BAB 3
ANALISIS KEBUTUHAN ALGORITMA

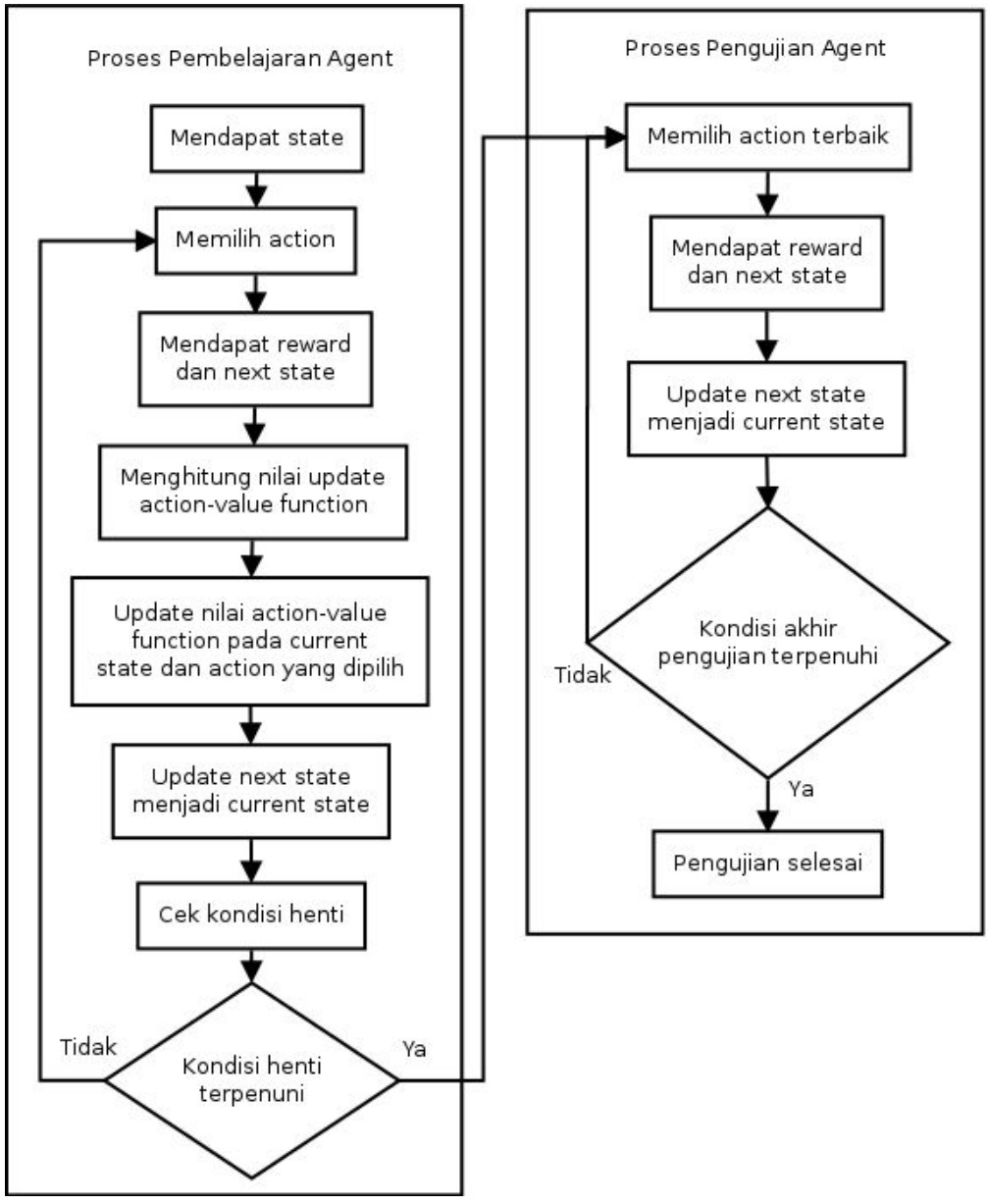
3.1. Analisis Masalah

Flappy Bird mempunyai ukuran *state* yang besar, mulai dari posisi burung dan posisi pipa yang selalu berubah setiap waktu, perubahan beberapa *pixel* saja merupakan state yang berbeda, maka penggunaan *function approximation* diharapkan akan membuat proses pembelajaran lebih cepat dan dapat mengurangi jumlah bobot yang disimpan di *memory* namun tidak mengurangi kemampuan *agent* dalam memainkan permainan *Flappy Bird* jika dibandingkan dengan penggunaan *Q-learning* saja. Maka dari itu bagaimana cara mengimplementasikan *Q-learning* yang dikombinasikan dengan *backpropagation* untuk memainkan permainan *Flappy Bird* lalu melihat pengaruh dari *Q-learning* yang dikombinasikan dengan *backpropagation* terhadap waktu pembelajaran dan bobot yang disimpan *agent* untuk memainkan permainan *Flappy Bird*.

3.2. Analisis Sistem

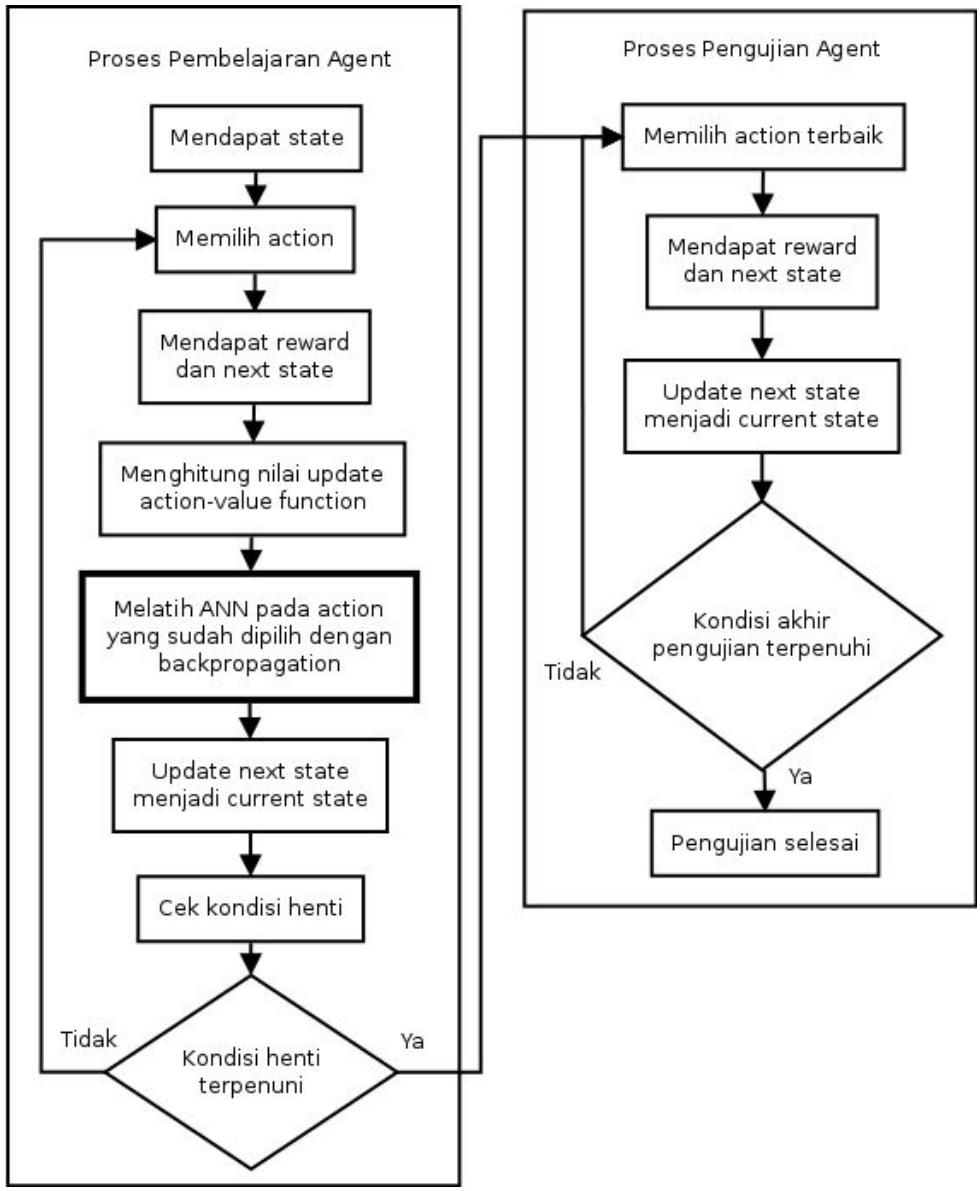
Sistem meliputi dua tahap yaitu proses pembelajaran *agent* dan proses pengujian *agent*. Proses pembelajaran *agent* dilakukan sampai kondisi henti terpenuhi. Jika kodisi henti sudah terpenuhi maka *agent* akan masuk ke proses pengujian. Pada proses pengujian, *agent* akan selalu memilih *action* terbaik, dan tidak lagi melakukan penghitungan nilai *update action value function* dan tidak lagi melakukan *backpropagation*.

Berbeda dengan proses pembelajaran *Q-learning* saja (Gambar 3.1) yang melakukan update pada nilai *action-value function* secara langsung seperti pada persamaan (2.2). *Q-learning* yang dikombinasikan dengan *backpropagation* (Gambar 3.2) melatih pola *state* dan nilai *update action-value function* sebagai nilai target atau *output* yang diharapkan dengan *backpropagation* pada *action* yang sudah dipilih. Adapun Blok diagram dari sistem *Q-learning* yang dikombinasikan dengan *backpropagation* bisa dilihat pada Gambar 3.2.



Gambar 3.1 Gambaran Umum Sistem *Q-learning*

Kondisi henti pada proses pembelajaran *Q-learning* adalah waktu pembelajaran yang sudah ditentukan. Sedangkan kondisi akhir pengujian adalah sampai agent mendapat *score* 1000.



Gambar 3.2 Gambaran Umum Sistem *Q-learning* yang Dikombinasikan dengan *Backpropagation*

Kondisi henti pembelajaran dari *Q-learning* yang dikombinasikan dengan *backpropagation* adalah berdasarkan *score* yang sudah ditentukan penjelasan lebih lanjut akan dipaparkan pada Bab 3.4. Sedangkan kondisi akhir pengujian adalah sampai agent mendapat *score* 1000. Detail dari proses pembelajaran dijelaskan lebih lanjut di bagian Analisis Metode *Q-Learning* yang dikombinasikan dengan *Backpropagation*.

3.3. Analisis *Input*

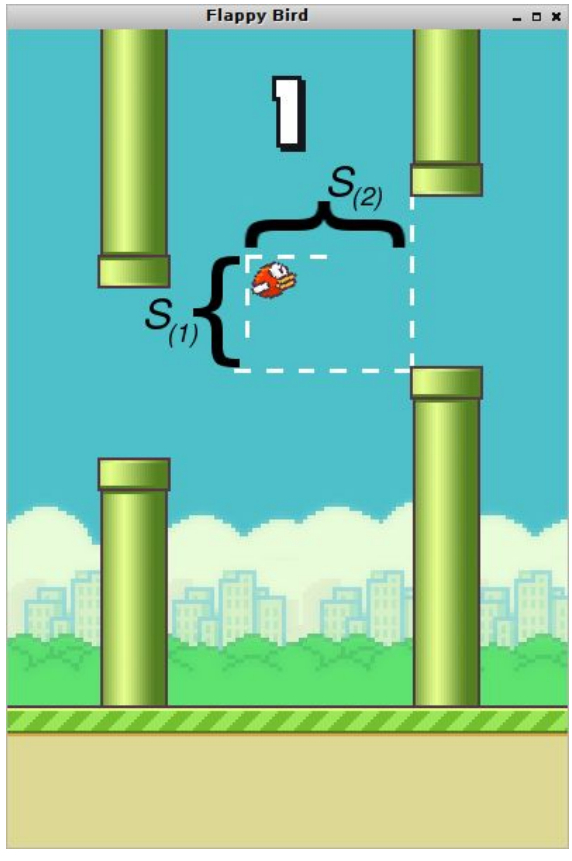
Pada penelitian implementasi dari *Q-learning* yang dikombinasikan dengan *backpropagation* ini, *input* yang digunakan antara lain:

1) *State*

State yang digunakan pada penelitian ini antara lain:

1. Selisih antara burung dan pipa bagian bawah didepannya $S_{(1)}$.
2. Jarak antara burung dan pipa didepannya $S_{(2)}$.

Gambar 3.3 menunjukan ilustrasi nilai yang digunakan untuk *state*.



Gambar 3.3 Ilustrasi Nilai untuk *State*

Selisih antara burung dan pipa bagian bawah didepannya didapat dari kordinat y pipa bagian bawah yang ada di depannya dikurang kordinat y burung.

Sedangkan jarak antara burung dan pipa didepannya didapatkan dari nilai kordinat x pipa didepannya dikurang kordinat x burung. Dua nilai tersebut dibagi dengan 100 sebelum dijadikan *input* ke *artificial neural network* sehingga perubahan bobotnya tidak terlalu jauh. Nilai 100 dipilih karena ukuran dari *frame* bernilai ratusan *pixel* sehingga nantinya akan menjadi nilai satuan.

2) *Action*

Action yang digunakan pada penelitian ini adalah terbang dan diam. Nilai *action* didefinisikan sebagai nilai *integer*, terbang = 1 dan diam = 0.

3) *Reward*

Reward yang digunakan pada penelitian ini antara lain:

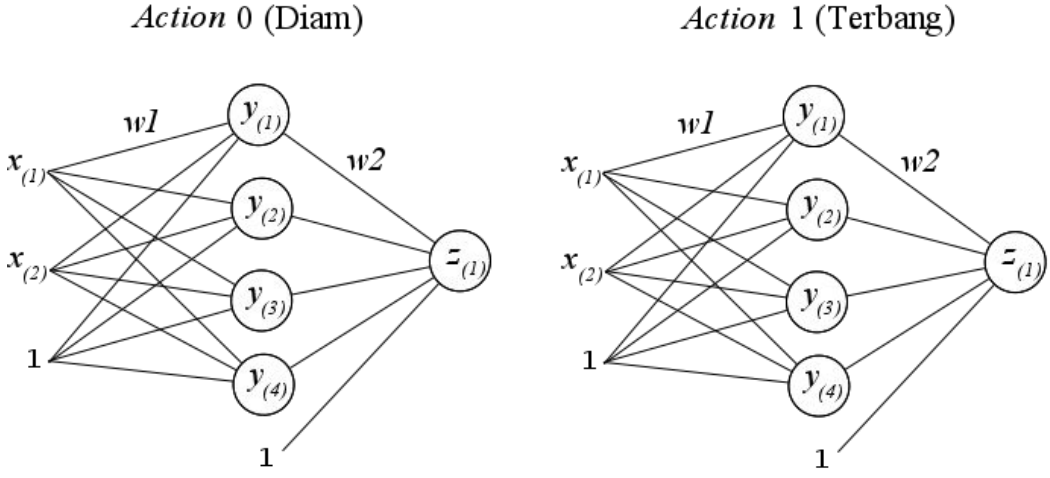
1. -1 ketika burung mati.
2. +0.1 ketika burung hidup.
3. +1 ketika *score* bertambah.

Penentuan *reward* dipengaruhi *activation function* yang digunakan. Penelitian ini menggunakan *sigmoid bipolar* dengan rentang nilai (-1,1). Jika dilihat dari tujuannya *agent* harus melewati pipa sebanyak banyaknya maka ketika berhasil melewati pipa harus diberi *reward* maksimal, dan ketika mati harus diberi *reward* seminimal mungkin. Ketika burung hidup namun belum melewati pipa, itu belum tentu situasi yang baik atau buruk bagi agent, namun pada penelitian ini diberi *reward* positif yang kecil. Nilai positif kecil dipilih agar ketika berhasil melewati pipa dan mendapat *reward* maksimal nantinya bisa melakukan perubahan *action-value function* lebih cepat, dan tidak terlalu jauh ketika mendapat *reward* minimal.

3.4. Analisis Metode *Q-Learning* yang Dikombinasikan dengan *Backpropagation*

Analisis metode dilakukan untuk menganalisis bagaimana proses pembelajaran dari *Q-learning* yang dikombinasikan dengan *backpropagation*

untuk memainkan *Flappy Bird*. Satu peristiwa pada penelitian ini adalah dari awal hingga akhir pembelajaran, sedangkan setiap langkahnya adalah setiap *frame* yang ditampilkan. Adapun arsitektur ANN yang digunakan terlihat pada Gambar 3.4.



Gambar 3.4 Arsitektur ANN yang Digunakan

Gambar 3.4 menunjukkan artisektur untuk satu *artificial neural network* tiap kemungkinan *action*. Masing-masing *artificial neural network* memiliki 2 *input* unit yang didapat dari *state*, 4 *hidden neuron* (penggunaan 4 hidden neuron merupakan pemilihan awal, nilai lainnya juga akan di uji pada bab selanjutnya), dan 1 *output neuron* sebagai nilai *action-value function*. Proses pembelajaran terdiri dari beberapa tahap antara lain:

1) Inisialisasi bobot masing-masing ANN dengan nilai *random*

Pertama *agent* akan melakukan inisialisasi bobot masing-masing ANN dengan nilai *random* antara 0 sampai 1.

2) Dapatkan *State*

Selanjutnya *agent* akan mendapat *state*, *state* yang didapat seperti yang sudah dijelaskan di bagian analisis input.

3) Pilih *Action* dari *State* dengan ε-*greedy*

Selanjutnya agent akan memilih action yang akan dilakukan dengan ε-*greedy.* Nilai ε (epsilon) yang digunakan adalah 0.0, ini berarti bahwa *agent* akan selalu

memilih *action* dengan nilai *action-value function* terbesar. Pemilihan nilai tersebut karena pada kasus ini *action* akan diminta sangat cepat tiap *frame* nya, jika ada pemilihan *random action* dan *action* yang diambil salah, maka itu akan sangat mengganggu proses pembelajaran, *agent* akan sulit mencari solusi yang optimal.

4) Dapatkan *Reward* dan *Next State*

Selanjutnya *agent* akan mendapat *reward* dan *next state* dari *action* yang sudah dilakukan.

5) Hitung nilai *update* untuk *action-value function*

Selanjutnya *agent* akan menghitung nilai *update action-value function* seperti pada persamaan (2.2) yang nantinya akan digunakan sebagai *output* yang diharapakan pada proses *backpropagation*. *Learning rate* yang digunakan adalah 0.9, karena dibutuhkan perubahan yang besar pada setiap nilainya untuk mencapai solusi optimal. Sedangankan *discount rate* yang digunakan adalah 0.9, karena agent harus lebih mementingkan *reward* dimasa depan daripada *reward* yang dekat. Nilai lain pada parameter ini juga akan diuji pada bab selanjutnya.

6) Latih ANN pada *action* yang sudah dipilih dengan *backpropagation*

Selanjutnya agent akan melatih ANN pada action yang sudah dipilih dengan *input state* dan *ountput* yang diharapkan didapat dari nilai *update action-value function*. *Learning rate* yang digunakan adalah 0.9 karena dibutuhkan perubahan yang besar pada setiap nilainya untuk mencapai nilai dari *action-value function* yang sebenarnya. Nilai lain pada parameter ini juga akan diuji pada bab selanjutnya.

7) *Update Next State* menjadi *Current State*

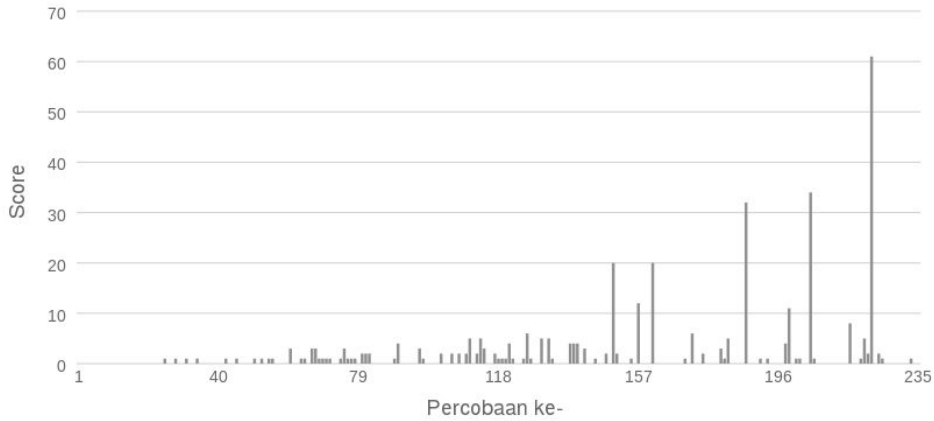
Selanjutnya *agent* akan melakukan *update next state* yang didapat menjadi *current state*.

8) Cek kondisi penyimpanan bobot dan kondisi henti

Kondisi henti diambil berdasarkan *score* yang didapat. Batas score yang dijadikan acuan adalah 30, ketika sudah mencapai batas *score* maka proses pembelajaran akan dihentikan, namun sebelumnnya bobot saat *score* 14 disimpan terlebih dahulu dan nantinya akan digunakan setelah proses pembelajaran selesai.

Jika *score* tidak mencapai 30 maka akan kembali ke langkah 3.

Kondisi henti tersebut ditentukan berdasarkan hasil percobaan yang dilakukan sebelum penentuan kondisi henti. Gambar 3.5 menunjukan salah satu hasil dari beberapa kali percobaan *agent* dalam pembelajaran. Grafik tersebut menunjukkan ada beberapa kali *agent* mendapat *score* yang relatif besar (*Score* 20, 32, 34, 61) diantara *score* yang lain kemudian setelah itu *score* dari percobaan percobaan selanjutnya akan mendapat *score* yang kecil beberapa kali dikarenakan ANN mengalami pelatihan berlebih di beberapa *state* dan mempengaruhi pola *state* lain yang tidak dilatih, setelah beberapa kali mendapat *score* kecil agent kembali mendapat *score* yang relatif besar. Bobot yang ada di tengah *score* yang relatif besar tersebut sudah bisa digunakan sebagai solusi optimal.



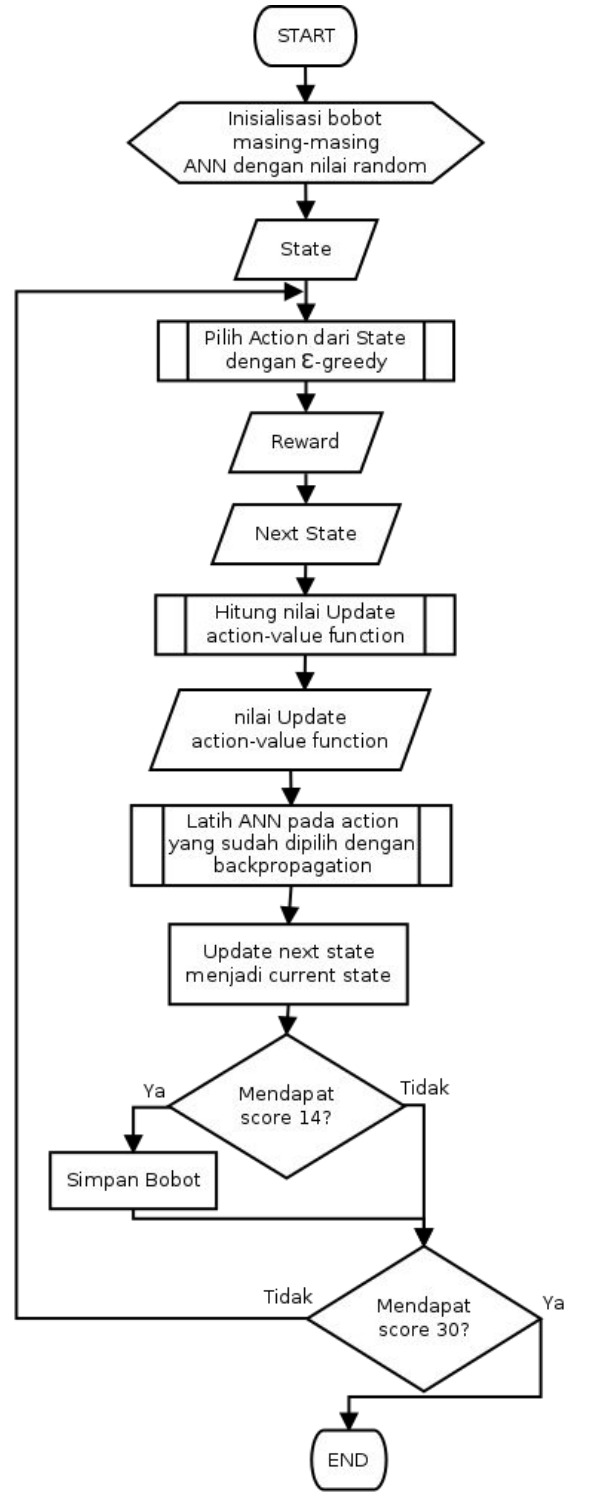
Gambar 3.5 Grafik Hasil Percobaan

Pemilihan *score* 14 sebagai kondisi penyimpanan bobot dan nilai 30 sebagai batas *score* pada kondisi henti adalah sebagai hipotesis awal karena nilai 30 dirasa sudah mencukupi dan tidak terlalu awal untuk diambil bobot pada nilai ditengahnya, *score* 14 diambil karena saat mendapat *score* 14 itu merupakan bobot ditengah saat nilai 30, karena sesaat sebelum mendapat *score* 15 masih dalam dalam kondisi *score* 14. Nilai lainnya juga akan diuji pada bab selanjutnya.

Adapun proses tambahan setiap 2 menit akan dilakukan pengecekan apakah dalam dua menit tersebut aget tidak mendapat score sama sekali, jika iya maka akan melakukan inisialisasi ulang bobot ANN. Inisialisasi ulang dibutuhkan

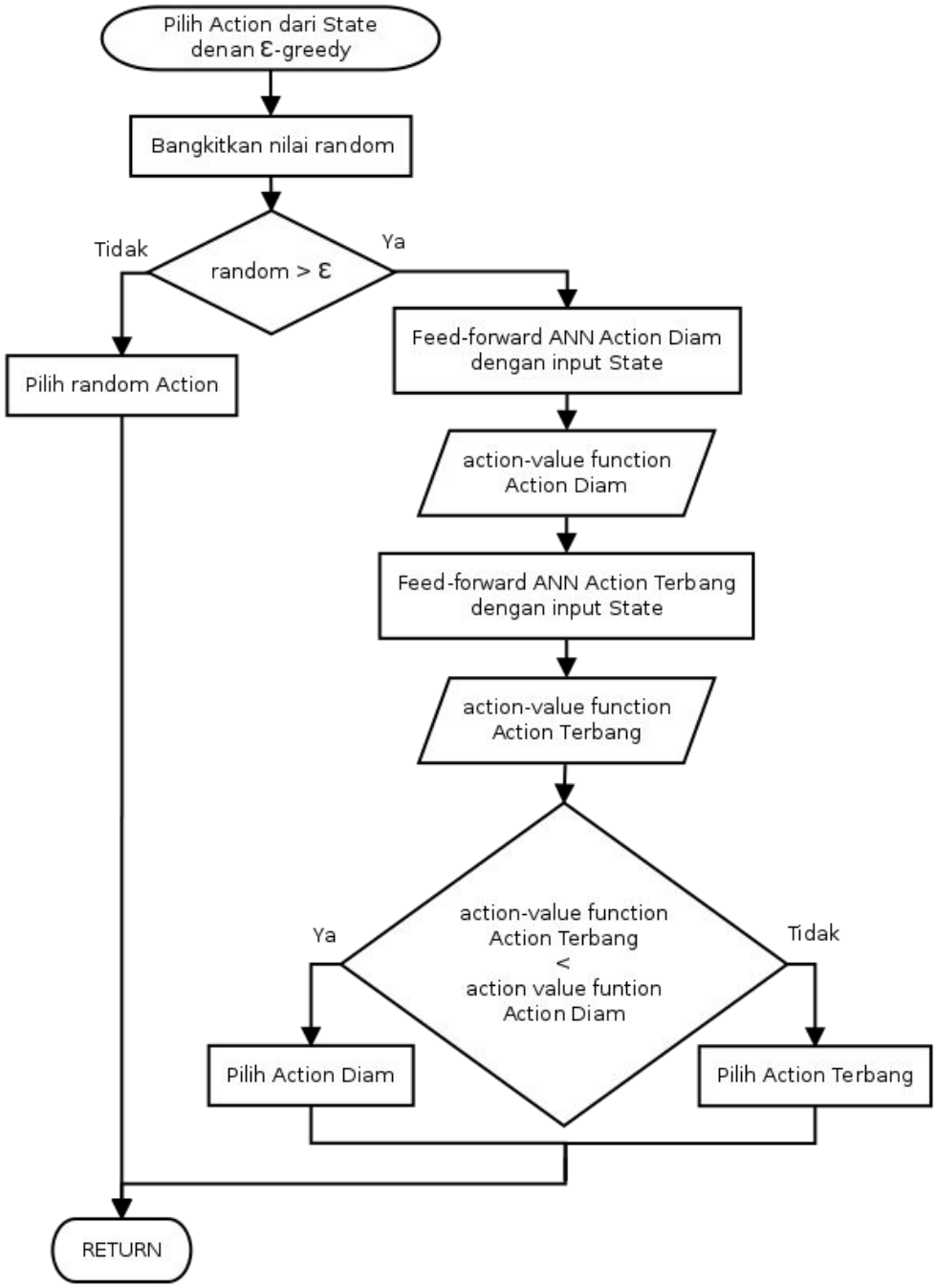
karena ada kondisi ketika pembelajaran tidak menemukan solusi hingga lebih dari 30 menit berdasarkan percoban sebelumnya. Waktu 2 menit dipilih karena dirasa tidak terlalu cepat dan tidak terlalu lama untuk memperkirakan bahwa proses pembelajaran tidak akan menemukan solusi. Nilai lainnya juga akan diuji pada bab selanjutnya.

Jika Proses pembelajaran selesai maka *agent* akan menggunakan bobot yang telah disimpan untuk proses pengujian. *Agent* tidak akan lagi memilih *action* dengan ε-*greedy*, tetapi akan selalu memilih *action* dengan nilai *action-value function* terbesar, *agent* juga tidak lagi menghitung nilai *update* untuk *action-value function* dan tidak lagi melatih ANN. Adapun *flowchart* dari proses pembelajaran *Q-learning* yang dikombinasikan dengan *backpropagation* terlihat pada Gambar 3.6 *Flowchart* Proses Pembelajaran.



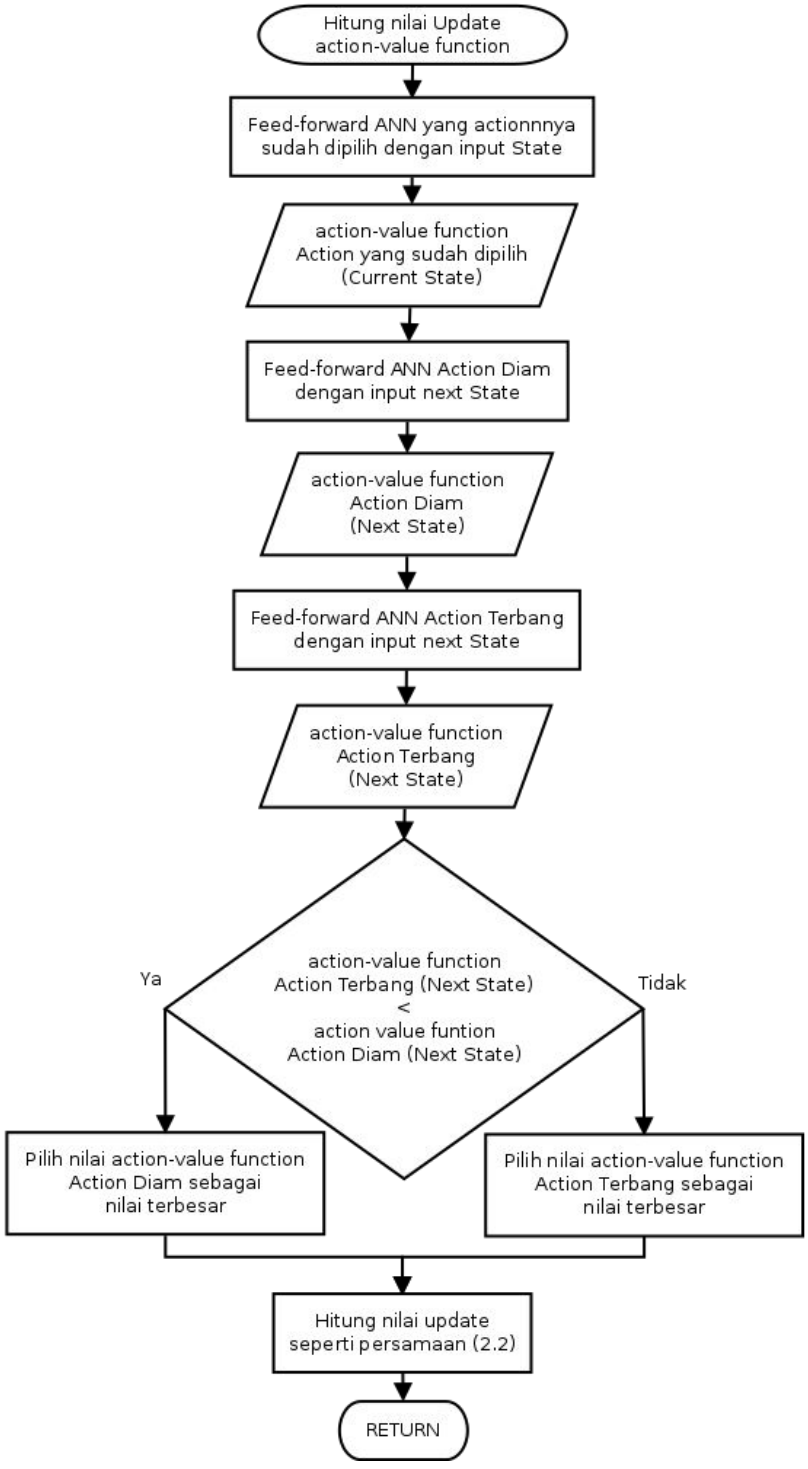
Gambar 3.6 *Flowchart* Proses Pembelajaran

Gambar 3.7 menunjukkan detil *flowchart* dari *predefined process* Pilih *Action* dari *State* dengan ε-*greedy* di Gambar 3.5 *Flowchart* proses pembelajaran.



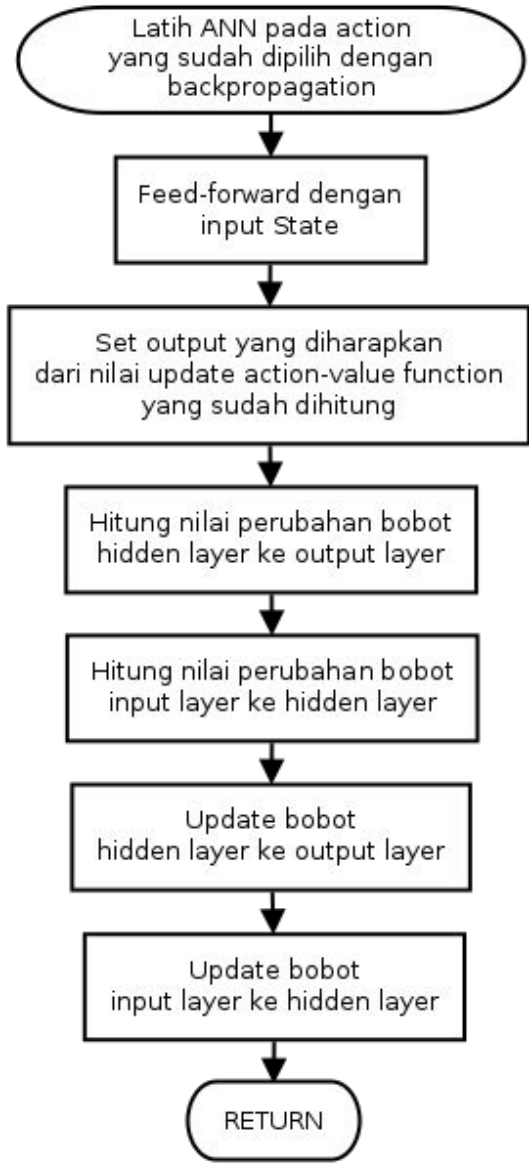
Gambar 3.7 Flowchart Pilih *Action* dari *State* dengan ε-*greedy*

Gambar 3.8 menunjukkan detil *flowchart* dari *predefined process* Hitung nilai *Update action-value function* di Gambar 3.6 *Flowchart* proses pembelajaran.



Gambar 3.8 Hitung Nilai *Update Action-Value Function*

Gambar 3.9 menunjukkan detil *flowchart* dari *predefined process* Latih ANN pada *action* yang sudah dipilih dengan *backpropagation* di Gambar 3.6 *Flowchart* proses pembelajaran.



Gambar 3.9 Latih ANN pada *Action* yang Sudah Dipilih dengan *Backpropagation*

Adapun contoh proses pembelajaran satu kali iterasi adalah sebagai berikut:

1) Inisialisasi bobot masing-masing ANN dengan nilai *random*

Inisialisasikan bobot *w1* dan *w2* pada masing-masing ANN dengan nilai *random* antara 0 sampai 1.

Action 0 (Diam)

$w1_{(1,1)} = 0.3737467985596139$ $w2_{(1,1)} = 0.6981433296189277$
$w1_{(1,2)} = 0.3272211118251288$ $w2_{(2,1)} = 0.6184307761972161$
$w1_{(1,3)} = 0.9529138643629589$ $w2_{(3,1)} = 0.992909937330567$
$w1_{(1,4)} = 0.5823802930171551$ $w2_{(4,1)} = 0.2334423547441814$
$w1_{(2,1)} = 0.13212418148869665$ $w2_{(5,1)} = 0.012739708821004059$
$w1_{(2,2)} = 0.3239285971112901$
$w1_{(2,3)} = 0.3393466407427368$
$w1_{(2,4)} = 0.15440917597409076$
$w1_{(3,1)} = 0.02705724989590974$
$w1_{(3,2)} = 0.06982251855540678$
$w1_{(3,3)} = 0.4996754871565079$
$w1_{(3,4)} = 0.9192497675928927$

Action 1 (Terbang)

$w1_{(1,1)} = 0.3143692393836913$ $w2_{(1,1)} = 0.5790454022610518$
$w1_{(1,2)} = 0.09314410902453651$ $w2_{(2,1)} = 0.5090528449625555$
$w1_{(1,3)} = 0.02879340957844745$ $w2_{(3,1)} = 0.5109558325241379$
$w1_{(1,4)} = 0.5364724047280168$ $w2_{(4,1)} = 0.42125968287763105$
$w1_{(2,1)} = 0.7832510663743999$ $w2_{(5,1)} = 0.1408635321980527$
$w1_{(2,2)} = 0.13009746335307137$
$w1_{(2,3)} = 0.8366555320866069$
$w1_{(2,4)} = 0.662049259269347$
$w1_{(3,1)} = 0.7008056647857615$
$w1_{(3,2)} = 0.7581777721646513$

$w1_{(3,3)}$ = 0.3871213745104467

$w1_{(3,4)}$ = 0.9536038571958243

2) Dapatkan *State*

Agent mendapat *state* dari *environment*. Seperti yang sudah dijelaskan di analisis input, state yang didapat antara lain:

1. Selisih antara posisi burung dan pipa bagian bawah didepannya didapat dari kordinat y pipa bagian bawah - kordinat y burung. Selanjutnya nilai tersebut dibagi 100.
   $S_{(1)}$ = (363-250)/100 = 1.13
2. Jarak antara burung dan pipa didepannya didapat dari kordinat x pipa didepannya - kordinat x burung. Selanjutnya nilai tersebut dibagi 100.
   $S_{(2)}$ = (300-135)/100 = 1.65

$x_{(1)}$ didapat dari $S_{(1)}$ yaitu 1.13.

$x_{(2)}$ didapat dari $S_{(2)}$ yaitu 1.65.

3) Pilih *Action* dari *State* dengan ε-*greedy*

Pertama bangkitkan bilangan *random* antara 0 sampai 1.

random = 0.4369708854999099

Karena nilai *random* > ε maka *agent* memilih *action* dengan nilai *action-value* terbesar.

Selanjutnya lakukan *feed-forward* pada masing-masing *artificial neural network* dengan *input state* yang didapat untuk mendapatkan nilai *action-value function*.

$x_{(1)}$ = 1.13

$x_{(2)}$ = 1.65

<u>*Action* 0 (Diam)</u>

Hitung nilai sigma dari *w1*x* seperti pada persamaan (3.1).

$$y_net_{(j)} = \sum_{i=1}^{a+1} w1_{(i,j)} x_{(i)} \tag{3.1}$$

$y_net_{(1)}$ = 1.13*0.3737467985596139 + 1.65*0.13212418148869665 + 1.0*0.02705724989590974

= 0.6673960317246229

$y_net_{(2)}$ = 1.13*0.3272211118251288 + 1.65*0.3239285971112901 + 1.0*0.06982251855540678

= 0.9740645601514308

$y_net_{(3)}$ = 1.13*0.9529138643629589 + 1.65*0.3393466407427368 + 1.0*0.4996754871565079

= 2.136390111112167

$y_net_{(4)}$ = 1.13*0.5823802930171551 + 1.65*0.15440917597409076 + 1.0*0.9192497675928927

= 1.8321146390595278

Hitung *output* dari setiap *neuron* di *hidden layer* seperti pada persamaan (3.2).

$$y_{(j)} = f(y_net_{(j)}) \tag{3.2}$$

Activation function yang digunakan adalah *bipolar sigmoid* seperti pada persamaan (2.2).

$y_{(1)}$ = $2/(1 + e^{-0.6673960317246229})-1$

= 0.3218396843179574

$y_{(2)}$ = $2/(1 + e^{-0.9740645601514308})-1$

= 0.45185781821820314

$y_{(3)}$ = $2/(1 + e^{-2.136390111112167})-1$

= 0.7887802367479924

$y_{(4)}$ = $2/(1 + e^{-1.8321146390595278})-1$

= 0.7240268956961655

Hitung nilai sigma dari *w2*y* seperti pada persamaan (3.3).

$$z_net_{(k)} = \sum_{j=1}^{b+1} w2_{(j,k)} y_{(j)} \tag{3.3}$$

$z_net_{(1)}$ = 0.3218396843179574*0.6981433296189277 +
0.4518578182182031 4*0.6184307761972161 +
0.7887802367479924*0.992909937330567 +
0.7240268956961655*0.2334423547441814 +
1.0*0.012739708821004059
= 1.4690789977521828

Hitung *output* dari setiap *neuron* di *output layer* seperti pada persamaan (3.4).

$$z_{(k)} = f(z_net_{(k)}) \tag{3.4}$$

Activation function yang digunakan adalah *bipolar sigmoid* seperti pada persamaan (2.2).

$z_{(1)}$ = $2/(1 + e^{-1.4690789977521828})-1$
= 0.625834715728151

Action 0 (Diam) pada *state* $S_{(1)}$ = 1.13 dan $S_{(2)}$ = 1.65 menghasilkan nilai *action-value function* 0.625834715728151.

Selanjutnya lakukan *feed-forward* pada ANN Action 1.
$x_{(1)}$ = 1.13
$x_{(2)}$ = 1.65

<u>*Action* 1 (Terbang)</u>
Hitung nilai sigma dari *w1*x* seperti pada persamaan (3.1).

$$y_net_{(1)} = 1.13*0.3143692393836913 + 1.65*0.7832510663743999 + 1.0*0.7008056647857615$$
$$= 2.3484071648070923$$
$$y_net_{(2)} = 1.13*0.09314410902453651 + 1.65*0.13009746335307137 + 1.0*0.7581777721646513$$
$$= 1.0780914298949453$$
$$y_net_{(3)} = 1.13*0.02879340957844745 + 1.65*0.8366555320866069 + 1.0*0.3871213745104467$$
$$= 1.8001395552769937$$
$$y_net_{(4)} = 1.13*0.5364724047280168 + 1.65*0.662049259269347 + 1.0*0.9536038571958243$$
$$= 2.6521989523329057$$

Hitung *output* dari setiap *neuron* di *hidden layer* seperti pada persamaan (3.2). *Activation function* yang digunakan adalah *bipolar sigmoid* seperti pada persamaan (2.2).

$$y_{(1)} = 2/(1 + e^{-2.3484071648070923})-1$$
$$= 0.8256150744735646$$
$$y_{(2)} = 2/(1 + e^{-1.0780914298949453})-1$$
$$= 0.49226526855349895$$
$$y_{(3)} = 2/(1 + e^{-1.8001395552769937})-1$$
$$= 0.716331844444487$$
$$y_{(4)} = 2/(1 + e^{-2.6521989523329057})-1$$
$$= 0.8682927849411683$$

Hitung nilai sigma dari *w2*y* seperti pada persamaan (3.3).

$$z_net_{(1)} = 0.8256150744735646*0.5790454022610518 +$$

0.49226526855349895*0.5090528449625555 +
0.716331844444487*0.5109558325241379 +
0.8682927849411683*0.42125968287763105 +
1.0*0.1408635321980527
= 1.6013118577137373

Hitung *output* dari setiap *neuron* di *output layer* seperti pada persamaan (3.4). *Activation function* yang digunakan adalah *bipolar sigmoid* seperti pada persamaan (2.2).

$z_{(1)} = 2/(1 + e^{-1.6013118577137373})-1$
$= 0.6644033109815832$

Action 1 (Terbang) pada *state* $S_{(1)}$ = 1.13 dan $S_{(2)}$ = 1.65 menghasilkan nilai *action-value function* 0.6644033109815832
Karena *action-value function Action* 1 (Terbang) lebih besar dari *action-value function Action* 0 (Diam), maka *agent* akan memilih *action* 1 (Terbang).

4) Dapatkan *Reward*

Pada *state* $S_{(1)}$ = 1.13 dan $S_{(2)}$ = 1.65 dan *agent* memilih *action* 1 (Terbang), *agent* mendapat *reward* +0.1.

5) Dapatkan *Next State*

Pada *state* $S_{(1)}$ = 1.13 dan $S_{(2)}$ = 1.65 dan *agent* memilih *action* 1 (Terbang), *agent* mendapat *next state* $S'_{(1)}$ = 1.28 dan $S'_{(2)}$ = 1.63.

6) Hitung nilai *update* untuk *action-value function*

Pertama hitung nilai *action-value function* pada *current state*, yang nantinya diperlukan untuk melakukan *update*.

Nilai $Q(S, A)$

Feed-forward pada *artificial neural network* yang *action* nya dipilih dengan *input state* yang didapat untuk mendapat nilai *action-value function*.

$x_{(1)} = 1.13$

$x_{(2)} = 1.65$

Hitung nilai sigma dari *w1*x* seperti pada persamaan (3.1).

$y_net_{(1)}$ = 1.13*0.3143692393836913 + 1.65*0.7832510663743999 + 1.0*0.7008056647857615

= 2.3484071648070923

$y_net_{(2)}$ = 1.13*0.09314410902453651 + 1.65*0.13009746335307137 + 1.0*0.7581777721646513

= 1.0780914298949453

$y_net_{(3)}$ = 1.13*0.02879340957844745 + 1.65*0.8366555320866069 + 1.0*0.3871213745104467

= 1.8001395552769937

$y_net_{(4)}$ = 1.13*0.5364724047280168 + 1.65*0.662049259269347 + 1.0*0.9536038571958243

= 2.6521989523329057

Hitung *output* dari setiap *neuron* di *hidden layer* seperti pada persamaan (3.2). *Activation function* yang digunakan adalah *bipolar sigmoid* seperti pada persamaan (2.2).

$y_{(1)} = 2/(1 + e^{-2.3484071648070923})-1$

= 0.8256150744735646

$y_{(2)} = 2/(1 + e^{-1.0780914298949453})-1$

= 0.49226526855349895

$y_{(3)} = 2/(1 + e^{-1.8001395552769937})-1$

$= 0.716331844444487$

$y_{(4)} = 2/(1 + e^{-2.6521989523329057})-1$

$= 0.8682927849411683$

Hitung nilai sigma dari *w2*y* seperti pada persamaan (3.3).

$z_net_{(1)} = 0.8256150744735646*0.5790454022610518 +$
$0.49226526855349895*0.5090528449625555 +$
$0.716331844444487*0.5109558325241379 +$
$0.8682927849411683*0.42125968287763105 +$
$1.0*0.1408635321980527$
$= 1.6013118577137373$

Hitung *output* dari setiap *neuron* di *output layer* seperti pada persamaan (3.4). *Activation function* yang digunakan adalah *bipolar sigmoid* seperti pada persamaan (2.2).

$z_{(1)} = 2/(1 + e^{-1.6013118577137373})-1$

$= 0.6644033109815832$

$Q(S, A) = 0.6961609686650201$

Selanjutnya hitung nilai *action-value function* terbesar untuk *next state* $\max_a Q(S', a)$

Feed-forward pada masing-masing *artificial neural network* dengan *input next state* untuk mendapat nilai *action-value function* yang selanjutnya akan dibandingkan untuk mendapat nilai yang terbesar.

<u>*Action* 0 (Diam)</u>

$x_{(1)} = 1.28$

$x_{(2)} = 1.63$

Hitung nilai sigma dari *w1*x* seperti pada persamaan (3.1).

$$
\begin{aligned}
y_net_{(1)} &= 1.28*0.3737467985596139 + 1.63*0.13212418148869665 + \\
&\quad 1.0*0.02705724989590974 \\
&= 0.7208155678787911 \\
y_net_{(2)} &= 1.28*0.3272211118251288 + 1.63*0.3239285971112901 + \\
&\quad 1.0*0.06982251855540678 \\
&= 1.0166691549829743 \\
y_net_{(3)} &= 1.28*0.9529138643629589 + 1.63*0.3393466407427368 + \\
&\quad 1.0*0.4996754871565079 \\
&= 2.272540257951756 \\
y_net_{(4)} &= 1.28*0.5823802930171551 + 1.63*0.15440917597409076 + \\
&\quad 1.0*0.9192497675928927 \\
&= 1.916383499492619
\end{aligned}
$$

Hitung *output* dari setiap *neuron* di *hidden layer* seperti pada persamaan (3.2). *Activation function* yang digunakan adalah *bipolar sigmoid* seperti pada persamaan (2.2).

$$
\begin{aligned}
y_{(1)} &= 2/(1 + e^{-0.7208155678787911})-1 \\
&= 0.3455731707733132 \\
y_{(2)} &= 2/(1 + e^{-1.0166691549829743})-1 \\
&= 0.4686465672878313 \\
y_{(3)} &= 2/(1 + e^{-2.272540257951756})-1 \\
&= 0.8131543145364393 \\
y_{(4)} &= 2/(1 + e^{-1.916383499492619})-1 \\
&= 0.7434692077202323
\end{aligned}
$$

Hitung nilai sigma dari *w2*y* seperti pada persamaan (3.3).

$z_net_{(1)}$ = 0.3455731707733132*0.6981433296189277 +
0.4686465672878313*0.6184307761972161 +
0.8131543145364393*0.992909937330567 +
0.7434692077202323*0.2334423547441814 +
1.0*0.012739708821004059
= 1.5247709752780874

Hitung *output* dari setiap *neuron* di *output layer* seperti pada persamaan (3.4). *Activation function* yang digunakan adalah *bipolar sigmoid* seperti pada persamaan (2.2).

$z_{(1)}$ = $2/(1 + e^{-1.5247709752780874})-1$
= 0.6424799137312227

Action 0 (Diam) pada *next state* $S'_{(1)}$ = 1.28 dan $S'_{(2)}$ = 1.63 menghasilkan nilai *action-value function* 0.6424799137312227

<u>*Action* 1 (Terbang)</u>
$x_{(1)}$ = 1.28
$x_{(2)}$ = 1.63
Hitung nilai sigma dari *w1*x* seperti pada persamaan (3.1).

$y_net_{(1)}$ = 1.28*0.3143692393836913 + 1.63*0.7832510663743999 +
1.0*0.7008056647857615
= 2.379897529387158
$y_net_{(2)}$ = 1.28*0.09314410902453651 + 1.63*0.13009746335307137 +
1.0*0.7581777721646513
= 1.0894610969815643
$y_net_{(3)}$ = 1.28*0.02879340957844745 + 1.63*0.8366555320866069 +
1.0*0.3871213745104467

$$= 1.7877254560720286$$

$$y_net_{(4)} = 1.28*0.5364724047280168 + 1.63*0.662049259269347 + 1.0*0.9536038571958243$$

$$= 2.719428827856721$$

Hitung *output* dari setiap *neuron* di *hidden layer* seperti pada persamaan (3.2). *Activation function* yang digunakan adalah *bipolar sigmoid* seperti pada persamaan (2.2).

$$y_{(1)} = 2/(1 + e^{-2.379897529387158})-1$$

$$= 0.8305629778087753$$

$$y_{(2)} = 2/(1 + e^{-1.0894610969815643})-1$$

$$= 0.49656045814430794$$

$$y_{(3)} = 2/(1 + e^{-1.7877254560720286})-1$$

$$= 0.7132963686691278$$

$$y_{(4)} = 2/(1 + e^{-2.719428827856721})-1$$

$$= 0.8763268134320341$$

Hitung nilai sigma dari *w2*y* seperti pada persamaan (3.3).

$$z_net_{(1)} = 0.8305629778087753*0.5790454022610518 + 0.49656045814430794*0.5090528449625555 + 0.7132963686691278*0.5109558325241379 + 0.8763268134320341*0.42125968287763105 + 1.0*0.1408635321980527$$

$$= 1.6081968151140644$$

Hitung *output* dari setiap *neuron* di *output layer* seperti pada persamaan (3.4). *Activation function* yang digunakan adalah *bipolar sigmoid* seperti pada persamaan (2.2).

$z_{(1)}$ = $2/(1 + e^{-1.6081968151140644})-1$

= 0.6663217747739034

Action 1 (Terbang) pada *state* $S'_{(1)}$ = 1.28 dan $S'_{(2)}$ = 1.63 menghasilkan nilai *action-value* 0.6663217747739034.

Action 1 (Terbang) > *Action* 0 (Diam), maka

$\max_a Q(S', a)$ = 0.6663217747739034

Hitung nilai *update* nya seperti pada persamaan (3.5).

$$Nilai\ update = Q(S, A) + \alpha(R + \gamma max_a Q(S', a) - Q(S, A)) \quad (3.5)$$

α = 0.9

γ = 0.9

Nilai update = 0.6644033109815832 + 0.9 *
(0.1+(0.9*0.6663217747739034)-0.6644033109815832)

= 0.6961609686650201

7) Latih ANN pada *action* yang sudah dipilih dengan *backpropagation*

Feed-forward

$x_{(1)}$ = 1.13

$x_{(2)}$ = 1.65

Hitung nilai sigma dari *w1*x* seperti pada persamaan (3.1).

$y_net_{(1)}$ = 1.13*0.3143692393836913 + 1.65*0.7832510663743999 +
1.0*0.7008056647857615

= 2.3484071648070923

$y_net_{(2)}$ = 1.13*0.09314410902453651 + 1.65*0.13009746335307137 +

$$
\begin{aligned}
& 1.0*0.7581777721646513 \\
= \; & 1.0780914298949453 \\
y_net_{(3)} = \; & 1.13*0.02879340957844745 + 1.65*0.8366555320866069 + \\
& 1.0*0.3871213745104467 \\
= \; & 1.8001395552769937 \\
y_net_{(4)} = \; & 1.13*0.5364724047280168 + 1.65*0.662049259269347 + \\
& 1.0*0.9536038571958243 \\
= \; & 2.6521989523329057
\end{aligned}
$$

Hitung *output* dari setiap *neuron* di *hidden layer* seperti pada persamaan (3.2). *Activation function* yang digunakan adalah *bipolar sigmoid* seperti pada persamaan (2.2).

$$
\begin{aligned}
y_{(1)} &= 2/(1 + e^{-2.3484071648070923})-1 \\
&= 0.8256150744735646 \\
y_{(2)} &= 2/(1 + e^{-1.0780914298949453})-1 \\
&= 0.49226526855349895 \\
y_{(3)} &= 2/(1 + e^{-1.8001395552769937})-1 \\
&= 0.716331844444487 \\
y_{(4)} &= 2/(1 + e^{-2.6521989523329057})-1 \\
&= 0.8682927849411683
\end{aligned}
$$

Hitung nilai sigma dari *w2*y* seperti pada persamaan (3.3).

$$
\begin{aligned}
z_net_{(1)} = \; & 0.8256150744735646*0.5790454022610518 + \\
& 0.49226526855349895*0.5090528449625555 + \\
& 0.716331844444487*0.5109558325241379 + \\
& 0.8682927849411683*0.42125968287763105 + \\
& 1.0*0.1408635321980527 \\
= \; & 1.6013118577137373
\end{aligned}
$$

Hitung *output* dari setiap *neuron* di *output layer* seperti pada persamaan (3.4). *Activation function* yang digunakan adalah *bipolar sigmoid* seperti pada persamaan (2.2).

$z_{(1)}$ = $2/(1 + e^{-1.6013118577137373})-1$

= 0.6644033109815832

Backpropagation

Output yang diharapkan diambil dari nilai *update* dari *action-value function* yang sudah dihitung sebelumnnya.

$t_{(1)}$ = 0.6961609686650201

α = 0.9

Hitung nilai delta untuk perubahan bobot *w2* seperti pada persamaan (3.6).

$$\delta_o_{(k)} = (t_{(k)} - z_{(k)}) f'(z_net_{(k)}) \quad (3.6)$$

Fungsi turunan yang digunakan seperti pada persamaan (2.3).

$\delta_o_{(1)}$ = (0.6961609686650201-0.6644033109815832) *
(0.5*(1+0.6644033109815832)*(1-0.6644033109815832))

= 0.008869409485044033

Hitung nilai perubahan bobot untuk *w2* seperti pada persamaan (3.7).

$$\Delta w2_{(j,k)} = \alpha \ \delta_o_{(k)} \ y_{(j)} \quad (3.7)$$

$\Delta w2_{(1,1)}$ = 0.9*0.008869409485044033*0.8256150744735646

= 0.006590446355278053

$\Delta w2_{(2,1)}$ = 0.9*0.008869409485044033*0.49226526855349895

= 0.0039294920178595365

$\Delta w2_{(3,1)}$ = 0.9*0.008869409485044033*0.716331844444487

= 0.0057180964099995185

$\Delta w2_{(4,1)}$ = 0.9*0.008869409485044033*0.8682927849411683

= 0.006931119836297248

$\Delta w2_{(5,1)}$ = 0.9*0.008869409485044033*1.0

= 0.00798246853653963

Hitung nilai sigma dari *w2**δ seperti pada persamaan (3.8).

$$\delta_h_net_{(j)} = \sum_{k=1}^{c} w2_{(j,k)}\delta_o_{(k)} \tag{3.8}$$

$\delta_h_net_{(1)}$ = 0.008869409485044033*0.5790454022610518

= 0.0051357907830853105

$\delta_h_net_{(2)}$ = 0.008869409485044033*0.5090528449625555

= 0.00451499813149954

$\delta_h_net_{(3)}$ = 0.008869409485044033*0.5109558325241379

= 0.004531876507428159

$\delta_h_net_{(4)}$ = 0.008869409485044033*0.42125968287763105

= 0.003736324626981502

Hitung nilai delta untuk perubahan bobot *w1* seperti pada persamaan (3.9).

$$\delta_h_{(j)} = \delta_h_net_{(j)} f'(y_net_{(j)}) \tag{3.9}$$

Fungsi turunan yang digunakan seperti pada persamaan (2.3).

$\delta_h_{(1)}$ = 0.0051357907830853105 *

(0.5*(1+0.8256150744735646)*(1-0.8256150744735646))

= 8.175145318013598E-4

$\delta_h_{(2)}$ = 0.00451499813149954 *

(0.5*(1+0.49226526855349895)*(1-0.49226526855349895))

= 0.001710450391028256

$\delta_h_{(3)}$ = 0.004531876507428159 *
(0.5*(1+0.716331844444487)*(1-0.716331844444487))
= 0.0011032143861131102

$\delta_h_{(4)}$ = 0.003736324626981502 *
(0.5*(1+0.8682927849411683)*(1-0.8682927849411683))
= 4.596942909060448E-4

Hitung nilai perubahan bobot untuk *w2* seperti pada persamaan (3.10).

$$\Delta w1_{(i,j)} = \alpha\ \delta_h_{(j)}\ x_{(i)} \tag{3.10}$$

$\Delta w1_{(1,1)}$ = 0.9*8.175145318013598E-4*1.13
= 8.314122788419828E-4

$\Delta w1_{(1,2)}$ = 0.9*0.001710450391028256*1.13
= 0.0017395280476757361

$\Delta w1_{(1,3)}$ = 0.9*0.0011032143861131102*1.13
= 0.001121969030677033

$\Delta w1_{(1,4)}$ = 0.9*4.596942909060448E-4*1.13
= 4.675090938514475E-4

$\Delta w1_{(2,1)}$ = 0.9*8.175145318013598E-4*1.65
= 0.0012140090797250194

$\Delta w1_{(2,2)}$ = 0.9*0.001710450391028256*1.65
= 0.00254001883067696

$\Delta w1_{(2,3)}$ = 0.9*0.0011032143861131102*1.65
= 0.0016382733633779685

$\Delta w1_{(2,4)}$ = 0.9*4.596942909060448E-4*1.65
= 6.826460219954765E-4

$\Delta w1_{(3,1)}$ = 0.9*8.175145318013598E-4*1.0
= 7.357630786212239E-4

$\Delta w1_{(3,2)}$ = 0.9*0.001710450391028256*1.0

= 0.0015394053519254304

$\Delta w1_{(3,3)}$ = 0.9*0.0011032143861131102*1.0

= 9.928929475017992E-4

$\Delta w1_{(3,4)}$ = 0.9*4.596942909060448E-4*1.0

= 4.137248618154403E-4

Update bobot *w2* seperti pada persamaan (3.11).

$$w2_{(j,k)} = w2_{(j,k)} + \Delta w2_{(j,k)} \tag{3.11}$$

$w2_{(1,1)}$ = 0.5856358486163299+0.006590446355278053

= 0.5922262949716070

$w2_{(2,1)}$ = 0.5129823369804151+0.0039294920178595365

= 0.5169118289982740

$w2_{(3,1)}$ = 0.5166739289341374+0.0057180964099995185

= 0.5223920253441370

$w2_{(4,1)}$ = 0.4281908027139283+0.006931119836297248

= 0.4351219225502250

$w2_{(5,1)}$ = 0.14884600073459234+0.00798246853653963

= 0.1568284692711320

Update bobot *w1* seperti pada persamaan (3.12).

$$w1_{(i,j)} = w1_{(i,j)} + \Delta w1_{(i,j)} \tag{3.12}$$

$w1_{(1,1)}$ = 0.3152006516625333+8.314122788419828E-4

= 0.3160320639413750

$w1_{(1,2)}$ = 0.09488363707221226+0.0017395280476757361

= 0.0966231651198879

$w1_{(1,3)}$ = 0.029915378609124483+0.001121969030677033

= 0.0310373476398014

$w1_{(1,4)}$ = 0.5369399138218682+4.675090938514475E-4

$= 0.5374074229157190$

$w1_{(2,1)} = 0.7844650754541249+0.0012140090797250194$

$= 0.7856790845338490$

$w1_{(2,2)} = 0.13263748218374832+0.00254001883067696$

$= 0.1351775010144250$

$w1_{(2,3)} = 0.8382938054499849+0.0016382733633779685$

$= 0.8399320788133620$

$w1_{(2,4)} = 0.6627319052913425+6.826460219954765E-4$

$= 0.6634145513133370$

$w1_{(3,1)} = 0.7015414278643827+7.357630786212239E-4$

$= 0.7022771909430030$

$w1_{(3,2)} = 0.7597171775165767+0.0015394053519254304$

$= 0.7612565828685010$

$w1_{(3,3)} = 0.3881142674579485+9.928929475017992E-4$

$= 0.3891071604054500$

$w1_{(3,4)} = 0.9540175820576398+4.137248618154403E-4$

$= 0.9544313069194540$

8) *Update Next State* menjadi *Current State*

$S_{(1)} = 1.28$

$S_{(2)} = 1.63$

9) Cek kondisi penyimpanan bobot

Jika *agent* mendapat *score* 14 maka bobot dari *artificial neural network* tersebut akan disimpan, jika tidak maka selanjutnya akan melakukan pengecekan kondisi henti.

10) Cek kondisi henti

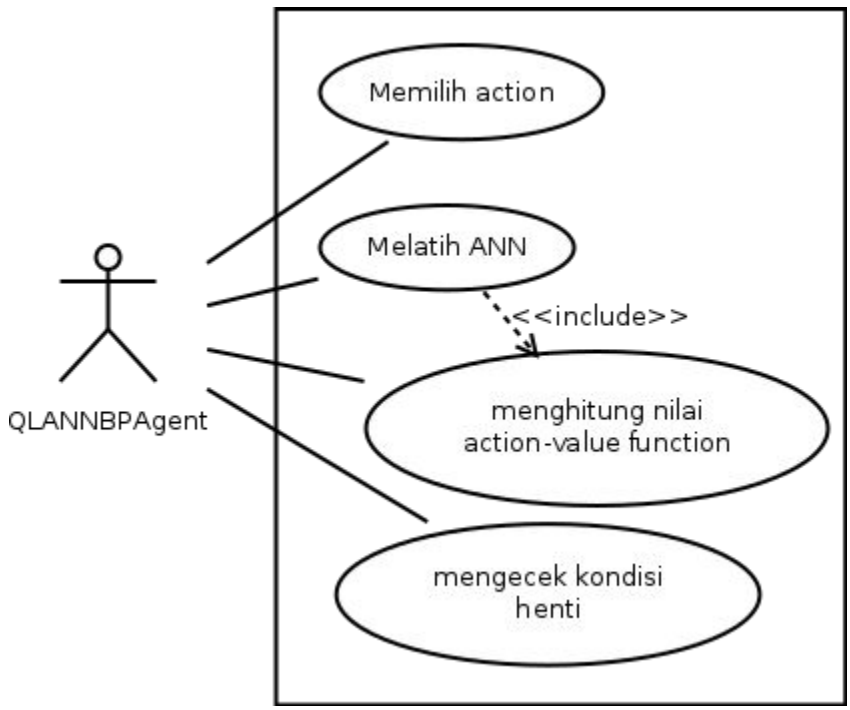
Jika *agent* mendapat *score* 30 maka poses pembelajaran akan dihentikan, jika tidak maka akan kembali ke langkah 3.

3.5. Analisis Kebutuhan Fungsional

Kebutuhan fungsional meliputi beberapa diagram UML, antara lain *use case diagram*, *activity diagram*, *class diagram* dan *sequence diagram*. Diagram tersebut dipergunakan agar rencana desain dan struktur lebih mudah dipahami.

3.5.1. *Use Case Diagram*

Use-case diagram menggambarkan unit fungsionalitas yang disediakan oleh sistem. Adapun *use case diagram* pada penelitian ini dapat dilihat pada Gambar 3.10 *Use Case Diagram*.



Gambar 3.10 *Use Case Diagram*

3.5.1.1. Definisi Aktor

Definisi Actor berfungsi untuk menjelaskan Actor yang terdapat pada Use Case Diagram. Adapun definisi aktor dari *use case diagram* yang digunakan

dapat dilihat pada Tabel 3.1 Definisi Aktor.

Tabel 3.1 Definisi Aktor

No	Aktor	Definisi
1	QLANNBPAgent	Aktor ini adalah *Agent* yang belajar memainkan permainan *Flappy Bird* menggunakan *Q-learning* yang dikombinasikan dengan *backpropgation*.

3.5.1.2. Definisi *Use Case*

Definisi *use case* berfungsi untuk mejelaskan fungsi *use case* yang terdapat pada *use case diagram*. Adapun definisi *use case* dari *use case diagram* yang digunakan dapat dilihat pada Tabel 3.2 Definisi *Use Case*.

Tabel 3.2 Definisi *Use Case*

No	Use Case	Definisi
1	Memilih action	*Agent* memilih *action* dari *state* yang diterima.
2	Melatih ANN	*Agent* melatih ANN dengan pola *state* dan nilai *update* dari *action-value function*.
3	Menghitung nilai action-value function	*Agent* menghitung nilai *action-value function* dari setiap *state* yang diterima.
4	Mengecek kondisi henti	*Agent* melakukan pengecekan apakah proses pembelajaran sudah selesai atau belum.

3.5.1.3. Skenario *Use Case*

Skenario setiap bagian pada *use case* menunjukkan proses apa yang terjadi pada setiap bagian didalam *use case* tersebut. Tabel 3.3 menunjukkan skenario dari *use case* memilih *action*.

Tabel 3.3 Skenario Memilih *Action*

Use Case Name	Memilih *action*	
Related Requirement	*None*	
Goal	*Agent* memilih *action* dari *state* yang didapat.	
Precondition	*Agent* membangkitkan bilangan *random*.	
Success End Condition	*Agent* me*return*kan *action* yang dipilih.	
Failed End Condition	*None*	
Primary Actor	QLANNBPAgent	
Main Flow	*Step*	*Action*
	1	*Agent* membangkitkan bilangan *random*.
	2	Agent akan memilih action bedasarkan bilangan random yang sudah dibangkitkan.
	3	*Agent* me*return*kan *action* yang dipilih.
Extension	*Step*	*Braching Action*
	2.1	Jika nilai *random* lebih besar dari nilai epsilon, maka agent akan memilih *action* dengan nilai *action-value* terbesar, jika tidak maka akan memilih *random action*.

Tabel 3.4 menunjukkan skenario dari *use case* melatih ANN.

Tabel 3.4 Skenario Melatih ANN

Use Case Name	Melatih ANN	
Related Requirement	Menghitung nilai *action-value function*	
Goal	*Agent* melatih ANN dengan pola *state* dan nilai *update* dari *action-value function*.	
Precondition	*Agent* menghitung nilai *action-value function*.	
Success End Condition	*Agent* selesai melatih ANN.	
Failed End Condition	*None*	
Primary Actor	QLANNBPAgent	
Main Flow	*Step*	*Action*
	1	Set output semua *neuron* di *hidden layer*.
	2	Set output semua *neuron* di *output layer*.
	3	Lakukan *backpropagation* dengan nilai *update action-value function* sebagai *output* yang diharapkan.
	4	*Update* semua bobot.
Extension	*Step*	*Braching Action*
	-	-

Tabel 3.5 menunjukkan skenario dari *use case* menghitung nilai *action-value function*.

Tabel 3.5 Skenarion Menghitung Nilai *Action-Value Function*

Use Case Name	Menghitung nilai *action-value function*	
Related Requirement	*None*	
Goal	*Agent* menghitung nilai *action-value function* dari setiap *state* yang diterima.	
Precondition	*Agent* meng*convert state* menjadi *array* untuk input ke ANN.	
Success End Condition	*Agent* mendapat nilai *action-value function* dari setiap *state* yang diterima.	
Failed End Condition	*None*	
Primary Actor	*None*	
Main Flow	*Step*	*Action*
	1	*Agent* meng*convert state* menjadi *array* untuk input ke ANN.
	2	*Feed-forward* untuk mendapat nilai *action-value function*.
	3	*Agent* mengambil nilai di *neuron* pada *output layer* sebagai nilai *action-value function*.
Extension	*Step*	*Braching Action*
	-	-

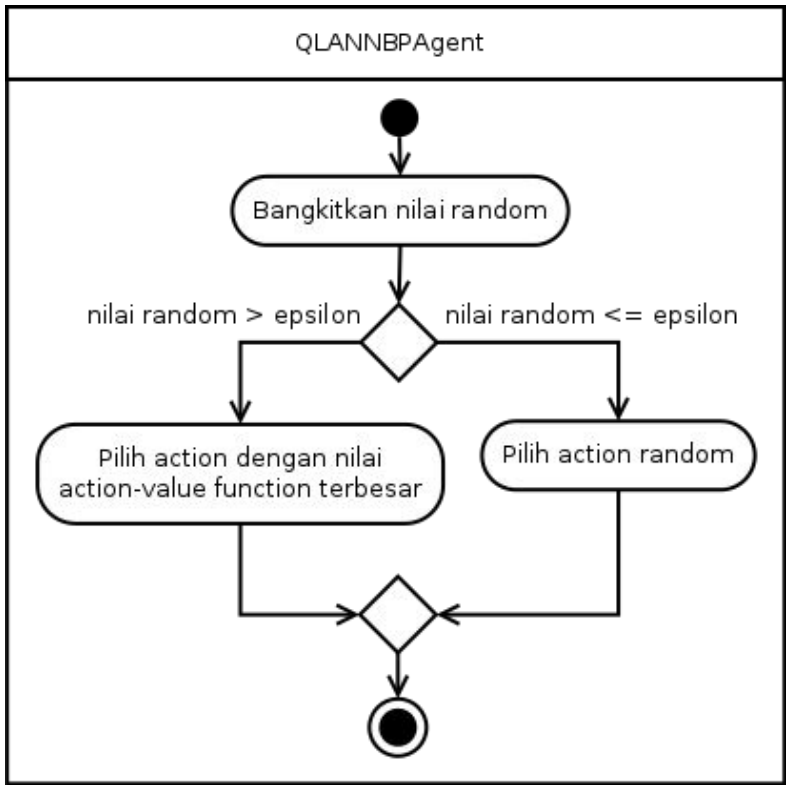
Tabel 3.6 menunjukkan skenario dari *use case* mengecek kondisi henti.

Tabel 3.6 Skenario Mengecek Kondisi Henti

Use Case Name	Mengecek kondisi henti	
Related Requirement	-	
Goal	*Agent* melakukan pengecekan apakah proses pembelajaran sudah selesai atau belum.	
Precondition	None	
Success End Condition	*Agent* mengetahui apakah proses pembelajaran sudah selesai atau belum.	
Failed End Condition	*None*	
Primary Actor	*None*	
Main Flow	*Step*	*Action*
	1	*Agent* mengecek *score* untuk penyimpanan bobot.
	2	*Agent* mengecek *score* untuk akhir pembelajaran.
Extension	*Step*	*Braching Action*
	1.1	Jika *agent* mendapat *score* 14 maka bobot akan disimpan.
	2.1	Jika *agent* mendapat *score* 30 maka proses pembelajaran akan dihentikan dan akan menggunakan bobot yang sudh disimpan.

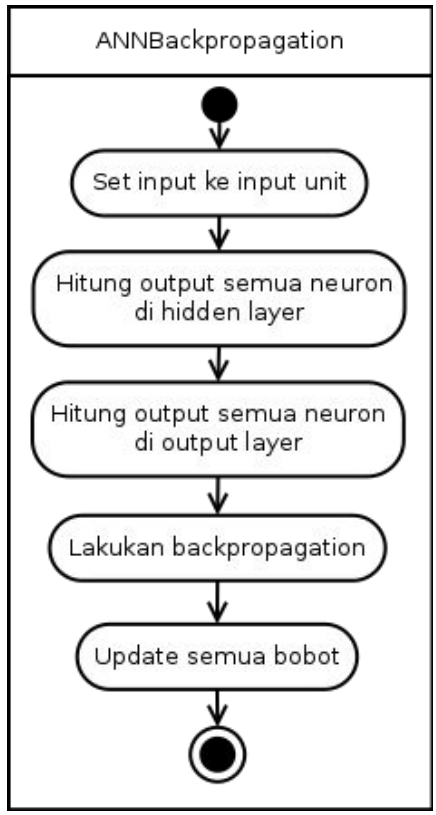
3.5.2. *Activity Diagram*

Activity diagram menunjukkan aliran kontrol antara dua atau lebih abjek saat memproses suatu kegiatan. Gambar 3.11 menunjukkan *activity diagram* memilih *action*.



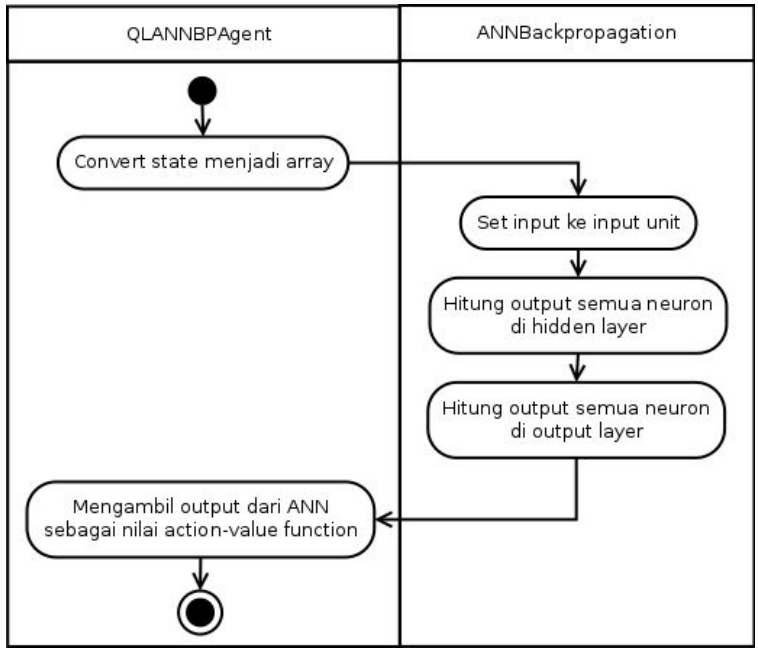
Gambar 3.11 *Activity Diagram* Memilih *Action*

Gambar 3.12 menunjukkan *activity diagram* melatih ANN.



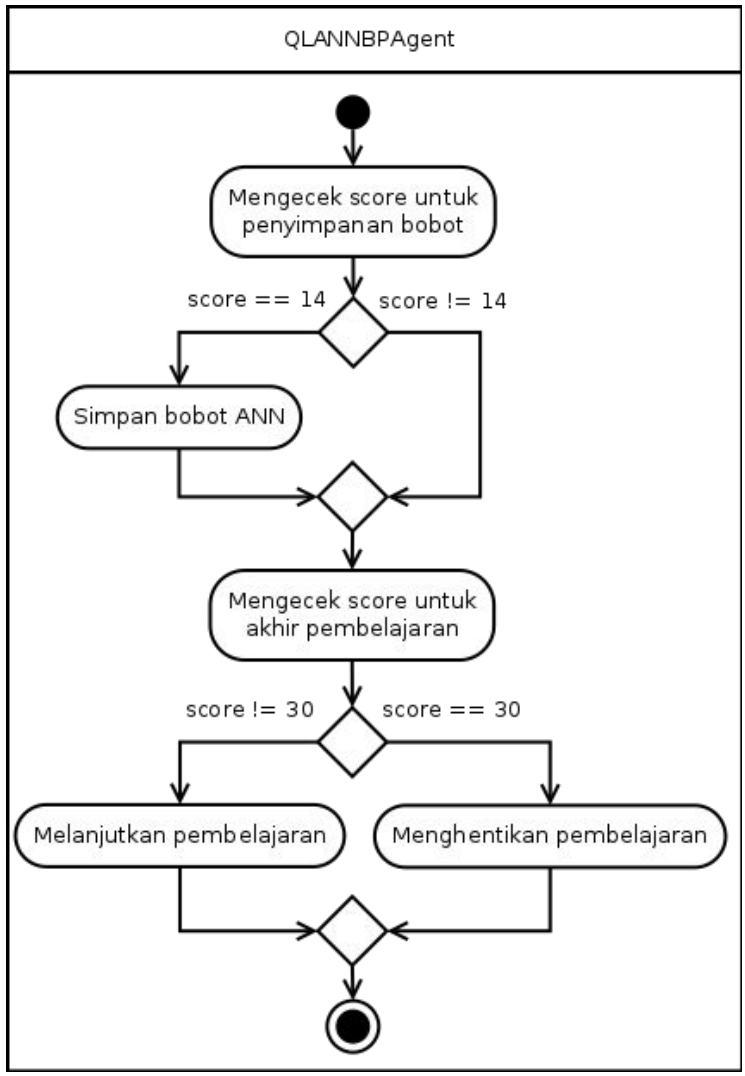
Gambar 3.12 *Activity Diagram* Melatih ANN

Gambar 3.13 menunjukkan *activity diagram* menghitung nilai *action-value function*.



Gambar 3.13 ***Activity Diagram*** **Menghitung Nilai** ***Action-Value Function***

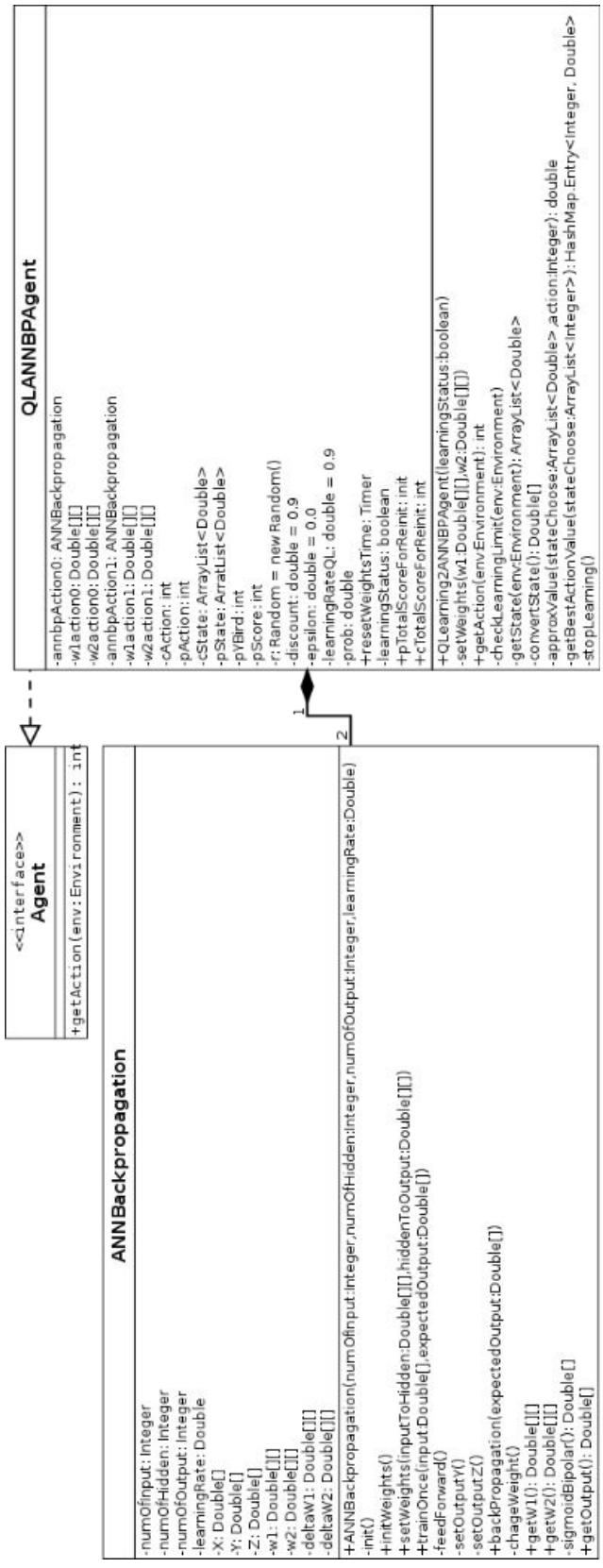
Gambar 3.14 menunjukkan *activity diagram* mengecek kondisi henti.



Gambar 3.14 *Activity Diagram* Mengecek Kondisi Henti

3.5.3. *Class Diagram*

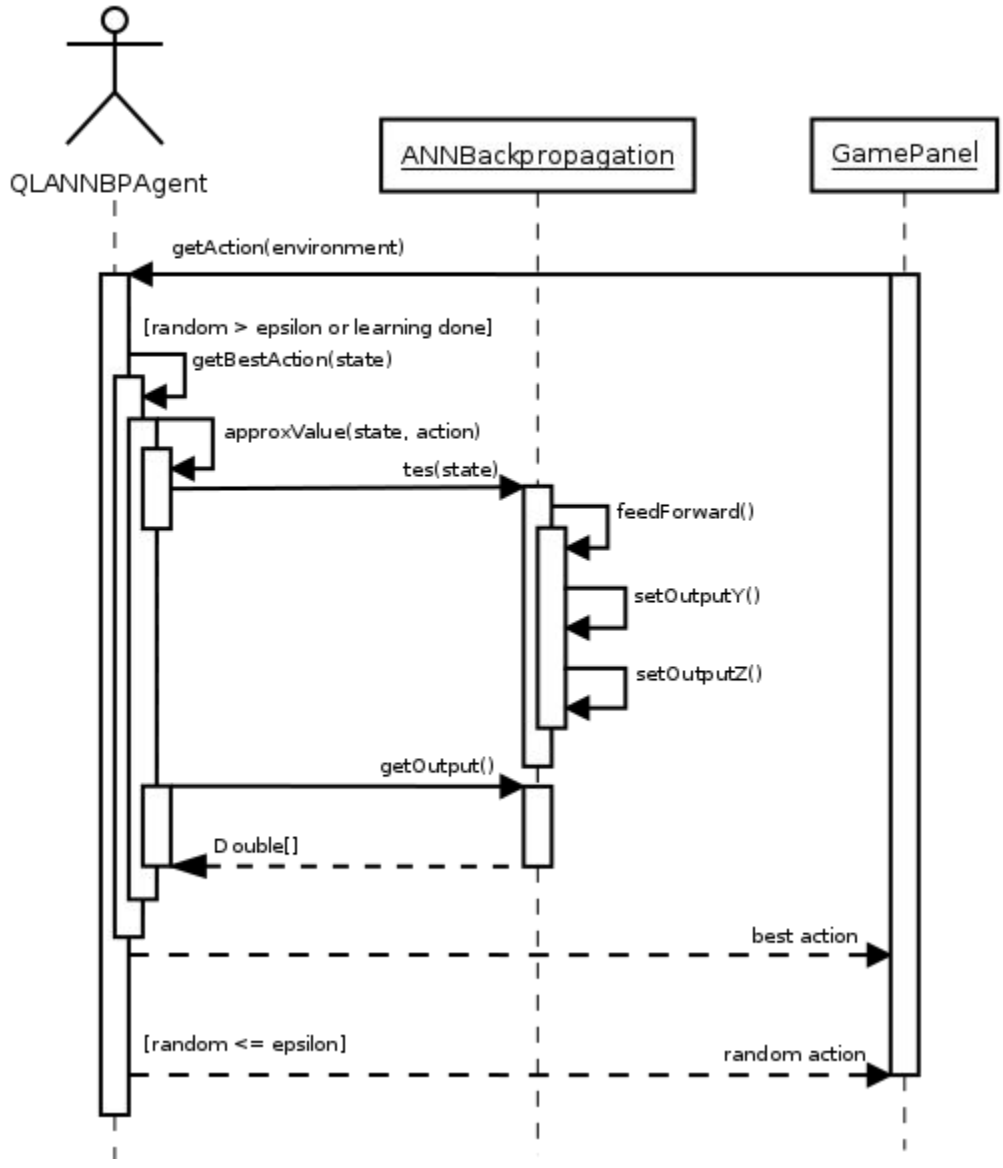
Class diagram menunjukkan bagaimana entitas yang berbeda berhubungan satu sama lain. Adapun *class diagram* pada penelitian ini dapat dilihat pada Gambar 3.15 *Class Diagram*.



Gambar 3.15 *Class Diagram*

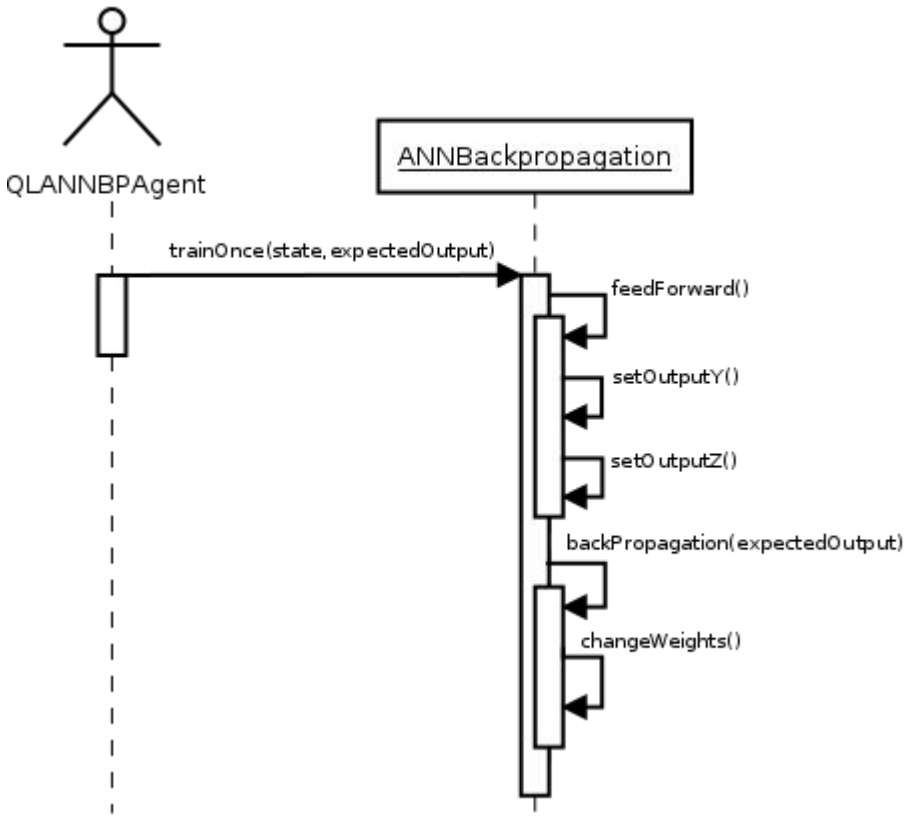
3.5.4. *Sequence Diagram*

Sequence diagram menunjukkan aliran rinci untuk *use case* tertentu atau bahkan hanya bagian dari *use case* tertentu. Gambar 3.16 menunjukkan *sequence diagram* memilih *action*.



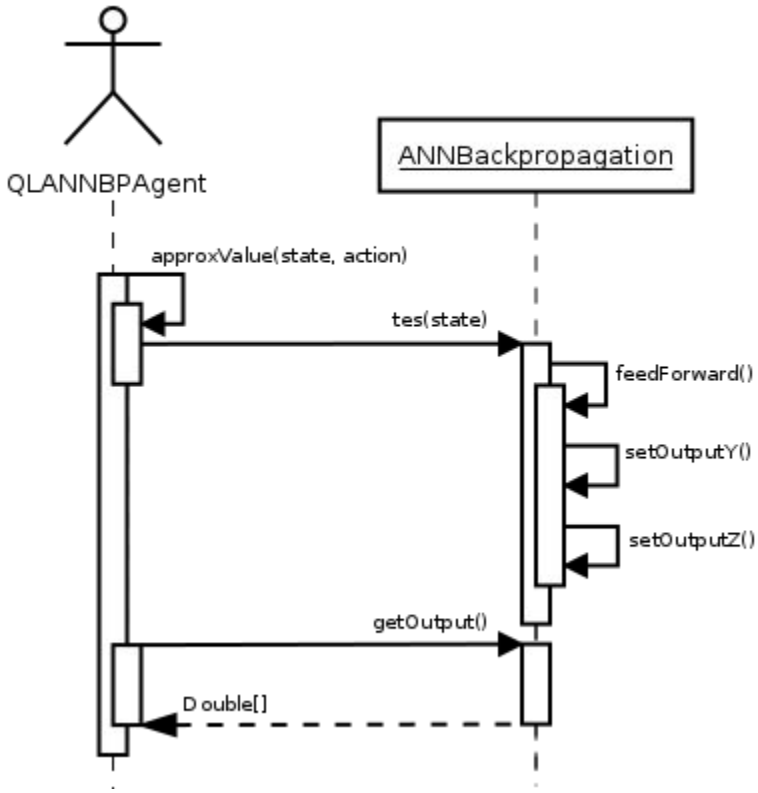
Gambar 3.16 *Sequence Diagram* Memilih *Action*

Gambar 3.17 menunjukkan *sequence diagram* melatih ANN.



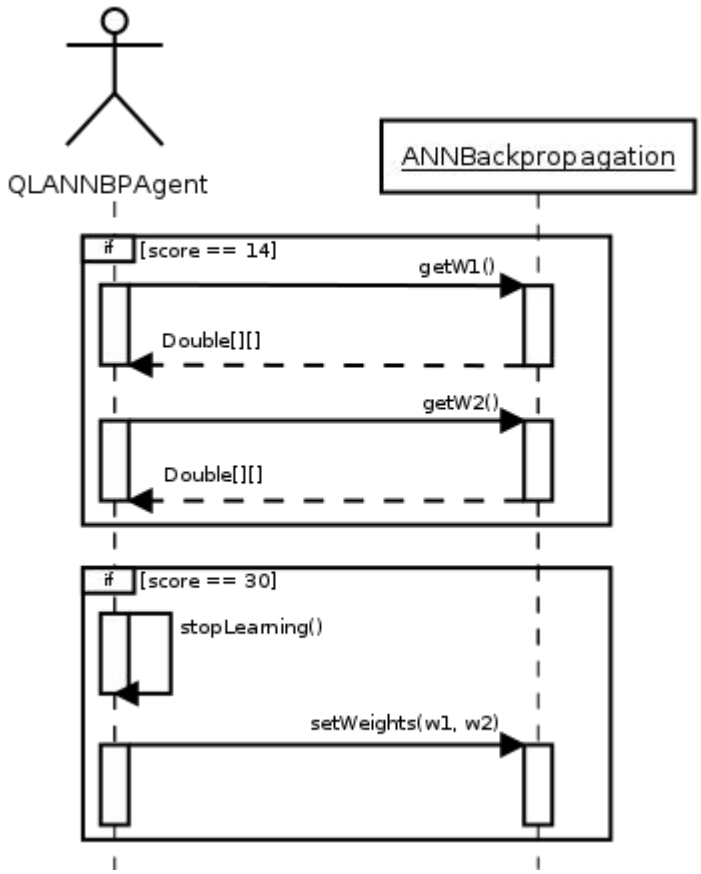
Gambar 3.17 *Sequence Diagram* Melatih ANN

Gambar 3.18 menunjukkan *sequence diagram* menghitung nilai *action-value function*.



Gambar 3.18 ***Sequence Diagram*** **Menghitung Nilai** ***Action-Value Function***

Gambar 3.19 menunjukkan *sequence diagram* mengecek kondisi henti.



Gambar 3.19 *Sequence Diagram* Mengecek Kondisi Henti

3.6. Analisis Kebutuhan Non-fungsional

Kebutuhan non-fungsional meliputi kebutuhan *hardware* dan kebutuhan *software* minimal agar implementasi algoritma pada penelitian ini dapat berjalan dengan baik.

3.6.1. Analisis Kebutuhan *Hardware*

Spesifikasi *hardware* yang dibutuhkan pada penelitian ini dapat dilihat pada Tabel 3.7 Spesifikasi Kebutuhan *Hardware*.

Tabel 3.7 Spesifikasi Kebutuhan *Hardware*

Hardware	Spesifikasi
Processor	*Dual Core* atau lebih tinggi
RAM	2GB atau lebih tinggi
Harddisk	10GB *Free Space* atau lebih
Monitor	Resolusi 1024x768 atau lebih tinggi
Keyboard & Mouse	*Standard*

3.6.2. Analisis Kebutuhan *Software*

Spesifikasi *software* yang dibutuhkan pada penelitian ini dapat dilihat pada Tabel 3.8 Spesifikasi Kebutuhan *Software*.

Tabel 3.8 Spesifikasi Kebutuhan *Software*

Software	Spesifikasi
Operating System	Dapat menjalankan *Java* 8
Development Kit	JDK 1.8.0