Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer e-ISSN: 2548-964X
Vol. 2, No. 2, Februari 2018, hlm. 703-714 http://j-ptiik.ub.ac.id

Implementasi *Adaptive AI* Pada *Game Turn-Based RPG* Dengan Menggunakan Metode *Hierarchial Dynamic Scripting*

Intishar Fadi Abdillah[1], Eriq Muh. Adams Jonemaro[2], Muhammad Aminul Akbar[3]

Program Studi Teknik Informatika, Fakultas Ilmu Komputer, Universitas Brawijaya
Email: [1]intishar7@gmail.com, [2]eriq.adams@ub.ac.id, [3]ameenul.46@gmail.com

Abstrak

Ada banyak metode yang digunakan oleh pengembang *game* dalam menerapkan *Artificial Intelligence*(AI) ke dalam *Non Playable Character*(NPC) yang bertujuan sebagai penghambat *gamer* dalam mencapai tujuan atau misinya. Metode Scripting adalah salah satu metode yang paling banyak digunakan pengembang game dalam merancang AI pada NPC dikarenakan prosesnya yang sederhana, fleksibel(mudah dimodifikasi), dan *powerful.* Metode *Scripting* merupakan cara untuk merancang *behaviour* NPC melalui kombinasi aturan atau *rule* yang dimodelkan dengan kalimat *if-then*. Kombinasi beberapa *rule* ini tentunya ditulis secara eksplisit pada *source code* atau file eksternal lain (biasa disebut sebagai *hard-coded*) yang mana tidak memungkinkan untuk merubah *behaviour* NPC ketika *game* telah dirilis. Hal ini menyebabkan NPC seringkali mudah dieksploitasi oleh gamer yang telah memahami pola dari behaviour NPC. Untuk mengatasi permasalahan tersebut, diperlukan suatu metode yang mampu menghasilkan *behaviour* NPC yang adaptif namun tetap memberikan sifat dari metode *Scripting* yakni sederhana, fleksibel, dan *powerful*. Oleh karean itu, metode *Hierarchial Dynamic Scripting* mencoba untuk menjawab permasalahan tersebut dalam penelitian ini. Metode *Hierarchial Dynamic Scripting* merupakan pengembangan lanjut dari metode *Dynamic Scripting* dengan menambahkan arsitektur *Hierarchial Task Network* di dalamnya. Prinsip utama dari metode *Dynamic Scripting* adalah memberikan *weight* atau bobot tertentu pada kumpulan *rule* dan mengkombinasikannya menjadi sebuah *script* yang dinamis. Hasil pengujian adaptifitas yang dilakukan pada game dengan genre *Turn-Based Role Playing Game* berdasarkan tiga parameter pengujian yakni efektifitas, efisiensi, dan variasi menunjukkan bahwa NPC dengan menggunakan metode *Hierarchial Dynamic Scripting* memiliki efektifitas yang tinggi dengan tolak ukur nilai rata-rata *fitness* sebesar 0.73 dan efisiensi yang tinggi dengan mencapai rata-rata nilai *turn point* sebesar 6 melawan semua taktik skenario dalam penelitian ini. Selain itu, pengujian nilai *break even point* menghasilkan nilai terbaik sebesar 0,5 untuk mendapatkan tingkat adaptifitas NPC yang baik.

Kata kunci: *dynamic scripting, hierarchial dynamic scripting, game AI, rulebase, learning AI, turn based RPG*

Abstract

There are many methods used by game developers in applying Artificial Intelligence (AI) into Non Playable Character (NPC) which aims to inhibit gamers in achieving their goals or mission. Scripting method is one of the most used method of game developers in designing AI in NPC because of its simple, flexible (easy to modify), and powerful process. Scripting method is a way to design behavior NPC through a combination of rules that are modeled with if-then sentences. The combination of these rules is explicitly written on source code or other external files (commonly referred to as hard coded) which is not possible to change the NPC behavior when the game has been released. This causes the NPC is often easily exploited by gamers who have understood the pattern of the NPC behavior. To overcome these problems, we need a method capable of producing an adaptive NPC behavior while still providing the nature of Scripting method that is simple, flexible, and powerful. Therefore, Hierarchial Dynamic Scripting method tries to answer the problem in this research. Hierarchial Dynamic Scripting method is an advanced development of the Dynamic Scripting method by adding Hierarchial Task Network architecture in it. The main principle of the Dynamic Scripting method is to assign specific weights to the set of rules and combine them into a dynamic script. Adaptive test results conducted in the game with the genre of Turn-Based Role Playing Game based on three parameters of

the test of effectiveness, efficiency, and variation shows that the NPC using Hierarchial Dynamic Scripting method has a high effectiveness with the average fitness value of 0.73 and High efficiency by achieving an average turn-point value of 6 against all scenario tactics in this study. In addition, testing the value of break even point yields the best value of 0.5 to get a good level of NPC adaptability.

Keywords: *dynamic scripting, hierarchial dynamic scripting, game AI, rulebase, learning AI, turn based RPG*

1. PENDAHULUAN

Dalam mengembangkan *non playable character*(NPC) atau agen AI(*Artificial Intelligence*), pengembang *game* masih menggunakan metode *scripting, finite state machines, rule-based system,* dan metode lain yang membutuhkan *domain knowledge* mengenai *game* yang dikembangkan. Hal ini menyebabkan timbulnya permasalahan seperti *pattern* atau *behaviour* dari agen AI terlalu mudah diprediksi dan repetitif. Hal ini menurunkan tingkat tantangan sekaligus nilai *entertainment* yang didapatkan *gamer* (Wang & Tan, 2015). Salah satu metode yang sering digunakan oleh pengembang *game* adalah metode *scripting*. Metode *scripting* merupakan cara untuk merancang *behaviour* agen AI melalui kombinasi aturan atau *rule* yang dimodelkan dengan kalimat *if-then*. Dengan mengkombinasikan beberapa *rule*, pengembang *game* dapat dengan mudah mendefinisikan dan memodifikasi *behaviour* dari agen AI. Adapun kelemahan utama dari metode *scripting* ini adalah semakin kompleks *environment game*, maka semakin kompleks pula kombinasi *rule* yang harus dibuat. Pengembang *game* harus mengkombinasikan *rule* sehingga teruji dalam semua skenario yang mungkin saja terjadi ketika *game* telah dirilis dan dimainkan oleh *gamer*. Hal ini tentu saja menimbulkan resiko yang tinggi agen AI dapat dieksploitasi dengan mudah pada skenario yang belum ditemukan oleh pengembang *game* (Price, 2011).

Berdasarkan hal tersebut, gagasan AI adaptif muncul di tengah perkembangan AI pada *game* saat ini. Dengan memanfaatkan teknik *learning* yang ada pada *Academic AI*(istilah untuk ilmu AI pada ranah akademis) seperti *artificial neural network*, *genetic algorithm*, serta teknik sejenis lainnya, para akademisi melakukan eksperimen untuk menerapkan teknik-teknik tersebut pada *game* AI yang hasilnya terbukti dapat menciptakan agen AI yang adaptif. Sementara itu, pengembang *game* masih belum berani dalam mengambil teknik *learning* yang telah dipublikasikan oleh akademisi dikarenakan beberapa aspek seperti tidak konsistennya *behaviour* agen AI sehingga terkadang menimbulkan *inferior behaviour* atau tindakan agen AI yang tidak jelas dan merugikan agen AI itu sendiri, memerlukan waktu yang tidak sedikit untuk menganalisis agen AI saat proses *testing game* dikarenakan sifat alami teknik *learning*, serta sulitnya proses pengumpulan data yang benar-benar mewakili *behaviour* dari *gamer* (Muñoz-Avila, et al., 2013).

Dengan melihat permasalahan tersebut, diperlukan sebuah metode yang adaptif namun prosesnya dapat diidentifikasi dan dianalisis agar tidak mengurangi kontrol pengembang *game* akan *behaviour* agen AI yang dikembangkan. Berdasarkan studi literatur pada konteks permasalahan ini, metode *Dynamic Scripting* memiliki kriteria sebagaimana disebutkan sebelumnya yaitu adaptif dan terkontrol. Prinsip utama dari metode *Dynamic Scripting* adalah memberikan *weight* atau bobot tertentu pada tiap *rule* dalam *rulebase* yang digunakan untuk membentuk *behaviour* AI. *Rule* dengan nilai bobot yang baik akan dipilih dan di-*generate* menjadi kombinasi *rule* atau *scripting*. Dengan kata lain, *scripting* yang dihasilkan bersifat dinamis sesuai bobot dari tiap *rule* dalam *rulebase*. Bobot tiap *rule* akan selalu diperbarui sesuai performanya ketika dieksekusi (Toubman, et al., 2014).

Penelitian mengenai keadaptifan agen AI menggunakan *Dynamic Scripting* telah banyak dilakukan. Sebagai referensi dan sumber dalam penelitian ini digunakan penelitian dengan rentang waktu lima tahun terakhir. Salah satu penelitian keadaptifan agen AI menggunakan *Dynamic Scripting* dilakukan oleh Febri Abdullah yang mana mengembangkan metode *Dynamic Scripting* dengan penambahan teknik *Macro Action*. *Macro Action* merupakan urutan-urutan beberapa aksi atau *rule* yang ditangani sebagai satu unit tunggal. *Macro Action* juga digunakan untuk merepresentasikan style permainan dari agen AI. Dengan

mengkombinasikan beberapa *macro action* dari *macro-base* menggunakan teknik *generation* dari *Dynamic Scripting*, memungkinkan untuk menggabungkan beberapa *style* permainan sehingga keberagaman dapat ditingkatkan (Abdullah, 2013). Hasil penelitian yang dilakukan Febri Abdullah berhasil membuktikan bahwa dengan penambahan teknik *Macro Action*, dapat menghasilkan keberagaman *script* dengan rata-rata nilai sebesar 0,95. Sementara itu, *Dynamic Scripting* tanpa *Macro Action* menghasilkan keberagaman yang lebih rendah yakni sebesar 0,75. Namun kelemahan dari metode *Dynamic Scripting Macro Action* adalah meskipun *script* yang dibentuk bersifat dinamis, akan tetapi kombinasi urutan aksi dalam sebuah *Macro Action* tidak dapat diubah(statis) sehingga secara alami pengembang game harus mendefinisikan secara manual kumpulan *style* bermain atau *Macro Action Base*. Oleh karena itu, pada penelitian ini diajukan metode *Hierarchial Dynamic Scripting* untuk menutup kekurangan metode *Dynamic Scriping Macro Action* sekaligus menguji keadaptifan agen AI menggunakan metode ini.

2. TURN BASED ROLE PLAYING GAME

Turn-Based Role Playing Game merupakan *sub-genre* dari *Role Playing Game*(RPG). Pada *turn-based RPG*, pemain mengontrol satu atau lebih *party* yang berisikan karakter dengan berbagai *skill* atau kemampuan khusus sesuai dengan *class* atau profesinya. Pertempuran dalam *turn-based RPG* dilakukan dengan cara penentuan *turn* atau giliran. Ada banyak variasi dari cara penentuan giliran. Salah satunya seperti permainan catur yaitu penentuan giliran selanjutnya dilakukan setelah pemain sebelumnya telah menyelesaikan aksinya. Dalam tiap gilirannya, pemain memilih aksi yang dilakukan oleh tiap karakter dalam *party* yang sedang dikontrol. Aksi ini bisa berupa mengeluarkan *skill* atau menyerang, bergerak, menggunakan item, dan aksi lainnya sesuai dengan konteks dan aturan dari game. Misi dari tiap pertempuran bermacam-macam seperti mengeliminasi semua musuh, bertahan hingga *turn* tertentu, dll. Tapi pada umumnya, misi tiap pertempuran adalah mengeliminasi semua musuh yang dihadapi.

Beberapa *class* atau profesi yang selalu ada dalam setiap game *turn-based RP*G adalah *Warrior*, *Wizard*, dan *Healer*. *Warrior* identik dengan keunggulan serangan fisiknya daripada serangan *magic* yang dimilikinya. Sedangkan *Wizard* memiliki keunggulan dalam serangan *magic* dan lemah dalam serangan fisik. Sementara itu, *Healer* merupakan profesi yang sangat efektif dalam memulihkan *hit-point*. *Hit-point* atau biasa disebut HP merupakan indikator yang menyatakan apakah suatu karakter masih aktif dan dapat melakukan aksinya. *Hit-point* dapat berkurang jika karakter terkena serangan lawannya. Jika *Hit-point* mencapai nol maka karakter dianggap "mati" atau hilang dari permainan. Sementara itu, *Mana-point* merupakan indikator apakah karakter masih dapat menggunakan *skill*-nya atau tidak. Setiap pengaktifan *skill*, karakter akan menggunakan *Mana-point* dengan besaran yang berbeda tergantung pada kualitas *skill* yang diaktifkan tersebut. Jika *Mana-point* mencapai nol, maka karakter tidak dapat mengaktifkan *skill*. Pada umumnya, *Mana-point* akan bertambah di tiap turn pertempuran atau dengan mengkonsumsi item tertentu dalam game.

Dalam penelitian ini dirancang sebuah *game turn based RPG* dengan skala kecil untuk menguji keadaptifan metode *Hierarchial Dynamic Scripting*. Game ini hanya dikembangkan pada aspek *battle game turn based RPG*. Setiap karakter(agen AI) memiliki atribut dan sejumlah *skill*. Nilai atribut dan *skill* yang dimiliki oleh tiap agen AI berbeda-beda sesuai dengan *class* atau profesi agen AI. Dalam penelitian ini, hanya menggunakan tiga profesi agen AI yaitu *Warrior*, *Wizard*, dan *Healer*. Atribut tiap profesi ditunjukkan pada Tabel 1.

Tabel 1. Atribut Agen AI Per Profesi

Profesi	HP	MP	DEF	STR
Wizard	200	130	3	30
Warrior	185	100	9	40
Healer	180	140	5	25

Pada Tabel 1, HP, MP, DEF, dan STR masing-masing merupakan *Hit Point*, *Mana Point*, *Defence*, dan *Strength*. DEF mempengaruhi besar *damage* yang diterima agen AI terhadap *skill* lawan. Semakin tinggi DEF, tingkat reduksi *damage* agen AI semakin tinggi. STR mempengaruhi besar *damage* dari suatu *skill* yang memiliki efek *damage*.

Semakin tinggi STR, maka semakin besar *damage* yang dikeluarkan oleh *skill*. Atribut yang dimiliki ketiga profesi tersebut sama tetapi berbeda dalam nilainya. Profesi *Warrior* memilik karakteristik nilai atribut DEF lebih besar daripada *Wizard* dan *Healer* namun nilai atribut MP-nya rendah. Sedangkan profesi *Healer* memiliki nilai atribut MP tinggi karena memiliki *skill* penambah HP yang berguna untuk memberi dukungan kepada *party*. Adapun daftar skill yang dimiliki tiap profesi ditunjukkan oleh Tabel 2.

Tabel 2. *Skill* Profesi *Wizard*

Skillname	*Category*	*MP Need*	*Min Damage*	*Max Damage*	*Status Effect*	*Status Damage/ Bonus*	*Status Duration*
Attack	Damage	0	11	12	-	-	-
Flare	Damage	20	22	25	-	-	-
Phoenix Storm	Damage	28	27	30	-	-	-
Temptation	Buff	20	-	-	ATKUP:*Increase* ATK	20	3
Eclipse	Divine	27	23	24	BLIND: *Enemy Action Miss*	15%	2

Tabel 3. *Skill* Profesi *Warrior*

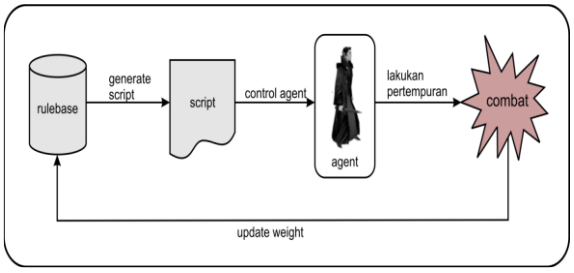
Skillname	*Category*	*MP Need*	*Min Damage*	*Max Damage*	*Status Effect*	*Status Damage/ Bonus*	*Status Duration*
Attack	Damage	0	12	13	-	-	-
Cascade Strike	Damage	19	20	21	-	-	-
Rains ‘O Blade	Damage	28	30	32	-	-	-
Demon’s Zeal	Buff	22	-	-	DEFUP:*Increase* DEF	30	3
Demon’s Blow	Divine	27	15	17	POISON: *Decrease Enemy* HP	9%	2

Tabel 4. *Skill* Profesi *Healer*

Skillname	*Category*	*MP Need*	*Min Damage*	*Max Damage*	*Status Effect*	*Status Damage/ Bonus*	*Status Duration*
Attack	Damage	0	9	10	-	-	-
Lightning Arc	Damage	18	15	17	-	-	-
Poseidon	Damage	22	24	27	-	-	-
Revitalize	Heal	35	-	-	HEAL:*Increase* HP	30	3
Purifier Fountain	Clear	25	-	-	CLEAR: *Remove poison, blind*	9%	2

3. DYNAMIC SCRIPTING

Dynamic Scripting memiliki siklus utama yaitu *script generation*, *perfom*, dan *evaluation*. Proses *script generation* dilakukan untuk meng-*generate script* yang berisi kombinasi *rule*. Proses pemilihan *rule* menggunakan metode random dengan mempertimbangkan *weight* atau bobot pada tiap *rule*. Semakin besar nilai bobot, maka semakin tinggi kemungkingan *rule* tersebut untuk dipilih mejadi bagian dari *script*. Script yang telah di-*generate* akan digunakan agen AI pada tahap *perform*.



Gambar 1. Alur Kerja *Dynamic Scripting*

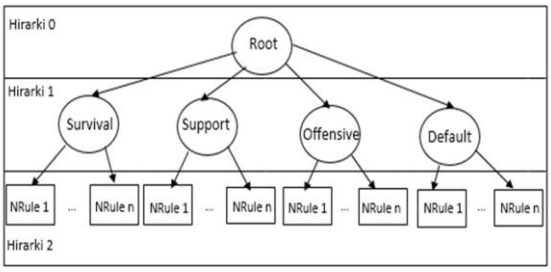
Proses evaluasi dilakukan ketika tahap *perform* selesai. Proses evaluasi pada *Dynamic Scripting* menggunakan fungsi *fitness* yang berupa nilai numerik antara 0 dan 1 dalam mengevaluasi performa agen AI pada tiap periode *perform* yang ditentukan oleh pengembang *game*. Hasil output dari fungsi *fitness* akan digunakan sebagai penentuan besarnya nilai *reward* atau *penalty*. Nilai *reward* atau *penalty* inilah yang akan didistibusikan pada atribut *weight* dari tiap *rule* yang diaktifkan pada periode *perform* sebelumnya. Alur kerja *Dynamic Scripting* ditunjukkan pada Gambar 1.

Pada Gambar 1, agen AI melakukan pertempuran dengan menggunakan *script* yang berasal dari proses *script generation*. Proses *script generation* merupakan proses pemilihan *rule-rule* pada *rulebase* secara terbatas untuk dijadikan sebuah *script*. *Script* direpresentasikan sebagai *array rule* dan memiliki ukuran tertentu untuk menampung *rule* terpilih. Kualitas sebuah *script* tergantung pada kombinasi *rule* terpilih di dalamnya. Setelah *script* dibentuk, agen AI masuk ke tahap *perform* untuk melakukan pertempuran. Pada tahap *perform*, pemilihan aksi atau *decision making* agen AI dipengaruhi oleh algoritma yang digunakan untuk mengeksekusi *script* yang dimiliki agen AI. Setelah tahap *perform* selesai, dilakukan proses evaluasi yang di dalamnya memuat dua proses yaitu penghitungan performa agen AI menggunakan fungsi *fitness* dan pembaruan nilai bobot tiap *rule* pada *rulebase*.

4. HIERARCHIAL DYNAMIC SCRIPTING

Metode *Hierarchial Dynamic Scripting*(HDS) merupakan metode pengembangan dari *Dynamic Scripting* dengan mengadaptasi arsitektur *Hierarchial Task Network* untuk membentuk sebuah *tree*. Oleh karena itu, tahapan pada modul HDS memiliki kesamaan dengan tahapan dalam metode *Dynamic Scripting*. Tahapan tersebut adalah *script generation*, *perform*, dan *evaluation*. Perbedaan metode *Dynamic Scripting* dan *Hierarchial Dynamic Scripting*, terdapat pada beberapa aspek seperti *rulebase*, model *script* yang digunakan, serta proses evaluasi khususnya pada tahap distribusi nilai bobot.

Arsitektur dari *Hierarchial Dynamic Scripting* direpresentasikan dalam sebuah struktur data *tree* yang ditunjukkan pada Gambar 2. *Node* pada hirarki teratas adalah *root*. Hirarki selanjutnya(*child* dari *root*) adalah *node* kategori yang merepresentasikan garis besar strategi atau kategori *behaviour* dari agen AI. Tiap *node* kategori dapat memiliki *child* berupa *node* kategori lainnya ataupun *node rule*. *Node rule* merupakan *node* yang merepresentasikan *rule* atau *behaviour* dasar dari agen AI. *Node rule* tidak memiliki *child* atau disebut juga *leaf*. Sementara itu, metode *Dynamic Scripting* diterapkan pada *node root* dan *node* kategori untuk mengevaluasi dan memilih *node rule* di dalamnya menjadi sebuah *script* yang digunakan oleh agen AI.



Gambar 2. Arsitektur Tree HDS Agen AI

Gambar 2 merupakan contoh ilustrasi arsitektur tree metode HDS. Pada Gambar 2 terdapat tiga kategori *behaviour* yang dapat digunakan oleh agen AI yaitu yaitu *behaviour Survival*, *Support*, *Offensive*, dan *Default*.

Penentuan kategori *behaviour* didasarkan pada karakteristik dan kemampuan agen AI. Secara garis besar, *skill* yang dimiliki oleh agen AI pada game turn based RPG dalam penelitian ini dapat dikategorikan menjadi beberapa tipe yaitu *Damage*, *Buff*, *Heal*, *Clear*, dan *Divine*. Tipe *Damage* dan *Divine* dapat mengurangi HP dan memberikan status efek kepada lawan. Maka dari itu, kategori *behaviour Offensive* dibentuk untuk menampung *rule-rule* yang mendefinisikan *skill* bertipe *Damage* dan *Divine*. Sementara itu, tipe *Buff*, *Heal*, dan *Clear* dapat dikategorikan dalam *Survival* dan *Support*. Kategori *behaviour* Survival berisi *rule-rule* yang mendefinisikan cara bertahan diri agen AI seperti mengaktifkan *buff Defense* pada *Warrior* ketika dalam keadaan kritis, mengembalikan HP diri sendiri pada *Wizard* dalam kondisi tertentu, dll. Kategori behaviour Support mendefinisikan cara memberikan dukungan kepada rekan satu tim agen AI seperti melakukan heal pada rekan yang memiliki HP rendah, dll.

4.1. Script Generation

Tahap *script generation* merupakan tahap untuk meng-*generate* sebuah *script* yang berisikan *rule-rule* dari *rulebase* agen AI pengguna *script* tersebut. Selain untuk menyimpan hasil ekstrak *rulebase*, *tree* HDS juga digunakan untuk menyimpan *node* terpilih di tiap *node* kategorinya. Proses *script generation* terjadi pada *node* kategori(memiliki *child*). *Node root* menyimpan *node* kategori terpilih sementara dan untuk tiap *node* kategori terpilih menyimpan *node rule* terpilih. Proses seleksi dimulai pada hirarki ke-0 atau *root* untuk menentukan *node* kategori yang akan digunakan pada *script*. Nilai bobot dari *node* kategori merupakan rata-rata dari nilai bobot *node child*-nya(hirarki ke-2). Khusus untuk *node* kategori *Default*, selalu ditambahkan saat seleksi hirarki ke-0. Setelah itu, proses seleksi berlangsung pada hirarki ke-1 untuk tiap *node* kategori terpilih. Jumlah pengambilan *node rule* tiap *node* kategori berbeda-beda. Pada penelitian ini, untuk kategori *rule Survival* hanya diambil satu *rule*, kategori *Support* dua *rule*, dan kategori *Offensive* tiga *rule*. Sementara itu, *node rule* pada *node* kategori *Default* langsung dipilih tanpa seleksi. Jumlah total *rule* dalam *script* tidak melebihi jumlah total *rule* per kategori yaitu enam *rule*. Setelah semua *node* kategori beserta *node rule* di dalamnya terpilih, *node* kategori pada *script* akan diurutkan secara *descending* berdasarkan nilai bobotnya. Hal ini berguna untuk mempercepat proses pemilihan kategori *behaviour* pada tahap *perform*. Metode seleksi *node* pada tahap *script generation* menggunakan teknik *random* dengan probabilitas berdasarkan nilai bobot *node*. Algoritma yang digunakan untuk menyeleksi *node rule* pada tiap *node* kategori memiliki langkah sebagai berikut:

1. Kosongkan daftar *node* terpilih atau *selectedNodes* pada *node* kategori saat ini.
2. Cek apakah *node* kategori merupakan *root.* Jika merupakan *root*, maka tambahkan *node* kategori *Default* secara langsung.
3. Jumlah semua nilai bobot *child node* dan simpan pada *sumweight.*
4. Mulai lakukan perulangan sebanyak batas pengambilan atau *maxSelected*. Selama jumlah *child node* dalam *selectedNodes* belum mencapai *maxSelected*, lakukan langkah no.5.
5. *Generate* nilai *random* mulai dari 0 sampai *sumweight* dan simpan nilai tersebut pada *fraction*. Simpan nilai bobot *child node* pertama dalam *child node* kategori saat ini pada *tempSum*. Cek nilai *tempSum* apakah lebih besar daripada nilai *fraction*. Jika lebih besar, maka ambil *child node* pertama tersebut dan masukkan ke dalam *selectedNodes.* Jika lebih kecil dari nilai *fraction*, tambahkan nilai *tempSum* dengan nilai bobot *child node* selanjutnya. Jika nilai *tempSum* setelah penambahan lebih besar daripada *fraction*, ambil *child node* yang menyebabkan nilai *tempSum* melebihi *fraction* dan simpan pada *selectedNodes*. Proses penambahan *tempSum* dengan nilai bobot *child node* dilakukan selama nilai *tempSum* kurang dari *fraction*.
6. Jika jumlah *child node* dalam *selectedNodes* telah mencapai *maxSelected*, hentikan perulangan. Dengan demikian proses seleksi *child node* pada *node* kategori selesai.

4.2. Perform

Tahap *perform* merupakan tahap dimana agen AI menggunakan *script* yang telah di-*generate* pada tahap *script generation* untuk memilih sebuah aksi yang dilakukan saat mendapat *turn*. Pemilihan aksi dalam *tree script* dimulai dari *node* kategori pertama pada hirarki ke-1 lalu akan dicek *node rule* di dalamnya dengan mempertimbangkan nilai bobot dan *Mana Point*(MP) yang dibutuhkan untuk

mengeksekusi *rule*. *Node* yang memiliki nilai bobot terbesar dan MP agen AI cukup untuk mengeksekusi *rule* pada *node* tersebut akan dipillih menjadi aksi yang akan dilakukan agen AI. Jika *node rule* pada *node* kategori tidak ada yang memenuhi kriteria pemilihan aksi maka pengecekan akan berlanjut pada *node* kategori selanjutnya. Jika dari *node* kategori yang ada tidak terdapat *node rule* yang dapat dieksekusi maka pengecekan akan berakhir pada *node* kategori *Default*. *Node rule* di dalam *node* kategori *Default* merupakan *rule* default sehingga pasti terpilih dan dieksekusi oleh agen AI.

4.3 Evaluation

Tahap *evaluation* memiliki dua sub-tahap di dalamnya yaitu *calculate fitness* dan *weight adjusment*. Tahap *calculate fitness* merupakan tahap penilaian performa agen AI HDS pada tahap *perform* sebelumnya dengan menggunakan fungsi *fitness*. Sedangkan tahap *weight adjusment* merupakan tahap pengubahan bobot tiap *node rule* pada *rulebase* dengan mempertimbangkan nilai fungsi *fitness* yang telah didapatkan pada tahap *calculate fitness*.

Fungsi *fitness* yang digunakan dalam menilai performa agen AI adalah $F(g), C(g), B(g), D(a)$, dan $F(a,g)$. Nilai dari tiap fungsi fitness haruslah bernilai 0 hingga 1. Fungsi $F(g)$ merupakan fungsi *fitness* untuk menghitung hasil pertarungan tim.

$$\mathrm{F}(g) = \sum_{c \in g} \begin{cases} 0 & g\ lost \\ \frac{1}{2Ng}\left(1 + \frac{hT(c)}{hO(c)}\right) & g\ won \end{cases} \quad (1)$$

Persamaan 1 merupakan fungsi *fitness* untuk menghitung hasil pertarungan tim. Pada Persamaan 1, g merupakan tim agen AI yang dievaluasi, sedangakan *c* merujuk pada agen AI yang dievaluasi. *Ng* merupakan total agen AI pada tim *g*. *hT* merupakan nilai HP terakhir dari agen *c*. *h0* merupakan nilai HP awal yang dimiliki agen *c*.

Fungsi *fitness* $C(g)$ digunakan untuk menghitung besar *damage* yang dihasilkan oleh tim yang dituliskan dalam Persamaan 2.

$$C(g) = \frac{1}{N \backsim g} \sum_{c \in \backsim g} \begin{cases} 1 & hT(c) \leq 0 \\ 1 - \frac{hT(c)}{h0(c)} & hT(c) > 0 \end{cases} \quad (2)$$

Pada Persamaan 2, beberapa variabel yang perlu diperhatikan adalah $N \backsim g$ dan *c*. terdapat perbedaan pada nilai yang dirujuk oleh dua atribut tersebut dengan Persamaan 4.1 sebelumnya. $N \backsim g$ merupakan negasi dari nilai Ng atau dapat diartikan merujuk pada jumlah tim lawan dari tim g. Begitu pula dengan nilai *c* yang merujuk pada agen AI anggota dari tim $\backsim g$(tim lawan dari tim g).

Fungsi *fitness* selanjutnya adalah fungsi *fitness* $B(g)$ yang digunakan untuk menghitung tingkat *survivalability* dari suatu tim yang dituliskan pada Persamaan 3.

$$B(g) = \frac{1}{2Ng} \sum_{c \in g} \begin{cases} 0 & hT(c) \leq 0 \\ 1 + \frac{hT(c)}{h0(c)} & hT(c) > 0 \end{cases} \quad (3)$$

Persamaan 3, hampir sama dengan Persamaan 1 dengan perbedaan pada operator "-" yang diganti dengan operator "+" dan nilai yang dirujuk oleh Ng dan c. Nilai Ng dan c pada Persamaan ini sama dengan Persamaan 1. Berdasarkan Persamaan 3, nilai 0 akan diberikan jika hT dari agen AI kurang dari sama dengan nol yang berarti jika suatu tim mengalami kekalahan maka $B(g)$ akan bernilai 0. Fungsi *fitness* berikutnya adalah $D(a)$ yang digunakan untuk menghitung tingkat *survivalability* dari agen AI. Fungsi $D(a)$ dituliskan dalam Persamaan 4.

$$D(a) = \frac{t(a)}{tB} \quad (4)$$

Pada Persamaan 4, $t(a)$ merupakan *turn* saat agen AI mengalami KO. Sementara tB adalah jumlah *turn* untuk mengakhiri satu pertarungan. Nilai tB berbeda-beda untuk tiap simulasi pertarungan. Sebagai contoh, agen AI mengalami KO pada *turn* ke-7 dan pertarungan berakhir pada *turn* ke-10. Berdasarkan Persamaan 4, maka nilai *fitness*nya adalah 7/10 yaitu 0,7.

Fungsi *fitness* terakhir yaitu $F(a,g)$ merupakan total dari semua fungsi *fitness* yang telah dijelaskan sebelumnya.

$$F(a,g) = \frac{1}{10}\left(5F(g) + B(g) + 2C(g) + 2D(a)\right) \quad (5)$$

Pada Persamaan 5, a merujuk pada agen AI dan g merujuk pada tim dari agen AI a. Pada Persamaan 5, nilai fungsi *fitness* $F(g)$ memiliki faktor pengali terbesar yaitu 5 sehingga memiliki pengaruh paling besar pada

fitness akhir agen AI daripada nilai fungsi *fitness* lainnya. Hal ini berarti kemenangan bagi suatu tim akan sangat mempengaruhi nilai *fitness* akhir agen AI. Begitu pula sebaliknya, kekalahan akan membuat nilai *fitness* akhir agen AI menjadi sangat rendah. Sementara itu, untuk fungsi $C(g)$ dan $D(a)$ memiliki faktor pengali 2 yang berarti bahwa bobot penilaian fungsi *fitness* besar *damage* dan *survivalability* agen AI sama besar.

Sub-tahap selanjutnya yaitu *weight adjusment*, menggunakan nilai $F(a,g)$ untuk menentukan besar perubahan bobot *node rule*.

$$\Delta w = \begin{cases} -\left(maxP\,\frac{b-F}{b}\right) & F<b \\ maxR\,\frac{F-b}{1-b} & F\geq b \end{cases} \quad (6)$$

Persamaan 6 merupakan fungsi untuk menghitung nilai Δw. Δw merupakan nilai perubahan bobot. F merupakan nilai dari fungsi *fitness* akhir agen AI atau $F(a,g)$. b atau break-even point merupakan nilai *threshold* dari fungsi *fitness* akhir yang menentukan apakah agen AI layak mendapat *reward* atau pinalti. $maxR$ merupakan maksimal *reward* yang dapat diberikan pada agen AI. Sementara $maxP$ merupakan maksimal pinalti yang dapat diberikan pada agen AI. Berdasarkan Persamaan 6, jika nilai F kurang dari b maka Δw merupakan penghitungan dari nilai pinalti dan sebaliknya jika nilai F lebih dari sama dengan b, maka Δw bernilai *reward*. Setelah Δw didapatkan, maka nilai bobot *node rule* akan ditambahkan dengan nilai Δw. Setiap *node rule* memiliki nilai maksimum dan minimum bobot. Jika setelah penambahan nilai bobot dengan Δw melebihi nilai maksimum maka nilai bobot disesuaikan menjadi nilai maksimum. Begitu juga sebaliknya ketika nilai bobot setelah penambahan kurang dari nilai minimum maka nilai bobot disesuaikan menjadi nilai minimum.

4.4. Rulebase

Agen AI HDS memerlukan komponen *rulebase* sebagai penyuplai kombinasi *rule* pada modul *scripting*nya. Setiap *rule* pada *rulebase* mendefinisikan *behaviour* agen AI. Untuk memenuhi kebutuhan metode HDS, maka kumpulan *rule* dalam *rulebase* harus dikategorikan ke dalam kategori tertentu. Dlam penelitian ini, kategori *rule* dalam *rulebase* dibagi menjadi beberapa kategori yaitu *Survival*, *Support*, *Offensive*, dan *Default*. Kategori *Survival* berisi *rule-rule* yang mendefinisikan *behaviour* bertahan dari agen AI. Kategori *Support* berisi *rule-rule* untuk memberikan dukungan pada agen AI dalam satu tim seperti menambah atribut HP agen AI yang kurang dari 50 persen jumlah HP maksimalnya, dll. Kategori *Offensive* berisi *rule-rule* yang mendefinisikan *behaviour* menyerang agen AI. Sementara kategori *Default* mendefinisikan *rule default* yang harus terdapat modul *scripting*. Format penulisan *rulebase* secara keseluruhan adalah sebagai berikut.

Tabel 5 Format Rulebase HDS

@Sup
rule support 1
...
rule support n
@Sur
rule survival 1
...
rule survival n
@Off
rule offensive 1
...
rule offensive n
@Def
rule default 1
...
rule default n

Pada penelitian ini, setiap *rulebase* disimpan dalam ekstensi *.txt*. Baris @Sup pada *rulebase* merepresentasikan blok kategori *Support*. Semua *rule* setelah baris @Sup hingga sebelum baris @Sur termasuk dalam kategori *Support*. Hal ini berlaku juga pada baris @Sur, @Off, dan @Def. Kategori *rule* pada tiap *rulebase* agen AI tidak selalu memuat empat kategori di atas namun dapat diubah sesuai keinginan pengembang *game* dan konteks dari *game* yang dikembangkan.

5. HASIL DAN PEMBAHASAN

Pengujian keadaptifan agen AI menggunakan metode Hierarchial Dynamic Scripting(HDS) didasarkan pada tiga parameter yakni efektifitas, efisiensi, dan variasi. Parameter efektifitas adalah parameter yang mengukur kualitas agen AI HDS dalam menyelesaikan pertarungan dalam *game Turn Based RPG*. Parameter efektifitas diukur dengan melihat nilai rata-rata *fitness* agen AI HDS pada tiap skenario uji. Agen AI HDS dikatakan efektif jika memiliki rata-rata nilai

fitness lebih besar daripada rata-rata nilai *fitness* agen AI statis. Parameter efisiensi dalam konteks penelitian ini adalah tingkat kecepatan agen AI HDS dalam beradaptasi dengan taktik agen AI statis. Untuk mengukur efisiensi, menggunakan istilah *turn point*. *Turn point* merujuk pada pertarungan ke-*i* dimana tim HDS berhasil mengungguli nilai rata-rata *fitness* agen AI statis pada minimal sepuluh pertarungan selanjutnya setelah pertarungan ke-*i*. Semakin rendah nilai *turn point*, maka semakin tinggi efisiensi agen AI HDS. Semetara itu, parameter variasi mengukur tingkat keberagaman *script* yang di-*generate* oleh agen AI dalam tiap kali pertarungan yang dilakukan. Selain menguji keadaptifan agen AI, dilakukan pula pengujian nilai parameter *break even point* untuk melihat pengaruhnya pada tingkat adaptifitas agen AI.

Pengujian dilakukan menggunakan beberapa skenario. Pada tiap skenario, dipertemukan dua tim yang masing-masing terdiri dari tiga agen AI di dalamnya. Tim pertama terdiri dari tiga agen AI HDS atau bisa kita sebut sebagai tim HDS. Tim kedua terdiri dari agen AI statis atau bisa kita sebut sebagai tim statis. Kedua tim memiliki komposisi profesi agen AI yang sama yaitu satu *Warrior*, satu *Wizard*, dan satu *Healer*. Simulasi pertarungan kedua tim pada tiap skenario dilakukan sebanyak 5 iterasi dan pada tiap iterasi dilakukan 150 pertarungan. Tiap skenario akan diuji dengan tiga nilai *break-even point* untuk melihat pengaruhnya pada adaptifitas agen AI. Tiga nilai *break-even point* yang diuji adalah 0,4; 0,5; dan 0,6. Adapun nilai parameter lain yang digunakan dalam pengujian ditunjukkan pada Tabel 6.

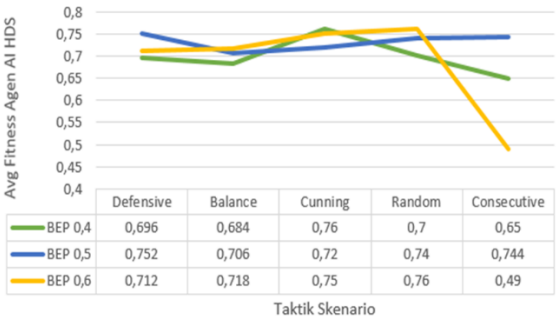
Tabel 6. Nilai Parameter Hierarchial Dynamic Scripting

Parameter	Nilai
initNodeWeight	100
minWeight	0
maxWeight	2000
maxReward	100
maxPenalty	40
break-even point	Diuji

Terdapat lima skenario pengujian yang dilakukan dalam penelitian ini untuk membuktikan tingkat keadaptifan agen AI HDS. Skenario pengujian disini diimplementasikan dalam bentuk pertarungan agen AI HDS melawan beberapa taktik agen AI statis. Skenario pertama adalah *Defensive*. Agen AI statis yang menggunakan taktik *Defensive* secara garis besar akan memprioritaskan *behaviour survivalability* dari dirinya dan juga mendukung rekan setimnya. Skenario kedua adalah *Cunning* yang memprioritaskan *skill* dengan *damage* berkala seperti *poison* serta status efek seperti *blind* untuk mencurangi lawan. Skenario ketiga adalah *Balance* yang memprioritaskan kesemibangan anrata *behaviour* bertahan dan menyerang. Skenario keempat adalah *Random* yang selalu mengganti taktiknya secara acak antara *Defensive*, *Cunning*, dan *Balance*. Skenario terakhir adalah *Consecutive* yang selalu mengganti taktiknya setelah mengalami kekalahan.

5.1. Hasil Pengujian

Hasil pengujian skenario dibedakan berdasarkan parameter pengujian yaitu efektifitas, efisiensi, dan variasi yang masing-masing ditunjukkan pada Gambar 3, 4, dan 5.

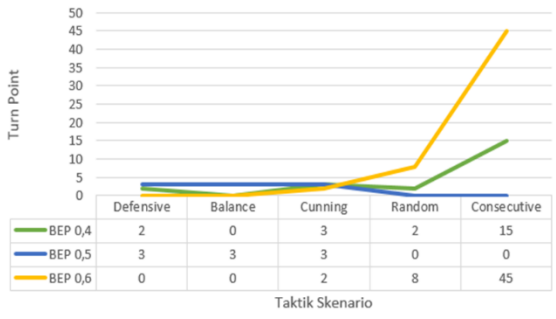


Gambar 3. Hasil Pengujian Efektifitas

Grafik hasil pengujian efektifitas pada Gambar 3 menunjukkan tingkat efektifitas agen AI HDS dengan acuan nilai rata-rata *fitness* agen AI HDS pada tiap pertarungan melawan lima taktik skenario dalam penelitian ini. Dengan melihat ketiga grafik pada Gambar 3, grafik BEP 0,6 mengalami penurunan yang drastis daripada grafik BEP lainnya. Agen AI dengan BEP 0,6 tidak bisa memberikan performa yang baik ketika melawan taktik skenario *Consecutive*. Hal ini diseBabkan oleh tingginya kriteria *fitness* yang harus dicapai oleh agen AI pada tiap pertarungan. Sebagaimana dijelasakan sebelumnya bahwa

BEP merupakan penentu atau *threshold* apakah performa agen AI dikatakan layak diberi *reward* atau pinalti. Dengan nilai BEP yang semakin tinggi, maka kualitas *script* yang dibentuk semakin baik sehingga meningkatkan efektifitas agen AI namun dampaknya adalah mengurangi jumlah *node rule* yang dapat dipilih karena tingginya kriteria *fitness*. Hal ini dapat kita lihat pada grafik BEP 0,6 yang selalu memiliki nilai *fitness* di atas 0,7 tetapi mengalami penurunan ketika melawan taktik skenario *Consecutive*. Penurunan tersebut menandakan kurangnya tingkat variasi pemilihan *node rule* pada *tree* HDS yang disebabkan tidak seimbangnya persebaran nilai bobot *node rule* sehingga agen AI tidak bisa beradaptasi dengan perubahan taktik pada taktik skenario *Consecutive*. Sementara itu, grafik BEP 0,5 memiliki kestabilan paling tinggi daripada kedua grafik BEP lainnya dengan kisaran nilai rata-rata *fitness* antara 0,7 dan 0,75. Dapat dikatakan bahwa agen AI HDS BEP 0,5 mampu mengatasi semua taktik skenario secara konsisten. Agen HDS BEP 0,5 memiliki tingkat efektifitas yang cukup tinggi(rata-rata *fitness* sebesar 0,7) pada semua taktik skenario diseBabkan oleh kriteria *fitness* yang tidak terlalu tinggi maupun rendah yakni 0,5. Berdasarkan hasil pengujian efektifitas, disimpulkan bahwa dengan menggunakn nilai *break even point* 0.5, agen AI HDS dapat menunjukkan efektifitas yang tinggi dan stabil saat melawan semua taktik skenario dalam penelitian ini.

Grafik pada Gambar 4 menunjukkan tingkat efisiensi agen AI HDS dengan melihat nilai *turn point*(TP) pada tiap taktik skenario. Terlihat bahwa TP agen AI dengan BEP 0,6 mengalami peningkatan TP pada taktik skenario *Consecutive* yakni 45. Hal ini menandakan bahwa BEP 0,6 baru bisa mengungguli taktik skenario Consecutive pada rata-rata simulasi pertarungan ke-45.



Gambar 4. Hasil Pengujian Efisiensi

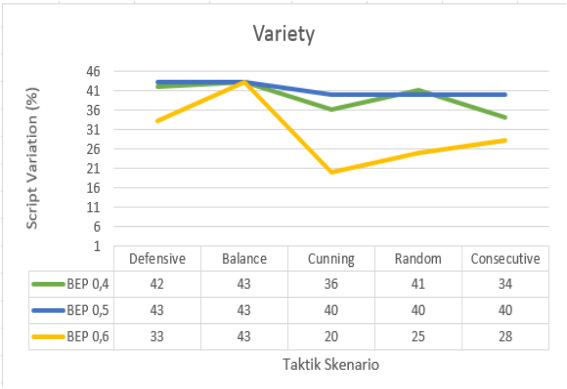
Sementara agen AI dengan BEP 0,5 menunjukkan efisiensi yang sangat tinggi dengan mencapai TP di bawah 10 untuk semua taktik skenario. Secara keseluruhan, agen AI HDS memerlukan kurang dari setengah total simulasi(di bawah 75 simulasi pertarungan) untuk dapat mengungguli agen AI statis. Hal ini diseBabkan oleh pengkategorian *node rule* menjadi beberapa *high level behaviour*. Sebagai contoh, suatu *script* dapat terdiri dari kombinasi *behaviour Survival* dan *Offensive* sehingga memungkinkan agen AI untuk melakukan pertahanan diri dan menyerang. Hal ini mengurangi efek *randomness* yang tinggi pada proses pembentukan *script* sehingga mempercepat proses *learning* agen AI HDS.

Selain itu, nilai BEP juga mempengaruhi TP yang dicapai agen AI. Semakin tinggi nilai BEP, maka proses evaluasi *node rule* semakin ketat sehingga kandidat *node rule* yang dapat dipilih menjadi bagian dari *script* menjadi sedikit. Kelebihan dari nilai BEP yang tinggi adalah mampu membentuk *script* dengan tingkat efektifitas yang tinggi sehingga sangat baik dalam melawan taktik skenario yang tidak berubah seperti *Defensive*, *Balance*, dan *Cunning*. Namun, nilai BEP yang tinggi menjadi masalah ketika melawan taktik skenario yang berubah-ubah seperti *Random* dan *Consecutive*. Hal ini dikarenakan agen AI harus beradaptasi ulang dengan perubahan taktik yang dilakukan agen AI statis. Dengan kandidat *node rule* yang semakin sedikit, proses adaptasi akan memakan waktu yang lama.

Grafik pada Gambar 5 menunjukkan tingkat variasi *script* yang dibentuk pada tiap simulasi pertarungan untuk semua taktik skenario. Pada Gambar 5 terlihat bahwa grafik BEP 0,6 mengalami fluktuasi yang besar dengan kisaran nilai variasi *script* antara 0,2 dan 0,4. Agen AI HDS BEP 0,6 mengalami penurunan variasi *script* yakni sebesar 0,2 saat melawan taktik skenario *Cunning* namun memiliki nilai *fitness* yang tinggi(Gambar 3) saat melawan taktik skenario tersebut. Pada pertarungan melawan taktik skenario *Consecutive*, agen AI HDS BEP 0,6 memiliki nilai variasi *script* yang rendah yaitu sebesar 0,28 dan nilai *fitness* yang rendah pula(Gambar 6.1). Hal ini menandakan bahwa tingkat variasi *script* sangat berpengaruh saat melawan taktik skenario *Consecutive* dan tidak terlalu berpengaruh pada skenario tanpa perubahan taktik.

Grafik BEP 0,4 mengalami fluktuasi

dengan kisaran nilai variasi *script* antara 0,34 dan 0,43. Agen AI HDS BEP 0,4 juga mengalami penurunan ketika melawan taktik skenario *Cunning* yaitu dengan nilai variasi sebesar 0,36. Akan tetapi, agen AI HDS BEP 0,4 memiliki nilai *fitness* yang tinggi(Gambar 6.1) saat melawan taktik skenario *Cunning*. Pada pertarungan melawan taktik skenario *Consecutive*, agen AI HDS BEP 0,4 memiliki nilai variasi *script* yang rendah dibanding taktik skenario lainnya yaitu sebesar 0,34 dan nilai *fitness* yang rendah pula(Gambar 3). Sementara itu, agen AI HDS BEP 0,5 menunjukkan nilai variasi yang stabil dengan kisaran nilai antara 0,4 dan 0,43. Agen AI BEP 0,4 tetap menghasilkan variasi *script* pada nilai 0,4 ketika melawan taktik skenario *Cunning* dan *Consecutive.*



Gambar 5. Hasil Pengujian Variasi *Script*

Secara keseluruhan, nilai variasi ketiga BEP di bawah 50% yang berarti jumlah variasi *script* yang dibentuk sekitar 60 jenis dari 150 simulasi pertarungan. Oleh karena itu dapat dikatakan bahwa agen AI HDS memiliki tingkat variasi relatif rendah. Berdasarkan hasil pengujian dapat disimpulkan bahwa nilai variasi *script* dapat disebabkan oleh beberapa faktor. Faktor pertama adalah tingginya nilai BEP menyebabkan proses evaluasi *script*, yang terdiri dari kombinasi *node rule*, hanya memberikan *reward* pada *script* dengan nilai *fitness* di atas 0,6. Hal ini menyebabkan persebaran nilai bobot *node rule* tidak seimbang karena hanya pasangan *node rule* tertentu yang memiliki nilai bobot tinggi. Faktor kedua adalah kurangnya *node rule* yang terdapat dalam blok kategori *behaviour*. Minimnya jumlah *node rule* pada blok kategori *behaviour* dapat mengurangi kombinasi *node rule* pada blok kategori *behaviour* tersebut dan kombinasi *node rule* secara keseluruhan dalam *script*. Faktor ketiga adalah kualitas dari tiap *rule* dalam *rulebase.* Kualitas dari *rule* yang tidak merata akan mengurangi banyak *node rule* yang dapat dipilih untuk membentuk *script*. Hal ini diseBabkan metode HDS secara alami memiliki probabilitas yang tinggi untuk memilih *node rule* dengan nilai bobot yang baik. Pada penelitian ini, *node rule* pada *rulebase Wizard* dengan *id* 4 dan 15 jarang dipilih untuk membentuk *script*. *Node rule* tersebut menyimpan *rule* untuk mengeluarkan *skill Eclipse* kepada agen AI lawan ketika dalam kondisi HP rendah. *Skill Eclipse* memberikan peluang *miss* pada serangan lawan sebesar 15% pada dua *turn* berikutnya. Namun, pada simulasi pertarungan yang dilakukan, efek *blind* jarang aktif dikarenakan peluang terjadinya rendah yakni sebesar 15% sehingga aksi agen AI *Wizard* menjadi sia-sia dan mempersulit keadaan tim HDS. Oleh karena itu, *node rule* ini seringkali mendapat pinalti.

6. KESIMPULAN

Berdasarkan hasil penelitian yang telah dilakukan, maka dapat disimpulkan beberapa poin sebagai berikut:

1. Metode *Hierarchial Dynamic Scripting*(HDS) diimplementasikan menjadi sebuah modul atau komponen dalam *game engine* Unity. Komponen ini nantinya akan dipasang pada *GameObject* yang merepresentasikan agen AI. Komponen HDS memiliki akses terhadap *rulebase* yang dalam penelitian ini disimpan dalam sebuah file *.txt*. Komponen HDS menyimpan *rulebase* ini ke dalam sebuah *tree* dengan mengelompokkannya berdasarkan kategori *rule* di dalamnya. Komponen HDS memiliki tiga tahap utama yaitu *Script Generation*, *Perform*, dan *Evaluation*. Tahap *Script Generation* merupakan tahap untuk melakukan *generate script* yang digunakan oleh agen AI. *Perform* adalah tahap untuk mengeksekusi *script* dan digunakan oleh agen AI untuk menentukan aksinya. Tahap evaluasi merupakan tahap penilaian performa agen AI dan pembaharuan nilai bobot tiap *node rule* dalam *tree* HDS.
2. Agen AI dengan menggunakan metode *Hierarchial Dynamic Scripting* hanya mampu memenuhi dua dari tiga parameter

pengujian adaptifitas yaitu efektifitas dan efisiensi. Agen AI dengan menggunakan *Hierarchial Dynamic Scripting* memenuhi kriteria efektif ditunjukkan dengan nilai rata-rata *fitness* sebesar 0,73 mengungguli nilai rata-rata fitness dari agen AI statis. Agen AI dengan menggunakan *Hierarchial Dynamic Scripting* memenuhi kriteria efisien ditunjukkan dengan nilai rata-rata *turn point* sebesar 6 yang berarti bahwa tim HDS mampu mengungguli semua taktik skenario rata-rata pada pertarungan ke-6 dari 150 simulasi pertarungan. Hal ini menandakan cepatnya proses adaptasi yang dilakukan agen AI HDS dalam melawan semua taktik skenario. Sementara itu, agen AI memiliki variasi yang tergolong rendah karena hanya menghasilkan nilai variasi rata-rata sebesar 39% atau dapat dikatakan bahwa agen AI rata-rata menggunakan 59 *script* yang berbeda dari 150 simulasi pertarungan. Rendahnya nilai variasi dipengaruhi oleh kualitas dan kuantitas dari *rule* dalam setiap *rulebase*. Dengan kualitas *rule* yang merata serta banyaknya opsi *rule* dalam *rulebase*, metode *Hierarchial Dynamic Scripting* mampu mencapai nilai variasi yang tinggi. Sementara itu, dari hasil pengujian nilai *break even point* didapatkan bahwa nilai sebesar 0,5 dapat menstabilkan tingkat efektifitas, efisiensi, dan variasi dari agen AI. Dengan nilai *break even point* sebesar 0.5, agen AI dapat menghasilkan nilai *fitness* yang tinggi dan stabil ketika melawan semua taktik skenario. Sementara itu, nilai variasi dan efisiensi yang dihasilkan ketika melawan semua taktik skenario bisa dikatakan tidak mengalami fluktuasi yang tinggi.

7. DAFTAR PUSTAKA

Abdullah, F., 2013. *Implementasi Fighting System Menggunakanan Dynamic Scripting dan Macro Action.* Malang: Universitas Brawijaya.

Kop, R., Toubman, A., Hoogendoorn, M. & Roessingh, J., 2015. Evolutionary Dynamic Scripting: Adaptation of Expert Rule Bases for Serious Games.

Muñoz-Avila, H. et al., 2013. Learning and game AI. *Dagstuhl Follow-Ups,* Volume VI.

Price, 2011. Effective Team Strategies using Dynamic Scripting.

Wang, D. & Tan, A., 2015. Creating Autonomous Adaptive Agents In A Real-Time First-Person Shooter Computer Game. *IEEE Transactions on Computational Intelligence and AI in Games,* VII(2), pp. 123-138.