Pembuatan *Game* dengan Menerapkan Metode *Decision Tree: UCB1,* untuk Menentukan Pemilihan *Strategy* dalam *AI.*

Erick Santoso.[1], Gregorius Satia Budhi M,T,[2], Rolly Intan Dr. Eng,[3]
Program Studi Teknik Informatika Fakultas Teknologi Industri Universitas Kristen Petra
Jl. Siwalankerto 121 – 131 Surabaya 60236
Telp. (031) – 2983455, Fax. (031) – 8417658
Email: m26412071@john.petra.ac.id[1], greg@petra.ac.id[2], rintan@petra.ac.id[3]

ABSTRAK
Dikarenakan seiring bertambahnya teknologi dalam dunia *game*, penelitian ini akan menggunakan metode AI pendukung, yaitu UCB1 untuk di aplikasikan kepada suatu *game*.UCB1 akan digunakan untuk perpindahan level *enemy* yang dapat beradaptasi terhadap *player*. Sehingga AI dapat mengimbangi permainan dari *player*. UCB1 bekerja dengan cara mengkomparasi hasil perhitungan beberapa *state*. Perhitungan tersebut juga berasal dari data *history* atau data hasil permainan sebelumnya untuk dijadikan patokan pemilihan keputusan dikedepannya.

Pengujian akan dilakukan dengan menerapkan metode UCB1 kedalam perpindahan level *enemy* dengan tambahan beberapa rule. Penambahan rule membuat perpindahan dari level tersebut lebih fleksibel dan lebih cepat.

Hasil Pengujian menunjukkan bahwa UCB1 dapat bekerja dengan cukup baik bila diterapkan kedalam perpindahan level kesusahan dari *enemy*. UCB1 dapat membuat level dari *enemy* berpindah sesuai dengan tingkat kesulitan dari *player*.

Kata Kunci: *Game, AI, Unity, Decision Tree*

ABSTRACT
So that the aim of this research is to testing how good is UCB1 method when implemented into the main method that the game works.

The test will test the UCB1 method work as the movement of enemy diffilcuty level. But with some rules that make this method become more flexible to use. So the enemy diffilcuty will adapt to player diffilcuties.

The result of this research is the UCB1 method can be use to enemy diffilcuty level movement. This method can make the enemy become more adapt to the player.

Keywords*:Game, AI, Unity, Decision Tree*

1. PENDAHULUAN
Seiring dengan perkembangan jaman dan kemajuan teknologi yang pesat membuat permainan menjadi lebih bervariasi. Saat ini, permainan atau *game* tidak hanya dapat dinikmati oleh anak-anak, tetapi juga oleh orang dewasa, bahkan tidak sedikit orang yang memanfaatkan *game* untuk bisnis. Semakin berkembangnya teknologi membuat orang semakin banyak menciptakan *game* yang dapat dikonsumsi pada berbagai *Console*, seperti *Playstation*, *Xbox*, hingga Komputer. Penelitian kali ini mencoba mengoptimalkan *AI* dalam *Real Time Games* dengan cara membuat *AI* lebih fleksibel terhadap situasi dan kondisi *player*. Dalam hal ini, *AI* dapat beradaptasi dengan tingkat permainan dari *player*. Dengan dilaksanakannya penelitian ini, harapannya akan dapat diketahui seberapa baik metode *UCB1* bila diterapkan dalam *Real Time Games* untuk pemilihan Strategy *AI* dalam menghadapi *player*.

2. LANDASAN TEORI
2.1. UCB1
UCB1 merupakan salah satu metode yang mempunyai potensi tinggi dalam penerapannya di berbagai *game.* UCB1 memantain rata-rata perhitungan *payoff* yang didapat ketika memainkan *arm.* Perhitungan di lakukan dengan rumus sebagai berikut:

$$v(i) = x(i) + \sqrt{\frac{k \ln(t)}{c(i)}} \qquad (1)$$

Keterangan:
- v(i) [float]: *payoff* yang didapat dari hasil perhitungan tiap *arm*
- x(i) [real]: hasil dari penggunaan *arm*
- c(i) [real]: berapa kali *arm* tersebut dijalankan
- k [real]: bilangan konstanta
- t [real]: waktu ketika menjalankan UCB1

Bilangan konstanta diisi dengan menggunakan nilai *default* 2. Besar atau kecilnya nilai konstanta *(k)* akan berpengaruh terhadap hasil *payoff (v)*. Jika semakin besar nilai konstanta, maka perpindahan antar *arm* akan lebih cepat. Sebaliknya jika nilai konstanta semakin kecil, maka perpindahan antar *arm* akan lebih lama. Nilai *t* dalam perhitungan UCB1 akan selalu ditambah setiap kali perhitungan UCB1 dijalankan.

Tabel 1 Tabel Contoh Perhitungan UCB1

	Rock			Paper			Scissors		
Time	c(i)	x(i)	v(i)	c(i)	x(i)	v(i)	c(i)	x(i)	v(i)
0	**0**	**0**	∞	0	0	∞	0	0	∞
1	1	0	0.00	**0**	**0**	∞	0	0	∞
2	1	0	1.18	1	1	2.18	**0**	**0**	∞
3	1	0	1.48	**1**	**1**	**2.48**	1	−1	0.48
4	1	0	1.67	**2**	**1**	**2.18**	1	−1	0.67
5	1	0	1.79	**3**	**1**	**2.04**	1	−1	0.79
6	1	0	1.89	**4**	**1**	**1.95**	1	−1	0.89
7	**1**	**0**	**1.97**	5	1	1.88	1	−1	0.97

Pada Tabel 1 di simulasikan bahwa *Player* akan mendapatkan asumsi terbaik yaitu untuk Kertas dari Time ke-2 sampai ke-6. Untuk Time 1,2,3, UCB1 akan mencoba semua kemungkinan terlebih dahulu untuk mendapatkan hasil *payoff* pertama [8].

2.2. *Game*
Game adalah sebuah sistem dimana didalamnya *player* akan berhadapan dengan konflik buatan yang diatur berdasarkan rule tertentu dan akan mengeluarkan sebuah hasil yang dapat diukur

[6]. Sebuah *Game* harus memiliki *goal* yang harus dicapai *player* dan dalam mencapai *goal* tersebut *player* harus di libatkan dalam proses menghasilkan *goal* tersebut dimana *player* harus merasa memiliki kontrol terhadap apa yang akan terjadi di dalam *game* tersebut [7]**.** *NPC* juga harus dapat bertindak layaknya sifat dari karakter tersebut. Sehingga *NPC* juga harus dapat bertindak dan berperilaku lebih *real* terhadap *player* dan situasi [12].

Sebuah permainan adalah sebuah sistem dimana pemain terlibat dalam konflik buatan, di sini pemain berinteraksi dengan sistem dan konflik dalam permainan merupakan rekayasa atau buatan, dalam permainan terdapat peraturan yang bertujuan untuk membatasi perilaku pemain dan menentukan permainan [1].

Game berasal dari kata bahasa inggris yang berarti dasar permainan. Permainan dalam hal ini merujuk pada pengertian kelincahan intelektual (*Intellectual Playability Game*) yang juga bisa diartikan sebagai arena keputusan dan aksi pemainnya [2].

Berdasarkan genre permainannya**,** *game* dapat dibagi menjadi beberapa kategori [14], yaitu:

- *Action – Shooting*: video *game* jenis ini sangat memerlukan kecepatan refleks, koordinasi mata-tangan, juga *timing*.
- *Fighting* (pertarungan): Jenis ini memang memerlukan kecepatan refleks dan koordinasi mata-tangan, tetapi inti dari *game* ini adalah penguasaan permainan, pengenalan karakter dan *timing*.
- *Action – Adventure*: *Game* jenis ini sudah berkembang jauh hingga menjadi genre campuran *action beat-em up*, dan sekarang jenis ini cenderung untuk memiliki visual 3D dan sudut pandang orang ke tiga.
- *Adventure*: Pemain dalam bergerak, berlari, melompat hingga memecut atau menembak tidak diperlukan di sini. *Video Game* murni petualangan lebih menekankan pada jalan cerita dan kemampuan berpikir pemain.
- Simulasi Konstruksi dan manajemen: Video *Game* ini seringkali menggambarkan dunia di dalamnya sedekat mungkin dengan dunia nyata dan memperhatikan dengan detil berbagai faktor.
- *Role Playing*: Video *game* cenderung bermain peran, memiliki penekanan pada tokoh atau peran perwakilan pemain di dalam permainan.
- Strategi: Video *game* jenis strategi, layaknya bermain catur, justru lebih memerlukan keahlian berpikir dan memutuskan setiap gerakan secara hati-hati dan terencana. *Game* jenis ini terbagi atas:
  - *Real time Strategy, game* berjalan dalam waktu sebenarnya dan serentak antara semua pihak dan pemain harus memutuskan setiap langkah yang di ambil saat itu juga
  - *Turn based Strategy, game* yang berjalan secara bergiliran, saat kita mengambil keputusan dan menggerakan pasukan, saat itu pihak lawan menunggu, begitu pula sebaliknya.
- Puzzle: Video *game* jenis ini sesuai namanya berintikan mengenai pemecahan teka-teki, baik itu menyusun balok, menyamakan warna bola, memecahkan perhitungan matematika, melewati labirin, hingga mendorong-dorong kota masuk ke tempat yang seharusnya, itu semua termasuk dalam jenis ini.
- Simulasi kendaraan: Video *Game* jenis ini memberikan pengalaman atau interaktifitas sedekat mungkin dengan kendaraan yang aslinya, meskipun terkadang kendaraan tersebut masih eksperimen atau bahkan fiktif, tapi ada penekanan khusus pada detil dan pengalaman realistik menggunakan kendaraan tersebut.
- Olahraga*:* Singkat padat jelas, bermain sport di PC atau konsol. Permainan ini dibuat dengan tingkat realistis yang tinggi.

2.3. AI

AI harus mempunyai 3 macam *decision problem*, yaitu *action, outcomes, dan preferences*, dimana ketiganya tersebut akan mempengaruhi bagaimana *AI* akan bertindak terhadap *player* [11]. Kecerdasan Buatan (AI) merupakan sebuah studi tentang bagaimana membuat komputer melakukan hal-hal yang pada saat ini dapat dilakukan lebih baik oleh manusia [5]. Program AI pertama yang bekerja ditulis pada 1951 untuk menjalankan mesin Ferranti Mark I di University of Manchester (UK): sebuah program permainan naskah yang ditulis oleh Christopher Strachey dan program permainan catur yang ditulis oleh Dietrich Prinz. John McCarthy membuat istilah "Kecerdasan Buatan" pada konferensi pertama pada tahun 1956, selain itu dia juga menemukan bahasa pemrograman Lisp. Alan Turing memperkenalkan "*Turing test*" sebagai sebuah cara untuk mengoperasionalkan test perilaku cerdas. Joseph Weizenbaum membangun ELIZA, sebuah chatterbot yang menerapkan psikoterapi [4]. *Soft computing* mengeksploitasi adanya toleransi terhadap ketidaktepatan, ketidakpastian, dan kebenaran parsial untuk dapat diselesaikan dan dikendalikan dengan mudah agar sesuai dengan realita [13].

2.4. C#

C# adalah bahasa pemrograman baru yang diciptakan oleh Microsoft yang dikembangkan dibawah kepemimpinan Anders Hejlsberg yang telah menciptakan berbagai macam bahasa pemrograman termasuk Borland Turbo C++ dan orland Delphi [3]. C# juga merupakan sebuah bahasa pemrograman yang berorientasi objek yang dikembangkan oleh Microsoft sebagai bagian dari inisiatif kerangka .NET Framework [9].

2.5. Unity 3D

Unity 3D adalah sebuah *game* engine yang berbasis cross-platform. Unity dapat digunakan untuk membuat sebuah *game* yang bisa digunakan pada perangkat komputer, ponsel pintar android, iPhone, PS3, dan bahkan X-BOX. Unity adalah sebuah sebuah tool yang terintegrasi untuk membuat *game*, arsitektur bangunan dan simulasi. Unity bisa untuk *game*s PC dan *game*s Online. Untuk *game*s Online diperlukan sebuah plugin, yaitu Unity Web *Player*, sama halnya dengan Flash *Player* pada Browser [10].

3. DESAIN SYSTEM

Pada bab ketiga dari skripsi ini akan dijelaskan berbagai perumusan masalah meliputi:

- Desain *Game*
  - Desain *Game* Keseluruhan / Storyline
  - Desain *Game* Component
  - Desain *Player*
  - Desain *Enemy* (AI)
  - Desain *Game*play
- Desain Pergantian Level
  - Perhitungan UCB1 Dengan Penambahan Rules

- Implementasi UCB1 Terhadap AI
- Desain Interface
  - Menu
  - Playing

3.1 Desain *Game* Keseluruhan / Storyline

Game yang akan dibuat bernama "Shoot For Your Life". *Game* ini berceritakan tentang seorang tentara (*Player*) yang terjebak dalam sebuah kota, dan dalam kota tersebut penuh dengan banyak *Zombie* (*Enemy*). Dalam perjalanan untuk keluar dari kota tersebut, tentara (*Player*) akan dihadang oleh berbagai macam jenis *zombie* (*Enemy*) yang perlu dikalahkan untuk membuka ke jalan berikutnya. Jika selama permainan *player* kalah terhadap *enemy*, maka stage akan diulang kembali, namun perhitungan tetap dilanjutkan

3.2 Pembuatan Stage Component

Pembuatan *Stage Component*disusun dengan tema *broken city*. Sehingga component diletakkan dengan berserakan di jalan–jalan

3.3 Desain *Player*

Player dapat melakukan tembakan kearah *enemy*. Ketika *player* membidik kearah *enemy*, maka akan keluar *crosshair* yang dapat diarahkan kepada *enemy*. *player* selama permainan berlangsung. Komponen–komponen tersebut adalah:

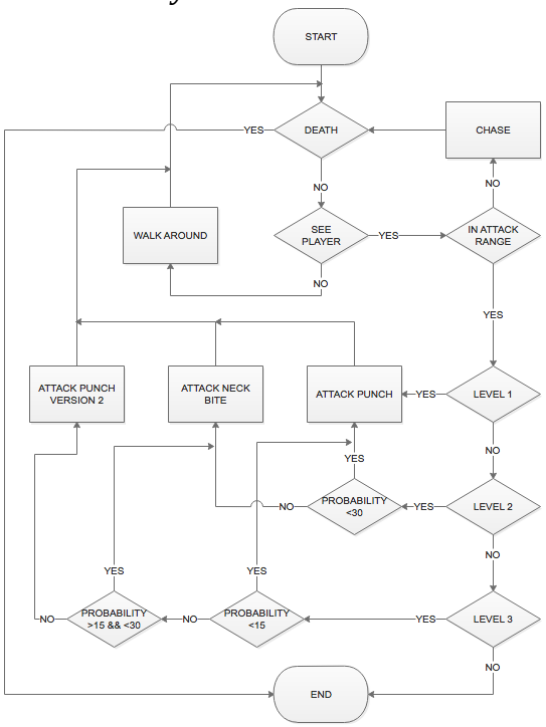
- *Life Point*: *Life point* yang dimiliki oleh *player* dalam bentuk angka.
- *Experience* (*Exp*): *Experience (exp)* di dalam *player* berguna untuk tingkat akurasi *player* dalam menembak *enemy*. *Exp* akan didapat dari setiap *enemy* yang telah dikalahkan atau dibunuh oleh *player*.
- Akurasi (*Accur*): Akurasi (*accur*) akan didapat dari setiap *exp* yang dikumpulkan. Setiap kali *exp* memenuhi jumlah tertentu, maka tingkat *accur* akan bertambah. Semakin tinggi tingkat *accur* yang dimiliki oleh *player*, maka *damage* yang diterima oleh *enemy* akan berpeluang semakin besar. Jumlah *Exp* yang dibutuhkan untuk meningkatkan level *Accur* adalah sebagai berikut:
  - *Accur* level 2: Membutuhkan 50 *exp*
  - *Accur* level 3: Membutuhkan 100 *exp*
  - *Accur* level 4: Membutuhkan 150 *exp*
  - *Accur* level 5: Membutuhkan 200 *exp*
  - *Accur* level 6: Membutuhkan 250 *exp*
  - *Accur* level 7: Membutuhkan 300 *exp*

  Accur dalam permainan berpengaruh terhadap seberapa tepat *player* menembak *enemy*. Berikut adalah presentase tingkat akurasi yang akan dimiliki oleh *player*:

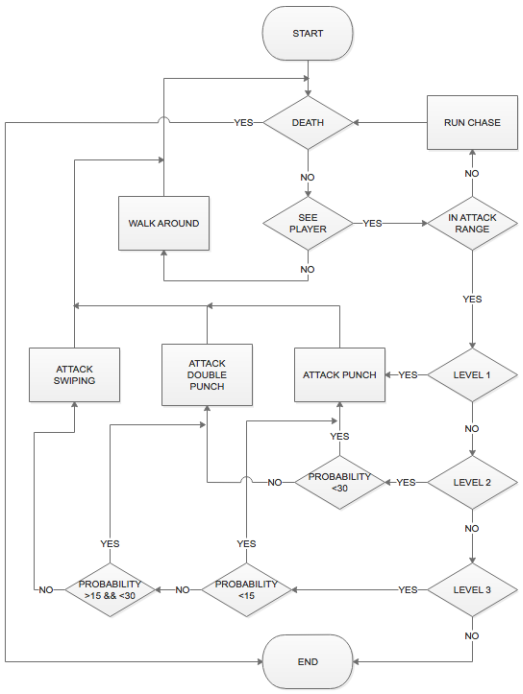
  - *Accur* level 1: Tingkat akurasi 30%, peluang *miss shot* 70%
  - *Accur* level 2: Tingkat akurasi 40%, peluang *miss shot* 60%
  - *Accur* level 3: Tingkat akurasi 50%, peluang *miss shot* 50%
  - *Accur* level 4: Tingkat akurasi 60%, peluang *miss shot* 40%
  - *Accur* level 5: Tingkat akurasi 70%, peluang *miss shot* 30%
  - *Accur* level 6: Tingkat akurasi 80%, peluang *miss shot* 20%
  - *Accur* level 7: Tingkat akurasi 90%, peluang *miss shot* 10%

- *Damage*: *Damage* yang dipunyai oleh *player* dalam bentuk angka. Damage awal dari *player* berkisar antara 9-15 per tembakan, dengan *accur* (akurasi) awal yaitu 30%. Dengan bertambahnya tingkat *accur* dari *player*, maka besar *accur* yang dimiliki oleh *player* juga bertambah.

3.4 Desain *Enemy* & AI



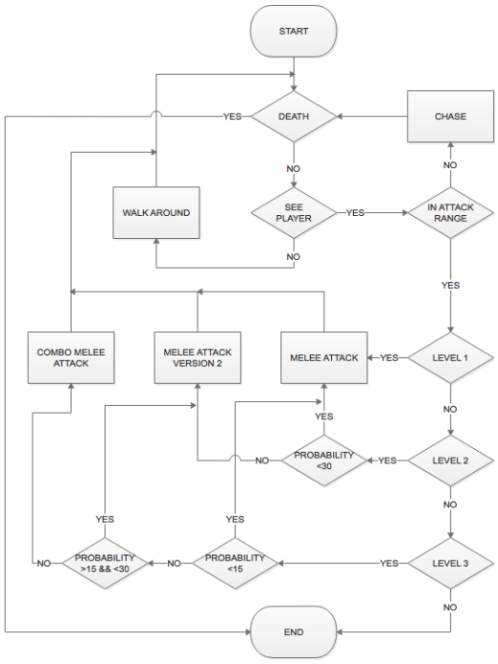
Gambar 1. *Flowchart enemy Puncher*



Gambar 2. *Flowchart enemy Catcher*

Pada Gambar 1 terlihat *state action* yang dimiliki oleh Puncher. Ketika Puncher melihat *player*, maka akan dicek apakah *player* berada dalam jarak serang dari Puncher, jika iya maka Puncher akan melakukan *attack*. Jika tidak dalam jarak serang, maka Puncher akan melakukan *state chase*. Selama Puncher berada dalam *state chase*, akan selalu dicek apakah *enemy* masih melihat *player* atau tidak

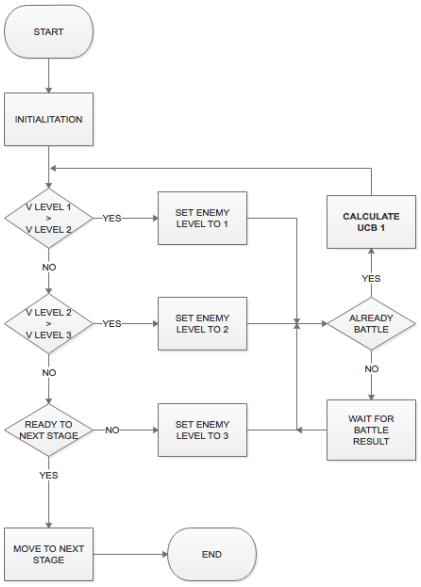
Pada Gambar 2 Catcher akan mulai bergerak atau mengejar ketika Catcher melihat *player*, kemudian akan dicek apakah *player* termasuk dalam jarak serang atau tidak. Jika tidak, maka Catcher akan masuk kedalam *state chase run*. Jika Catcher berada dalam jarak serang, maka Catcher akan melakukan serangan.



Gambar 3. *Flowchart enemy Tanker*

Pada Gambar 3 Tanker akan bergerak atau mengejar jika Tanker melihat *player*, dan kemudian akan dicek, apakah *player* berada dalam jarak serang atau tidak. Jika *player* berada dalam jarak serang, maka Tanker akan melakukan serangan (*attack*). Jika *player* tidak berada dalam jarak serang, maka Tanker akan masuk kedalam *state chase*.

3.5 Desain UCB1

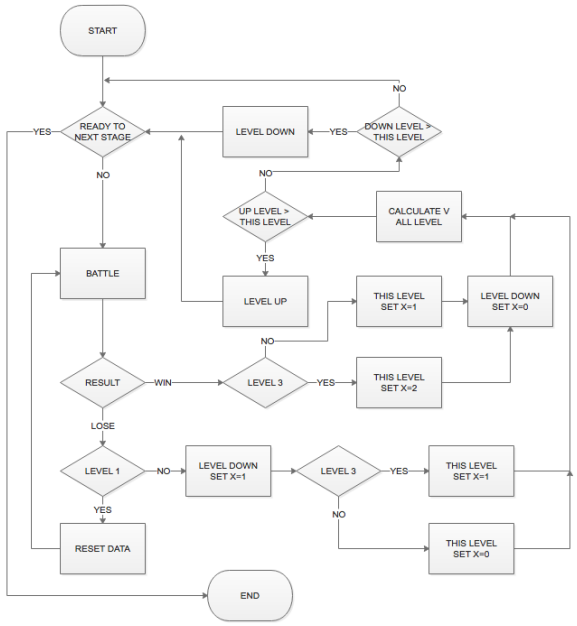


Gambar 4. Implementasi UCB1 Terhadap *Enemy*

Pada *game* ini, metode UCB1 dipakai didalam pergantian level yang diletakkan pada *enemy*. *Enemy* akan mempunyai level dan tingkatan yang berbeda. Semakin tinggi tingkatan dari *enemy*, maka semakin tinggi pula *life point*, *damage*, dan *speed*. Pada Gambar 4 akan terlihat implementasi UCB1 terhadap *enemy*.

3.6 Alur Perhitungan UCB1

Pada Gambar 5. *A*lur perhitungan UCB1 juga mengalami sedikit perubahan pada susunan perhitungannya. Metode UCB1 pada dasarnya adalah metode yang digunakan untuk penentuan keputusan, langkah apa yang terbaik untuk dilakukan.



Gambar 5. Alur Perhitungan UCB1

Sedangkan pada *game* ini, metode UCB1 digunakan untuk perpindahan level *enemy*. UCB1 akan menghitung probabilitas *Enemy* dalam perpindahan level. Rules–rules yang ditambahkan dalam metode UCB ini adalah sebagai berikut:

- Ketika permainan dimulai, maka *enemy* akan berada di level 1 dan di *stage* pertama.
- Selama *enemy* berada di level 1, maka kolom di level 3 tidak akan di isi (NULL). Ketika permainan dimulai, maka *enemy* akan berada di level 1. Ketika hasil perhitungan (*v*) level 2 lebih tinggi, maka level *enemy* akan naik ke level 2.
- Ketika permainan dimulai, maka level 2 di bagian *used* (c) akan langsung di set dengan 1. Dan pada bagian *result* (x) di level 2 diset 0. Pergantian berguna agar pergantian antara level 1 dan level 2 tidak terlalu lama.
- Ketika *enemy* berada pada level 2, maka perhitungan di level 3 dimulai. Nilai *used* (c) yang berada di level 3 akan diganti dengan 1. Nilai *result* (x) pada level 1 akan diganti dengan 0.
- Selama *enemy* belum pernah masuk ke level 3, maka *used* (x) pada level 3, akan di isi dengan 0. Tetapi ketika *enemy* sudah pernah berada dalam level 3, maka ketika *enemy* kembali (*player kalah*) ke level 2, maka nilai *result* (x) di level 3 akan diset 1.
- Ketika terjadi kesamaan perhitungan terhadap 2 level yang berbeda, maka *enemy* akan tetap berada pada level yang sebelumnya.

Ketika *enemy* berada pada level 3, maka nilai *used* (x) di level 3 akan berganti dengan 2. Dan nilai *used* (x) yang berada di level 1 akan diganti dengan -1. Hal ini diperlukan agar ketika komparasi nilai hasil perhitungan, tidak terjadi konflik antara level 1, level 2, dan level 3.

- Disimulasikan bahwa *player* telah mencapai *enemy* level 3, dan kemudian *player* kalah. *Player* tidak akan selalu langsung melawan *enemy* level 2, namun dapat mencoba kembali. Jika *player* kembali kalah, maka level dari *enemy* akan turun.

4. PENGUJIAN SYSTEM

Pada bab ini akan dijelaskan mengenai pengujian system dan pengujian metode yang digunakan. Pengujian yang dilakukan berupa pengujian terhadap metode UCB1 yang diterapkan kedalam AI. Pengujian metode UCB1 di uji kedalam beberapa situasi. Kemudian pada bab ini juga akan menguji *enemy* movement di *chasing state* dan *patrol state*. *Enemymovement* juga akan di uji ketika melakukan *attack* terhadap *player* yang dibedakan berdasarkan *level* dari *enemy* tersebut. Pada bab ini juga akan diuji peningkatan *level accuracy* dari *player*.

4.1. Pembuatan Model *Player*

Gambar 6. Desain *Player*

Pergerakan *player* pada Gambar 6. menggunakan *Character Controller*. Dengan menggunakan *Character Controller*, *player* akan secara otomatis dpat bergerak sesuai dengan setting yang berada di Unity. *Player* akan membawa dua jenis tembakan, yaitu *Pistol* dan *Rifle*. *Player* dapat memilih menggunakan senjata *pistol* atau *rifle*.

4.2. Pembuatan Model *Enemy* & Movement

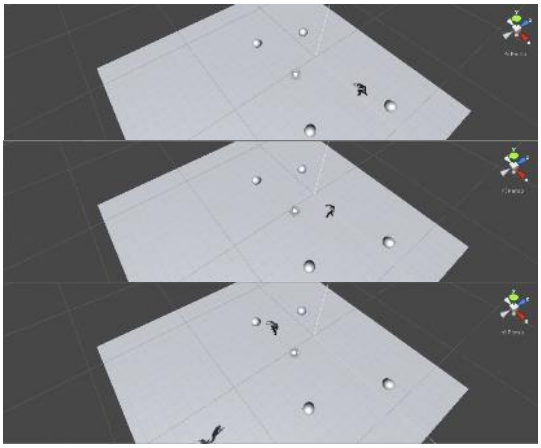
Pembuatan *Enemy Model* dibentuk sesuai dengan tipe dari *enemy* tersebut**.**

Gambar 7. Desain *Enemy* Puncher

Gambar 8. Desain *Enemy* Cathcer

Gambar 9. Desain *Enemy* Tanker

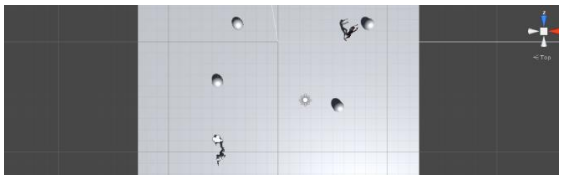
Pada Gambar 7 terlihat desain *enemy puncher*. Gambar 8 terlihat desain *enemy catcher*. Pada Gambar 9 terlihat desain *enemy* tanker. Enemy *movement* pada Gambar 10 dan Gambar 11, diterapkan kedalam permainan ini dibagi menjadi dua *state*, yaitu *state chasing* dan *state patrol*. Dalam permainan ini *waypoint* menggunakan *game Object* berupa *sphere* yang nantinya akan dihilangkan *material texture*-nya, agar tidak terlihat dalam permainan. Pemilihan *waypoint* diatur secara random agar *enemy* dapat bergerak fleksibel.



Gambar 10. *Enemy* Movement Patrol

Ketika *attack*, *enemy* mempunyai tiga tipe jenis serangan dengan damage yang berbeda – beda. Semakin tinggi level *enemy*, maka semakin banyak pula tipe serangan dari *enemy*. Ketika *enemy* berada pada level pertama, maka *enemy* akan sepenuhnya menjalankan jenis serangan pertama. Ketika *enemy* berada pada level kedua, maka *enemy* akan dapat menjalankan dua jenis serangan yang berbeda dengan presentase 80% untuk jenis serangan kedua, dan 20% jenis serangan pertama. Ketika *enemy* berada pada level ketiga, maka *enemy* dapat menjalankan tiga jenis serangan dengan presentase menjalankan serangan ketiga sebesar 70%, jenis serangan kedua dan pertama masing – masing mempunyai presentase sebesar 15%.

Gambar 11. *Enemy* Movement Chase



Gambar 12. Ketika *Enemy* Tidak melihat *Player*

Pada *state chasing*, *enemy* akan mengejar *player* jika *player* berada dalam jangka lihat dari *enemy* (*Line Of Sight*) yang terlihat pada Gambar 12. Apabila *player* keluar dari jarak pandang *enemy*, maka *enemy* akan menjalankan *state patrol*.

4.2. Pengujian Implementasi UCB1

Pengujian terhadap UCB1 dimulai dengan menginisialisasi nilai awal sesuai yang pada desain sistem, Pada Gambar 5.1 akan terlihat inisialisasi awal dari permainan dan perhitungan sama. Kemudian metode UCB1 tersebut disimulasikan menang selama enam kali. Kemudian hasil dari perhitungan akan dibandingkan dengan hasil perhitungan di level yang lainnya. Baris berwarna kuning menandakan adanya pergantian *level* musuh. Hal ini disebabkan karena hasil perhitungan (v) di *level* pertama lebih kecil daripada di *level* kedua. Perhitungan UCB1 baru akan dijalankan ketika variable *entry* menjadi *true*. Perubahan variable *entry* menjadi *true* disebabkan ketika *player* telah mengalahkan *enemy* atau *player* telah dikalahkan oleh *enemy*. Ketika menjalanakan perhitungan UCB1, akan dicek apakah *player* menang atau kalah dalam melawan *enemy*. Jika hasilnya *player* dapat mengalahkan *enemy*, maka variabel *last_result* akan diubah menjadi 1, jika *player* kalah melawan *enemy*, maka variabel *last_result* akan diubah menjadi 0. Selanjutnya akan dicek dimana level *enemy* sekarang dan akan dilakukan penggantian pada variabel *c*, *x*, dan *t*.

- *Player* menang terhadap *enemy*
  - Jika *enemy* sekarang berada di level 1, maka variabel *1c* akan ditambah 1, kemudian variabel *1x* akan diubah menjadi 1. *2x* dan *3x* akan diubah menjadi 0.
  - Jika *enemy* sekarang berada di level 2, maka variabel *2c* akan ditambah 1, kemudian variabel *2x* akan diubah menjadi 1. *1x* akan diubah menjadi 0. *3x* akan dicek apakah *enemy* pernah berada di level 3. Jika *enemy* pernah berada di level 3 maka *3x* akan diubah menjadi 1. Jika *enemy* tidak pernah berada di level 3 maka *3x* akan diubah menjadi 0.
  - Jika *enemy* sekarang berada di level 3, maka variabel *3c* akan ditambah 1, kemudian variabel *3x* akan diubah menjadi 2. *1x* dan *3x* akan diubah menjadi 0.

- *Player* kalah terhadap *enemy*
  - Jika *enemy* sekarang berada di level 1, maka variabel *1c* akan ditambah 1, kemudian variabel *1x* akan diubah menjadi 0. *2x* dan *3x* akan diubah menjadi 0.
  - Jika *enemy* sekarang berada di level 2, maka variabel *2c* akan ditambah 1, kemudian variabel *2x* akan diubah menjadi 0. *1x* dan *3x* akan diubah menjadi 0.
  - Jika *enemy* sekarang berada di level 3, maka variabel *3c* akan ditambah 1, kemudian variabel *3x* akan diubah menjadi 1. *1x* akan diubah menjadi 0, dan variable *2x* akan diubah menjadi 1.

Setelah merubah variabel-variabel dari setiap level. Maka akan dilakukan proses perhitungan UCB1 dengan memanggil fungsi *calculatev*. Hasil dari perhitungan UCB1 tersebut akan dimasukkan kedalam variabel 1v (level 1), 2v (level 2), dan 3v (level 3). Kemudian akan dikomparasi hasil dari perhitungan tersebut.

5. Kesimpulan dan saran

5.1 Kesimpulan

Berdasarkan hasil dari pengujian, dapat disimpulan beberapa hal sebagai berikut: Metode UCB1 dapat digunakan sebagai metode untuk perpindahan level, namun penggunaan metode UCB1 bila diterapkan sebagai metode utama membutuhkan beberapa *rules* tambahan agar dapat bekerja secara maksimal

Penggunaan metode UCB1 membutuhkan modifikasi agar dapat digunakan dengan baik. Inisialisasi awal dibutuhkan agar metode dapat berjalan dengan maksimal. Reset dibutuhkan ketika *enemy* kembali ke level pertama, dikarenakan akan membutuhkan waktu yang banyak agar dapat naik ke level berikutnya

5.2 Saran

Setelah mengevaluasi seluruh skripsi ini, berikut beberapa saran yang saya berikan adalah banyak metode–metode yang dapat digunakan sebagai metode yang dapat beradaptasi. UCB1 merupakan metode yang cukup dapat beradaptasi. Sehingga UCB1 membutuhkan pengembangan lebih lanjut agar dapat digunakan secara maksimal

Sedikitnya metode yang dapat digunakan untuk beradapatasi dengan *player*. Sehingga harus membuat suatu metode lagi ataupun memodifikasi metode baru agar dapat digunakan untuk membuat metode yang dapat beradaptasi

6. DAFTAR PUSTAKA

[1] Avedon, E. M. and Smith, B. S.. 1971. "The Structural Elements of *Game*s.". The Study of *Game*s. New York: Ishi Press, 419-426

[2] Dill, Kevin. 2014. "What is *game* AI?". *Game* AI Pro. New York: CRC Press, 3-10.

[3] Hejlsberg, Anders. 2004. The C# Programming Languange 2nd Edition. Addison-Wesley Professional. New York

[4] Negnevitsky, M. 2011. Artificial Intelligence A Guide to Intelligent Systems 3rd Edition. Kanada: Pearson Education Canada.

[5] Rich and Knight 1991. Artificial Intelegence. Hill Publishing Company Limited.

[6] Salen, K. & Zimmerman, E. 2003. Rules of Play: *Game* Design Fundamentals. The MIT Press.

[7] Schwab, B. 2014. AI postmortem: Hearthstone. Lecture, *Game* Developer's Conference AI Summit 2014, San Francisco, CA.

[8] Sturtevant, R.N. 2015. "Monte Carlo Tree Search and Related Algorithms for *Game*s.". *Game* AI Pro 2. New York: CRC Press, 296-311.

[9] Stellman, Andrew 2007. Head First C#. O'Reilly Media. Newston, US

[10] Suvak, J. 2014.Learn Unity 3D Programming with UnityScript: Unity's JavaScript for Beginners. New York: Apress.

[11] Tadelis, Steven. 2013. *Game* Theory: An Introduction.United Kingdom: Princeton University Press.

[12] Welsh, R. 2015. "Looking for Trouble Making NPCs Search Realistically". *Game* AI Pro 2. New York: CRC Press, 305-312.

[13] Zadeh, A. Lofti 1994. Fuzzy Logic, neural networks, and soft computing. Association for Computing Machinery, Inc.(3). 77-85.

[14] *Game*Ranking-Video *Game* Reviews from around the Internet 2016. http://www.*game*rankings.com\