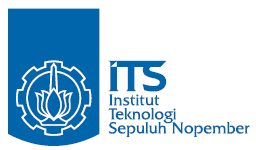
**JURUSAN TEKNIK INFORMATIKA**

**FAKULTAS TEKNOLOGI INFORMASI**

**INSTITUT TEKNOLOGI SEPULUH NOPEMBER**

**USULAN TUGAS AKHIR**

**1. IDENTITAS PENGUSUL**

Nama : Mubdi Nur Hafidh

NRP : 5105100086

Dosen Wali : Sarwosri, S.Kom, MT

**2. JUDUL TUGAS AKHIR**

**Penerapan Kecerdasan Buatan Adaptif pada Permainan Strategi Waktu Nyata Section 52 menggunakan Silverlight**

**3. LATAR BELAKANG**

Dalam permainan strategi waktu nyata terdapat kecerdasan buatan yang menggerakkan suatu unit, tugas kecerdasan buatan ini mengatur strategi untuk memberikan perlawan kepada pemain. Selain pengaturan strategi kecerdasan buatan juga mengatur jalannya pergerakan suatu unit dari suatu titik menuju ke titik tujan. Suatu unit dalam melakukan pergerakan menuju titik tujuan akan melakukan *pathfinding* (pencarian jalur) sehingga unit tersebut mampu mencapai titik tujuan dengan benar.

Untuk permainan yang kompleks seperti permainan strategi penggabungan kecerdasan buatan sulit untuk dilakukan dikarenakan banyaknya pilihan ditiap putaran permainannya. Untuk mengimplementasikan kecerdasan buatan pada permainan yang kompleks, kebanyakan *developer* menggunakan cara *script*, yaitu list dari *rule-rule* yang ada dijalankan secara berurutan. *Script* merupakan teknik pilihan pada industri permainan untuk mengimplementasikan kecerdasan buatan permainan, dikarenakan sifatnya yang dapat dimengerti, dapat diprediksi, mampu beradaptasi terhadap kondisi tertentu, mudah di implementasikan, dikembangkan dengan mudah, dan dapat digunakan oleh non-programer.

*Script* biasanya statis dan cenderung panjang serta kompleks, yang mengakibatkan dua masalah. Pertama karena kompleksitasnya, *script* kemungkinan akan mengandung kelemahan yang dapat dimanfaatkan oleh pemain-manusia untuk mengalahkan musuh yang seharusnya memiliki tingkat kesulitan yang tinggi dengan mudah. Kedua karena sifatnya yang statis, *script* tidak dapat menangani strategy pemain-manusia yang tidak tampak, dan tidak dapat mengukur tingkat kesuliatan yang disajikan oleh kecerdasan buatan permainan kepada pemain-manusia baik yang masih pemula dan yang mahir.

Kecerdasan buatan yang terlalu kuat atau terlalu lemah dapat membuat suatu permainan strategi menjadi tidak menarik. Padahal salah satu nilai hiburan pada sebuah permainan yang dapat membuatnya menjadi menarik adalah perlawanan yang diberikan oleh kecerdasan buatan dari unit musuh yang digerakkan oleh komputer. Semakin seimbang perlawanan yang diberikan oleh kecerdasan buatan akan membuat pemain merasa tertarik untuk memainkan.

Tugas akhir ini membahas mengenai implementasi kecerdasan buatan adaptif permainan dalam sistem pertempuran pada permainan strategi waktu nyata Section 52 untuk mengendalikan pergerakan unit yang dikendalikan oleh komputer. Sehingga unit yang dikendalikan oleh komputer dapat menyesuaikan diri susuai dengan keadaan pertempuran yang sedang berlangsung saat itu. Karena kecerdasan buatan yang dapat berkembang sesuai cara bermain pemainnya dapat membuat suatu permainan menjadi menarik. Dalam implementasinya Section 52 akan menggunakan *silverlight* sebagai *framework* pembuatan permainan. *Silverlight* merupakan *framework* aplikasi untuk menulis dan menjalankan *rich internet application*, dengan tujuan dan fitur yang sama seperti *flash*.

*Silverlight* sendiri dipilih sebagai *framework* pembuatan permainan karena permainan berbasis web sudah mulai berkembang dan memiliki peminatnya tersendiri serta masih jarangnya *silverlight* digunakan untuk pembuatan permainan. Selain itu nantinya diharapkan dapat megeksplorasi kemampuan-kemampuan yang dimiliki oleh *silverlight* sehingga mampu membuat permainan berbasis web yang menarik.

**4. RUMUSAN MASALAH**

Permasalahan yang diangkatdalam tugas akhir ini adalah sebagai berikut:

* Bagaimana merancang dan mengaplikasikan kecerdasan buatan adaptif permainan pada tiap unit agar dapat mengambil keputusan saat akan melakukan tindakan.
* Bagaimana menerapkan *pathfinding* pada pergerakan unit dari suatu titik menuju titik tujuan.
* Bagaimana membuat *battle system* yang bisa digunakan dalam strategi waktu nyata.
* Bagaimana membuat *battle system* tersebut bisa di uji tersendiri dan bisa berjalan dengan baik sewaktu digabung dengan modul utama.

**5. BATASAN MASALAH**

Ruang lingkup permasalahan yang dibahas dalam Tugas Akhir ini adalah sebagai berikut:

* Menggunakan A\* untuk *pathfinding*.
* Implementasi menggunakan *Silverlight 3*.
* Menggunakan *Dynamic Scripting* untuk kecerdasan buatan adaptif.

**6. TUJUAN**

Tugas akhir ini memiliki tujuan sebagai berikut:

* Mengimplementasikan kecerdasan buatan adaptif pada tiap unit sehingga dapat menyusun strategy sesuai situasi yang terjadi saat itu.
* Membuat tiap unit dapat bergerak menuju titik yang telah ditentukan dengan benar.
* Membuat *battle system* strategi waktu nyata yang sesuai untuk permainan Section 52.
* Membuat modul *battle system* tersebut dapat diuji secara terpisah dan bisa berjalan baik sewaktu digabung dengan modul utama.

**7. RINGKASAN**

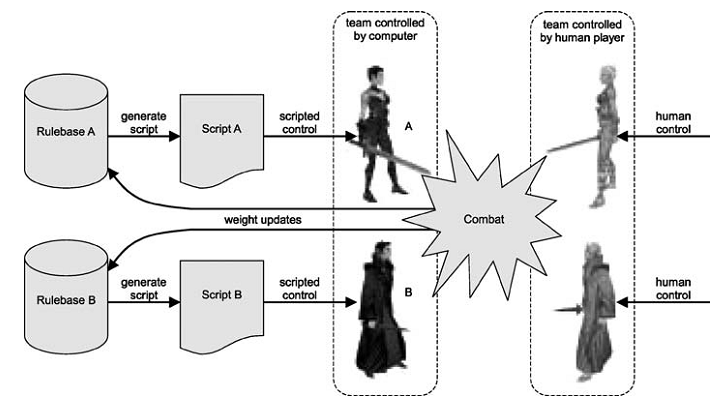
Kecerdasan buatan adaptif permainan merupakan kecerdasan buatan permainan yang dapat berkembang atau berubah – ubah sesuai dengan kondisi/situasi yang sedang terjadi, sehingga nantinya dapat memberikan perlawanan yang seimbang sesuai dengan pemainnya. Beberapa permainan komersial telah menerapkan kecerdasan buatan adaptif, seperti Max Payne 2, Black & White, Forza Motorsport, dan masih banyak yang lainnya. Max Payne 2 merupakan permainan berjenis aksi yang menekankan pada baku tembak dengan sudut pandang orang ke-tiga. Max Payne 2 memperkenalkan sesuatu yang disebut *dinamic difficulty setting* yang merupakan bentuk kecerdasan buatan adaptif, informasi yang ada pada dunia permainan di ekstrak untuk mengukur tingkat kemampuan pemain dan memberikan respon berupa penyesuaian tingkat kesuliatan musuh yang dihadapi oleh pemain.

Black & White merupakan permainan berjenis simulasi, kecerdasan buatan adaptif pada Black & White diterapkan pada suatu makhluk yang bisa dilatih oleh pemain. Makhluk pada permainan ini dilatih melalui sistem *reward* dan *punishment* menggunakan algoritma *reinforcement learning* yang seiring dengan jalannya permainan akan menentukan sifat makhluk yang dilatih apakah menjadi makhluk yang jahat atau baik.

Forza Mororsport merupakan permainan berjenis balap, semua musuh yang digerakkan komputer pada mode *arcade* dan *carrer* telah menerapkan kecerdasan buatan adaptif yang pada permainan ini disebut *Drivatar*. Drivatar bekerja dengan mempelajari tindakan pemain sewaktu melintasi lintasan, kebiasaan dalam mengemudi yang dilakukan pemain, dan strategi yang diterapkan pemain di tiap putaran balap. Dari hal-hal tadi dibuatlah kelakuan kendaraan yang dapat mengimbangi pemain. Selain mode *arcade* dan *carrer* terdapat juga suatu fitur *Train Drivatar* yaitu fitur untuk membuat tiruan pengemudi pemain. Di fitur ini drivatar akan mempelajari cara mengemudi pemain dalam beberapa putaran balap yang nantinya akan dibuat kelakuan kendaraanya semirip mungkin dengan cara mengemudi pemainnya mulai dari kebiasaan pengereman pada saat memasuki tikungan, kesalahan yang biasa dilakukan, kebiasaan pemilihan jalur pada lintasan balap, dan sebagainya sehingga benar – benar terbentuk pengemudi tiruan dari pemain.

Untuk menerapkan kecerdasan buatan adaptif permainan dapat dilakukan dengan beberapa teknik, selain dengan teknik yang digunakan pada permainan yang telah dibahas masih banyak teknik yang dapat digunakan salah satunya dengan *Dynamic Scripting*. *Dynamic scripting* merupakan sebuah teknik untuk memperoleh pembelajaran secara *online* oleh unit yang digerakkan komputer. Dalam *dynamic scripting*, *script* dibuat secara *online*, yaitu selama permainan berlangsung, berdasarkan *rule-rule* yang diekstrak dari sebuah *rulebase*. Pada teknik ini proses adaptasi berjalan dengan pemberian *reward* atau pemberian *penalty* suatu *rule-rule* sesuai dengan peranannya terhadap hasil yang diperoleh.[1]

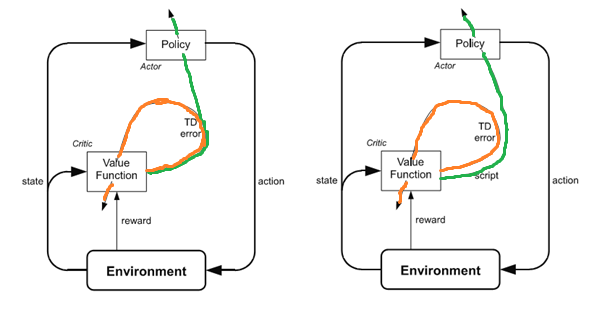
*Dynamic scripting* mengatur beberapa *rulebase*, dimana tiap unit memiliki sebuah *rulebase* tersendiri. Setiap sebuah unit baru dihasilkan, *rulebase-rulebase* digunakan untuk membuat *script* baru yang mengendalikan tindakan unit tersebut. *Rule-rule* yang meliputi sebuah *script* *controlling* sebuah unit tertentu diekstrak dari *rulebase* yang sesuai dengan unit tersebut. Kemungkinan suatu rule dipilih untuk sebuah *script* dipengaruhi oleh nilai *weight* yang tercantum pada tiap rule. Tujuan *dynamic scripting* ini adalah untuk mengadaptasi *weight* pada *rulebase* sehingga kesesuaian perilaku yang diharapkan didefinisikan oleh *script* yang dihasilkannya meningkat secara terus menerus, walaupun dalam situasi yang berubah-ubah. Kesesuaian biasanya didefinisikan sebagai kemungkinan suatu tim yang merupakan asal dari suatu unit memenangkan permainan. Adaptasi suatu *rulebase* berlangsung dengan mengubah nilai *weight* untuk menggambarkan peluang keberhasilan atau kegagalan suatu *rule* yang terkait dalam *script*. Perubahan *weight* sendiri ditentukan oleh *weight-update function*.[1]



Gambar 1. Ilustrasi Dynamic Sripting [2]

Gambar 1 menjelaskan mengenai terjadinya sebuah pertempuran menggunakan *dynamic scripting*. Diawal sebuah pertempuran, sebuah *script* baru dihasilkan untuk tiap unit yang dikendalikan komputer, secara acak memilih sejumlah *rule-rule* dari *rulebase* yang berkaitan. *Dynamic scripting* menggunakan sebuah mekanisme pemilihan, yaitu terdapat relasi linier antara kemungkinan suatu rule dipilih dan *weight*-nya yang berkaitan. Urutan *rule* mana yang ditempatkan pada *script* bergantung pada bagian aplikasi. Sebuah mekanisme prioritas dapat digunakan agar suatu *rule* diutamakan daripada *rule* lainnya. Mekanisme yang seperti ini hanya dibutuhkan bila urutan *rule-rule* dan tindakan dideskripsikan terlebih dahulu oleh pengetahuan bagian.[1]

Dalam penyelesaian suatu pertempuran, *weight-weight* dari *rule-rule* digunakan selama pertempuran diadaptasi sesuai kontribusinya terhadap hasil yang diperoleh. *Rule-rule* yang mengantarkan pada keberhasilan diberi *reward* dengan penambahan *weight*, seedangkan *rule-rule* yang mengantarkan pada kegagalan diberi *penalty* dengan pengurangan *weight*. Penambahan atau pengurangan setiap *weight* dikompensasikan untuk menjaga total *weight* konstan dengan cara mengurangi atau menambah semua sisa *weight*. Pendistribusian *weight-weight* inilah yang merupakan fitur penting dari *dynamic scripting*, karena membuat semua *rule* pada *rulebase* belajar di tiap *update*.[1]



Gambar 2.Perbandingan arsitektur actor-critic (kiri) dan arsitektur Dynamic scripting (kanan) [2]

Gambar 2 menjelaskan mengenai arsitektur pada *actor-critic method* dan *dymanic scripting*. *Dynamic scripting* sebenarnya berbasis pada *reinforcement learning* yang arsitekturnya mirip dengan arsitektur *actor-critic*. Serupa dengan *actor-critic method*, *dynamic scripting* merupakan sebuah *method* kontrol yang berdasar pada *policy* yang mengatur stuktur data terpisah untuk *acto*r, yang berupa *script* dan untuk *critic*, yang berupa *rulebase* dengan mekanisme penyesuaian *weight* yang diasosiasikan. Penyesuaian *weight* pada *dynamic scripting* ditentukan dengan sebuah *fitness function* yang menggabunagkan *state* dan sebuah *reward* atau *penalty* untuk performa tim di *environment*.

Perbedaan *dynamic scripting* dan *actor-critic* terletak pada dua cara, yaitu:

* *Dynamic scripting* melakuakan *update policy* dengan cara mengekstrak *rule* dari *rulebase*, sedangkan *actor-critic method* melakukan *update policy* secara langsung dari *TD error*.
* *Dynamic scripting* pada tugas akhir ini melakuakan *update* fungsi nilai sewaktu efek rangkaian tindakan dapat diukur, sedangkan *actor-critic method* biasanya melakukan *update* fungsi nilai setelah tiap tindakan. [1]

Karena *reinforcement learning* standar seperti *TD-learning* umumnya membutuhkan eksplorasi yang luas dan terus-menerus untuk menghasilkan tingkahlaku yang berhasil, hal ini membuat *reinforce learning* tidak efektif dibanding *dynamic scripting*.

Mulanya, *dynamic scripting* digunakan untuk membuat *script* pada tim musuh dalam *computer role-playing games* (CRPG). Dikarenakan adanya perbedaan antara *script* pada permainan CRPG dan strategi waktu nyata, dilakukanlah sbuah modifikasi terhadap algoritma *dynamic scripting* untuk sebuah permainan strategi waktu nyata yang pada intinya mengenai dua aspek sebagai berikut:

1. Pada CRPG *rule-rule* dibedakan berdasarkan perbedaan tipe musuh (misalnya: *warrior, wizzard*, dan sebagainya), pada strategi waktu nyata *rule-rule* dibedakan berdasarkan perbedaan *game states* (kondisi permainan yang berlangsung).
2. *Rule-rule* untuk kecerdasan buatan musuh diadaptasi saat sebuah terjadi perubahan *state* dan *rule-rule* dievaluasi berdasarkan *fitness* untuk *state* yang sebelumnya dan *fitness* untuk keseluruahn permainan. [2]

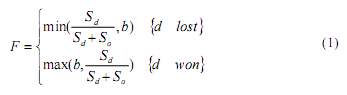


Gambar 3. Arsitektur umum Permainan Section 52

Gambar 3 menjelaskan mengenai arsitektur umum permainan section 52 dimana permainan dapat dilakuakn secara multiplayer. Perhitungan kecerdasan buatan dilakuakn terpusat pada server permainan, sehingga client tidak perlu melakukan perhitungan cukup dengan mengirimkan data yang dibutuhkan dalam perhitungan saja maka server yang akan melakukan perhitungan kecerdasan buatan dan mengirimkan hasil perhitungan kepada client. Perhitungan yang dilakukan meliputi pengecekan state menggunakan persamaan (1) dan persamaan (2). Setelah pengecekan *state* dilakukan akan ditentukan keputusan pergerakan yang akan diambil, setelah itu dilakukan pemberian skor menggunakan persamaan (3). Selanjutnya melakukan *update weight* yang nantinya digunakan dalam mengubah *state* dan membantu pengambilan keputusan menggunakan persamaan (4). Selain itu saat terjadi pergerakan unit dari suatu titik menuju titik tujuan akan dilakukan perhitungan pathfinding menggunakan persamaan (5).

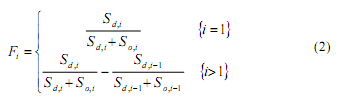
Berikut beberapa fungsi yang digunakan dalam *dynamic scripting*.

*Overall fitness function* *F* untuk pemain *d* yang dikendalikan *dynamic scripting* (untuk menyingkat disebut “pemain dinamis”) memiliki sebuah nilai dalam range [0,1]. Didefinisikan sebagai berikut:



Dalam persamaan (1), *Sd* merepresentasikan skor untuk pemain dinamis, *So* merepresentasikan skor untuk pemain dinamis milik musuh, dan *b* [0,1] merupakan *break-event point*. Pada *break event point,* *weight* tidak berubah.[2]

Untuk pemain dinamis, *state* *fitness* *Fi* untuk *state* *i* didefinisikan sebagai berikut:



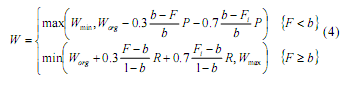
Dalam persamaan (2), *Sd,x* merepresentasikan skor pemain dinamis setelah state *x* dan *So,x* merepresentasikan skor pemain dinamis milik musuh setelah state *x*.[2]

*Score function* merupakan *domain-dependent*, dan harus mampu menggambarkan kekuatan relatif dari dua pemain yang berhadapan dalam permainan. Didefinisikan skor *Sx* untuk pemain *x* sebagai berikut:

*Sx* = 0.7*Mx* + 0.3*Bx* (3)

Dalam persamaan (3), *Mx* merepresentasikan *military point* untuk pemain *x*, yaitu banyaknya point yang diberikan untuk membunuh unit dan mengancurkan bangunan. *Bx* merepresentasikan *building point* untuk pemain *x*, yaitu banyaknya point yang diberikan untuk membuat pasukan dan membangun bangunan.[2]

*Weight-update function* mengubah *fitness function* menjadi *weight adaptation* untuk tiap *rule* dalam *script*. *Weigh–update function* *W* untuk pemain dinamis didefinisikan sebagai berikut:



Dalam persamaan (4), *W* merupakan nilai weight yang baru, *Worg* merupakan nilai weight asal, *P* merupakan *penalty* maksimum, *R* merupakan *reward* maksimum, *Wmax* merupakan nilai *weight* maksimum, *Wmin*merupakan nilai *weight* minimum, *F* merupakan *overall fitness* dari pemain dinamis, *Fi*merupakan *state fitness* untuk pemain dinamis pada *state i*, dan *b* merupakan *break-event point*.[2]

*Pathfinding* merupakan metode pencarian jalur(*plotting*)yang dilakuakn oleh aplikasi komputer. Dalam sebuah game metode *pathfinding* yang sering digunakan adalah A\*(dibaca “*A star*”). A\* menggunakan sebuah best-first search dan mencari jalur yang least-cost dari initial node menuju goal node. A\* menggunakan sebuah fungsi *distance-plus-cost heuristic* (biasanya dinotasikan dengan *f(n)*) untuk menentukan *node-node* yang dilalui dalam *tree*.[3]

*f(n)* = *g(n)* + *h(n)* (5)

* *-g(n)* merupakan biaya pergerakan untuk bergerak dari *node* asal ke *node n* yang dilalui.
* -*h(n)* merupakan biaya *heuristic* yaitu perkiraan biaya pergerakan dari *node n* ke tujuannya.

Untuk menentukan *h(n)* dapat dilakukan dengan berbagai cara, yaitu dengan *manhattan distance, diagonal distance, euclidian distance, euclidian distance squared, breaking ties, dan searching for an area*. Dikarenakan permainan Section 52 merupakan permainan 2D, cara manhattan distance sudah cukup untuk biaya *heuristic*-nya. Manhattan distance merupakan perkiraan untuk memperoleh heuristic yang digunakan untuk grid persegi dimana dapat melakuakn pergerakan ke empat arah. Manhattan dihitung dengan menjumlahkan banyaknya persegi yang dilalui secara horizontal dan vertikal, kemudian dikalaikan dengan skala yang sesuai dengan biaya pergerakan. [4]

*h(n) = D \* (abs(n.x-goal.x) + abs(n.y-goal.y))* (6)

* *D* merupakan skala yang sesuai dengan biaya pergerakan.
* *abs(n.x-goal.x)* merupakan jumlah persegi yang dilalui secara horizontal.
* *abs(n.y-goal.y)* merupakan jumlah persegi yang dilalui secara vertikal.

**8. METODOLOGI**

Metodologi yang digunakan dalam Tugas Akhir ini adalah sebagai berikut:

1. Studi Literatur

Tahap dimana pencarian dan pemahaman literatur, teori-teori yang berkaitan dengan rumusan masalah yang bisa digunakan untuk pembuatan permainan.

1. Perancangan perangkat lunak

Tahap dimana dilakukan perancangan terhadap hasil analisis dari studi literatur yang telah dilakukan.

1. Pembuatan perangkat lunak

Pada tahap ini dilakukan implementasi dari rancangan yang telah dibuat menjadi perangkat lunak.

1. Uji coba dan evlauasi

Tahap untuk melakukan uji coba terhadap perangkat lunak yang telah dibuat untuk mengetahui masalah - masalah yang muncul dan melakukan perbaikan apabila hasil yang dinilai tidak sesuai dengan tujuan.

1. Penyusunan laporan tugas akhir

Tahap akhir yaitu menyusun laporan yang berisi dokumentasi mulai dari dasar teori, proses yang telah dilakukan, dan hasil-hasil yang telah didapatkan selama pengerjaan tugas akhir.

Sistematika laporannya sebagai berikut:

BAB I. PENDAHULUAN

Bab ini berisi latar belakang masalah, tujuan pembuatan tugas akhir, permasalahan, batasan masalah, metodologi yang digunakan dan sistematika penyusunan tugas akhir.

BAB II. TINJAUAN PUSTAKA

Bab ini membahas beberapa teori penunjang yang berhubungan dengan pokok pembahasan dan mendasari pembuatan tugas akhir ini.

BAB III. METODOLOGI PENGEMBANGAN PERANGKAT LUNAK

Bab ini membahas secara detail tentang analisis sistem, desain dan perancangan peragkat lunak serta implementasi perangkat lunak dan disertai dengan potongan source code yang penting dalam sistem tersebut.

BAB IV. UJI COBA DAN EVALUASI

Bab ini membahas uji coba dari aplikasi yang dibuat dengan melihat output yang dihasilkan oleh aplikasi, dan evaluasi untuk mengetahui kemampuan aplikasi.

BAB V. PENUTUP

Bab ini berisi kesimpulan dari hasil uji coba yang dilakukan serta saran untuk pengembangan aplikasi selanjutnya.

**9. JADWAL PEMBUATAN TUGAS AKHIR**

Tugas akhir ini dikerjakan menurut jadwal sebagai berikut:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| No. | Tahapan | Bulan | | | | | |
| Maret | April | | Mei | | Juni |
| 1 | Studi literatur |  |  |  |  |  |  |
| 2 | Analisis dan Desain Perangkat Lunak |  |  |  |  |  |  |
| 3 | Implementasi Perangkat Lunak |  |  |  |  |  |  |
| 4 | Uji Coba dan Evaluasi |  |  |  |  |  |  |
| 5 | Penyusunan Buku Tugas Akhir |  |  |  |  |  |  |

**10. DAFTAR ACUAN**

[1] Ponsen Marc, Spronck Pieter, Kuyper Ida Sprinkhuizen, Postma Eric “**Adaptive game AI with dynamic scripting**”, SpringerScience

[2] Ponsen Marc, Spronck Pieter, “**IMPROVING ADAPTIVE GAME AI WITH EVOLUTIONARY LEARNING**”, Lehigh University, USA, Maastricht University, The Netherlands

[3] Hart, P. E.; Nilsson, N. J.; Raphael, B. (1968). "**A Formal Basis for the Heuristic Determination of Minimum Cost Paths**"

[4] <http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html> **.** (diakses maret 2012). **“Heuristics for grid maps from Amit’s thoughts on pathfinding”**