4. IMPLEMENTASI SISTEM

Pada bab ini, akan dibahas implementasi sistem sesuai analisa dan desain sistem pada bab sebelumnya.

Tabel 4.1 Daftar Segmen Program dan Flowchart

SEGMEN PROGRAM	GAMBAR (FLOWCHART)
4.1	3.3
4.2, 4.3	3.5, 3.6
4.4, 4.5	3.7
4.6	3.8
4.8, 4.9, 4.10	3.10
4.11	3.11

4.1 Instalasi *Open Source Library*

Dalam implementasi sistem ini, digunakan dua *open source library*, yaitu *NEAT-Python* dan *Neataptic.js.*

4.1.1 Instalasi NEAT-Python

NEAT-Python merupakan sebuah *open source library* dari bahasa pemrograman *python*. *Library* ini nantinya akan digunakan untuk menjalankan proses NEAT.

Untuk meng-*install NEAT-Python*, dibutuhkan *pip*, yang merupakan *package installer* untuk bahasa pemrograman *python*. Jika *pip* sudah ter-*install*, cukup ketikkan **`pip install neat-python`** pada *terminal* jika menggunakan *linux* atau *macOS*, atau *command prompt* jika menggunakan *windows.*

4.1.2 Instalasi *Neataptic.js*

Untuk melakukan *backpropagation*, dibutuhkan *open source library* lainnya. Pada penelitian ini, library yang digunakan untuk melakukan

backpropagation adalah *Neataptic.js*, yang merupakan *library* dari bahasa pemrograman *javascript* yang pada penelitian ini akan berjalan diatas *nodejs*.

Sama seperti *NEAT-Python*, *Neataptic.js* juga membutuhkan *npm*, yaitu sebuah *package installer* pada bahasa pemrograman *javascript*, untuk melakukan instalasi. Jika *npm* sudah terinstall, cukup ketikkan **`npm install neataptic`** pada *terminal* jika menggunakan *linux* atau *macOS*, atau *command prompt* jika menggunakan windows

4.2 *Data Pre-processing*

Sebelum data dapat digunakan, harus dilakukan *pre-processing* terlebih dahulu untuk mengekstrak *features* yang akan digunakan. Seperti yang sudah dijelaskan pada Bab 3, pada awalnya format data adalah *json*. Proses ini menggunakan bahasa pemrograman *php*. *Output* dari proses *pre-processing* ini berbentuk *.txt* yang nanti akan digunakan pada proses NEAT.

Segmen Program 4.1 Pre-processing pada data dengan feature berupa player ratings dan team ratings

```
$data = json_decode(file_get_contents("dataset/datafile/season14-
15/season_stats.json"),true);
$fInput = array();
$fOutput = array();
$readCount = 0;

foreach($data as $key => $values){
    $input = array();
    $output = array();
    $readCount++;
    foreach($values as $key2 => $values2){
        foreach($values2 as $key3 => $values3){
            if($key3 == 'team_details'){
                foreach($values3 as $key4 => $values4){
                    if($key4 == 'team_rating'){
                        array_push($input,$values4/10);
                    }
                }
            }
            if($key3 == 'aggregate_stats'){
                $noGoal = false;
                foreach($values3 as $key4 => $values4){
                    if($key4 == 'goals'){
```

```
                        array_push($output,$values4);
                        $noGoal = true;
                    }

                }
                if($noGoal == false){
                    array_push($output,0);
                }
            }
            if($key3 == 'Player_stats'){
                foreach($values3 as $key4 => $values4){
                    foreach($values4 as $key5 => $values5){
                        if($key5 == 'player_details'){
                            foreach($values5 as $key6 => $values6){
                                if($key6 == 'player_rating' ){
                                    if($values6 == 'Sub'){
                                        break;
                                    }else if($key6 == 'player_rating'){
                                        array_push($input,$values6/10);
                                    }
                                }
                            }
                        }
                    }
                }
            }

        }
    }
    array_push($fInput,$input);
    array_push($fOutput,$output);
}

$input_str = '[';
$inputCount = 0;
$outputCount = 0;

foreach($fInput as $key => $values){
    $inputCount++;
    $input_str =  $input_str.'(';
    foreach($values as $key2 => $values2){
        $input_str = $input_str.$values2.', ';
    }
    $input_str =  $input_str.'),';

}

$input_str =  $input_str.']';
```

```
$output_str = '[';

foreach($fOutput as $key => $values){
    $outputCount++;
    $output_str =  $output_str.'(';
    foreach($values as $key2 => $values2){
        $output_str = $output_str.$values2.', ';
    }
    $output_str =  $output_str.'),';

}

$output_str =  $output_str.']';
$myfile = fopen("inputTestWithTeam.txt", "w") or die("Unable to open file!");
fwrite($myfile, $input_str);
$myfile = fopen("OutputTest.txt", "w") or die("Unable to open file!");
fwrite($myfile, $output_str);
```

Setelah proses *pre-processing* selesai, barulah data siap digunakan untuk proses selanjutnya, yaitu NEAT.

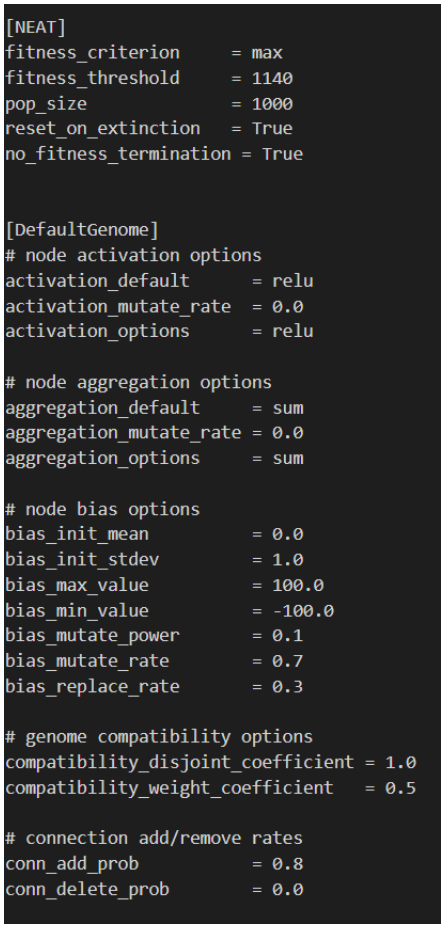
4.3 Implementasi NEAT

Proses NEAT dimulai dengan melakukan pengaturan *config*, yang merupakan aturan-aturan yang berlaku selama proses NEAT berlangsung, seperti kemungkinan terjadinya mutasi, jumlah populasi, dan lain-lain. Contoh *config* pada NEAT dapat dilihat pada Gambar 4.1.

Setelah melakukan pengaturan *config*, akan dijalankan fungsi `load_data()` untuk menginputkan data yang telah diproses melalui proses *pre-processing*.

Akan ada 4 data yang akan digunakan, yaitu data untuk *input* dan *output* untuk proses *training* dan data *input* dan *output* untuk proses *testing*.

Jika data sudah diterima dan *config* sudah ditetapkan, proses neat akan berjalan dengan menjalankan fungsi `start()`, yang menerima *config* dan mengaplikasikannya kepada sistem.



```
[NEAT]
fitness_criterion     = max
fitness_threshold     = 1140
pop_size              = 1000
reset_on_extinction   = True
no_fitness_termination = True


[DefaultGenome]
# node activation options
activation_default      = relu
activation_mutate_rate  = 0.0
activation_options      = relu

# node aggregation options
aggregation_default     = sum
aggregation_mutate_rate = 0.0
aggregation_options     = sum

# node bias options
bias_init_mean          = 0.0
bias_init_stdev         = 1.0
bias_max_value          = 100.0
bias_min_value          = -100.0
bias_mutate_power       = 0.1
bias_mutate_rate        = 0.7
bias_replace_rate       = 0.3

# genome compatibility options
compatibility_disjoint_coefficient = 1.0
compatibility_weight_coefficient   = 0.5

# connection add/remove rates
conn_add_prob           = 0.8
conn_delete_prob        = 0.0
```

Gambar 4.1 Contoh *file config* pada NEAT

Segmen Program 4.2 Fungsi `load_data()`

```
def load_data():
    train_inputs = f=open("inputTestWithTeam.txt","r")
    if(f.mode == 'r'):
        xor_inputs = eval(f.read())
    train_outputs = f=open("outputTest.txt","r")
    if(f.mode == 'r'):
        xor_outputs = eval(f.read())
    valid_inputs = f=open("validInputWithTeam.txt","r")
    if(f.mode == 'r'):
        valid_inputs = eval(f.read())
    valid_outputs = f=open("validOutputTest.txt","r")
    if(f.mode == 'r'):
        valid_outputs = eval(f.read())
```

Segmen Program 4.3 Fungsi `start()` untuk menjalankan proses NEAT

```
def start(config_file):
    config = neat.Config(neat.DefaultGenome, neat.DefaultReproduction,
                         neat.DefaultSpeciesSet, neat.DefaultStagnation,
                         config_file)
    p = neat.Population(config)
    p.add_reporter(neat.StdOutReporter(True))
    stats = neat.StatisticsReporter()
    p.add_reporter(stats)
    p.add_reporter(neat.Checkpointer(50))
    pe = neat.ParallelEvaluator(4,fitness_function2)
    winner = p.run(pe.evaluate,5000)
    print('\nBest genome:\n{!s}'.format(winner))
    print('\nOutput:')
    calculate_winner(winner,config)
    node_names = {-1:'A', -2: 'B', 0:'A XOR B'}
    visualize.extract_net(config,winner)
    visualize.draw_net(config, winner, view=False, node_names=node_names)
    visualize.plot_stats(stats, ylog=False, view=False)
    visualize.plot_species(stats, view=False)
```

Fungsi `neat.ParallelEvaluator()` menerima sebuah *fitness function* yang akan digunakan untuk mengevaluasi populasi pada NEAT. Akan ada 2 *fitness function* yang akan digunakan pada penelitian ini.

Segmen Program 4.4 *Fitness function* pertama

```
def eval_genome1(genome, config):
    fitness = 1140
    net = neat.nn.FeedForwardNetwork.create(genome, config)
    for xi, xo in zip(xor_inputs, xor_outputs):
        output = net.activate(xi)
        output[0] = round((output[0]*10))
        output[1] = round((output[1]*10))
        if(xo[0]*10 == output[0] and xo[1]*10 == output[1] ):
            genome.fitness += 1
        else :
            if(xo[0]*10 > xo[1]*10 and output[0] > output[1] ):
                genome.fitness += 0.5
            if(xo[0]*10 < xo[1]*10 and output[0] < output[1] ):
                genome.fitness += 0.5
            if(xo[0]*10 == xo[1]*10 and output[0] == output[1] ):
                genome.fitness += 0.5
    return fitness
```

Segmen Program 4.5 *Fitness function* kedua

```
def eval_genome2(genome, config):
    fitness = 1140
    net = neat.nn.FeedForwardNetwork.create(genome, config)
    for xi, xo in zip(xor_inputs, xor_outputs):
        output = net.activate(xi)
        fitness -= (output[0] - xo[0]) ** 2
        fitness -= (output[1] - xo[1]) ** 2
    return fitness
```

Pada *fitness function* pertama, masing masing *genome* akan diberikan nilai fitness sebesar 1 jika skor yang dihasilkan benar, sedangkan jika skor salah tetapi pemenang yang diprediksi benar, nilai fitness yang diberikan adalah 0.5. Sedangkan pada *fitness function* kedua, nilai *fitness* dihitung berdasarkan *sum of squared errors*.

Fungsi `neat.parallelEvaluator()` kemudian akan di*passing*kan kepada fungsi `run()` untuk menjalankan proses NEAT. Fungsi `run()` juga menerima parameter berupa jumlah iterasi yang ingin dijalankan.

Setelah fungsi `run()` selesai dijalankan, yang menandakan proses NEAT telah selesai, fungsi ini akan meng*outputkan* genome terbaik dengan nilai *fitness* terbaik. Dari genome terbaik ini kemudian akan di*construct* sebuah *neural network* yang akan di*passing*kan ke fungsi `calculate_winner()`

Fungsi `calculate_winner()` bertujuan untuk melakukan proses *testing* terhadap *genome* terbaik yang dihasilkan oleh NEAT.

Segmen Program 4.6 Fungsi `calculate_winner()`

```
def calculate_winner(winner,config):
    winner_net = neat.nn.FeedForwardNetwork.create(winner,config)
    correct_predict = 0
    correct_winner = 0
    for xi, xo in zip(valid_inputs, valid_outputs):
        output = winner_net.activate(xi)
        output[0] = round((output[0]))
        output[1] = round((output[1]))
        if(xo[0] == output[0] and xo[1] == output[1] ):
            correct_predict += 1
        if(xo[0] > xo[1] and output[0] > output[1] ):
```

```
            correct_winner += 1
        if(xo[0] < xo[1] and output[0] < output[1] ):
            correct_winner += 1
        if(xo[0] == xo[1] and output[0] == output[1] ):
            correct_winner += 1
        print("input {!r}, expected output {!r}, got {!r}".format(xi, xo, output
))
    print("Prediction accuracy = {!r} ".format(correct_predict))
    print("Winner accuracy = {!r} ".format(correct_winner))
```

Setelah semua proses pada NEAT selesai, genome terbaik akan diekstrak untuk dioptimasi menggunakan backpropagation. Proses ekstraksi ini terjadi pada fungsi `visualise.extract_net()`

Segmen Program 4.7 Fungsi `extract_net()`

```
def extract_net(genome,config):
    node_dict = dict()
    node_obj = []

    for k in config.genome_config.input_keys:
        node_dict = collections.defaultdict(dict)
        node_dict['key'] = k
        node_dict['bias'] = 0
        node_dict['activation'] = ''
        node_dict['type'] = 'input'
        node_obj.append(node_dict)
    for i in genome.nodes:
        node_dict = collections.defaultdict(dict)
        if genome.nodes[i].key > 1 :
            node_dict['key'] = genome.nodes[i].key
            node_dict['bias'] = genome.nodes[i].bias
            node_dict['activation'] = genome.nodes[i].activation
            node_dict['type'] = 'hidden'
            node_obj.append(node_dict)
    for i in genome.nodes:
        node_dict = collections.defaultdict(dict)
        if genome.nodes[i].key == 1 or genome.nodes[i].key == 0:
            node_dict['key'] = genome.nodes[i].key
            node_dict['bias'] = genome.nodes[i].bias
            node_dict['activation'] = genome.nodes[i].activation
            node_dict['type'] = 'output'
            node_obj.append(node_dict)

    with open('node_data.json', 'w') as outfile:
        json.dump(node_obj, outfile)
    conn_dict = dict()
    conn_obj = []
```

```
    for i in genome.connections:
        if genome.connections[i].enabled == True:
            conn_dict = collections.defaultdict(dict)
            conn_dict['from'] = genome.connections[i].key[0]
            conn_dict['to'] = genome.connections[i].key[1]
            conn_dict['weight'] = genome.connections[i].weight
            conn_obj.append(conn_dict)
    with open('conn_data.json', 'w') as outfile:
        json.dump(conn_obj, outfile)
```

Fungsi `extract_net()` akan menghasilkan 2 *output* berformat *json*, yaitu *node_data.json* yang berisi informasi dari *node* dan *conn_data.json* yang berisi informasi koneksi yang menghubungkan satu *node* dengan *node* lainnya.

4.4 Implementasi *Backpropagation*

Setelah semua proses NEAT selesai, proses selanjutnya adalah *backpropagation*. Langkah pertama pada prose *backpropagation* ialah menyiapkan data untuk *training* dan *testing*, serta melakukan konsturksi *network* berdasarkan output dari `extract_net()` pada NEAT. Semua proses *backpropagation* menggunakan bahasa pemrograman *javascript*

Segmen Program 4.8 fungsi `prepare_data()` untuk menyiapkan data berdasarkan tipe *feature* yang digunakan

```
prepareData(type){
  if(type == 1){
      this.trainingData = fs.readFileSync('./dataset/inputTestPlayerOnly.json');
      this.validData = fs.readFileSync('./dataset/validTestPlayerOnly.json');
   }else if(type == 2){
      this.trainingData = fs.readFileSync('./dataset/inputTestWithTeam.json');
      this.validData = fs.readFileSync('./dataset/validTestWithTeam.json');
   }else if(type == 3){
      this.trainingData = fs.readFileSync('./dataset/inputTestWithTeamAndPos.jso
n');
      this.validData = fs.readFileSync('./dataset/validTestWithTeamAndPos.json')
;
        }else{
            console.log("unknown type");
        }
    }
```

Segmen Program 4.9 fungsi `build()` untuk mengkonstuksi *network*

```
build(node_json , conn_json){
        let contents = fs.readFileSync(node_json);
        this.node_data = JSON.parse(contents);

        this.node_count = 0;
        for(let i in this.node_data){
            this.node_count++;
        }

        this.nodes = Array(this.node_count);
        for(let i = 0 ; i < this.nodes.length ; i++){
            this.nodes[i] = new Node(this.node_data[i].type , this.node_data[i].
key , this.node_data[i].bias)
        }
        contents = fs.readFileSync(conn_json);
        this.conn_data = JSON.parse(contents);

        for(let i in this.conn_data){
            for(let j = 0 ; j < this.nodes.length ; j++){
                if(this.conn_data[i].from == this.nodes[j].key){
                    for(let h = 0 ; h < this.nodes.length ; h++){
                        if(this.conn_data[i].to == this.nodes[h].key){
                            if(j != h){
                                this.nodes[j].connect(this.nodes[h] , this.conn_
data[i].weight);
                            }

                        }
                    }
                }
            }
        }
        this.network = architect.Construct(this.nodes);
    }
```

Setelah *network* berhasil dikonsturksi dan data telah disiapkan, proses *training* dan proses *testing* dapat dijalankan.

Segmen Program 4.10 fungsi `backprop()` untuk menjalankan proses *training*

```
backprop(){
        let backprop = true;
        let input = JSON.parse(this.trainingData)
        let opt = {
            log: 100,
```

```
            error: 0,
            iterations: 100000,
            rate: 0.001,
        }
        this.network.train(input,opt,propagate);
}
```

Pada segmen ini, `backprop` diatur ke `true` sebagai tanda untuk melakukan proses *training*. Object `opt` berisi konfirgurasi pada proses *training*. Selanjutnya, fungsi `network.train()` yang berasal dari *library neataptic.js* akan dijalankan untuk memulai proses *training*.

Setelah proses *training* selesai dijalankan, akan dilakukan proses *testing* menggunakan fungsi `calculate_accuracy()` yang bertujuan untuk mengukur tingkat akurasi dari *network*. Fungsi ini menggunakan data *testing* yang tidak dikenali oleh *network*.

Segmen Program 4.11 fungsi `calculateAccuracy()` untuk mengukur akurasi dari *network*

```
calculateAccuracy(){
        let set = JSON.parse(this.testingData);
        for(let i = 0 ; i < set.length ; i++){
            let input = set[i].input;
            let target = set[i].output;
            let output = this.network.activate(input, true);
            let temOutput = output;

            temOutput[0] = Math.round(temOutput[0]);
            temOutput[1] = Math.round(temOutput[1]);
            console.log('output = '+temOutput+' target = '+target);
            if(Math.round(output[0]) == target[0] && Math.round(output[1]) == ta
rget[1]){
                correct++;
            }
            if(Math.round(output[0]) > Math.round(output[1]) && target[0] > targ
et[1]){
                win++;
            }
            if(Math.round(output[0]) < Math.round(output[1]) && target[0] < targ
et[1]){
                win++;
            }
```

```
            if(Math.round(output[0]) == Math.round(output[1]) && target[0] == ta
rget[1]){
                win++;
            }
        }

        console.log("Prediction correct = "+correct)
        console.log("winner correct = "+win)
    }
```