat naik pada awal-awal program dijalankan

Jadi untuk mengimbangi agar ada pengembangan high score yang cepat diawal dan diakhir maka kedua code digabungkan menjadi seperti pada gambar 3.7

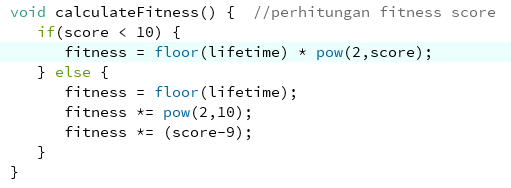


Gambar 3.7



Gambar 3.8

Gambar 3.8 menunjukan grafik high scorenya. Pada grafik penambahan di awal tidak begitu kentara karena penambahan high score di bagian belakang sangat besar. Agar AI tidak berfokus dengan bertahan hidup tidak bisa dengan menghilangkan koefisien lifetime pada function calculate fitness di class snake. Gambar 3.9

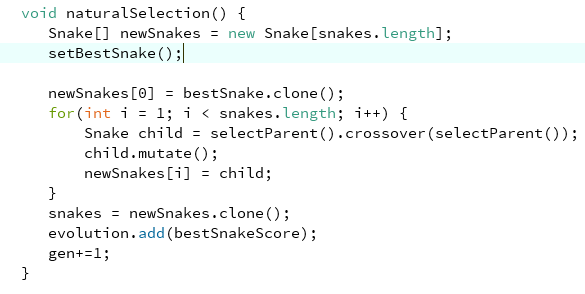


Gambar 3.9

Hal ini karena snake perlu melakukan ‘gerakan percuma’ seperti berputar-putar untuk melakukan scan.

Experiment 2

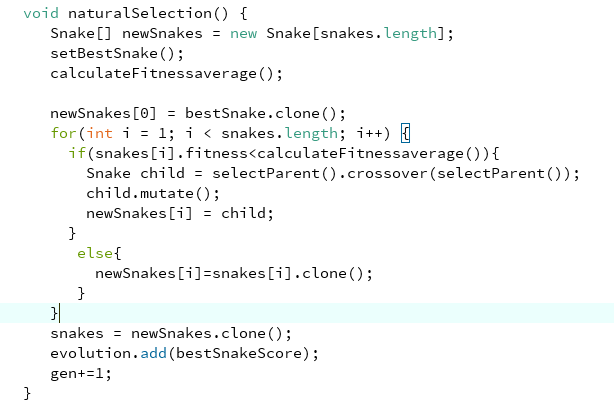
Mengclone semua snake yang diatas rata-rata.



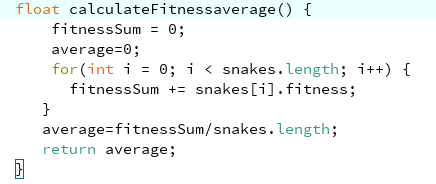
Gambar 3.10

Gambar 3.10 adalah kondisi awal function natural selection pada class population. Jadi newsnake akan diberi fitness yang terbaik pada index ke 0 lalu sisanya yang bukan merupakan best akan di cross over dan di mutasi.

Menurut logika hal ini kurang begitu baik karena banyak nilai fitness yang sebenarnya baik tapi bukan best akan terbuang. Maka dari itu saya mengimprove menjadi seperti gambar 3.11



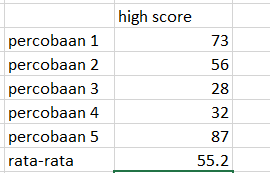
Gambar 3.11



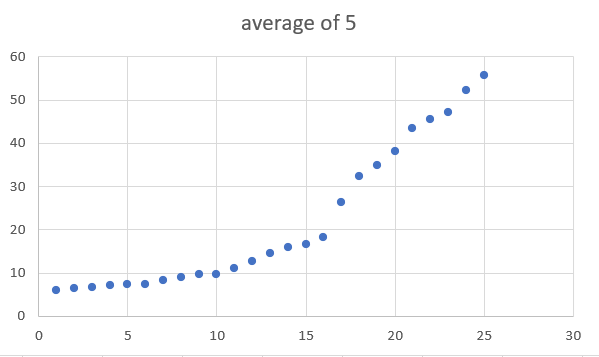
Gambar 3.12

Jadi pada gambar 3.11 if digunakan untuk membandingkan apakah nilai fitness snake lebih besar atau lebih kecil dari rata-rata. Apabila nilai fitnessnya lebih besar dari rata-rata akan di clone. Sedangkan yang dibawah rata-rata akan di crossover lalu di mutase.

Gambar 3.12 adalah fungsi yang akan dipanggil pada if di gambar 3.11. jadi pada function ini for akan menjumlah seluruh nilai fitness lalu totalnya akan dibagi dengan banyaknya snake.



Gambar 3.13

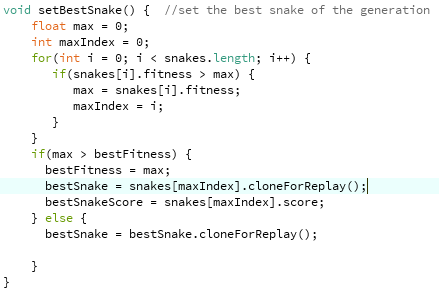


Gambar 3.14

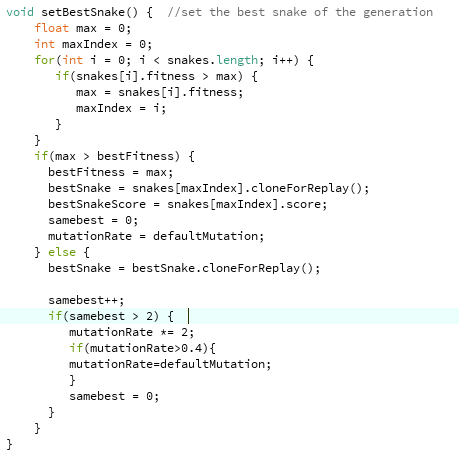
Gambar 4.4 adalah rata-rata high score pada generasi 25. Sedangkan gambar 4.5 adalah grafik dari 5 kali percobaan. Pada grafik ini menunjukan bahwa semakin banyak generasi semakin secap pula penambahan high scorenya sehingga kebalikan dari gambar 1.3.

**Jeremy Hanov - C14190215**

Mengubah dan membandingkan tingkat mutasi jika snake tetap menjadi yang terbaik selama 3 generasi menjadi 2 kali lipat, dibandingkan yang sebelumnya yaitu rate mutasi konstan 10%. Sebelumnya, operator mutasi mempertahankan keragaman dengan menukar salah satu gen dalam kromosom dengan nilai kebalikannya, sebagai contoh; jika kromosomnya memiliki nilai biner 0 dan 1 maka jika titik mutasi terpilih memiliki nilai 1 yang terjadi adalah nilai akan ditukar menjadi nilai 0. Pada kasus ini, kemungkinan mutasi tidak akan melebihi 40%, jika kemungkinan mutasi melebihi 40% maka akan di reset menjadi 10% kembali. Dan juga, jika best snake sudah terganti sebelum rate mutasi melebihi 40%, maka rate mutasi akan kembali ke 10%.



Gambar 4.1



Gambar 4.2

Pada Gambar 4.1, terlihat bahwa kodingan belum memiliki perubahan tingkat mutasi jika snake menjadi yang terbaik selama 3x berturut-turut. Pada Gambar 4.2, terjadi perubahan terhadap tingkat kemungkinan mutasi jika snake menjadi yang terbaik selama 3x berturut-turut. Berdasarkan data yang saya ambil, yang terjadi sebelum perubahan adalah

|  |  |  |
| --- | --- | --- |
| Generation | Mutation Chance | Highscore |
| 1 | 10 | 3 |
| 2 | 10 | 7 |
| 3 | 10 | 7 |
| 4 | 10 | 7 |
| 5 | 10 | 7 |
| 6 | 10 | 7 |
| 7 | 10 | 7 |
| 8 | 10 | 7 |
| 9 | 10 | 7 |
| 10 | 10 | 7 |
| 11 | 10 | 9 |
| 12 | 10 | 9 |
| 13 | 10 | 14 |
| 14 | 10 | 22 |
| 15 | 10 | 24 |
| 16 | 10 | 31 |
| 17 | 10 | 31 |
| 18 | 10 | 44 |
| 19 | 10 | 44 |

|  |  |  |
| --- | --- | --- |
| 20 | 10 | 44 |

Tabel 4.1

|  |  |  |
| --- | --- | --- |
| Generation | Mutation Chance | Highscore |
| 1 | 10 | 5 |
| 2 | 10 | 5 |
| 3 | 10 | 5 |
| 4 | 20 | 5 |
| 5 | 10 | 7 |
| 6 | 10 | 7 |
| 7 | 10 | 7 |
| 8 | 20 | 7 |
| 9 | 10 | 8 |
| 10 | 10 | 8 |
| 11 | 10 | 8 |
| 12 | 20 | 8 |
| 13 | 10 | 14 |
| 14 | 10 | 16 |
| 15 | 10 | 24 |
| 16 | 10 | 24 |
| 17 | 10 | 29 |
| 18 | 10 | 34 |
| 19 | 10 | 38 |

|  |  |  |
| --- | --- | --- |
| 20 | 10 | 41 |

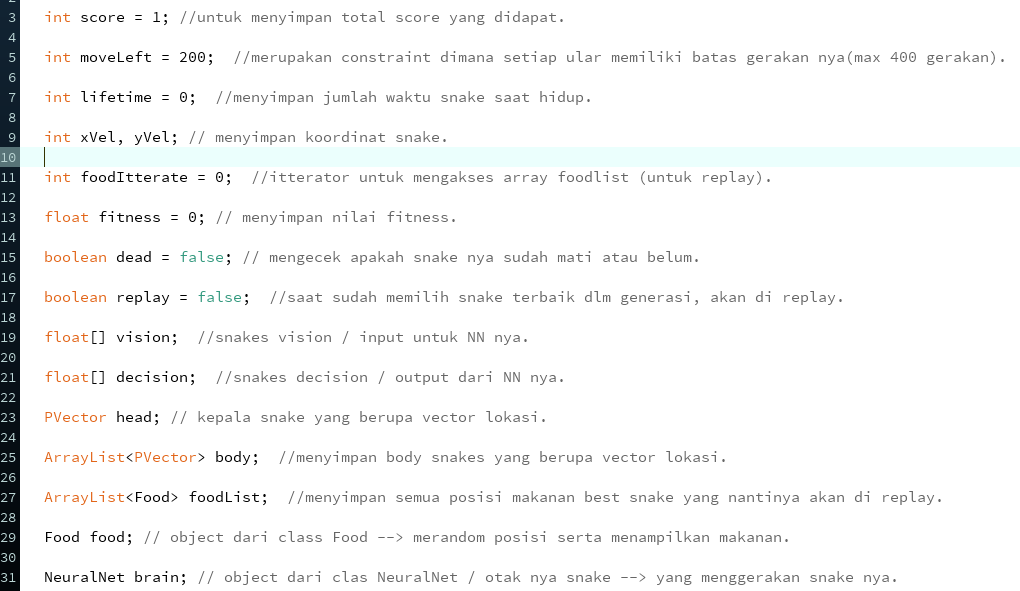
Tabel 4.2

Jika kita bandingkan hasil percobaan antara tabel 4.1 dan tabel 4.2, maka kita dapat menemukan perbedaan yang signifikan pada awal snake bekerja. Tabel 4.1 terlihat lebih konstan dan lonjakan yang lebih tinggi terhadap highscorenya, sedangkan Tabel 4.2 terlihat lebih laju dan tidak berulang, untuk perubahan highscorenya. Ini dikarenakan rasio mutasi yang diberikan Tabel 4.2 lebih tinggi. Tidak untuk menyebut bahwa semakin tinggi rate mutasi semakin baik, tetapi hasil yang diberikan semakin berbeda. Untuk highscore pada generasi ke 20, Tabel 4.1 lebih unggul 3 poin dibanding Tabel 4.2.

**Welliam Sastradipura - c14190194**

Eksperimen 1 (Menghapus Constraint moveLeft) :

Dalam Pembuatan Class Snake, terdapat variabel-variabel yang di deklarasi seperti,

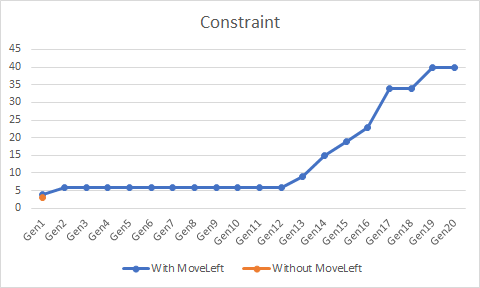


**Gambar 5.1**

Dari gambar 5.1, apa yang terjadi jika kita tidak menggunakan constraint moveLeft? berikut penjelasan nya:

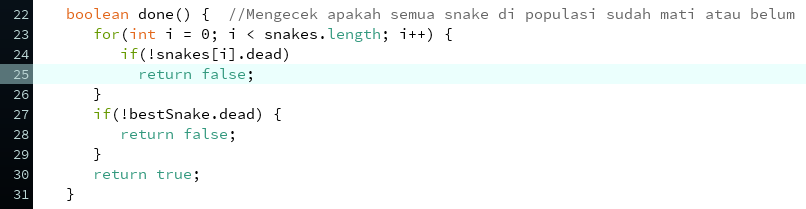
|  |  |  |
| --- | --- | --- |
| Generation | With MoveLeft | Without MoveLeft |
| Gen1 | 4 | 3 |
| Gen2 | 6 | - |
| Gen3 | 6 | - |
| Gen4 | 6 | - |
| Gen5 | 6 | - |
| Gen6 | 6 | - |
| Gen7 | 6 | - |
| Gen8 | 6 | - |
| Gen9 | 6 | - |
| Gen10 | 6 | - |
| Gen11 | 6 | - |
| Gen12 | 6 | - |
| Gen13 | 9 | - |
| Gen14 | 15 | - |
| Gen15 | 19 | - |
| Gen16 | 23 | - |
| Gen17 | 34 | - |
| Gen18 | 34 | - |
| Gen19 | 40 | - |
| Gen20 | 40 | - |

**Tabel 5.1**



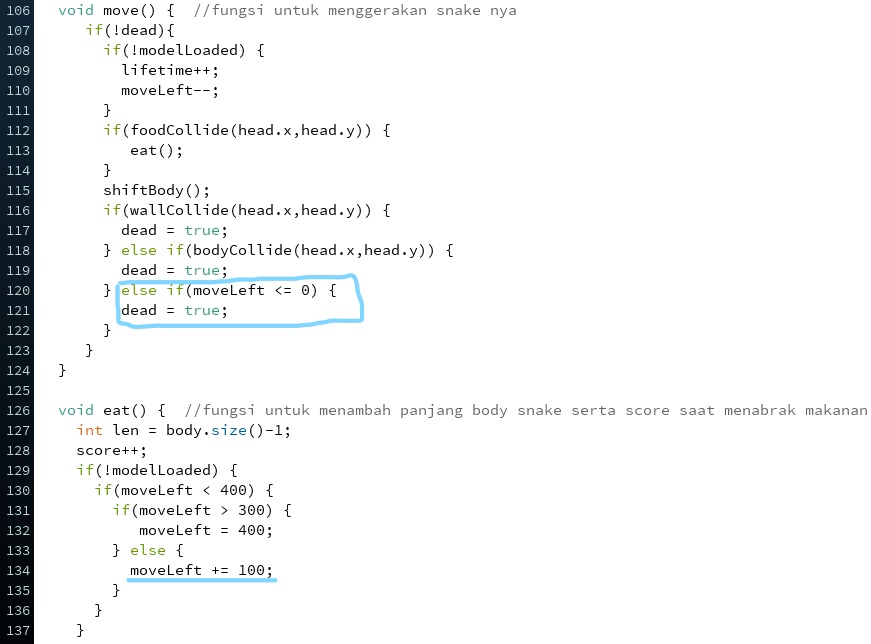
**Chart 5.1**

Bisa dilihat, perbandingan antara menggunakan Constraint moveLeft dan tidak. Pada Tabel 5.1, game Snake yang menggunakan Constraint moveLeft berkembang setiap generasi nya, atau bisa dibilang proses Genetic Algorithm nya berjalan dengan lancar. Sedangkan game Snake yang tidak menggunakan constraint moveLeft, tidak berkembang dan terus berada di generasi 1. Ada 1 hal yang menyebabkan masalah tersebut, yaitu proses GA menuju generasi berikut nya akan dilakukan pada saat semua snake mati.



**Gambar 5.2**

Bisa dilihat di gambar 5.2, suatu fungsi dalam Class Population(GA) yang mengecek apakah semua snake sudah mati atau belum, dan jika belum maka permainan snake akan terus berlanjut. Beberapa snake dari 1 populasi fokus untuk bertahan hidup daripada memakan makanan atau bisa dibilang infinite looping. Oleh karena itu diperlukan sebuah ‘Constraint’ yang membatasi pergerakan snake yaitu moveLeft.



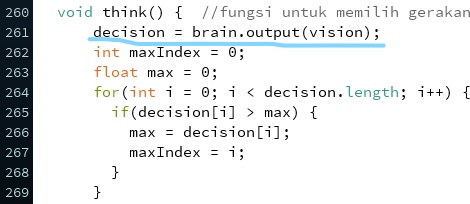
**Gambar 5.3**

Dari gambar 5.3, Jika snake sudah melewati batas pergerakan nya, maka snake akan langsung mati (di Function move), dan jika snake memakan makanan maka sisa pergerakan yang dimiliki snake tersebut akan di +100 (di Function eat).

Perlu di ingat juga bahwa perhitungan Fitness Score didapat dari variabel Score dan LifeTime snake, dimana kedua nilai tersebut didapatkan dari hasil permainan neural net dan crossover. Semakin lama snake bertahan hidup, semakin besar pula Fitness Score yang dihasilkan.

Eksperimen 2 (Membandingkan ReLu dan Sigmoid function) :

Saat snake ingin membuat decision, dibutuhkan output dari Neural Net nya seperti gambar di bawah.



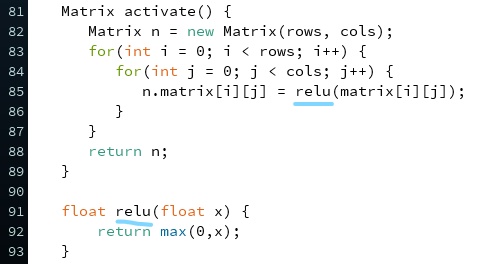
**Gambar 5.4**



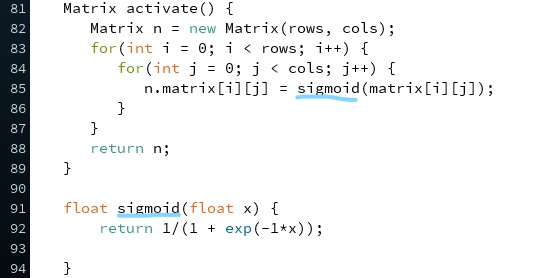
**Gambar 5.5**

Fungsi Output (gambar 5.5) dalam Class Neural Net digunakan untuk melakukan perkalian nilai nodes(dari input) dan weight(nilai random) snake, kemudian ditambahkan dengan bias dan dilakukan fungsi matematika yaitu ReLu di hasil akhir perhitungan. Setelah selesai, fungsi output akan me-return matrix 4x1 dimana nanti nya di fungsi think (gambar 5.4) akan mencari nilai terbesar di matrix tersebut, kemudian index nilai terbesar akan menjadi acuan pergerakan snake.

Eksperimen ini bertujuan untuk mengganti fungsi matematika dalam Class Matrix yaitu ReLu dengan Sigmoid dan melihat perbedaan nya.



**Gambar 5.6**

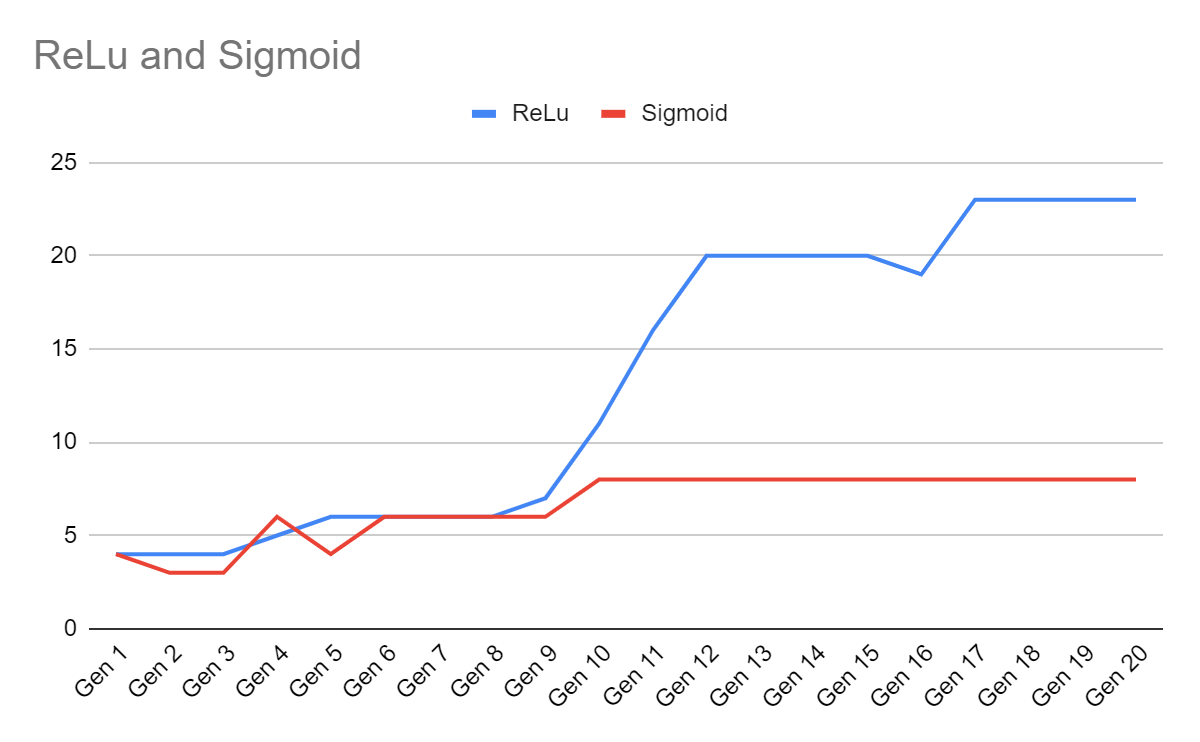


**Gambar 5.7**

Gambar 5.6 adalah fungsi activate yang menggunakan ReLu sedangkan gambar 5.7 adalah fungsi activate menggunakan Sigmoid. Berikut adalah data hasil percobaan :

|  |  |  |
| --- | --- | --- |
| Generasi | ReLu | Sigmoid |
| Gen 1 | 4 | 4 |
| Gen 2 | 4 | 3 |
| Gen 3 | 4 | 3 |
| Gen 4 | 5 | 6 |
| Gen 5 | 6 | 4 |
| Gen 6 | 6 | 6 |
| Gen 7 | 6 | 6 |
| Gen 8 | 6 | 6 |
| Gen 9 | 7 | 6 |
| Gen 10 | 11 | 8 |
| Gen 11 | 16 | 8 |
| Gen 12 | 20 | 8 |
| Gen 13 | 20 | 8 |
| Gen 14 | 20 | 8 |
| Gen 15 | 20 | 8 |
| Gen 16 | 19 | 8 |
| Gen 17 | 23 | 8 |
| Gen 18 | 23 | 8 |
| Gen 19 | 23 | 8 |
| Gen 20 | 23 | 8 |

**Tabel 5.2**

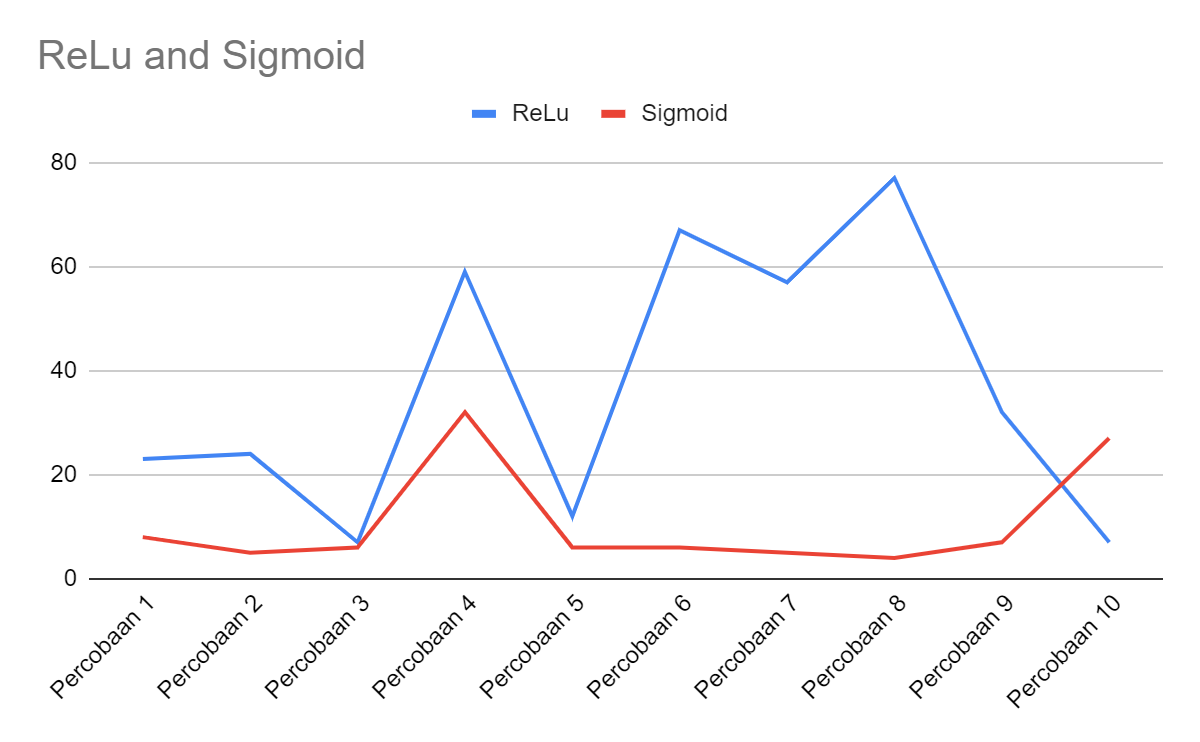


**Chart 5.2**

Tabel 5.2 dan Chart 5.2 adalah hasil perbandingan Score dari best snake dalam 20 generasi. Bisa dilihat bahwa game Snake yang menggunakan ReLu jauh lebih tinggi perkembangan nya dari pada menggunakan Sigmoid.

|  |  |  |
| --- | --- | --- |
| Percobaan | ReLu | Sigmoid |
| Percobaan 1 | 23 | 8 |
| Percobaan 2 | 24 | 5 |
| Percobaan 3 | 7 | 6 |
| Percobaan 4 | 59 | 32 |
| Percobaan 5 | 12 | 6 |
| Percobaan 6 | 67 | 6 |
| Percobaan 7 | 57 | 5 |
| Percobaan 8 | 77 | 4 |
| Percobaan 9 | 32 | 7 |
| Percobaan 10 | 7 | 27 |
| **Rata-rata** | **36.5** | **10.6** |

**Tabel 5.3**

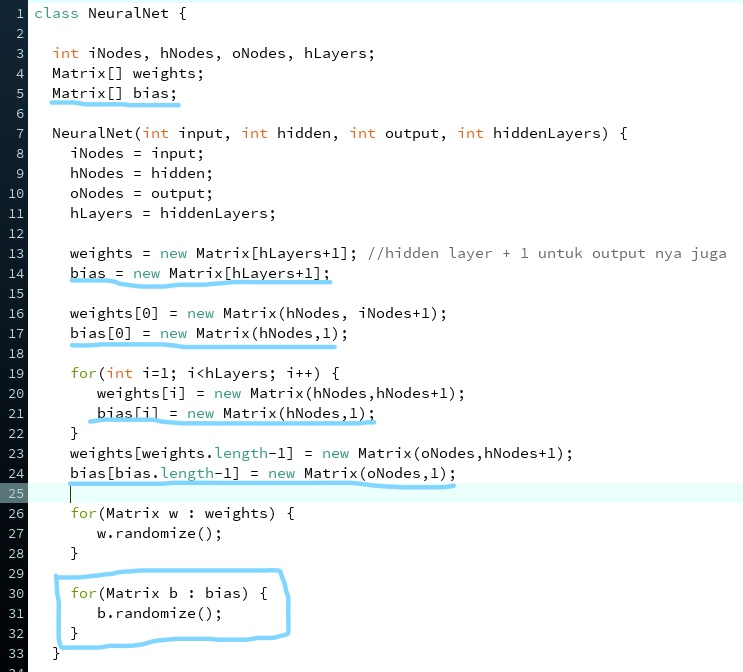


**Chart 5.3**

Tabel 5.3 dan Chart 5.3 adalah perbandingan High Score game snake di generasi ke 20 dalam 10 kali percobaan. Bisa dilihat bahwa rata-rata score game snake yang menggunakan ReLu lebih tinggi dibandingkan Sigmoid.

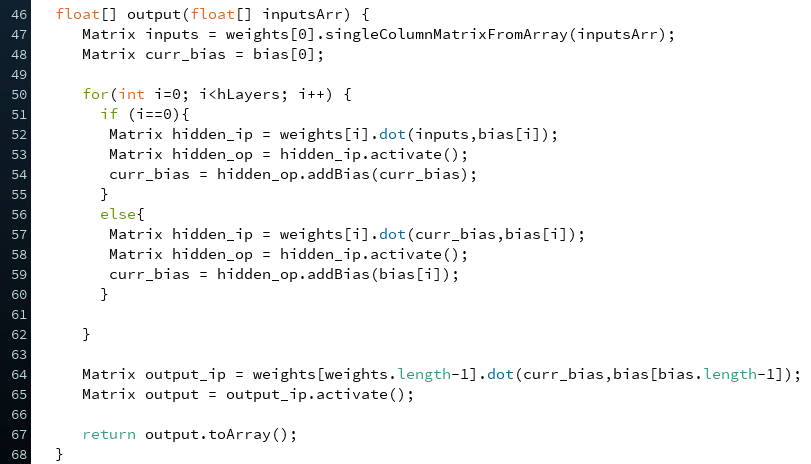
Kesimpulan yang diambil adalah fungsi ReLu lebih baik dan cepat perkembangannya daripada fungsi Sigmoid.

Eksperimen 3 (Meng-Crossover Bias dalam Neural Net) :

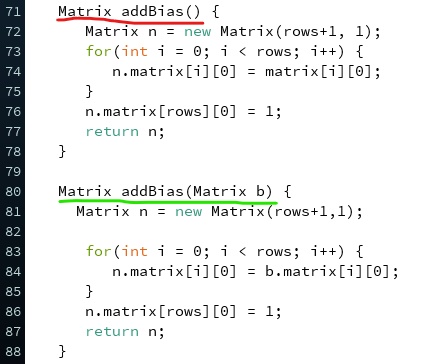
Saat melakukan penghitungan nilai output di class Neural Net (gambar 5.5), nilai bias adalah 0. Dalam eksperimen ini, nilai Bias akan di inisialisasi dan di random tiap nilai nya, kemudian di crossover tiap generasi dan melihat apa yang akan terjadi pada permainan snake. 

**Gambar 5.8**

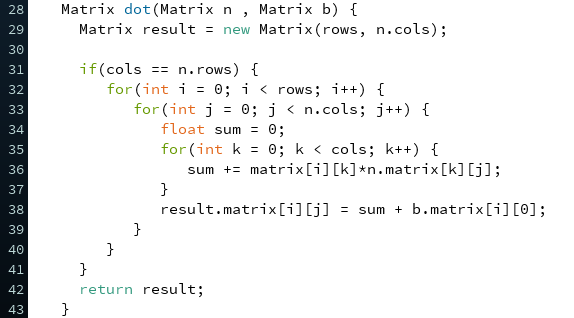
Gambar 5.8 adalah hasil modifikasi dari kodingan sebelum nya dimana ada variabel Matrix[] bias (garis biru) untuk menyimpan nilai-nilai bias nya saat di random. Kemudian di dalam constructor, matrix bias di inisialisasi ukuran serta nilai matrix nya, sama seperti inisialisasi matrix weights hanya saja jumlah kolom matrix bias selalu 1.



**Gambar 5.9**

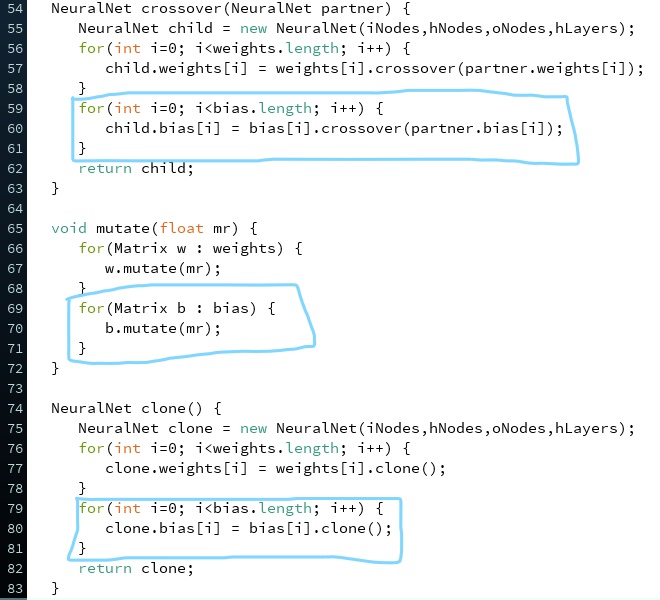


**Gambar 5.10**



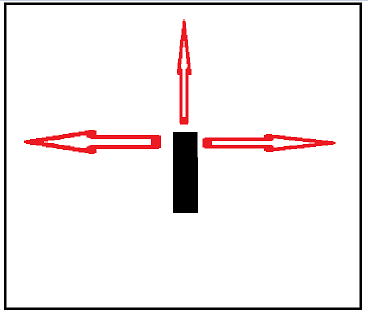
**Gambar 5.11**

Gambar 5.9 adalah fungsi output di class Neural Net yang juga sudah dimodifikasi sesuai kebutuhan. Gambar 5.10 adalah fungsi addBias di class Matrix yang digunakan di fungsi output Neural Net, dimana ada beberapa perubahan juga. Fungsi addBias (garis merah) adalah fungsi yang lama, sedangkan addBias dengan parameter (garis hijau) adalah fungsi yang baru. Gambar 5.11 adalah fungsi di class Matrix yang digunakan juga di fungsi output class Neural Net untuk mengalikan nilai nodes dengan weights, yang kemudian ditambah dengan nilai bias.



**Gambar 5.12**

Gambar 5.12 adalah penambahan matrix bias pada fungsi crossover, mutasi, dan clone. Setelah itu permainan snake bisa di Run.



**Gambar 5.13**

Setelah di run, ternyata pergerakan snake nya menjadi parah / kurang fleksibel. Snake akan terus bergerak ke arah yang dipilih secara konstan hingga menabrak tembok seperti gambar 5.13.

Kesimpulan nya adalah nilai bias yang digunakan sebaik nya tetap 0, tidak perlu diinisialisasi dan diberi nilai random serta di crossover dan mutasi. Akibat modifikasi tersebut menyebabkan pergerakan snake menjadi kurang fleksibel dimana snake akan bergerak 1 arah saja hingga menabrak tembok.