Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer e-ISSN: 2548-964X
Vol. 3, No. 9, September 2019, hlm. 8630-8639 http://j-ptiik.ub.ac.id

Optimasi *Collision Detection* Pada *2D Spaceship Game* Menggunakan Metode *Quadtree*

Andhi Indra Lestya Wicaksono[1], Eriq Muhammad Adams Jonemaro[2], Muhammad Aminul Akbar[3]

Program Studi Teknik Informatika, Fakultas Ilmu Komputer, Universitas Brawijaya
Email: [1]andyindra27@gmail.com, [2]eriq.adams@ub.ac.id, [3] muhammad.aminul@ub.ac.id

Abstrak

Permainan video *2D Spaceship* merupakan permainan klasik yang dapat dimainkan sejak jaman *Gameboy*, *Game* ber-*genre spaceship* ini masih populer di kalangan semua umur, pada *2D spaceship game* ini, *player* atau pemain diberi objektif untuk menghindari *obstacle* hingga mengalahkan *NPC*. *Game* ber-*genre* ini memerlukan sebuah *collision detection*, sebuah algoritma untuk mengetahui bertabrakannya 2 objek, seperti peluru dengan *NPC* atau *player* dengan *NPC* atau juga peluru dengan *obstacle*. Metode yang digunakan untuk memeriksa pertumpukan antara 2 objek dalam penelitian ini adalah menggunakan *quadtree*, yaitu metode pemisahan *area* pada *game* yang bertujuan untuk memfokuskan *collision detection* pada *area* tertentu saja, metode ini berjalan ketika ada 2 objek yang bisa bertumpukan berada pada 1 *node* (*area)* yang sama, jika pada suatu *area* tidak ada 2 objek yang dapat bertumpukan, metode *collision detection* tidak akan bekerja, *Quadtree* lebih optimal daripada metode *collision detection* yang langsung diterapkan kedalam game (*Bruteforce)* karena deteksi pertumpukannya ke semua wilayah sekaligus yang menyebabkan metode dengan *quadtree* menggunakan *resource* lebih sedikit dan *frame per second* naik saat dimainkan, hasil dari pengujian menyatakan bahwa algoritma *Quadtree* mampu mendobrak *FPS* menjadi lebih tinggi, rata-rata *FPS* jika menggunakan *bruteforce* tercatat sebesar 70,06 *FPS* sedangkan rata rata *FPS* dengan menggunakan *Quadtree* bisa mencapai 268,1 *FPS*

Kata kunci : *2D game, spaceship game, collision detection, quadtree, collidable object*

Abstract

2D Spaceship Game is a classic game since Gameboy's came out, but even it's an old type of game, many people still play it until now. This game gives you objective to survive an attack against obstacle or NPC. this genre needs a collision detection, an algorithm that allows you to detect collision between 2 collidable objects, the method uses quadtree, a method that allows certain area divided by itself in order to focusing collision detection, this method works if there is 2 collidable objects in the same place at the same time. So if there's no object or just 1 object in certain area, the collision detection algorithm won't work, this method is more optimal than just use the collision detection and implement it straight to the game (bruteforce) because the collision detection it's not focused in certain area and the algorithm will detect in all areas. That means the quadtree use less resource and the frame per second will increase, After being tested for couple times, the Quadtree Algorithm able to bust the FPS up, the average FPS for Bruteforce is 70.06 FPS, and the average FPS for Quadtree Collision Detection is 268.1 FPS

Keyword : *2D game, spaceship game, collision detection, quadtree, collidable object*

1. PENDAHULUAN

Video game atau bisa disebut permainan video adalah salah satu hiburan yang sudah banyak dinikmati oleh banyak golongan masyarakat, mulai dari *cooking mama* hingga *Grand Theft Auto V*, tidak hanya anak anak saja yang bisa menikmati *video game* namun orang dewasa juga bisa menikmatinya,*video game* juga sangat populer di jaman sekarang, bahkan di Indonesia saja survei yang dilakukan pada 2017 lalu, tercatat bahwa potensi nilai industrI *game* Tanah Air mencapai kisaran 800 juta dollar AS atau sekitar Rp 11 triliun. (Cipto Adiguno, 2017).

Permainan video masa kini tidak hanya berpacu pada bentuk *3D* dengan pengaplikasian teknologi *AR* dan *VR*, namun *game* berbentuk *2D* dengan gambar *pixelate* bisa mendobrak pasar *game* seperti *StardewValley* dan *RimWorld* (Kukuh Basuki, 2017). Lalu ada sebuah *subgenre* dalam permainan video yang bernama *2D spaceship*, walaupun merupakan genre klasik, genre permainan video ini sekarang mulai semakin popular semenjak perkembangan dan kepopularan *mobile game* dan banyaknya game platform pada tahun 2010 (Jones, et al, 2014).

Algoritma *collision detection* merupakan proses pengecekan apakah dua buah atau lebih objek spasial saling bertumpuk atau tidak, dan *collision detection* merupakan teknik deteksi tabrakan untuk mengetahui objek-objek apa saja yang bersentuhan dalam bidang koordinat tertentu (Lui Haekal Fasha, 2018). Di penelitian ini, peneliti menggunakan metode pemisahan *quadtree* untuk memeriksa dan menentukan bertabrakannya 2 buah objek pada game *2D spaceship.*

Berbeda dengan metode lain seperti *priori detection* yaitu pengecekan tumpukan sebelum tumpukan tersebut terjadi dan *post detection* yaitu pengecekan tumpukan setelah tumpukan tersebut terjadi, dan berbeda pula dengan metode *bruteforce* yang mendeteksi semua daerah dari *game* tanpa membagi daerah yang akan dideteksi yang membuat mekanisme tidak optimal dalam pengecekan *collider*.

Bounding Box, atau bisa disebut juga *Bounding Circle*, yang berarti setiap obyek yang direpresentasikan dengan *bounding box* memiliki titik pusat dan radius. Untuk menguji apakah terjadi tumpukan antar dua obyek tersebut yang perlu dilakukan adalah membandingkan jarak antara dua titik pusat dengan jumlah dari dua jari-jari (r1 & r2) (Arif Nurdiyanto, 2018).

Quadtree menggunakan sistem percabangan pohon yang menghasilkan deteksi lebih optimal dengan membagi menjadi 4 bagian kemudian memeriksa bertabrakannya objek berdasarkan bagian tersebut (Steven Lambert, 2012). Membuat performa permainan lebih baik untuk dimainkan, dan metode *quadtree* ini cocok dilakukan pada *2D game* karena bersifat 2 dimensi yang mempunyai wilayah x dan y, berbeda dengan *3D game* yang mempunyai 3 dimensi yaitu x, y, dan z yang lebih cocok dilakukan dengan *octree*

2. LANDASAN KEPUSTAKAAN

2.1 *Spaceship Game*

Spaceship game atau bisa disebut dengan *space shooter game* adalah salah satu genre dari *game*, pemain atau *player* diharuskan untuk mengatur pergerakan karakter dalam *game*, yang bisa menyimpan data (*save*) dan melanjutkan permainan (*load*) (Ansari Aaqib, 2018).

Game ini berbentuk *arcade* yang biasanya *stage oriented* dengan tujuan menyelesaikan misi, tetapi ada juga yang berbentuk *endless* yang bertujuan untuk menjadi *game* kompetitif dengan sistem *scoring*, meski *game space shooter* dianggap klasik namun masih banyak masyarakat yang masih mempermainan *game* bertema *spaceship*

Space Shooter Game banyak dipilih karena hanya mengandalkan ketangkasan untuk menyelesaikan game ini. Apalagi *game* yang sudah berbasis *android* yang memang di tujukan untuk *smartphone* maupun *tablet* yang sekarang banyak digunakan (Lia Musfiroh, 2014).

Gambar 1 *Spaceship game*

2.2 *Collision Detection*

Algoritma *collision detection* adalah proses pengecekan apakah beberapa buah objek spasial saling bertumpuk / bertabrakan 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 (Nugraha, 2013). Pada penelitian ini, penulis menggunakan *obstacle* berupa meteor, *player* berupa pesawat, dan *projectile* berupa tembakan dari *2D Spaceship Game* sebagai objek yang dapat bertumpuk atau *collide*

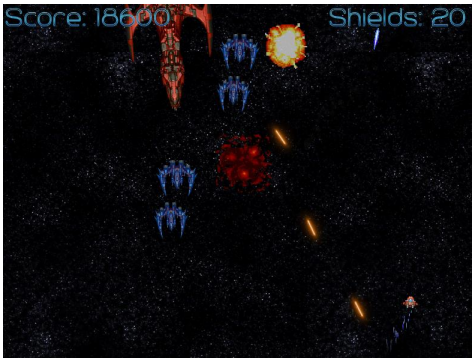
2.2.1 Obstacle

Obstacle yang ada didalam *game* adalah tantangan dalam *game* yang berbentuk halangan, *obstacle* pada game bisa diartikan sebagai penghalang pada medan yang sudah ditetapkan atau disusun yang dapat menghalangi pemain untuk mencapai tujuan atau menang dan mendapatkan sebuah reaksi pada salah satu objek yang ditentukan (Lui Haekal Fasha, 2018). Terlihat pada gambar 2 bahwa *obstacle* telah disusun oleh pembuat game yang berbentuk bundar berwarna hijau

Gambar 2 Obstacle

2.2.2 *Projectile*

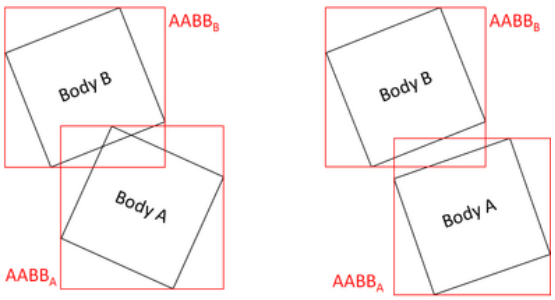
Projectile merupakan salah satu alat praktikum gerak parabola yang dapat digunakan dalam menentukan jangkauan terjauh, tinggi maksimum, dan waktu tempuh peluru dalam gerak parabola (Badru Salam, 2018). *Projectile* pada penelitian ini menggunakan tembakan berisi peluru, seperti yang dapat dilihat pada gambar 3 *projectile* berupa objek pipih berwarna kuning yang keluar dari *player*



Gambar 3 Projectile

2.2.3 *Bounding Box*

Bounding Box merupakan suatu metode dari *Collision Detection* atau pemeriksaan pertumpukan antar dua objek dalam suatu *game*, biasanya objek tersebut merupakan *projectile*, *NPC*, pendeteksian pada *bounding box* terpaku pada sumbu yang sejajar dan dengan cara ini bisa menghemat sumber daya *CPU* untuk pengujian bentuk yang lebih kompleks (Patah Herwanto, 2016).



Gambar 4 *Bounding Box*

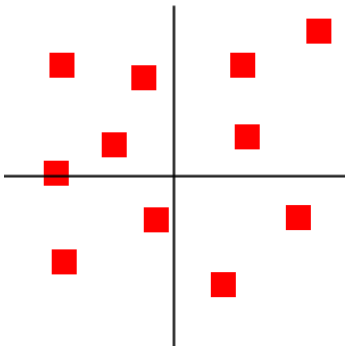
2.3 *Quadtree*

Algoritma *Quadtree* adalah algoritma yang berbentuk persegi yang terbagi empat atau berbentuk pohon *m-ray* yang tiap simpulnya memiliki 4 anak cabang, berbeda dengan algoritma *Octree*, *Quadtree* di implementasikan kepada *game* bergenre *2D*

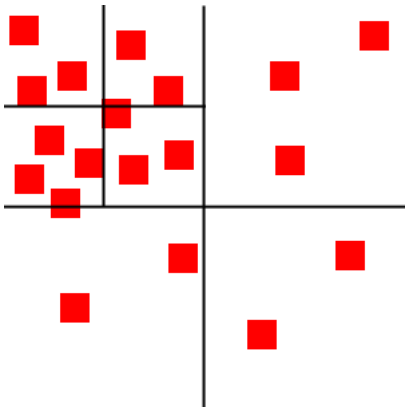
Algoritma ini dapat membagi arena permainan menjadi empat bagian yang berukuran sama. Apabila dua objek atau lebih berada pada satu wilayah yang sama, bagi lagi wilayah tersebut menjadi empat bagian yang berukuran sama dan Pembagian wilayah terus dilakukan hingga masing-masing tidak ada objek yang berada pada wilayah yang sama. Pembagian wilayah ini dilakukan secara rekursif. (Robbi Rahim, 2014).

2.3.1 Metode *Quadtree*

Metode *Quadtree* berawal dengan membagi area *window* menjadi 4 bagian atau *node*, kemudian mendeteksi objek yang ada pada salah satu bagian tersebut dengan membuat batasan berapa objek yang dibolehkan pada satu *node*, jika pada salah satu *node* mempunyai objek lebih dari yang diperbolehkan maka *node* tersebut akan membuat *child node* dengan membagi 4 bagian dari *node parent* tersebut, skema ini akan dibuat secara berulang secara dinamis seperti gambar 5 dan gambar 6



Gambar 5 *Parent Quadtree*



Gambar 6 *Child Quadtree*

Pada gambar telah dijelaskan bahwa objek (gambar kotak berwarna merah) memasuki beberapa *node* (garis hitam yang memisahkan daerah) yang memiliki kapasitas 4 objek, *node* yang mempunyai objek lebih dari kapasitas akan memisah diri menjadi *child node* yang mempunyai 4 wilayah seperti yang dapat dilihat pada gambar 6

2.4 *Bruteforce*

Brute force adalah suatu bentuk algoritma yang lempang (*straightforward*) dalam memecahkan suatu masalah. *Brute force* biasanya didasarkan pada pernyataan masalah dan pemecahan masalah secara intuitif ataupun sesuai konsep. (Ecky Putrady, 2011).

Algoritma Brute force bisa dikatakan metode yang kurang baik atau kurang cerdas, dikarenakan semua objek akan memeriksa satu sama lain yang berbeda jenis

2.4.1 Metode *Bruteforce*

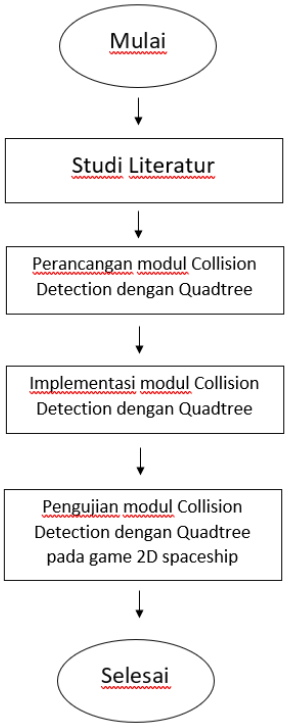
Ketika game dijalankan, metode *Bruteforce* juga akan berjalan dengan cara mengecek semua objek yang ada dengan total jenis objek yang ada, dan pada semua area, tidak hanya pada *window game*, namun juga diluar *window game*

2.5 *Frame Per Second*

FPS atau *Frame per second* adalah gambar yang di ter-*render* dalam hitungan per-detik, semakin banyak gambar yang bisa di *render* dalam satu detik, maka *visual* pada *game* semakin terlihat halus, *Frame rate* sebesar 3 atau 7 akan sangat tidak nyaman dimainkan aau bahkan tidak layak untuk dimainkan, *FPS* yang *playable* dimulai dengan 30 *FPS* sampai dengan 60 *FPS* (Claypool, 2007).

3. METODOLOGI

Pada bab ini akan menjelaskan langkah langah yang akan dilakukan dalam penyususnan skripsi. Penelitian ini bersifat implementatif dengan megimplementasikan *Collision Detection* p ada *2D Spaceship Game* menggunakan *Quadtree*. Metode penelitian yang digunakan adalah :



Gambar 7 Metodologi

Tahap pertama yaitu Studi Literatur, meliputi pencarian ilmu baik dari sumber tertulis seperti buku, jurnal, artikel, dokumen-dokumen relevan dengan permasalahan yang sama. Sehingga solusi yang akan di hasilkan memiliki dukungan yang kuat dari informasi – informasi yang di dapatkan dari studi kepustakaan ini.

Tahap kedua yaitu Perancangan Modul *Collision Detection* dengan *Quadtree* akan dibuat dengan dasar *Quadtree* yang akan di implementasikan pada *2D spaceship game*. Tahapan metode ini membuat algoritma yang berbentuk kode (code) yang dapat diimplementasikan pada *Unity* dengan Bahasa Pemograman *C#*, Setelah implementasi kode selesai, dilanjutkan dengan proses pengujian algoritma dengan tujuan menguji algoritma yang telah dibuat agar sesuai dengan tujuan dari penelitian yang dilakukan.

Tahap ketiga yaitu Implementasi dari Collision Detection yang menggunakan *Quadtree* akan di implementasikan dengan bahasa *C#* yang akan dibuat dan dianalisis pada *Unity*, asset yang akan digunakan sudah dipersiapkan. Langkah awal yang dilakukan adalah membuat algorimta berbentuk kode *Quadtree* untuk di implementasikan pada *Unity* dengan tujuan mendeteksi pertumpukan antara 2 objek.

Tahap keempat yaitu Pada sub bab ini akan dijelaskan tentang cara pengujian daripada penelitian yang dilakukan, pengujian penelitian ini akan berbentuk beberapa tabel dengan tujuan menguji validitas, dan pengujian performa pada *2D Spaceship Game* dengan mencantumkan *FPS* atau *Frame Per Second* ,dan membandingkan pengimplementasian *Quadtree* dalam deteksi pertumpukan atau *Collision Detection* dengan implementasi *bruteforce* pada *2D spaceship game*

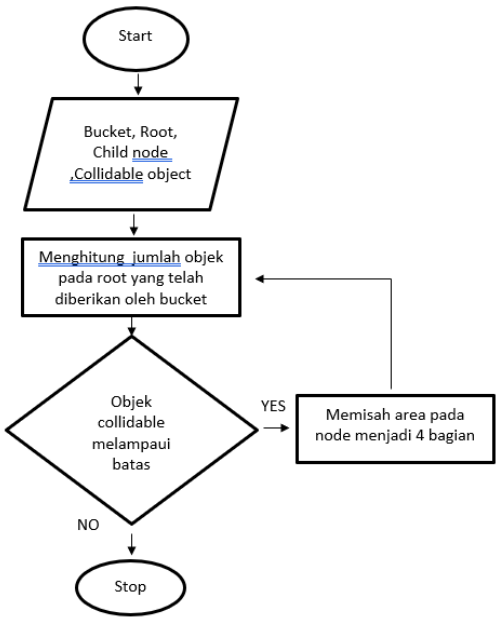
4. PERANCANGAN

4.1 Perancangan *Collision Detection* menggunakan *Quadtree*

mengenai rancangan Quadtree yang akan diimplemetasikan pada Collision Detection yang dapat digunakan pada game 2D Spaceship

4.1.1 Perancangan alur *Quadtree*

Pada sub bab ini akan dijelaskan mengenai bagaimana rancangan pemisahan menggunakan Quadtree pada metode Detection Collision



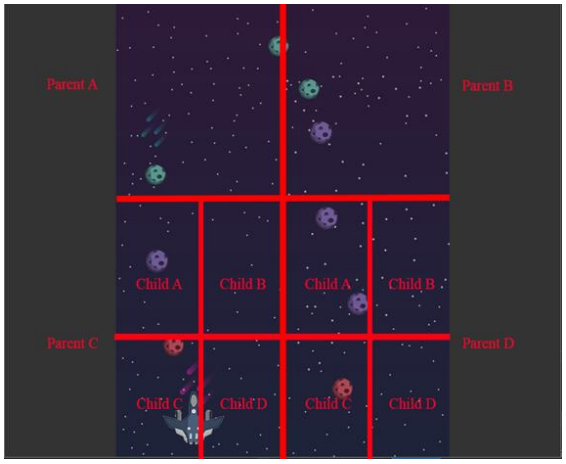
Gambar 8 Flowchart alur Quadtree

Seperti yang dijelaskan di flowchart diatas, untuk membuat pemisahan *quadtree* diperlukan sebuah *collidable* objek, objek disini bisa dibilang seperti *player*, *npc*, ataupun

obstacle, tapi penulis menentukan objek dalam penelitian ini berupa *player*, *projectile*, dan *obstacle*, kemudian dibutuhkan posisi objek untuk mengetahui dan mengatur di *node* mana objek tersebut akan disimpan, dapat dilihat juga pada gambar 5 dan 6, *node* membuat *child node* jika objek pada sebuah *node* melebihi kapasitas yang sudah ditentukan

Posisi disini adalah bagian dari *window game* yang telah dipisah oleh algoritma *quadtree* menjadi 4 dengan ukuran yang sama rata, seperti yang dapat di lihat di gambar 9

Proses dimulai dengan menghitung objek pada node parent, jika objek melebihi dari batas yang telah ditentukan oleh peneliti maka *node parent* akan memisah daerah menjadi 4 yang disebut *node child*, proses ini berulang terus menerus, pada kasus ini peneliti mempunyai batas objek tidak lebih dari 3 objek

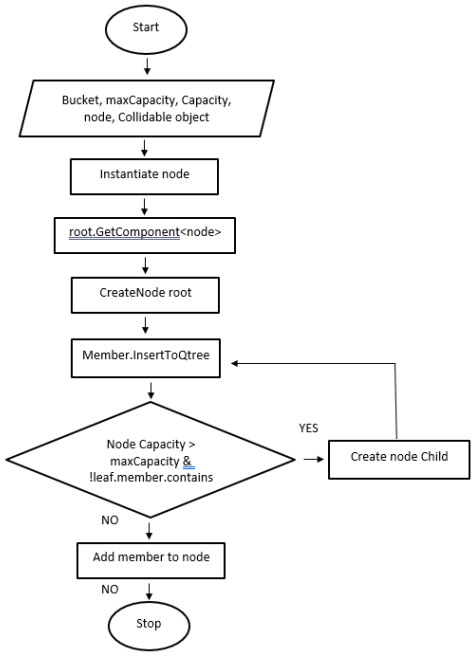


Gambar 9 *Area Quadtree game* Spacecape 2

Pada gambar 5 menjelaskan tentang pembagian area pada *quadtree* yaitu mempunyai *parent* yang memisah 1 layar *window* menjadi 4 bagian yang berarti terbuat 4 *parent* dan memisah lagi jika kapasitas objek melebihi dari yang diperbolehkan, pada gambar 9, wilayah didalam garis merah disebut dengan node sedangkan *obstacle*, *projectile*, dan *player* disebut dengan objek

4.1.1.1 Perancangan *Script Quadtree*

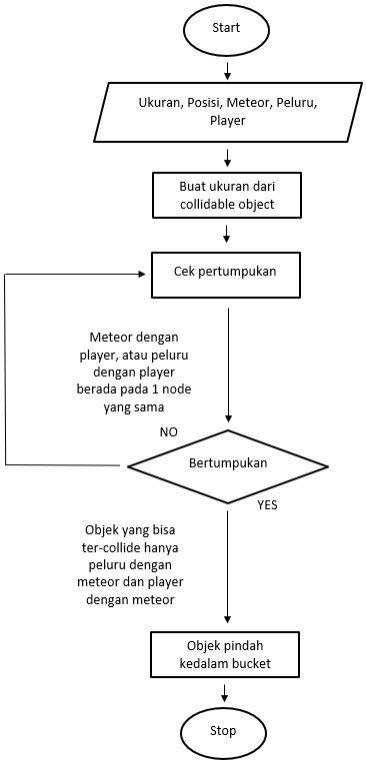
Pada Sub bab ini akan dijelaskan tentang algoritma berjalannya script perancangan pembuatan *node child* atau *node parent* serta pembagian *area* yang akan menjadi basis dari *Quadtree*



Gambar 10 *Flowchart Script Quadtree*

4.1.2 Perancangan alur *Collision Detection*

Pada sub bab Ini akan dibahas bagaimana algoritma *Collision Detection* akan diimplemetasikan pada *game 2D spaceship*



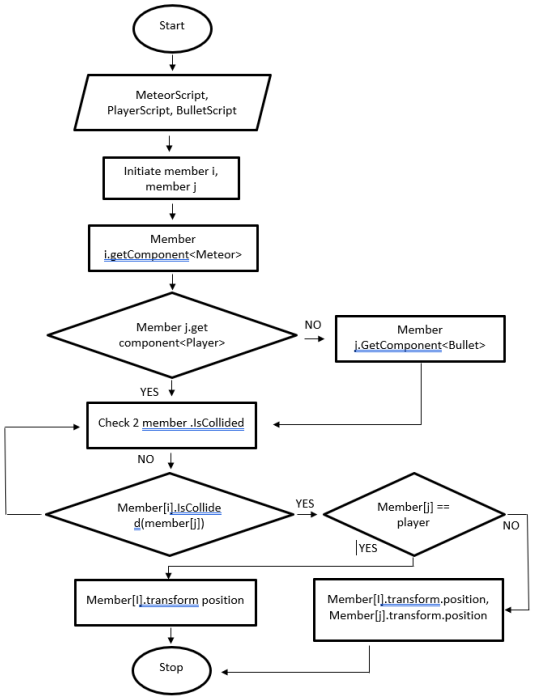
Gambar 11 *Flowchart* Alur *Collision Detection*

Pada flowchart diatas dijelaskan bahwa *Collision Detection* memerlukan sebuah

collidable objek dengan ukuran dan posisi nya, objek yang bisa ter-*collide* antara lain *player* dengan *obstacle* atau *projectile* dengan *obstacle*, setelah itu cek pertumpukan antara *collidable object*,

4.1.2.1 *Perancangan* Script Collision Detection

Pada sub bab ini membahas bagaimana rancangan *script* tentang algoritma *Collision detection*, mulai dari inisiasi sampai dengan *collide*



Gambar 12 *Flowchart Script Collision Detection*

Gambar 12 menjelaskan *flow* dari *Script Collision detection*, yaitu dengan meng inisiasi 2 *member* yaitu *member* i, *member* j, kemudian mengisi *member* i dengan *script* meteor, kemudian menanyakan apakah member j diisi dengan *script* dari *player* atau *script* dari *bullet*, member j hanya mempunyai 2 pilihan, memiliki isi dari *script player* atau *script bullet*,

kedua pilihan tersebut nantinya akan di cek apakah ter-*collide* dengan member i yang berisi *script* meteor, jika tidak maka akan terus mengecek pertumpukan objeknya, jika 2 *member* tersebut *collide*, maka akan menanyakan apakah member j berisi *player* atau *bullet*, jika berisi *player* maka member i akan kembali ke *area bucket*, jika member j berisi *bullet* maka *member* i dan j akan kembali ke *area bucket*

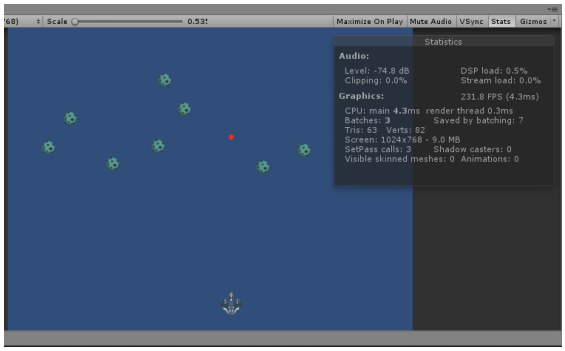
5. PENGUJIAN

5.1 Pengujian *Stress Testing* dengan pengaplikasian *Quadtree*

Pada bab ini akan dilakukan pengujian dengan cara *stress testing* yaitu melihat *FPS* atau *frame per second* dari *Collision Detection* yang di implementasikan kedalam *Quadtree* dengan menambahkan objek meteor dari objek *collidable*, penulis akan melakukan 3x pengujian, dengan 3 jumlah objek *collidable* yang berbeda pada area *quadtree*

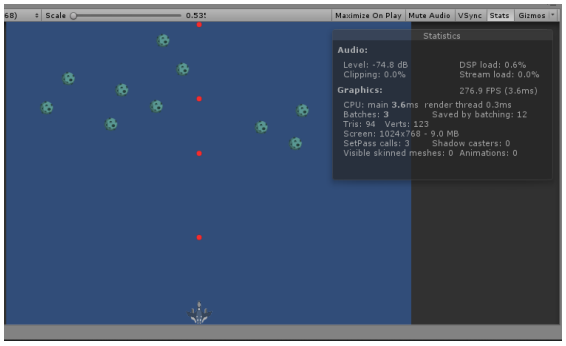
Tabel 1 Pengujian *Stress Testing* dengan *pengaplikasian Quadtree*

Pengujian ke-	**Jumlah Objek *Collidable***	***Frame per Second***
1	10	231.8 fps
2	15	276.6 fps
3	20	295.8 fps



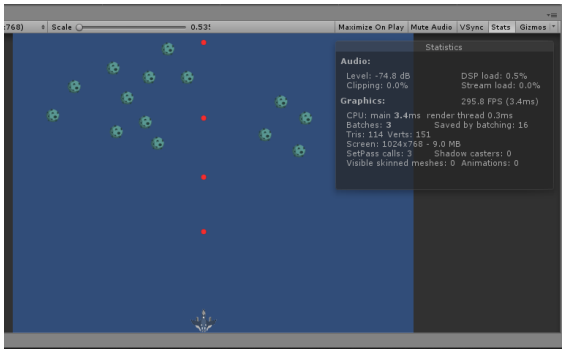
Gambar 13 Pengujian 10 *collidable* object menggunakan *quadtree*

Pada gambar 13 terlihat bahwa ada sebanyak 10 *collidable object* yang berada pada area *quadtree*, dan dapat dilihat juga hasil jadi *FPS* yaitu 231.8 *frame per second*



Gambar 14 Pengujian 15 *collidable* object menggunakan *quadtree*

Pada gambar 14 pengujian dilakukan dengan 15 *collidable object* pada waktu yang bersamaan, hasil *FPS* menunjukan di angka 276.9 *frame per second*



Gambar 15 Pengujian 20 *collidable object* menggunakan *quadtree*

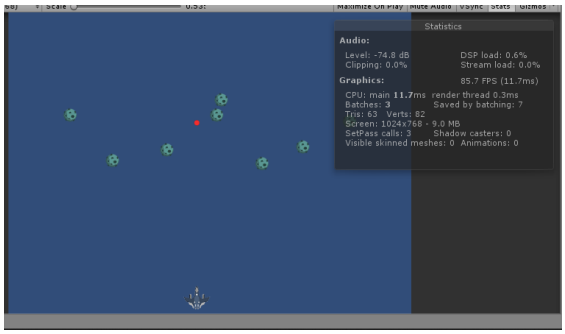
Pada gambar 15, *collidable object* semakin banyak yaitu ada 20 objek secara bersamaan masuk kedalam area *quadtree*, *FPS* menunjukan di angka 295.8 *frame per second*

5.2 Pengujian *Stress Testing dengan Bruteforce*

Pada sub bab ini, pengujian dilakukan dengan cara *bruteforce*, yaitu metode pemeriksaan *collision* dengan cara memeriksa seluruh area pada *game*, pengujian dilakukan sebanyak 3x, dengan 3 jumlah *collidable object* yang berbeda, sehingga dapat menghasilkan *FPS* atau *frame per second* yang berbeda beda yang akan dibandingkan

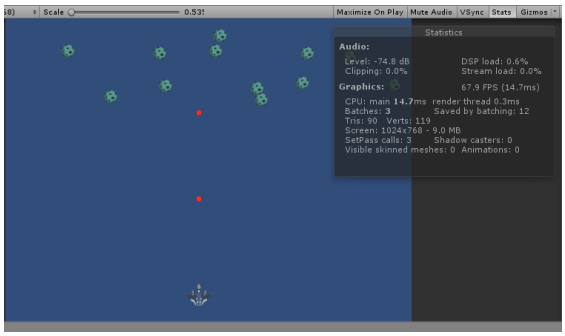
Tabel 2 Pengujian *Stress Testing* dengan *bruteforce*

Pengujian ke-	Jumlah objek *Collidable*	*Frame per Second*
1	10	85.7
2	15	67.9
3	20	56.6



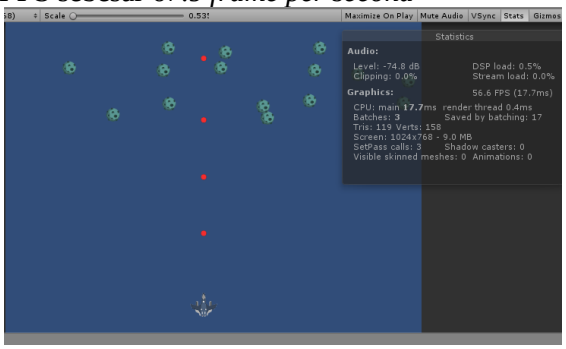
Gambar 16 Pengujian 10 collidable object menggunakan bruteforce

Pada gambar 16 dijelaskan bahwa didapat hasil *FPS* sebesar 85.7 *frame per second* dengan jumlah object sebanyak 10 dalam waktu bersamaan



Gambar 17 Pengujian 15 collidable object menggunakan bruteforce

Pada gambar 17 dapat dilihat ada 15 *collidable object* pada area *game*, dan mendapatkan hasil *FPS* sebesar 67.9 *frame per second*



Gambar 18 Pengujian 20 *collidable* object menggunakan *bruteforce*

Pada gambar 18 ini terdapat 20 *object collidable* di area *game*, mendapatkan hasil *FPS* 56.6 *frame per second*

6. KESIMPULAN DAN SARAN

6.1 Kesimpulan

Pada penelitian optimasi *collision detection* menggunakan *quadtree* pada *2D spaceship game* sudah berhasil dilaksanakan dan penulis telah menarik kesimpulan sebagai berikut :

1. Algoritma *collision detection* yang diterapkan pada algoritma *quadtree* dengan cara memanggil algoritma *collision detection* hanya pada saat *node* atau *leaf* pada *quadtree* mempunyai member berisi *collidable object*, yaitu antara *obstacle* dengan *projectile* dan/atau *player* dengan *obstacle*, Semakin banyak objek pada area *quadtree*, semakin banyak pula pembagian area maka semakin mengecil area dari setiap *node* yang diperiksa *collision* atau pertumpukannya, maka semakin cepat pula sebuah objek masuk dan keluar dari area *node* tersebut. Dikarenakan *collision detection* dilakukan hanya pada suatu *node* tertentu saja, maka semakin kecil wilayah *node* dan semakin singkat deteksi pertumpukan objek sehingga mempersedikit penggunaan *resource* yang menyebabkan kenaikan *frame per second.*, Metode ini juga mempunyai kelemahan yaitu dalam peletakan *collidable object*, karena metode ini menekankan tentang pemisahan daerah, jika ada *collidable object* yang berada tepat di 2 *node*, *collidable object* tersebut tidak terdeteksi sampai objek tersebut telah berpindah *area*

2. Pengujian dilakukan dengan perbandingan performa *FPS* atau *Frame per second* antara metode *collision detection* yang dimasukan kedalam algoritma *quadtree* dengan metode *collision detection* yang langsung di aplikasikan kedalam game

6.2 Saran

Pada penelitian optimasi *collision detection* menggunakan *quadtree* pada *2D spaceship game* , penulis memberikan saran sebagai berikut :

1. Menggunakan metode *collision detection* yang diterapkan kepada metode *quadtree* untuk pembuatan *game 2D* selain *game* ber-*genre spaceship*, dikarenakan lebih optimal dibandingkan dengan metode *collision detection* yang langsung di implementasikan kedalam *game.*

2.Menggunakan metode *collision detection* selain *bounding box* untuk diimplementasikan kepada *quadtree* sebagai perbandingan keoptimalan daripada metode yang telah dilakukan di penelitian ini

3. Untuk penelitian selanjutnya, disarankan untuk mengganti algoritma tentang pemasukan *member* yang berisi *collidable object* dengan tujuan memperbaiki kelemahan algoritma yang digunakan pada penelitian ini

7. DAFTAR PUSTAKA

. C. H. Chien and J. K. Aggarwal, A normalized quadtree representation, Comput. Vision Graphics Image Process. X,1984,331-346.

Claypool, K. T., & Claypool, M. (2007). On frame rate and player performance in first person shooter games. *Multimedia Systems*, *13*(1), 3–17. https://doi.org/10.1007/s00530-007-0081-1

Fasha, L. H., & Gufroni, M. (2018). IMPLEMENTASI ALGORITMA COLLISION DETECTION PADA, *3*(1), 58–65.

G. Schrack, “Finding neighbors of equal size in linear quadtrees and octrees in constant time,” CVGIP Image Underst., vol. 55, no. 3, pp. 221–230, 1992.

H. Samet, “Neighbor finding techniques for images represented by quadtrees,” Comput. Graph. Image Process., vol. 18, no. 1, pp. 37–57, 1982.

Hosam, O. (2015). Developing Simple Games with OpenGL, (October). https://doi.org/10.13140/RG.2.1.1685.8324

J. Vörös, “A strategy for repetitive neighbor finding in images represented by quadtrees,” Pattern Recognit. Lett., vol. 18, no. 10, pp. 955– 962, 1997

K. Aizawa, K. Motomura, S. Kimura, R. Kadowaki, and J. Fan, “Constant time neighbor finding in quadtrees: An experimental result,” in Communications, Control and Signal Processing, 2008. ISCCSP 2008. 3rd International Symposium on, 2008, pp. 505–510.

Kumar Sharma, D., & Vatta, S. (2015). Optimizing the Search in Hierarchical Database using Quad Tree, *4*(4), 221–226.

Lin, M., & Gottschalk, S. (1998). Collision detection between geometric models: A survey. *Proceedings of IMA Conference on Mathematics of Surfaces*, 1–20. https://doi.org/10.1.1.45.3926

Musfiroh, L., Jazuli, A., & Latubessy, A. (2014). Penerapan Algoritma Collision Detection Dan Boids Pada Game Dokkaebi Shooter. *Prosiding SNATIF*, 217–224. Retrieved from http://jurnal.umk.ac.id/index.php/SNA/article/view/148/148

Putrady, E. (2011). Optimasi Collision Detection Menggunakan Quadtree.

Rahim, R. (2014). Algoritma Quadtree Untuk Pendeteksian Tubrukan Pada Permainan Adventure of Upik, *III*(2), 13–18.

Salmon, R. A., Armstrong, M. G., & Jolly, S. J. E. (2011). Higher Frame Rates for More Immersive Video and Telvision. *BBC Research White Paper 209*, (October).

Steven Lambert. (2012). QUICK TIP: USE QUADTREES TO DETECT LIKELY COLLISIONS IN 2D SPACE https://gamedevelopment.tutsplus.com/tutorials/quick-tip-use-quadtrees-to-detect-likely-collisions-in-2d-space--gamedev-374 [diakses 4 April 2019]

Studi, P., Informatika, T., Informasi, F. T., & Stikubank, U. (2018). PENERAPAN METODE COLLISION DETECTION PADA GAME, 978–979.

Vanderhye, P. C. (2001). E O S T H 0 C N | W - E X S H J NW L, *1*(12).

Wicaksono, A., Hariadi, M., & N, S. M. S. (2013). STRATEGI MENYERANG NPC GAME FPS MENGGUNAKAN FUZZY FINITE STATE MACHINE, 25–30.

Zhou, M. (2014). Project Report : Collision Detection using Quadtree, 1–4.