BAB III

ANALISIS DAN PERANCANGAN

3.1 Analisis Sistem

Analisis sistem merupakan penguraian dari suatu sistem yang utuh ke dalam bagian-bagian komponennya dengan maksud untuk mengidentifikasikan dan mengevaluasi permasalahan-permasalahan, kesempatan dan hambatan - hambatan yang terjadi dan kebutuhan yang diharapkan sehingga dapat diusulkan perbaikan-perbaikan.

Pembangunan perangkat lunak terdiri dari beberapa tahap, salah satunya adalah tahap menganalisa sistem. Sistem yang akan dianalisis untuk membangun perangkat lunak permainan *snake* menggunakan algoritma *branch and bound*. Analisis sistem yang akan dibahas mengenai cara kerja algoritma *branch and bound* yang diimplementasikan pada permainan *snake*.

3.1.1 Analisis Masalah

Sejatinya permainan ini merupakan *single player game* atau permainan yang dimainkan sendiri. Dalam permainan ini komputer hanya memunculkan ayam secara random di layar untuk dimakan. Kesuksesan permainan ini bergantung kepada kecepatan dan perhitungan sang pemain agar ular yang dikontrolnya tidak terjebak dinding atau bagian tubuhnya sendiri.

3.2 Analisis Kebutuhan Fungsional

3.2.1 Identifikasi Aktor

Tahap pertama yang dilakukan dalam melakukan analisis berorientasi objek menggunakan UML adalah menentukan aktor atau pengguna sistem. Kata aktor dalam konteks UML, menampilkan peran (*roles*) yang pengguna (atau sesuatu diluar sistem yang dikembangkan yang dapat berupa perangkat keras, *end user*, sistem yang lain, dan sebagainya).

3.3 Analisis Kebutuhan Non Fungsional

Analisis kebutuhan non fungsional adalah sebuah langkah dimana seorang pembangun perangkat lunak menganalisis sumber daya yang akan menggunakan perangkat lunak yang dibangun. Analisis Kebutuhan Non Fungsional untuk penerapan algoritma *branch and bound* pada permainan *snake*, yaitu :

1. Analisis Pengguna
2. Analaisis Perangkat Lunak
3. Analisis Perangkat Keras

3.3.1 Analisis Pengguna

Perangkat keras dan perangkat lunak yang ada tidak akan berguna apabila tidak ada perangkat pikir yang mengoperasikannya. Perangkat pikir yang terlibat dalam sistem adalah pemain/*player* untuk memainkan permainan *snake*.

3.3.2 Analisis Perangkat Lunak

Kebutuhan perangkat lunak yang digunakan untuk bisa mengoperasikan aplikasi yang akan dibangun adalah :

a. Windows 7 Service Pack 1

b. Adobe Flash Player

3.3.3 Analisis Perangkat Keras

Perangkat keras yang digunakan untuk memainkan permainan ini antara lain adalah :

a. Processor Intel Core 2 Quad 2.5 GHz

b. RAM 8 Gb

c. Harddisk 500 Gb

d. CD/DVD-Rom

e. Monitor

f. Keyboard and Mouse

g. Speaker

3.3.4 Pembangunan Aplikasi Game Snake

3.3.4.1 Gameplay

Tugas utama pemain (*gamer*) dalam memainkan Aplikasi ini yaitu bertahan hidup agar pemain tidak menabrak dinding atau tubuh sendiri, pemain disini digambarkan dengan karakter ular. Pemain dapat memakan ayam sebanyak-banyaknya sehingga pemain memperoleh *score* (nilai).

Pemain (*gamer*) dapat memulai permainan dengan memilih pilihan "Mulai" dimana nantinya pemain dapat langsung memainkan ular (*snake*), dan juga disediakan pilihan "Nilai Tertinggi" yaitu pilihan untuk melihat *score-score* yang sudah tercetak sebagai hasil dari permainan yang telah dilakukan oleh pemain dan pilihan lainnya adalah pilihan "Pembuat", sebagai keterangan yang

menyediakan informasi mengenai pembuat *game* ini, dan juga pemain dapat memilih "Cara Main", sebagai keterangan bagaimana cara bermain *game snake* ini.

3.3.5 Sasaran User dan Kegunaan Aplikasi

Game yang akan dibuat ditujukan untuk *user* semua umur, alasannya adalah untuk menghibur atau sebagai sarana refreshing untuk mengurangi kejenuhan dalam bekerja atau berkegiatan lainnya.

Game yang dibuat merupakan *game* PC (*Personal Computer*) atau bisa disebut juga *game desktop*.

3.3.6 Analisis Penggunaan Algoritma Branch and Bound pada game Snake

3.3.6.1 Metode

Algoritma yang dipakai dalam pemodelan ini adalah algoritma *branch and bound*. Algoritma ini sejatinya adalah algoritma pencarian solusi dengan pencarian melebar atau *breadth first search* (BFS). Dalam algoritma BFS solusi dicari dengan membentuk pohon ruang status yang merupakan pohon dinamis.

Simpul-simpul di dalam pohon diinamis yang memenuhi kendala menyatakan status persoalan. Suatu operator mentransformasikan persoalan dari sebuah status ke status yang lain. Solusi persoalan dinyatakan dengan satu atau lebih status yang disebut status solusi. Status solusi yang merupakan simpul daun disebut status tujuan. Himpunan semua status solusi disebut ruang solusi. Seluruh simpul di dalam pohon dinamis disebut ruang status. Dan pohonnya dinamakan juga pohon ruang status. Akar pada pohon ruang status menyatakan status awal sedangkan daun menyatakan status solusi.

BFS mencari solusi persoalan pada pohon ruang status yang dibentuk secara dinamis dengan cara semua simpul pada aras *d* dibangkitkan terlebih dahulu sebelum simpul-simpul pada aras *d*+1. Simpul BFS memerlukan sebuah antrian untuk menyimpan simpul-simpul yang akan dibangkitkan. Simpul-simpul yang dibangkitkan disimpan di belakang antrian.

3.3.6.2 Algoritma *Branch and Bound*

Pada algoritma *Branch and Bound*, pencarian ke simpul solusi dapat dipercepat dengan memilih simpul hidup berdasarkan nilai ongkos (*cost*). Setiap simpul hidup diasosiasikan dengan sebuah ongkos yang menyatakan nilai batas (*bound*). Batas ini dapat berupa batas bawah (*lower bound*) ataupun batas atas (*upper bound*) masing-masing menyatakan batas maksimum atau batas minimum yang diperlukan untuk membangkitkan sebuah simpul.

Pemberian nilai batas akan ideal jika kita mengetahui letak simpul solusi, namun untuk kebanyakan persoalan letak simpul solusi tidak diketahui. Sehingga nilai batas untuk setiap simpulnya umumnya hanya berupa taksiran ataupun perkiraan.

Untuk membantu pemberian taksiran nilai umumnya digunakan fungsi heuristik yang dapat berupa fungsi apapun asalkan fungsi tersebut mampu memodelkan ongkos (*cost*) untuk membantu mencapai simpul solusi. Fungsi heuristik untuk menghitung taksiran nilai tersebut dinyatakan secara umum sebagai:

$$c(i) = f(i) + g(i)$$

dengan

c(*i*) = ongkos untuk simpul *i*

f(*i*) = ongkos mencapai simpul *i* dari akar

g(*i*) = ongkos mencapai simpul tujuan dari simpul (*i*)

Nilai c digunakan untuk mengurutkan pencarian. Jika pencarian adalah pencarian dengan biaya terkecil maka simpul berikutnya yang dipilih untuk diekspansi adalah simpul yang memiliki c minimum. Bila sebaliknya, simpul yang diekspansi adalah simpul yang memiliki c maksimum.

Skema umum algoritma *branch and bound* adalah sebagai berikut :

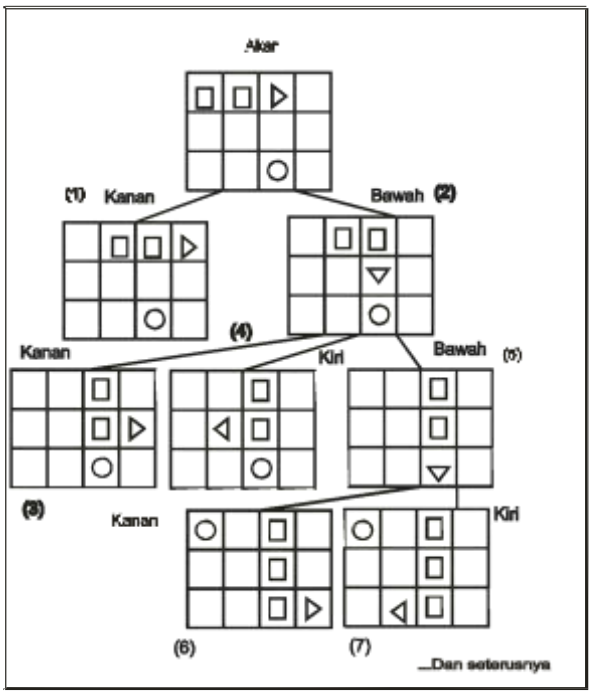
1. Masukkan simpul akar ke dalam antrian Q. Jika simpul akar adalah simpul solusi maka solusi telah ditemukan. Stop.
2. Jika Q kosong, tidak ada solusi. Stop.
3. Jika Q tidak kosong, pilih dari antrian Q simpul *i* yang mempunyai c(*i*) paling kecil (jika *least cost search*) atau c(*i*) paling besar (jika *most cost search*)
4. Jika simpul *i* adalah simpul solusi, berarti solusi sudah ditemukan, stop. Jika simpul *i* bukan simpul solusi, maka bangkitkan semua anakanaknya. Jika *i* tidak mempunyai anak, kembali ke langkah 2.
5. Untuk setiap anak *j* dari simpul *i,* hitung c(*j*) dan masukkan semua anak- anak tersebut ke dalam antrian Q.
6. Kembali ke langkah 2.

3.3.6.3 Modifikasi *Branch and Bound*

Untuk memodelkan perilaku *snake* ini diperlukan beberapa modifikasi karena algoritma *branch and bound* murni belum memadai untuk memodelkan perilaku *snake*.

Modifikasi pertama adalah tujuan (*goal*) dari algoritma tersebut. Pada *branch and bound* murni, algoritma mencari simpul tujuan dengan sesedikit mungkin membangkitkan simpul anak. Sedangkan tujuan dari permainan ini adalah kita bertahan selama mungkin dalam permainan *snake* ini sehingga tujuan utama dalam algoritma ini adalah membuat simpul sedalam mungkin.

Namun simpul yang dibangkitkan harus menjamin bahwa *snake* tidak mati. Simpul yang dibangkitkan pula harus menjamin bahwa *snake* berhasil mendapatkan ayam dengan jumlah langkah yang sesedikit mungkin.



Gambar 3.1 Pembangkitan simpul yang diinginkan dalam permainan *snake*

Dalam gambar ini diperlihatkan pembangkitan simpul yang diinginkan dalam permainan *snake*. Simbol kotak melambangkan bagian tubuh dari *snake*. Simbol segitiga melambangkan kepala dan arah gerak *snake*. Simbol lingkaran melambangkan ayam yang harus diambil *snake*.

Pada akar terlihat kondisi awal saat permainan pertama dimulai. Simpul yang bisa dibangkitkan adalah simpul dengan gerak ular selanjutnya adalah ke kanan atau ke bawah karena simpul dengan gerak ular ke atas atau ke kiri dapat mengakibatkan ular mati.

Pada aras kedua simpul yang ingin diteruskan adalah simpul dengan gerak ular ke bawah karena ayam berada di bawah sehingga simpul tersebut merupakan awal dari simpul dengan ongkos termurah untuk mencapai ayam. Kemudian pada aras terakhir dengan model AI ini ular akan bergerak ke kanan dan seterusnya sehingga ular tidak bisa bergerak lagi.

3.3.6.4 Fungsi Heuristik yang Digunakan

Untuk memodelkan perilaku yang diinginkan seperti di atas diperlukan fungsi pembangkitan ongkos yang sesuai. Fungsi pembangkitan ongkos ini merupakan kunci dari algoritma *branch and bound* karena algoritma ini akan membangkitkan simpul sesuai dengan ongkos simpul tersebut.

Dalam hal ini fungsi heuristik yang digunakan adalah :

$$\mathbf{c}(i) = \mathbf{f}(i) + \mathbf{g}(i) + \mathbf{h}(i)$$

dengan

c(*i*) = ongkos untuk simpul ***i***

f(*i*) = jarak dari kotak yang dituju dengan simpul *i* ke ayam (jarak dihitung dengan rumus jarak dua titik dalam bidang koordinat kartesius)

g(*i*) = ongkos jika simpul ***i*** mengarah ke dinding atau kebagian tubuh dari ular, bernilai sangat besar karena kita ingin menghindari tabrakan

h(*i*) = adalah ongkos yang dibangkitkan jika ular masuk ke wilayah yang dibuatnya.

Simpul yang dibangkitkan adalah simpul dengan **c(*i*)** terkecil.

Dalam permainan ular ini setiap kotak mempunyai koordinat tertentu dengan ketentuan sebagai berikut :

0,0	0,1	0,2	–
1,0			
–			

Gambar 3.2 Aturan koordinat bidang permainan *snake*

Sehingga pada aras kedua di gambar 1 nilai f(*i*) untuk bergerak ke kanan adalah

$$\mathbf{f(1)} = \sqrt{(2-0)^2} + (2\sqrt{-3})^2 = 5$$

namun karena nilai akar cenderung mempersulit perhitungan maka ongkos untuk **f(2)** dapat dikuadratkan sehingga menjadi

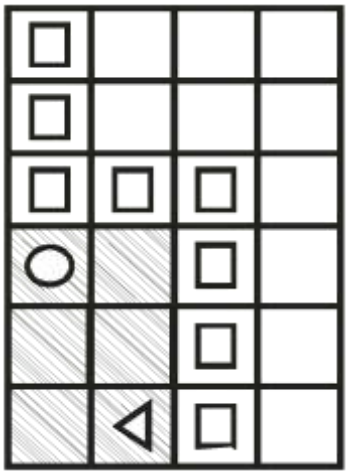
f(1) = 5

Untuk nilai **g(1)** adalah 0 karena simpul **g(1)** tidak menabrak dinding. Namun, tinjau apabila simpul yang bergerak ke atas dicoba dibuat dari simpul

akar maka nilai **g** untuk simpul tersebut adalah 99999 nilai ini sangat besar karena kita tidak ingin ular bergerak menabrak dinding. Nilai **g** untuk simpul yang akan menabrak tubuh ular juga akan bernilai 99999.

Untuk nilai **h** tinjau simpul **7** pada simpul tersebut ular sudah membuat wilayah yaitu segi empat dari koordinat (0,0) sampai koordinat (2,1). Nilai **h** akan dibangkitkan berdasarkan wilayah yang sudah dibuat tersebut. Pembangkitan nilai **h** ini harus dilakukan dengan hati-hati karena kita tidak ingin ular terkurung dalam wilayah yang dibuatnya sendiri. Jika ular sudah terkurung maka ular akan mati dan hal ini kita hindari.

Untuk definisi lengkap pembangkitan nilai **h** tinjau kasus berikut :



Gambar 3.3 Contoh kasus pembangkitan nilai h

Pada kasus ini ular sudah membuat wilayah yaitu kotak-kotak yang diarsir. Jumlah wilayah yang dimiliki adalah **9** karena perhitungan wilayah berdasarkan kuadrat panjang atau lebar tertinggi. Hal ini agar gerakan ular selalu mengeliminasi wilayah dengan gerakan spiral. Untuk gerakan selanjutnya, nilai **h** harus dihitung. Nilai **h** dihitung berdasarkan perhitungan berikut:

$$\mathbf{h(}\boldsymbol{i}\mathbf{) = l - j(}\boldsymbol{i}\mathbf{) + k(}\boldsymbol{i}\mathbf{)}$$

dengan

j(*i*) = jarak dari kotak yang dicapai oleh simpul ***i*** dengan ekor ular

k(*i*) = wilayah yang dibuat jika ular bergerak ke simpul ***i***

l = panjang badan ular

Pada kasus seperti gambar tiga simpul yang bisa dibangkitkan adalah simpul dengan arah gerak ke kiri atau ke atas mari kita tinjau *cost* untuk kedua simpul tersebut.

f(atas) = 2

g(atas) = 0

h(atas) = 9-16 + 9

c(atas) b f(atas) + **g(atas)** + **h(atas) =** 4

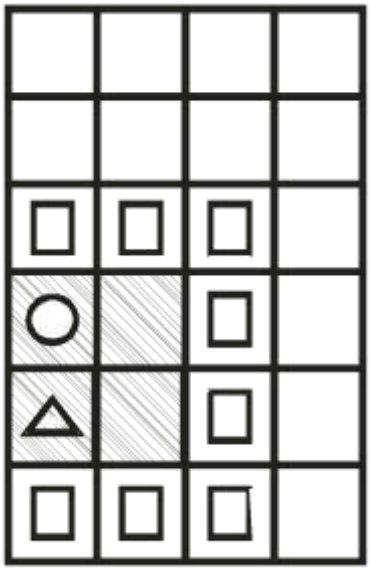
f(kiri) = 2

g(kiri) = 0

h(kiri) = 9-16 + 4

c(kiri) = **f(kiri)** + **g(kiri)**+**h(kiri)** = 1

Dari perhitungan tersebut ular akan bergerak ke kiri. Dari sana simpul yang bisa dibangkitkan hanyalah simpul dengan gerakan ular ke atas, maka ular akan ke atas. Sehingga posisi ular akan seperti :



Gambar 3.4 Contoh kasus pembangkitan nilai h

Dari posisi ini ular bisa bergerak ke atas atau ke kanan. Namun bila ular bergerak ke atas maka ular akan mati karena ular akan bertambah panjang bila memakan ayam dan akan terbentur oleh tubuhnya sendiri. Dalam dua kasus ini tinjau perhitungan berikut:

f(atas) = 0

g(atas) = 0

h(atas) = 9-1 + 4

c(atas) = f(atas) + g(atas) + h(atas) = 12

f(kanan) = 2

g(kanan) = 0

h(kanan) = 9 – 5 + 4

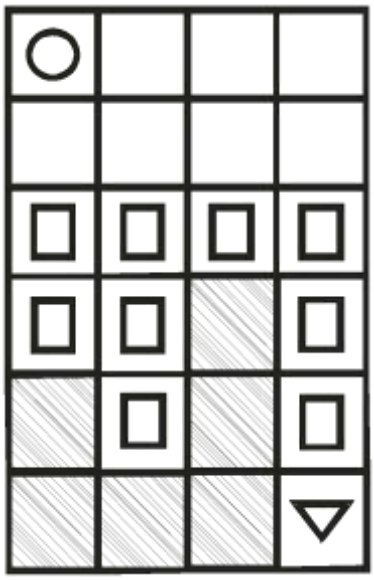
c(kanan) = f(kanan) + g(kanan) + h(kanan) = 10

Dengan perhitungan seperti ini, ular akan bergerak ke arah kanan dan ular akan selamat dari jebakan. Selanjutnya ular akan bergerak ke atas kemudian ke kanan.

Fungsi heuristik ini cukup efektif untuk menghindari kejadian seperti ini. Namun tidak selamanya fungsi **h** dihitung karena fungsi **h** akan dihitung jika dan hanya jika ular sudah mempunyai wilayah. Ular memiliki wilayah jika dan hanya jika dia sudah "mengurung" sejumlah kotak atau dengan kata lain bagian tubuhnya sudah menempel dengan dua dinding yang tidak berseberangan. Untuk kasus belum memiliki wilayah fungsi **f** dan **g** sudah cukup mewakili.

3.3.6.5 PENGETESAN

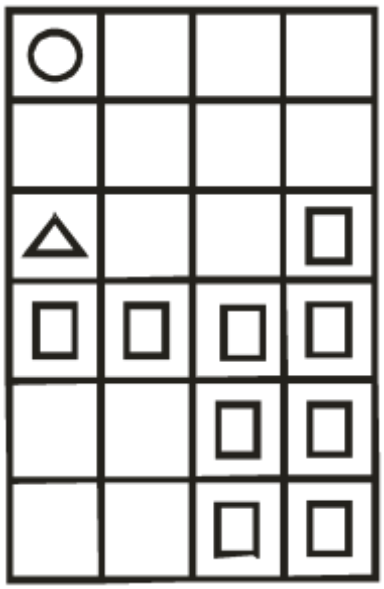
Pengetesan lebih lanjut dilakukan dengan kasus berikut :



Gambar 3.5 Contoh kasus untuk pengetesan

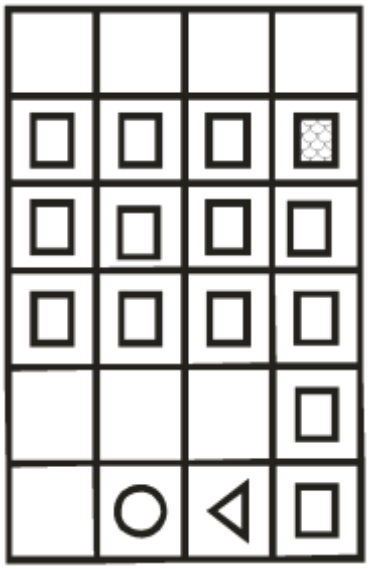
Dengan algoritma ini langkah-langkah ular seperti berikut : Langkah pertama ular akan bergerak ke kiri karena itu adalah satu-satunya langkah yang

mungkin. langkah berikutnya adalah ke atas, lalu ke atas lagi, kemudian ke kiri, lalu ke kiri, kemudian ke atas sehingga status ular tersebut akan seperti :



Gambar 3.6 Hasil pengetesan

Namun untuk panjang ular yang sangat besar, algoritma ini gagal memberikan hasil yang memuaskan karena langkah-langkah awal sebelum ular memiliki wilayah mengakibatkan ular terkurung di wilayahnya sehingga apapun kombinasi pergerakan tidak dapat membuat ular dapat keluar dari wilayah tersebut. Hal itu ditemukan dalam kasus berikut :



Gambar 3.7 Contoh kasus gagal

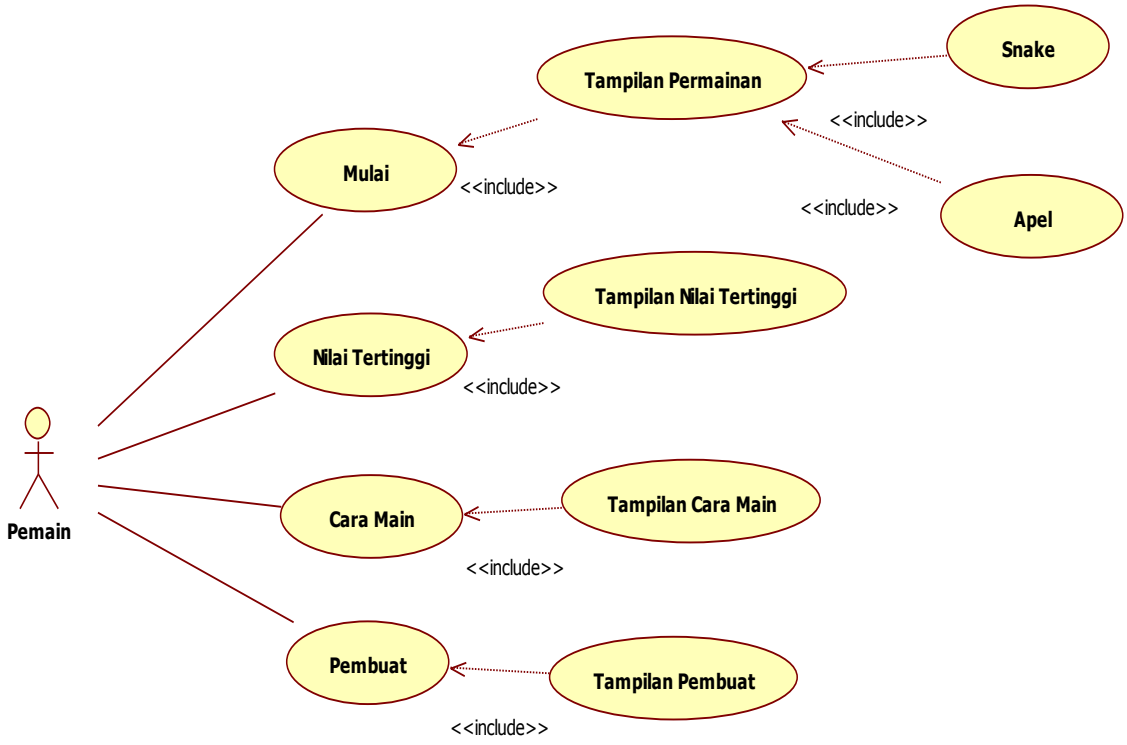
Algoritma ini tidak bisa mencegah agar ular tidak mengalami keadaan seperti ini karena panjang ular yang besar seringkali mengganggu perhitungan dan dalam kasus belum memiliki wilayah algoritma ini hanya mencari jarak terdekat ke ayam sehingga gerakan pada awal-awal terkesan “ceroboh” karena tidak memikirkan akibat dari gerakan tersebut.

3.4 Perancangan

Perancangan ini bertujuan untuk menerapkan solusi pemecahan masalah yang telah diajukan pada analisis sistem.

3.5 Pendefinisian Fungsi Game

Berikut ini adalah perancangan proses-proses yang terdapat pada *game Snake*:



Gambar 3.8 Use Case Diagram Tampilan Utama (User)

3.6 Definisi Use Case

Definisi Use Case berfungsi untuk menjelaskan proses yang terdapat pada setiap Use Case.

Tabel 3.1 Definisi Use Case

NO	Use Case	Deskripsi
1	Mulai	Proses untuk memulai permainan aplikasi
2	Nilai Tertinggi	Proses untuk melihat nilai tertinggi pada game tersebut
3	Cara Main	Proses untuk melihat cara bermain aplikasi game tersebut
4	Pembuat	Proses untuk melihat informasi pembuat aplikasi game tersebut

3.7 Skenario Use Case

Skenario setiap bagian pada use case menunjukkan proses apa yang terjadi pada setiap bagian didalam use case tersebut, dimana user memberikan perintah pada setiap bagian dan respon apa yang diberikan oleh sistem kepada user setelah user memberikan perintah pada setiap bagian – bagian use case.

Tabel 3.2 Scenario Use Case Pilihan Mulai

Identifikasi	
Nomor	1
Nama	Mulai
Tujuan	Memulai Permainan Aplikasi
Deskripsi	Proses untuk memulai permainan aplikasi
Aktor	Pemain

Skenario Utama	
Kondisi awal	Pengguna berada di menu utama aplikasi game Snake
Aksi Aktor	Reaksi Sistem
Memilih menu Mulai	Aplikasi merespon dengan cara masuk ke sistem dari aplikasi permainan
Kondisi akhir	Aplikasi masuk ke dalam permainan Snake

Tabel 3.3 Scenario Use Case Pilihan Nilai Tertinggi

Identifikasi	
Nomor	2
Nama	Nilai Tertinggi
Tujuan	Melihat Nilai Tertinggi pada permainan
Deskripsi	Proses untuk melihat nilai tertinggi pada permainan Snake
Aktor	Pemain
Skenario Utama	
Kondisi awal	Pengguna berada di menu utama aplikasi game Snake
Aksi Aktor	Reaksi Sistem
Memilih menu Nilai Tertinggi	Aplikasi merespon dengan cara masuk ke tampilan nilai tertinggi

Kondisi akhir	Aplikasi masuk ke dalam informasi nilai tertinggi permainan Snake

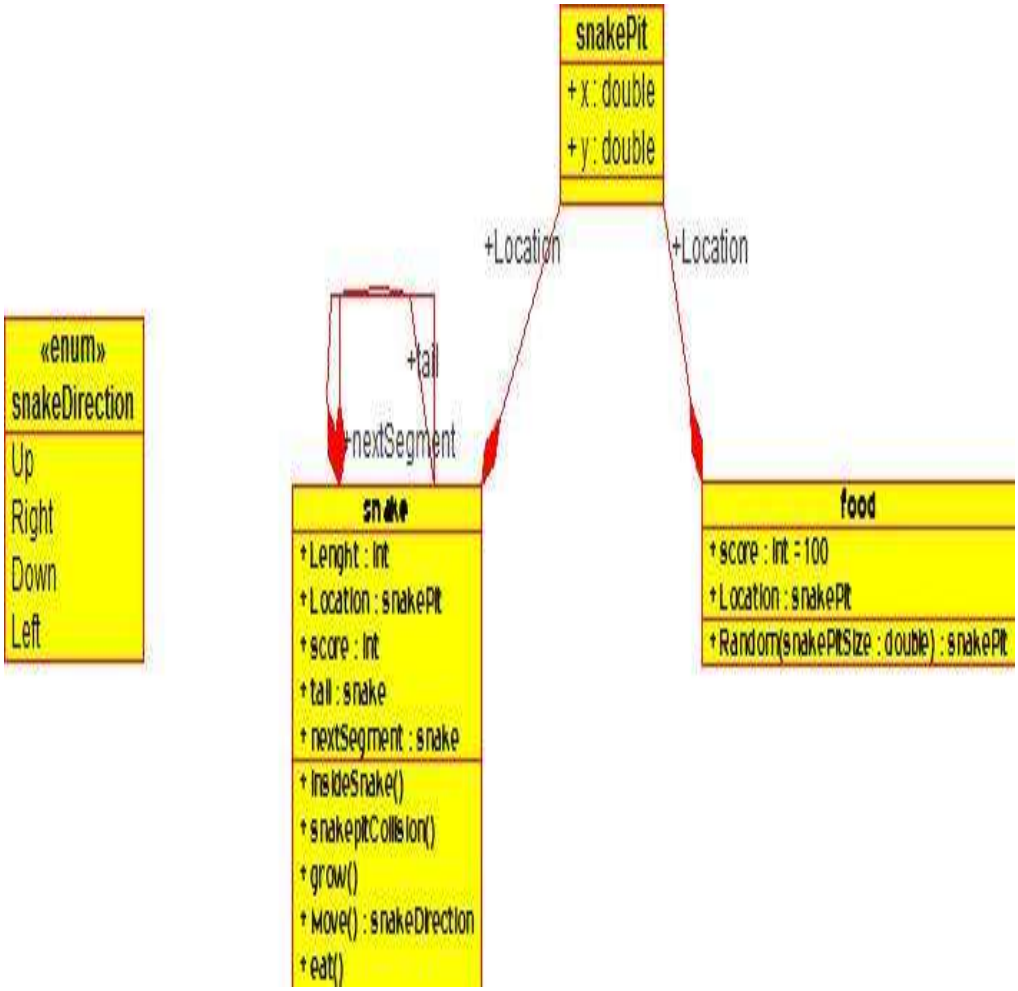
Tabel 3.4 Scenario Use Case Pilihan Cara Main

Identifikasi		
Nomor	3	
Nama	Cara Main	
Tujuan	Melihat informasi tentang cara bermain game Snake	
Deskripsi	Proses untuk melihat informasi mengenai cara bermain game	
Aktor	Pemain	
Skenario Utama		
Kondisi awal	Pengguna berada di menu utama aplikasi game Snake	
Aksi Aktor		Reaksi Sistem
Memilih menu Cara Main		Aplikasi merespon dengan cara masuk ke tampilan cara main
Kondisi akhir	Aplikasi masuk ke dalam informasi cara main permainan Snake	

Tabel 3.5 Scenario Use Case Pilihan Pembuat

Identifikasi		
Nomor	4	
Nama	Pembuat	
Tujuan	Melihat informasi mengenai pembuat aplikasi	
Deskripsi	Proses untuk melihat informasi mengenai pembuat aplikasi	
Aktor	Pemain	
Skenario Utama		
Kondisi awal	Pengguna berada di menu utama aplikasi game Snake	
Aksi Aktor		Reaksi Sistem
Memilih menu Pembuat		Aplikasi merespon dengan cara masuk ke tampilan pembuat
Kondisi akhir	Aplikasi masuk ke dalam informasi pembuat aplikasi game	

3.8 Class Diagram



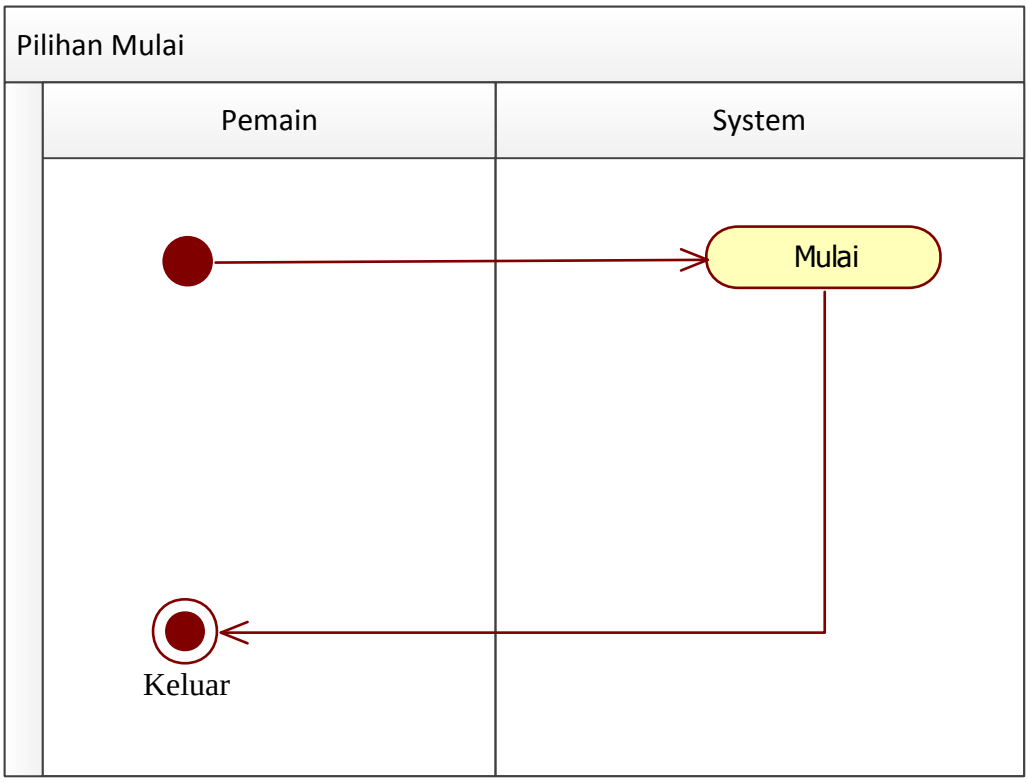
Gambar 3.9 Class Diagram *Snake*

3.9 Activity Diagram

3.9.1 Proses View Mulai

Tabel 3.6 Penjelasan View Mulai

Deskripsi	Tampilan ini menampilkan pilihan mulai, dimana pemain langsung memasuki dalam aplikasi permainan.
Normal Flow	1. Pemain memilih Mulai, lalu bermain permainan 2. Pemain keluar dari permainan

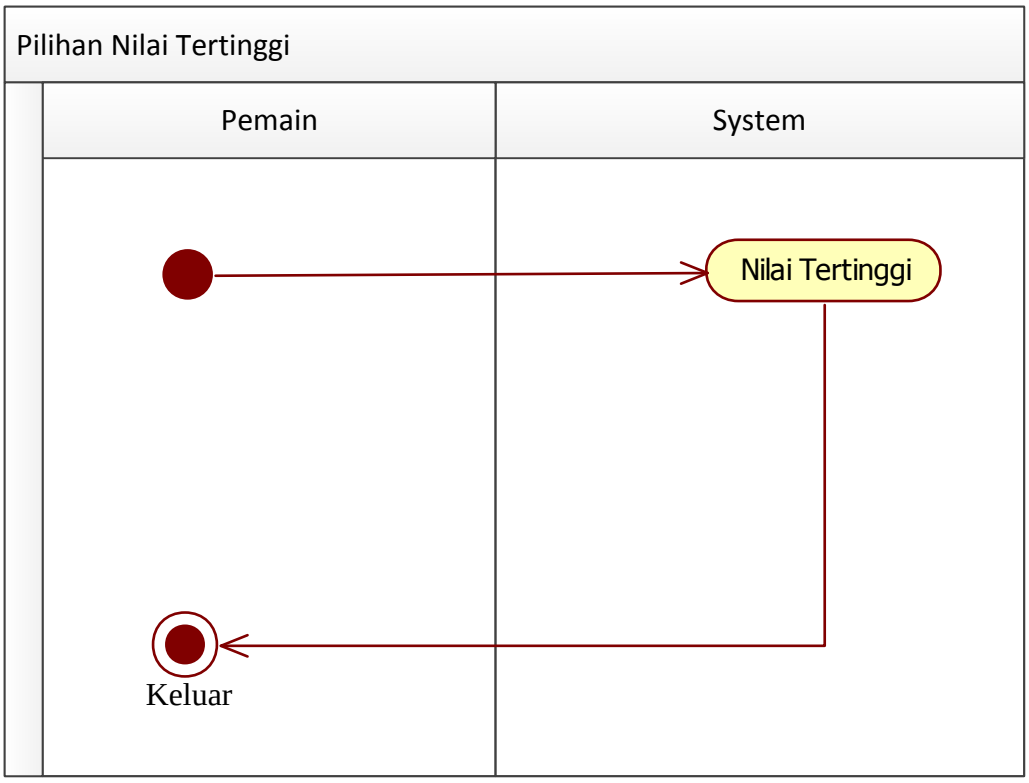


Gambar 3.10 Activity Diagram Pada Proses View Mulai

3.9.2 Proses View Nilai Tertinggi

Tabel 3.7 Penjelasan View Nilai Tertinggi

Deskripsi	Tampilan ini merupakan tampilan untuk melihat nilai tertinggi pada permainan.
Normal Flow	1. Pemain memilih pilihan nilai tertinggi melihat nilai tertinggi permainan 2. Pemain keluar dari tampilan nilai tertinggi

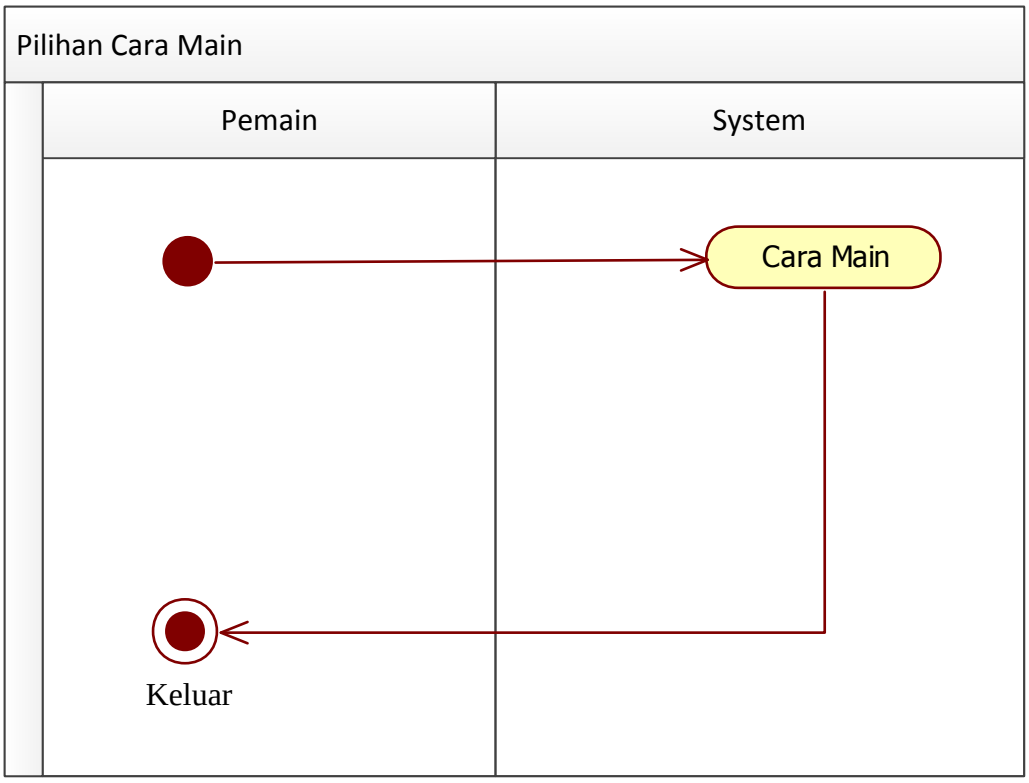


Gambar 3.11 Activity Diagram Pada Proses View Nilai Tertinggi

3.9.3 Proses View Cara Main

Tabel 3.8 Penjelasan View Cara Main

Deskripsi	Tampilan ini merupakan tampilan untuk menampilkan informasi cara memainkan game
Normal Flow	1. Pemain memilih pilihan cara main melihat cara bermain 2. Pemain keluar dari tampilan cara main

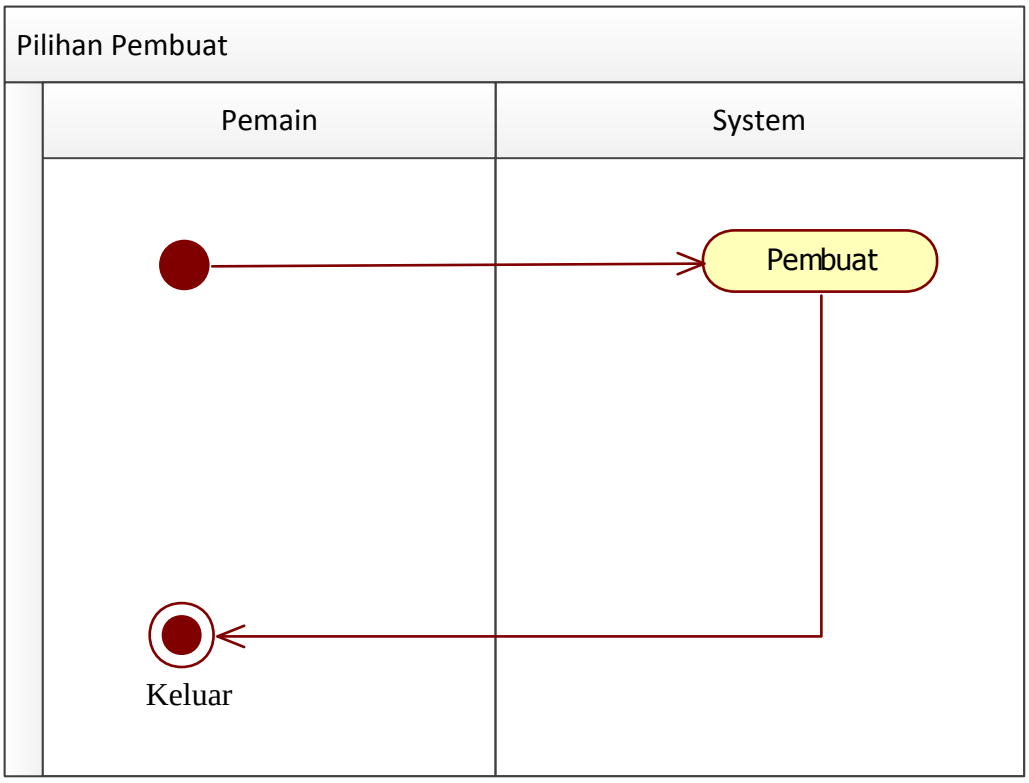


Gambar 3.12 Activity Diagram Pada Proses View Cara Main

3.9.4 Proses View Pembuat

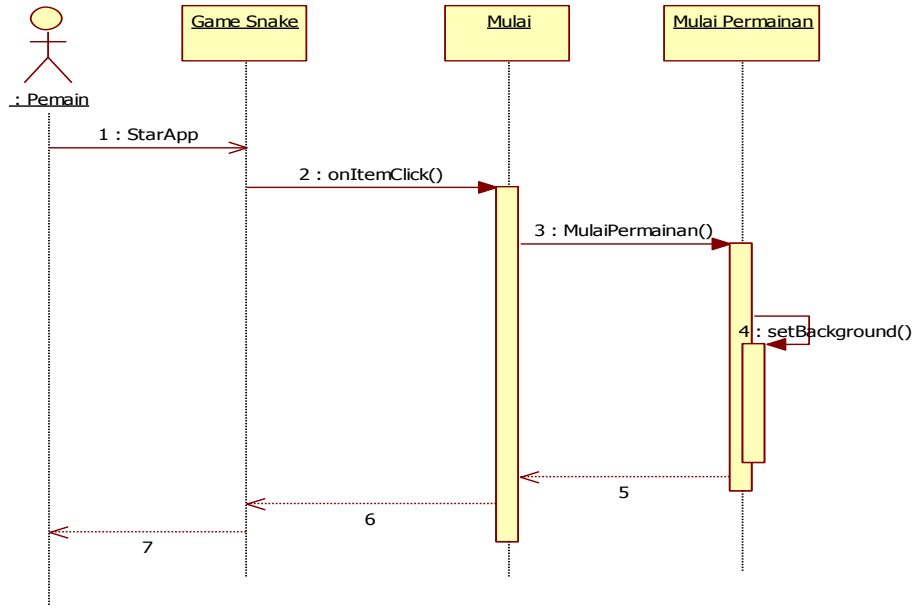
Tabel 3.9 Penjelasan View Pembuat

Deskripsi	Tampilan ini merupakan tampilan yang menampilkan keterangan atau informasi mengenai pembuat aplikasi
Normal Flow	1. Pemain memilih pilihan pembuat untuk melihat informasi mengenai pembuat aplikasi. 2. Pemain keluar dari tampilan pembuat

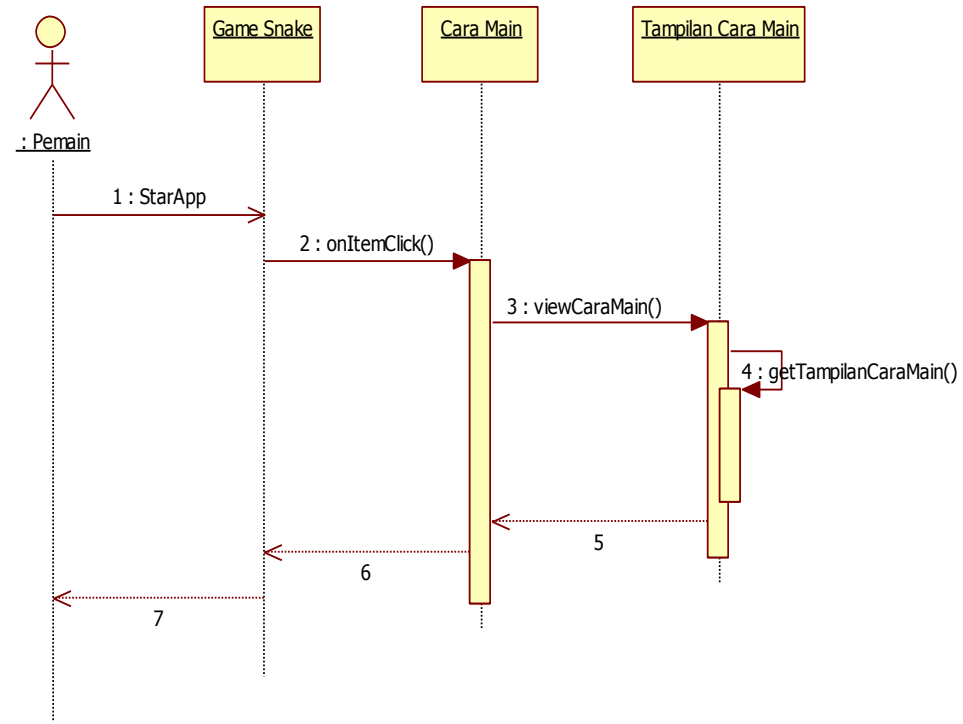


Gambar 3.13 Activity Diagram Pada Proses View Pembuat

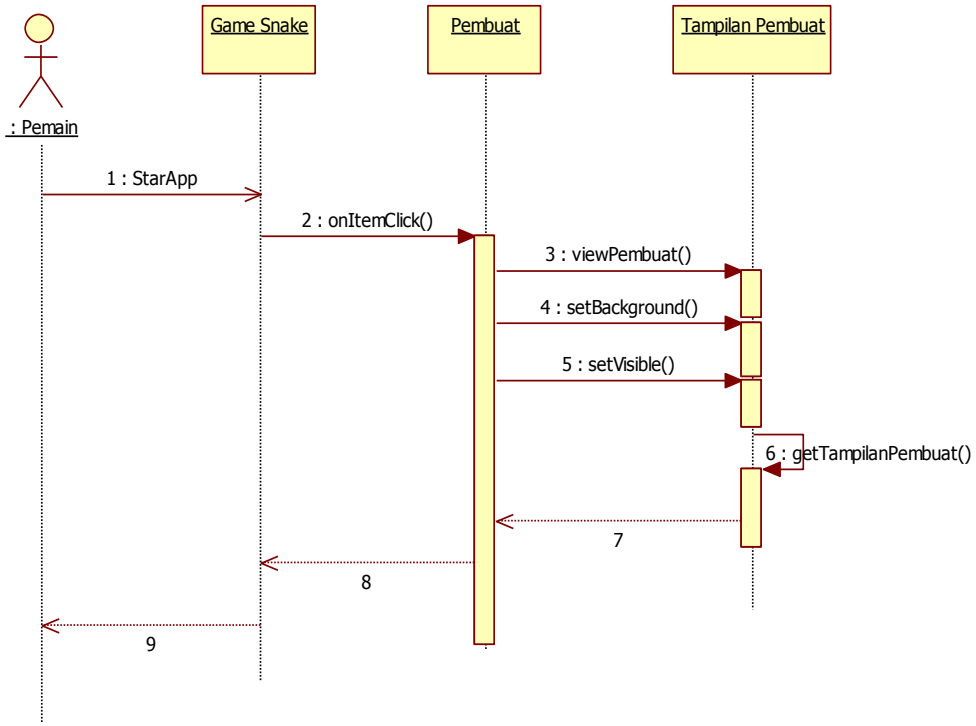
3.10 Perancangan Sequence Diagram



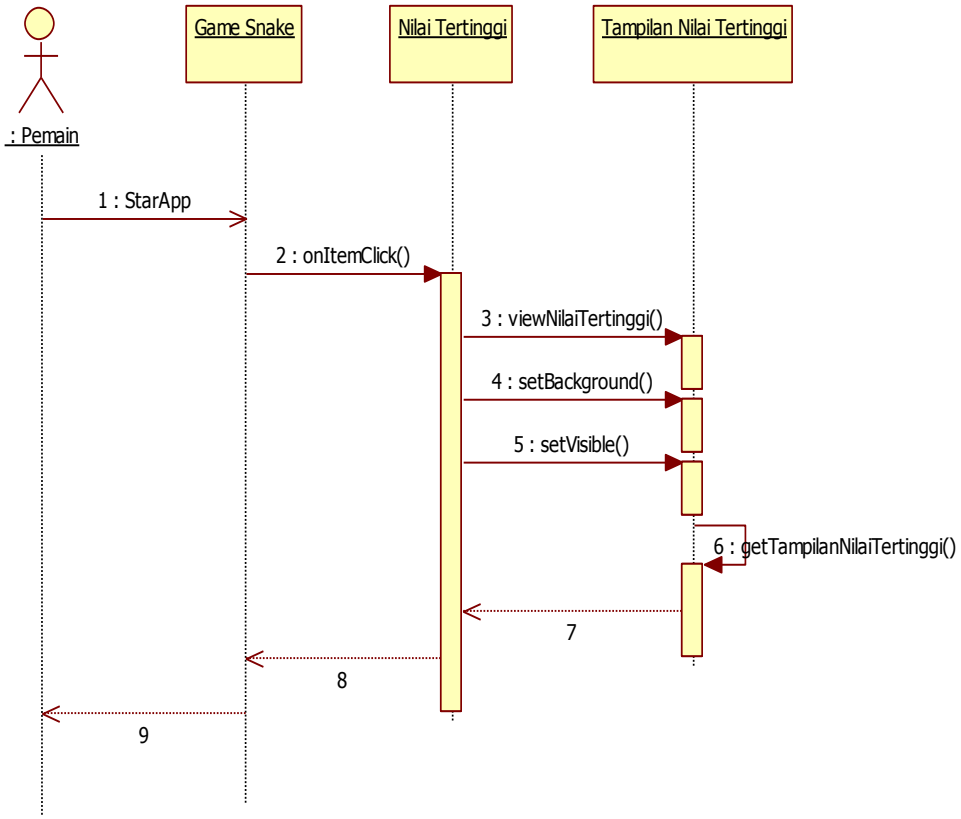
Gambar 3.14 Sequence Diagram Mulai



Gambar 3.15 Sequence Diagram Cara Main



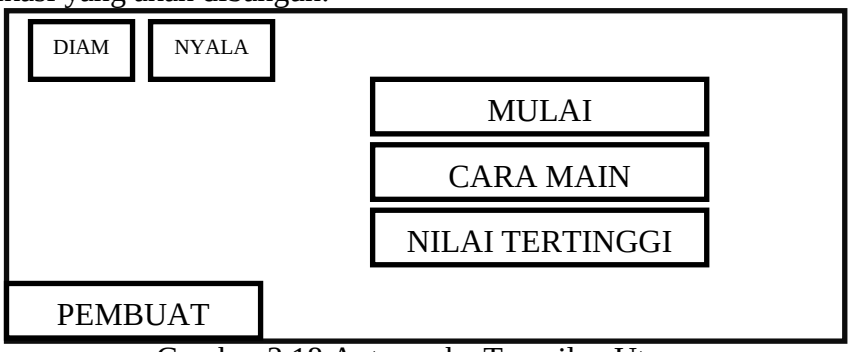
Gambar 3.16 Sequence Diagram Pembuat



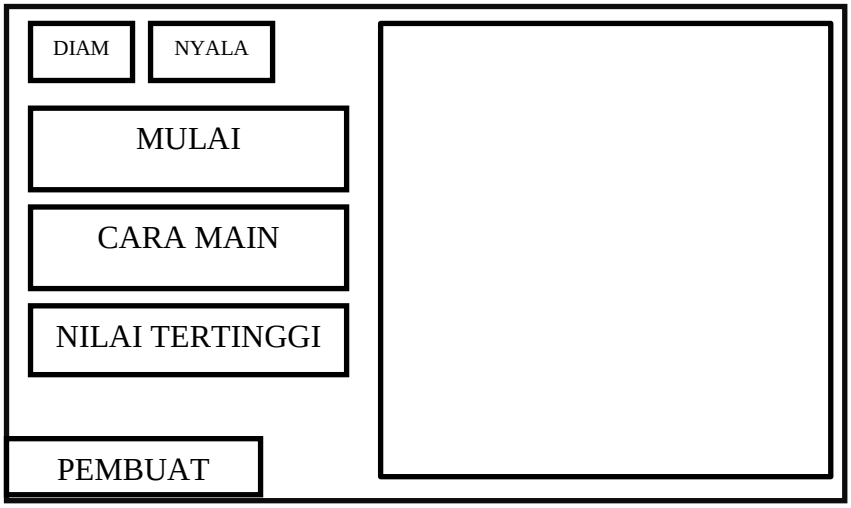
Gambar 3.17 Sequence Diagram Nilai Tertinggi

3.11 Perancangan Antarmuka

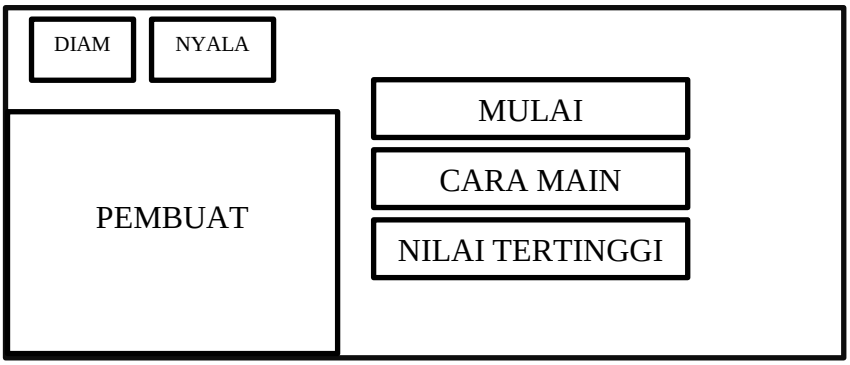
Perancangan antarmuka dibutuhkan untuk mewakili keadaan sebenarnya dari aplikasi yang akan dibangun, berikut akan disajikan perancangan antarmuka dari aplikasi yang akan dibangun:



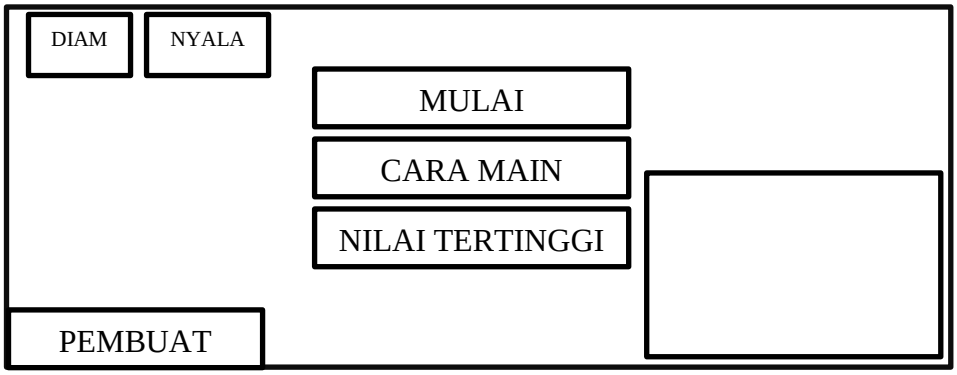
Gambar 3.18 Antarmuka Tampilan Utama



Gambar 3.19 Antarmuka Cara Main



Gambar 3.20 Antarmuka Pembuat



Gambar 3.21 Antarmuka Nilai Tertinggi