Penggunaan Struktur Data *Quad-Tree* dalam Algoritma *Collision Detection* pada *Vertical Shooter Game*

Rizky Maulana Nugraha - 13508083
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
If18083@students.if.itb.ac.id

***Abstract*—Makalah ini menjelaskan hasil pengujian implementasi struktur data *quad-tree* pada algoritma *collision detection* menggunakan strategi *divide and conquer*. Pengujian ini dikhususkan untuk implementasi pada permainan dengan genre *vertical shooter game*. Pengujian dilakukan menggunakan tiga parameter uji, yaitu kedalaman *quad-tree*, banyaknya objek spasial yang dicek dan ukuran ruang spasial yang dipartisi oleh *quad-tree*. Penulis berkesimpulan struktur data ini sangat efisien mengindeks objek-objek spasial pada *vertical shooter game* yang memiliki jumlah objek spasial dibawah 1000. Selain itu, struktur data ini juga umum digunakan untuk *Graphic Information Systems* dan aplikasi *layouting* seperti**

***Index Terms*— Quad-tree, Collision detection, Vertical Shooter Game, Divide and Conquer.**

I. PENDAHULUAN

Quad-tree adalah struktur data yang digunakan untuk *spatial indexing*. Struktur data lain yang digunakan untuk *spatial indexing* adalah *binary-tree* dan *oct-tree.* Ketiga struktur inilah yang biasa digunakan untuk *spatial indexing*. *Quad-tree* adalah struktur pohon yang memiliki empat anak atau cabang. Pada penggunaan ini, tiap cabang mewakili posisi atau daerah. Jadi tiap cabang memiliki empat cabang atau empat daerah. Penjelasan mengenai *spatial indexing* akan diberikan di bab Dasar Teori. *Collision Detection* atau pengecekan tumbukan bisa dilakukan menggunakan algoritma ini. Struktur data ini sangat dibutuhkan oleh aplikasi yang membutuhkan *spatial indexing* seperti proses *layouting* gambar, simulasi fisik, dan pengecekan tumbukan pada *game*. Penulis terutama tertarik dengan dengan aplikasi *quad-tree* pada *game*. *Game* yang dimaksud adalah *game* bergenre *vertical shooter*. Pada permainan ini pemain diberikan satu karakter (misalnya pesawat) yang bisa dikendalikan. Selama permainan, pemain harus menghindari tembakan misil milik lawan, dan pemain pun harus menembakkan misilnya ke lawan. Jika pemain terkena misil milik lawan, maka karakter pemain akan mati atau hancur.

Penulis pernah mengimplementasi algoritma untuk pendeteksian tumbukan pada game ini, namun hasilnya kurang baik. Algoritma yang dimaksud menggunakan struktur data *binary-tree* yang digunakan untuk membuat indeks posisi objek dalam *game*. Namun, pada prakteknya pohon indeks yang dibuat menjadi tidak seimbang (bukan pohon seimbang) sehingga proses pencarian objek tetap lambat. Namun, pada *quad-tree* pohon yang dibentuk adalah pohon seimbang.

Penulis memilih topik makalah ini karena permasalahan pendeteksian tumbukan yang efektif pada *vertical shooter game* sudah menjadi bahan penelitian sebelumnya. Kurangnya referensi dari internet dan buku juga menjadi motivasi penulis untuk mencari algoritma yang efisien untuk pendeteksian tumbukan. Struktur data *quad-tree* itu sendiri baru diketahui penulis setelah mengikuti kuliah "Strategi Algoritma" dan "Grafika Komputer dan Visualisasi".

II. DASAR TEORI

Spatial Indexing

Spatial Indexing adalah proses pembuatan indeks untuk objek-objek spasial. Contoh objek spasial adalah persegi, dan objek dua dimensi lain. Objek tiga dimensi juga adalah objek spasial. Secara umum, yang dimaksud objek spasial adalah objek yang menempati ruang. Misalnya dalam sistem pemetaan atau *Geographic Information Systems*, *spatial indexing* digunakan untuk menyimpan objek spasial agar lebih mudah di tampilkan pada layar, diklip dan diproses.

Struktur data yang biasa digunakan untuk *spatial indexing* adalah *binary-tree, quad-tree, dan oct-tree*. *Binary-tree* dan *quad-tree* biasa digunakan untuk menampilkan objek dua dimensi, sedangkan *oct-tree* biasa digunakan untuk objek tiga dimensi.

Quad-tree

Quad-tree adalah struktur data pohon. Setiap cabang atau *node* memiliki empat anak atau *node*. Pohon yang dibentuk oleh *quad-tree* adalah pohon seimbang. Pohon seimbang artinya setiap anak *node* memiliki jumlah anak *node* yang sama. Untuk *quad-tree*, setiap *node* memiliki anak empat *node*. Jika *node* memiliki anak, maka jumlah anaknya harus empat. Jika pohon tersebut bertambah kedalamannya, maka kedalaman tersebut harus terisi penuh. Konsekuensinya, jika n adalah kedalaman maka 4^n

adalah jumlah *node* pada kedalaman ini.

Pada *spatial indexing*, setiap *node* mewakili daerah tertentu. Pembagian daerah ini biasanya dilakukan dengan membagi daerah menjadi persegi. Setiap persegi dibentuk oleh empat persegi yang lebih kecil. Misalkan daerah total yang akan diindeks adalah persegi berukuran n kali n, maka tiap *node* anaknya akan mewakili persegi berukuran n/2 kali n/2, yaitu persegi pojok kiri atas, pojok kiri bawah, pojok kanan atas, dan pojok kanan bawah. Kemudian tiap persegi bisa dibagi lagi menjadi persegi yang lebih kecil hingga batas ketelitian yang diinginkan (batas persegi yang paling kecil.

Collision Detection

Collision detection atau pendeteksian tumbukan adalah proses pengecekan apakah beberapa buah objek spasial saling bertumpuk atau tidak. Jika ternyata ada paling sedikit dua buah objek yang bertumpuk, maka kedua objek tersebut dikatakan saling bertumpukkan. Pada ruang spasial dua dimensi. Objek yang bertumpuk berarti objek spasialnya beririsan.

Teknik pendeteksian tumbukan bisa dikelompokkan menjadi dua macam yaitu *priori detection* dan *post detection*. *Priori detection* adalah pengecekan tumbukan sebelum tumbukan tersebut terjadi, sedangkan *post detection* adalah pengecekan tumbukan setelah tumbukan tersebut terjadi.

Pada *vertical shooter game*, pendeteksian yang digunakan adalah *post detection* karena tumbukan baru ditangani setelah tumbukan tersebut terjadi. Namun, jika *game* tersebut memiliki *AI (Artificial Intelligence)* maka pendeteksian harus berjenis *priori detection* agar AI dapat berreaksi agar terhindar dari tumbukan.

Algoritma pendeteksian tumbukan menggunakan *quad-tree* ini adalah algoritma *indexing*. Jadi, sudah jelas kolisi dideteksi setelah objek spasial diindeks. Karena sudah diindeks, artinya objek sudah bertumbukan.

Proses pengindeksan ini harus dilakukan dengan sangat cepat karena permainan ini sangat mengandalkan respons pemain. Jika pendeteksian terjadi sangat lambat, maka fps *rate (Frame per second rate)* permainan bisa turun sangat drastis. FPS adalah ukuran seberapa cepat gambar di layar diperbarui. Jika FPS permainan rendah, maka permainan tersebut dianggap mengalami *lag* (telat). Tentu saja ini harus dihindari. Jika pengecekan tumbukan ini dilakukan menggunakan algoritma *brute-force*, pengecekan terjadi sangat lambat. Andaikata ada n buah objek spasial maka dengan aturan jabat tangan, ada n^2 pengecekan yang dilakukan. Jumlah ini bisa juga didapat dari fakta bahwa setiap objek harus dicek apakah bertumbukan dengan objek lain, dengan kata lain kita bisa menghitung jumlah pengecekan menggunakan kombinatorika yaitu C_2^n, yaitu kompleksitasnya $O(n^2)$. Dengan menggunakan strukur pohon, maka kompleksitas algoritmanya diperbaiki menjadi O(n log n)

Algoritma

Algoritma yang digunakan mengguna
kan struktur data ini adalah *divide and conquer*. Penggunaan struktur data *quad-tree* tergolong algoritma jenis ini karena struktur data ini berfungsi membagi ruang pencarian menjadi empat secara rekursif hingga kedalaman tertentu, yang menjadi ciri khas algoritma jenis *divide and conquer*. Seperti yang sudah disebutkan sebelumnya, kompleksitas asimptotik algoritma ini adalah O(n . log n).

III. IMPLEMENTASI

Implementasi Quad-Tree

Penulis akan mengimplementasi *quad-tree* pada bahasa pemrograman C# menggunakan *framework* XNA. Implementasi ini akan diuji dengan berbagai kondisi pengujian.

Quad-tree yang dibuat adalah kelas pohon biasa yang memiliki empat anak (*node)*. Implementasi pengujian menggunakan ruang spasial berukuran 256x256 pixel yang menjadi ukuran standar layar pada *game vertical shooter*. Semestinya *game vertical shooter* memiliki layar yang lebih besar tingginya. Namun, pada lingkungan pengujian ini ruang spasial dibuat persegi agar pembagian lebih mudah dilihat.

Objek spasial yang akan disimpan adalah objek sederhana berbentuk persegi juga. Sedangkan data yang disimpan hanya berdasarkan posisi *upper-left* persegi tersebut. Jika tumbukan dinyatakan terjadi maka persegi yang bertumbukan berada pada segmen yang sama.

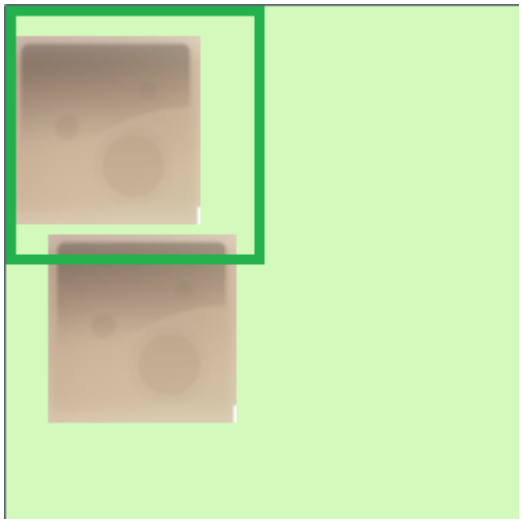
IV. PENGUJIAN

Implementasi yang sudah disebutkan tadi akan diuji dengan parameter yang berbeda. Parameter yang membedakan diantaranya kedalaman *quad-tree*, jumlah objek dan ukuran ruang spasial yang digunakan. Sedangkan parameter yang akan dilihat sebagai dampak parameter uji adalah UPS dan FPS. *Updates per Second* atau UPS adalah jumlah *update* yang dilakukan per detik. UPS menunjukkan seberapa cepat algoritma tersebut bekerja hingga tidak membuat proses *update* ruang spasial menjadi tertinggal. Nilai UPS yang besar dan stabil menunjukkan proses ruang spasial diperbarui dengan lancar dan berarti algoritma yang digunakan cukup bagus. Sedangkan, *Frames per Second* atau FPS adalah jumlah frame yang diganti per detiknya. FPS menunjukkan seberapa cepat layar digambar ulang. Nilai FPS yang besar berarti menunjukkan proses penggambaran ulang cukup bagus dan tidak terhambat oleh proses *updates.* Pada teorinya, sebenarnya algoritma pendeteksian kolisi hanya mempengaruhi UPS saja dan tidak mempengaruhi FPS. Untuk tiap pengujian, persegi yang digunakan sebagai objek dalam ruang spasial sama ukurannya dengan ukuran segmen *quad-tree* yang digunakan. Ukuran segmen *quad-tree* adalah ukuran terkecil yang digunakan

quad-tree untuk mengelompokkan objek spasial. Contohnya, jika ukuran segmen 1x1 pixel, maka algoritma dapat mendeteksi objek spasial berukuran 1x1 pixel yang bertumbukan dengan objek spasial lain yang berukuran 1x1 pixel.

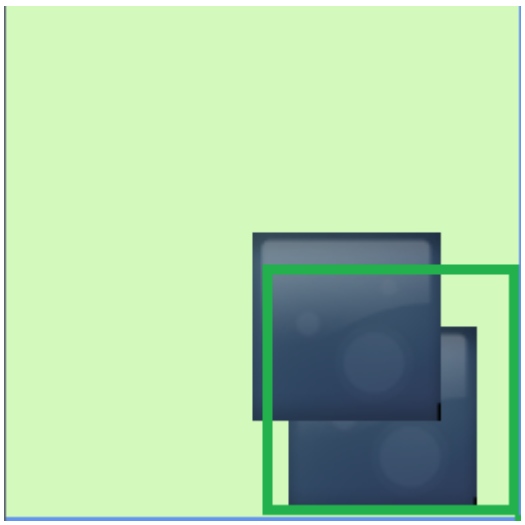
Pengaruh kedalaman quad-tree

Penulis akan mencoba beberapa kedalaman *quad-tree* pada ruang spasial dengan ukuran 256x256 pixel. Ukuran kedalaman sangat mempengaruhi akurasi pendeteksian tumbukan. Makin besar kedalaman *quad-tree* yang dibuat berarti makin akurat pendeteksian yang dilakukan. Jika ukuran layar cukup kecil bahkan bisa dilakukan pendeteksian per-pixel. Pada bab dasar teori, penulis telah menjelaskan bahwa algoritma *divide and conquer* yang digunakan akan membagi ruang spasial menjadi empat. Dengan ruang spasial berukuran 256x256 pixel dan kedalaman 1 maka ruang spasial akan dibagi menjadi empat persegi berukuran 128x128 pixel. Persegi apapun yang *upper-left*-nya ada di segmen yang sama akan dianggap bertumbukan.



Gambar 1. Kedalaman 1, warna abu-abu menunjukkan bahwa objek dinyatakan bertumbukan oleh algoritma

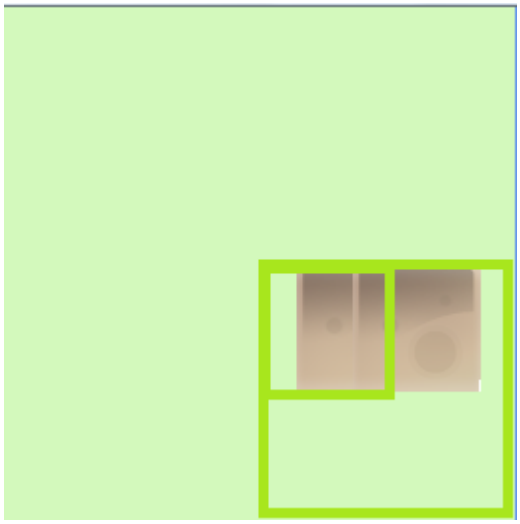
Pada gambar diatas, kedalaman yang digunakan adalah 1. Artinya ukuran segmennya 128x128. Pada contoh diatas, kedua kotak dinyatakan bertumbukan (dengan cara ditandai warnanya menjadi abu-abu oleh program). Kotak dinyatakan bertumbukan walaupun sebenarnya terpisah. Ini dikarenakan *upper-left* dari kedua kotak tersebut menempati segmen yang sama.



Gambar 2. Kedalaman 1, warna gelap menunjukkan bahwa objek tidak dinyatakan bertumbukan oleh algoritma

Pada gambar diatas, kedalaman yang digunakan tetap satu, kedua kotak terlihat bertumbukan (beririsan) tetapi program tidak mendeteksi kedua kotak bertumbukan karena *upper-left* kedua kotak berada pada segmen yang berbeda. Kotak yang paling atas berada pada segmen kiri atas, sedangkan kotak paling bawah berada pada segmen kanan bawah pada *quad-tree*. Pada kedua contoh diatas dengan kedalaman 1, UPS dan FPS-nya adalah 60. UPS atau FPS 60 adalah ukuran standar pada *game*. Nilai ini 60 karena dibatasi oleh *framework* XNA.

Berikutnya, penulis mencoba kedalaman 2 yang artinya ukuran segmennya adalah 64x64 pixel.



Gambar 3. Kedalaman 2

Pada gambar diatas segmen yang mendeteksi tumbukan menjadi lebih sempit. Dari gambar, bisa dilihat bahwa semakin kecil kedalaman semakin besar akurasi tumbukan. Pada pengujian ini, dengan kedalaman 8 maka ukuran segmen 1x1 pixel dengan jumlah kotak tetap 2. UPS program tetap 60 dengan kedalaman 8.

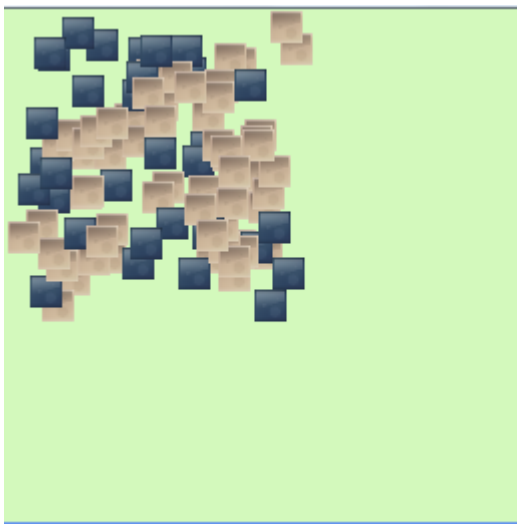
Pengaruh jumlah objek

Jumlah objek yang dimaksud adalah jumlah objek spasial yang harus dideteksi oleh algoritma. Pada contoh sebelumnya, jumlah objek adalah dua dengan kedalaman *quad-tree* 8. Penulis mencoba menggunakan 5000 objek spasial dengan ukuran 1x1 pixel, dan masih bernilai 60 UPS dengan stabil. Namu saat dicoba menggunakan 6000 objek, UPS menjadi tidak stabil. Makin banyak jumlah objek makin mempengaruhi UPS dan lamanya algoritma pendeteksian tumbukan selesai. Namun, untuk *game vertical shooter* 5000 objek spasial yang mampu ditangani algoritma ini sudah menunjukkan kinerja yang cukup baik.

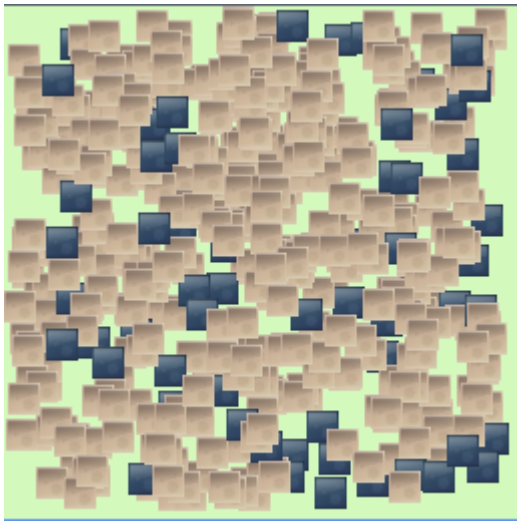
Ukuran ruang spasial

Ukuran ruang spasial yang digunakan pada dua pengujian sebelumnya adalah 256x256. Secara teori memperbesar ruang spasial empat kali sama saja ukuran segemennya dengan menambah kedalaman sebanyak satu kedalaman. Jadi, dengan ukuran ruang spasial 1024x1024 (diperbesar 16 kali), dan 10 kedalaman akan menghasilkan segmen yang sama dengan ruang spasial 256x256 dan kedalaman 8. Setelah dicoba, untuk menghasilkan akurasi yang sama pada ruang spasial 1024x1024 dengan kedalaman 10 dan jumlah objek 5000, membuat UPS menjadi kecil. Jadi untuk tingkat akurasi yang sama, makin besar ukuran ruang spasialnya, performa makin berkurang karena makin banyak daerah yang harus dipartisi. Secara teori, memperbesar ukuran spasial empat kali lipat akan menambah jumlah segmen empat kali lipat juga dengan akurasi yang sama. Menambah segmen empat kali lipat berarti menambah penggunaan memori empat kali lipat, dan menambah waktu pencarian objek sebesar $\log_4(n)$ dengan n adalah rasio perbesaran ruang spasialnya.

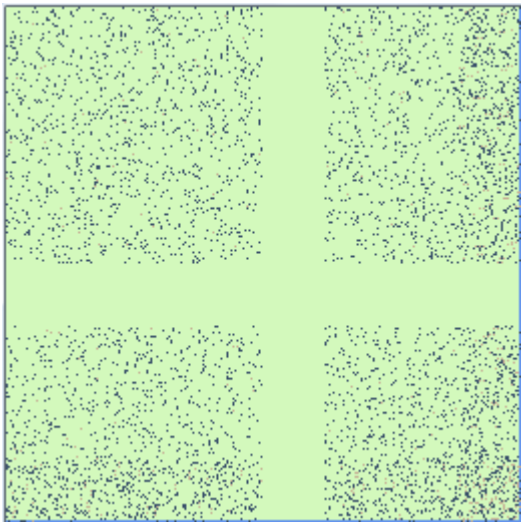
Berikut adalah beberapa *screenshoot* hasil uji yang lain.



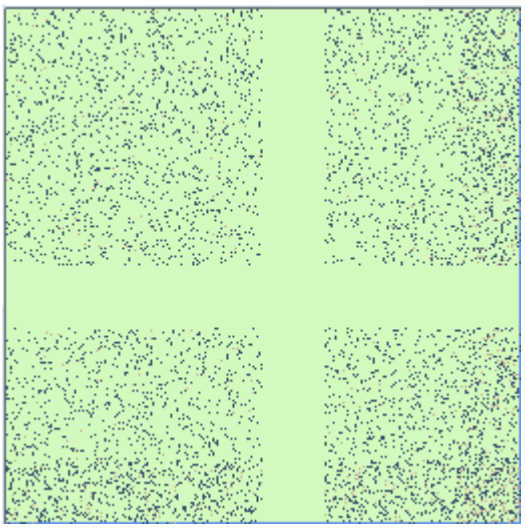
Gambar 4. Ukuran 256x256, kedalaman 4, 100 objek, ukuran segmen 16x16 pixel, UPS 60



Gambar 5. Ukuran 256x256, kedalaman 4, 500 objek, ukuran segmen 16x16 pixel, UPS 60



Gambar 4. Ukuran 256x256, kedalaman 8, 5000 objek, ukuran segmen 1x1 pixel, UPS 60



Gambar 4. Ukuran 256x256, kedalaman 8, 6000 objek, ukuran segmen 1x1 pixel, UPS rata-rata 3

V. KESIMPULAN DAN SARAN

Hasil pengujian dengan tiga parameter uji tersebut memberikan informasi yang cukup tentang performa algoritma *collision detection* menggunakan *divide and conquer* dan struktur data *quad-tree*. Untuk *game vertical shooter* biasa, jumlah objek spasial rata-rata tidak melebihi seribu buah. Penggunaan algoritma ini sudah lebih dari cukup untuk *game* dengan genre ini. Strategi lain bisa digunakan untuk *collision detection game vertical shooter*, salah satu caranya menggunakan *bounding box*. Objek spasial dapat diindeks hanya berdasarkan *bounding-box*-nya, sedangkan untuk pengecekan lebih lanjut untuk geometri yang lebih rumit dapat dilakukan lebih lanjut setelah mendapat list objek spasial dalam segmen yang dicek tersebut. Ukuran segmen pun tidak harus kecil, bisa sekitar 32x32 pixel saja dengan ukuran layar 256x256 pixel, yang berarti kedalaman yang digunakan hanya sekitar 3 atau 4. Optimasi lebih lanjut juga bisa dilakukan, misalnya menggunakan ukuran berupa kelipatan dua agar pengindeksan bisa menggunakan *bit-shifting.*

Jika pendeteksian 5000 objek dengan ketelitian 1x1 pixel saja bisa dilakukan dengan sangat efisien, maka algoritma ini bisa sangat efisien juga untuk aplikasi *layouting* seperti browser ataupun CAD.

REFERENCES.

[1] Henning Eberhardt, Vesa Klumpp, Uwe D. Hanebeck, *Density Trees for Efficient Nonlinear State Estimation*, Proceedings of the 13th International Conference on Information Fusion, Edinburgh, United Kingdom, July, 2010.

[2] Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf (2000). *Computational Geometry* (2nd revised ed.). Springer-Verlag. ISBN 3-540-65620-0. Chapter 14: Quadtrees: pp. 291–306

[3] Raphael Finkel and J.L. Bentley (1974). "Quad Trees: A Data Structure for Retrieval on Composite Keys". *Acta Informatica*

[4] Tomas G. Rokicki (2006-04-01). "An Algorithm for Compressing Space and Time". Retrieved 2009-05-20

[5] http://en.wikipedia.org/wiki/Collision_detection, 5 December 2010.

[6] http://en.wikipedia.org/wiki/Quad_tree, 5 December 2010

[7] http://www.kyleschouviller.com, 5 December 2010

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 29 April 2010

Rizky Maulana Nugraha - 13508083