JURUSAN TEKNIK INFORMATIKA

FAKULTAS TEKNOLOGI INFORMASI

INSTITUT TEKNOLOGI SEPULUH NOPEMBER

USULAN TUGAS AKHIR

1. IDENTITAS PENGUSUL

NAMA	**:**	**MOCHAMAD YASIN**
NRP	**:**	**5110100157**
DOSEN WALI	**:**	**Imam Kuswardayan, S.Kom., M.T.**
DOSEN PEMBIMBING	**:**	**1. Diana Purwitasari, S.Kom., M.Sc.**
		2. Imam Kuswardayan, S.Kom., M.T.

2. JUDUL TUGAS AKHIR

"Pemodelan *Tree* pada *Game* Sosial Android *Food Merchant Saga* dengan Behaviour Trees dan Algoritma A* untuk Penentuan Langkah NPC (*Non Playable Character*)"

3. LATAR BELAKANG

NPC (*Non Playable Character*) adalah karakter yang ada di dalam *game* tetapi tidak dikendalikan oleh pemain manusia [2]. NPC bisa bergerak dan bertingkah laku sesuai dengan keadaan lingkungan permainan layaknya manusia terhadap keadaan lingkungannya. Unsur yang bisa membuat NPC terlihat seperti manusia adalah kecerdasan buatan yang diterapkan pada NPC. Salah satu bagian kecerdasan buatan yang digunakan pada NPC adalah penentuan tindakan dan menemukan jalur pergerakan, sehingga NPC terlihat seperti manusia yang bisa bertindak dan menentukan jalurnya sendiri. NPC adalah salah satu media yang tepat untuk menerapkan kecerdasan buatan.

Saat ini banyak *game* yang memiliki detail grafis tinggi (3D) dan tema cerita yang lebih nyata, tetapi perilaku NPC membuat pemain frustasi [2]. Dengan desain grafis 2D dan tema cerita simulasi jual beli kuliner Nusantara, *game Food Merchant*

Saga akan menerapkan NPC yang menarik dan berperan dalam *game* sebagai pembeli. Peran NPC adalah sebagai karakter yang sangat berpengaruh pada jalannya *game* ini, karena pemain sebagai pemilik Pujasera (Pusat Jajanan Serba Ada) akan mendapat uang dari NPC. Uang tersebut digunakan untuk *upgrade* dan menjadi ukuran keberhasilan pemain dalam mengelola Pujaseranya.

NPC pada *game* ini nantinya bisa menentukan tindakan yang akan dilakukan sesuai dengan keadaan lingkungan dan sifat karakternya. Selain itu juga bisa menemukan jalur menuju kedai, kasir, kursi atau pintu. Untuk menentukan tindakan akan digunakan pemodelan *tree* dengan algoritma Behaviour Trees. Sedangkan untuk menemukan jalur menuju ke tempat yang dituju akan digunakan algoritma A*.

4. RUMUSAN MASALAH

Beberapa permasalahan yang digunakan sebagai rumusan masalah dalam Tugas Akhir ini adalah sebagai berikut.

1) Bagaimana memodelkan penentuan perilaku atau tindakan pada NPC *game Food Merchant Saga* dengan Behaviour Trees.
2) Bagaimana cara implementasi algoritma A* untuk *pathfinding* pada pergerakan NPC *game Food Merchant Saga*.

5. BATASAN MASALAH

Permasalahan yang dibahas dalam Tugas Akhir ini memiliki beberapa batasan antara lain.

1) Peta yang digunakan untuk pengujian NPC adalah peta dengan bentuk *tile-based*.
2) Tidak ada koordinasi atau interaksi antar NPC.
3) Perangkat lunak Unity yang digunakan untuk pengujian adalah versi 4.3.4. dengan bahasa pemrograman C#.

6. TUJUAN PEMBUATAN TUGAS AKHIR

Tujuan dari pembuatan Tugas Akhir ini adalah untuk membuat NPC pada *game Food Merchant Saga* lebih menarik dan menerapkan pemodelan perilaku NPC dalam bentuk *tree* menggunakan Behaviour Trees, serta menerapkan algoritma A* untuk menemukan jalur agar NPC bisa berjalan menuju kedai, kasir, kursi atau pintu.

7. MANFAAT TUGAS AKHIR

Manfaat dari hasil pembuatan Tugas Akhir ini adalah menjadikan NPC pada *game Food Merchant Saga* lebih menarik dengan memiliki kecerdasan buatan untuk menentukan tindakan atau perilaku dan menemukan jalur menuju kedai, kasir, kursi atau pintu. Sehingga perilaku atau tindakan NPC bisa terlihat seperti manusia saat membeli makanan.

8. TINJAUAN PUSTAKA

Berikut adalah fakta dan dasar teori yang digunakan sebagai penunjang pengerjaan Tugas Akhir ini.

8.1 Algoritma Behaviour Trees

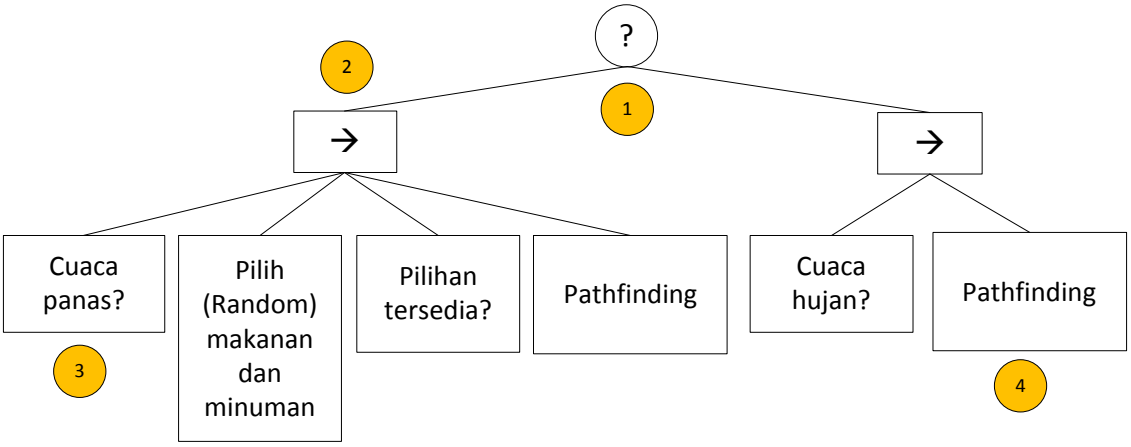
Beberapa tahun yang lalu, Behaviour Trees meningkat kepopulerannya dalam hal seleksi aksi untuk NPC pada *game-game* komersial. *Game-game* yang sudah menggunakan Behaviour Trees adalah Halo 2 [6], Halo 3 [5], Spore [4] dan GTA (*Grand Theft Auto*) [1]. Behaviour Trees tidak memiliki definisi yang formal karena digunakan terutama pada industri *game*. Champandard telah menulis mengenai Behaviour Trees dalam situsnya yaitu http://www.AiGameDev.com. Knafla juga telah mendeskripsikan Behaviour Trees dengan detail [2].

Behaviour Trees adalah DAG (*Directed Acyclic Graph*) yang terdiri dari berbagai macam *nodes*. Sesuai dengan namanya, Behaviour Trees berbentuk *tree*. Traversal pada Behaviour Trees menggunakan DFS (*Depth First Search*) sehingga penempatan *behaviour* dimulai dari kiri ke kanan secara menurun sesuai dengan prioritas. Hal tersebut untuk memastikan *behaviour* yang paling penting dieksekusi terlebih dahulu. Ketika *node-node* dieksekusi, mereka akan mengembalikan nilai yaitu *success*, *failure* atau *running* [2]. Pada Gambar 1 adalah contoh untuk Behaviour Trees. Macam-macam *nodes* pada Behaviour Trees adalah sebagai berikut.

- *Leaf Nodes*
  - *Conditions*
    Disimbolkan dengan kotak persegi panjang dengan isi teks berupa kalimat tanya. Contoh *Conditions* pada Gambar 1 *node* nomor 3.
  - *Actions*
    Disimbolkan dengan kotak persegi panjang dengan isi teks berupa kata kerja. Contoh *Actions* pada Gambar 1 *node* nomor 4.
- *Composite Nodes* (dengan 2 atau lebih sub*tree*)
  - *Selector*
    Disimbolkan dengan lingkaran yang berisi tanda tanya berfungsi untuk memilih "A atau B". Jika salah satu anak (sub*tree*) mengembalikan nilai *success* maka proses akan langsung berhenti

dan mengembalikan nilai *success.* Contoh *Selector* pada Gambar 1 *node* nomor 1.

- *Sequence*
  Disimbolkan dengan kotak persegi panjang dengan tanda panah mengarah ke kanan yang berfungsi untuk menjalankan "A kemudian B". Jika salah satu anak (sub*tree*) mengembalikan nilai *failure* maka proses akan langsung berhenti dan mengembalikan nilai *failure*. *Sequence* hanya akan mengembalikan nilai *success* jika semua *nodes* anaknya bernilai *success* dan dijalankan berurutan dari kiri ke kanan sesuai dengan prioritas yang diinginkan. Contoh *Sequence* pada Gambar 1 *node* nomor 2.



Gambar 1. Contoh Behaviour Trees.

8.2 Algoritma A*

Algoritma A* adalah improvisasi untuk versi *point-to-point* algoritma Dijkstra. Algoritma A* hadir pada tahun 1968 sebagai versi *directed point-to-point* algoritma Dijkstra. Sejak saat itu, algoritma A* menjadi acuan untuk semua algoritma pencarian *point-to-point graph*, terutama untuk *pathfinding*. Perbedaan utama antara A* dengan Dijkstra adalah pada nilai augmentasi CSF dengan nilai *heuristic* untuk memilih *open node* selanjutnya yang akan dilewati [3]. Berikut adalah ringkasan algoritma A*.

1. Memasukkan *node* awal ke dalam *Open List*.
2. Langkah berikut diulang.
   a. Menghitung nilai F dan mencari nilai F paling kecil dari *node-node* yang ada di dalam *Open List*.
   b. Memindahkan *node* terpilih tersebut ke *Closed List*.
   c. Untuk setiap 8 *node* yang mengelilingi *node* terpilih dilakukan.

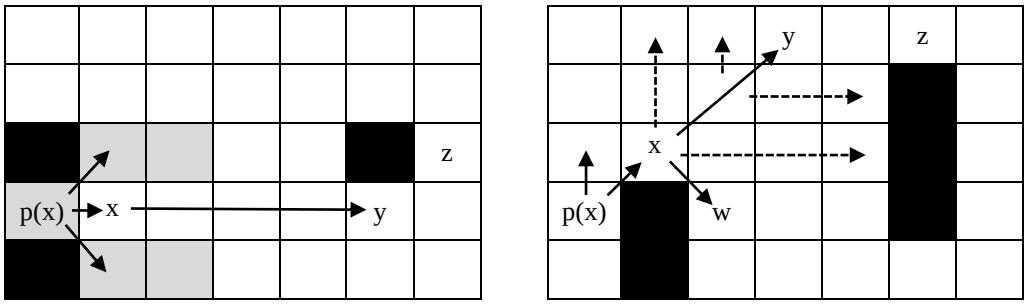
- Jika telah berada di dalam *Closed List* atau berupa *obstacle* yaitu meja, tembok atau kursi maka diabaikan. Selain itu lakukan langkah berikut.
- Jika belum berada di dalam *Open List* maka dimasukkan ke dalam *Open List*. *Node* yang sekarang dijadikan *parent*. Kemudian nilai F, G dan H dihitung.
- Jika sudah berada di dalam *Open List*, dilakukan pengecekan untuk melihat bahwa jalur ini menuju *node* tersebut lebih baik. Digunakan nilai G dengan ketentuan bahwa nilai G semakin rendah lebih baik.

d. Berhenti ketika salah satu kondisi berikut terpenuhi.
- *Node* tujuan telah dimasukkan ke dalam *Closed List* yang berarti bahwa jalur sudah ditemukan.
- Gagal menemukan *node* tujuan dan *Open List* telah kosong yang berarti jalur tidak ditemukan.

3. Jalur kemudian disimpan dan dilakukan *backtracking* dari *node* tujuan dengan berjalan dari setiap *node* ke *node parent* sampai mencapai *node* awal.

8.3 *Jumping Points*

Untuk mempercepat pencarian pada A* dapat dilakukan dengan pemilihan yang selektif pada *node* tertentu (*jump points*) [7]. Pada Gambar 2 adalah contoh dari *jump points*. Pencarian perluasan yang selektif akan dilakukan pada *node* x yang memiliki *parent* $p(x)$. Sedangkan *node* y adalah *jump points* yang disebut *successor*. *Node* yang berdekatan dengan *node* x disebut *neighbours*(x) dan masing-masing adalah n. Kotak warna abu-abu pada Gambar 2(a) adalah *node* yang telah dipangkas (*pruning*) untuk mempercepat pencarian *jump points*. Kemudian untuk menentukan *node* mana yang perlu untuk dipangkas, maka diperlukan *pruning rules*.



(a) (b)

Gambar 2. Contoh *Jump Points Straight* (a) dan *Diagonal* (b).

Dalam *pruning rules* digunakan pembandingan pada 2 jalur yaitu jalur dari $p(x)$ ke n dengan melewati *node* x dan tanpa melewati *node* x $(n\backslash x)$. Ada

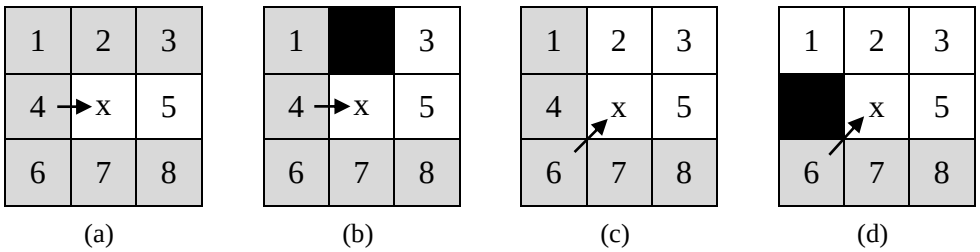
2 macam pergerakan berdasarkan arah dari $p(x)$ ke x yaitu *straight moves* dan *diagonal moves*. Perumusannya adalah sebagai berikut.

Straight moves: $length(\langle p(x), \dots, n\rangle \setminus x) \leq length(\langle p(x), x, n\rangle)$ (1)

Pada Gambar 3(a) adalah contoh dari *straight moves.*

Diagonal moves: $length(\langle p(x), \dots, n\rangle \setminus x) < length(\langle p(x), x, n\rangle)$ (2)

Pada Gambar 3(c) adalah contoh dari *diagonal moves*.



Gambar 3. Contoh *Pruning Rules.*

Hasil dari *straight moves* dan *diagonal moves* adalah kumpulan *node* yang disebut *natural neighbours*. Syarat di atas adalah untuk *neighbours*(x) yang tidak ada *obstacle*. Ketika *neighbours*(x) terdapat *obstacle* maka akan ada *node* yang *forced*. Syarat untuk *node* yang *forced* adalah sebagai berikut.

1. n bukan merupakan *natural neighbours* dari x.
2. $length(\langle p(x), x, n\rangle) < length(\langle p(x), \dots, n\rangle \setminus x)$.

Pada Gambar 3(b) dengan *straight moves* n = 3 menjadi *forced* dan Gambar 3(d) dengan *diagonal moves* n = 1 menjadi *forced*.

Node y adalah *jump points* jika y meminimalkan nilai k dari $y = x + k\vec{d}$ ($\vec{d}$ adalah arah pergerakan dari $p(x)$ ke x). Syarat untuk menentukan *node* yang akan dijadikan *jump points* adalah jika salah satu dari kondisi berikut terpenuhi.

1. *Node* y adalah *node* tujuan.
2. *Node* y memiliki minimal satu *neighbour* yang *forced*.
3. $\vec{d}$ adalah *diagonal move* dan terdapat *node* $z = y + k_i\vec{d}_i$, $k_i \in N$ langkah pada arah $\vec{d}_i \in \{\vec{d}_1, \vec{d}_2\}$ dan z adalah *jump points* dari y dengan kondisi 1 atau kondisi 2.

Pada Gambar 2(b) adalah contoh *jump points* yang didapat dari kondisi 3.

Algoritma 1 Identifikasi *Successors*
Parameter: x: *current node*, s: *start*, g: *goal*
1: $successors(x) \leftarrow \emptyset$
2: $neighbours(x) \leftarrow prune(x, neighbours(x))$
3: **for all** $n \in neighbours(x)$ **do**
4: $n \leftarrow jump(x, direction(x, n), s, g)$
5: add n to $successors(x)$
6: **return** $successors(x)$

Algoritma 2 Fungsi *jump*
Parameter: x: *initial node*, $\vec{d}$: *direction*, s: *start*, g: *goal*
1: $n \leftarrow step(x, \vec{d})$
2: **if** n is an obstacle or is outside the grid **then**
3: **return** *null*
4: **if** $n = g$ **then**
5: **return** n
6: **if** $\exists\, n' \in neighbours(n)$ s.t. n' is forced **then**
7: **return** n
8: **if** d is diagonal **then**
9: **for all** $i \in \{1,2\}$ **do**
10: **if** $jump(n, \vec{d}_i, s, g)$ is not *null* **then**
11: **return** n
12: **return** $jump(n, \vec{d}, s, g)$

Gambar 4. Algoritma untuk *Jump Points*.

Algoritma pada Gambar 4 digunakan untuk mencari *jump points* dengan penjelasan langkah berikut.

1. Mencari *neighbours* yang dipangkas dengan melakukan *straight moves* dan *diagonal moves*.
2. Menyimpan *neighbours* yang masih tersisa setelah dilakukan *pruning* (Algoritma 1 baris 2).
3. Menyimpan setiap *neighbour* n ke dalam kumpulan *successors* untuk x dengan menjalankan fungsi *jump* (Algoritma 1 baris 3:5).
4. Untuk mengidentifikasi *jump points* digunakan Algoritma 2.
5. Fungsi *jump* dimulai dengan *stepping* ke arah $\vec{d}$ dari *node* x dan periksa n sesuai syarat *jump points* (Algoritma 2 baris 1).
6. Ketika memenuhi syarat *jump points* maka *return* nilai n (Algoritma 2 baris 5, 7 dan 11).
7. Jika tidak maka melangkah lagi ke arah $\vec{d}$ dengan n yang baru (Algoritma 2 baris 12).
8. Rekursi akan berhenti ketika ada *obstacle* dan tidak ada langkah lagi (Algoritma 2 baris 3).

9. Setelah *straight jump points* tidak ditemukan maka *diagonal jump points* baru dieksekusi (Algoritma 2 baris 9:11).

8.4 *Game* Sosial dan Teknologi

Game sosial adalah *game* yang dimainkan dengan teman-teman yang berada pada suatu jaringan sosial sehingga pemain bisa mengundang atau mengajak teman untuk melakukan aktivitas tertentu dalam *game*. *Game* sosial pada umumnya bersifat *asynchronus* dan untuk memainkannya pemain tidak perlu *online* secara bersama-sama.

Unity merupakan sebuah ekosistem pengembangan *game* dengan mesin *rendering* yang kuat, terintegrasi, disuplai dengan satu set lengkap alat intuitif dan alur kerja yang cepat untuk membuat *game* 2D ataupun 3D yang interaktif, penggunaan *multiplatform* dengan mudah serta menyediakan ribuan aset berkualitas yang siap pakai yang ada pada *Unity Store* [8].

Android adalah sistem operasi berbasis Linux yang dikembangkan secara khusus untuk perangkat *touchscreen* seperti telepon pintar maupun komputer tablet. Sistem operasi ini pertama kali dikembangkan oleh Android, Inc. hingga Google mengambil alih pada tahun 2005. Pada Oktober 2008, telepon genggam pertama dengan menggunakan Android diluncurkan, yaitu T-Mobile G1.

9. RINGKASAN ISI TUGAS AKHIR

Game Food Merchant Saga adalah *social mobile game* yang bergenre simulasi bisnis kuliner Indonesia dalam sebuah Pujasera (Pusat Jajanan Serba Ada). Pemain akan diberi modal berupa sebuah Pujasera kecil yang hanya berisi kedai bakso dan kedai nasi pecel. Kemudian seiring dengan meningkatnya level Pujasera, maka luas bangunan Pujasera juga akan bertambah. Pemain bisa membeli kedai baru dan bisa memproduksi menu makanan baru. Pemain bisa meningkatkan kapasitas produksi makanan dan menambah jumlah jenis dan variasi makanan di setiap kedai. Syarat untuk mendapatkan poin yang berguna untuk naik level adalah menyelesaikan daftar tugas yang tersedia. Pemain akan mendapatkan uang dari setiap penjualan makanan dan penyelesaian misi.

Game Food Merchant Saga dibangun menggunakan teknologi Unity 2D dan untuk NPC pada *game* ini adalah pembeli yang akan membeli makanan di Pujasera pemain. NPC memiliki kecerdasan buatan yang menggunakan algoritma Behaviour Trees untuk menentukan perilaku dan tindakan dari NPC sesuai dengan keadaan lingkungan. Misalnya lingkungan sekitar Pujasera dalam keadaan hujan, maka NPC lebih memilih untuk membeli makanan atau minuman yang hangat. Untuk gambaran Behaviour Trees pada NPC dalam *game* ini terdapat pada lampiran. Kemudian NPC juga akan menggunakan algoritma A* untuk menemukan jalan dari pintu masuk menuju ke kedai, ke kasir untuk membayar, mencari tempat duduk yang kosong dan keluar dari Pujasera.

Pada awal permainan, pemain hanya akan diberi Pujasera dengan luas yang kecil. Seiring berjalannya permainan, pemain bisa memperluas area Pujaseranya. Sehingga untuk algoritma *pathfinding* A* dibutuhkan modifikasi agar pencarian jalur bisa lebih cepat. Modifikasi yang akan dilakukan adalah dengan menggunakan *Jumping Points* [7].

10.METODOLOGI

Berikut adalah langkah-langkah untuk mengerjakan Tugas Akhir ini.

a. Penyusunan Proposal Tugas Akhir

Tahap awal adalah penyusunan proposal Tugas Akhir. Proposal ini mengajukan gagasan dalam pembuatan kecerdasan buatan pada NPC dengan menggunakan algoritma Behaviour Trees untuk menentukan perilaku dan tindakan NPC serta algoritma A* untuk menemukan jalur ke tempat yang diinginkan NPC.

b. Studi Literatur

Pada tahap ini dilakukan pencarian informasi dari literatur yang diperlukan untuk menyelesaikan NPC yang akan dibuat. Literatur yang digunakan berasal dari paper, buku, internet maupun materi-materi lain yang berhubungan dengan algoritma Behaviour Trees dan A*.

c. Perancangan Sistem

Pada tahap ini dilakukan perancangan sistem pada NPC yang akan dibuat. Perancangan ini meliputi perancangan perilaku pada NPC dengan Behaviour Trees dan perancangan A* untuk *pathfinding*.

d. Implementasi Perangkat Lunak

Tahap implementasi ini adalah tahap untuk mengimplementasikan Behaviour Trees dan A* pada NPC dalam *game Food Merchant Saga*. Kakas bantu yang digunakan adalah Unity dan menggunakan bahasa pemrograman C#.

e. Pengujian dan Evaluasi

Pada tahap ini dilakukan pengujian untuk menguji Behaviour Trees dan A* pada NPC agar sesuai dengan metode yang digunakan saat game dijalankan. Kemudian jika ada kesalahan atau kekurangan maka bisa segera dilakukan perbaikan.

f. Penyusunan Buku Tugas Akhir

Pada tahap ini dilakukan penyusunan laporan yang menjelaskan dasar teori dan metode yang digunakan dalam Tugas Akhir ini serta hasil dari implementasi aplikasi perangkat lunak yang telah dibuat. Sistematika penulisan buku Tugas Akhir secara garis besar antara lain:

1. Pendahuluan
    a. Latar Belakang
    b. Rumusan Masalah
    c. Batasan Tugas Akhir
    d. Tujuan
    e. Metodologi
    f. Sistematika Penulisan
2. Tinjauan Pustaka
3. Desain dan Implementasi
4. Pengujian dan Evaluasi
5. Kesimpulan dan Saran
6. Daftar Pustaka

11. JADWAL KEGIATAN

Jadwal rencana pengerjaan Tugas Akhir ini dijelaskan pada Tabel 1.

Tabel 1. Rancangan Jadwal Kegiatan Pengerjaan Tugas Akhir.

Tahapan	2014																			
	Februari				Maret				April				Mei				Juni			
Penyusunan Proposal Tugas Akhir																				
Studi Literatur																				
Perancangan Sistem																				
Implementasi Perangkat Lunak																				
Pengujian dan Evaluasi																				
Penyusunan Buku Tugas Akhir																				

12. DAFTAR PUSTAKA

[1] A. J. Champandard, "Behavior Trees for Next-Gen Game AI," 2008. [Online]. Available: http://aigamedev.com/insider/presentations/behavior-trees/. [Accessed 15 Februari 2014].

[2] A. Johansson, Affective Decision Making in Artificial Intelligence, Norrkoping, 2012.

[3] B. Anguelov, Video Game Pathfinding and Improvements to Discrete Search on Grid-based Maps, Pretoria, 2011.

[4] C. Hecker, "My liner notes for spore/Spore Behavior Tree Docs," 2007. [Online]. Available: http://chrishecker.com/My_Liner_Notes_for_Spore/Spore_Behavior_Tree_Docs. [Accessed 15 Februari 2014].

[5] D. Isla, "Building a Better Battle - The Halo 3 AI Objectives System," in *Game Developers Conference*, 2008.

[6] D. Isla, "Handling Complexity in The Halo 2 AI," in *Game Developers Conference*, 2005.

[7] D. Harabor and A. Grastien, Online Graph Pruning for Pathfinding on Grid Maps, 2011.

[8] Unity, "Unity - Game engine, tools and multiplatform," Unity3D, 2014. [Online]. Available: http://unity3d.com/unity. [Accessed 15 Februari 2014].