IMPLEMENTASI ALGORITMA SIMPLIFIED MEMORY-BOUNDED A* UNTUK PENCARIAN KATA PADA PERMAINAN WORD SEARCH PUZZLE

Asih Joko Purnomo[1]

[1] Program Studi Teknik Informatika
Fakultas Teknik dan Ilmu Komputer Universitas Komputer Indonesia
Jl. Dipatiukur No 112-116 Bandung - Indonesia
E-mail : jokopurnomoa@gmail.com

ABSTRAK

Permainan *word search puzzle* adalah permainan untuk mencari kata yang tersembunyi pada papan permainan yang disusun dalam bentuk matriks. Kata-kata tersebut dapat disusun secara horizontal, vertikal maupun tersusun dengan lebih dari satu ruas garis yang terhubung secara horizontal dan vertikal. Pencarian kata yang tersusun dengan lebih dari satu ruas garis memiliki karakteristik yang sama dengan permasalahan *pathfinding*, sehingga membutuhkan suatu algoritma *pathfinding* untuk melakukan pencarian.

Algoritma *Simplified Memory-Bounded A* (SMA*)* adalah salah satu algoritma *pathfinding* yang dapat digunakan untuk melakukan pencarian kata pada pada permainan *word search puzzle*. Algoritma SMA* memiliki kelebihan pada penggunan memori yang lebih sedikit, hal ini dikarenakan penggunaan memori dibatasi hingga jumlah simpul tertentu.

Berdasarkan hasil pengujian bahwa semakin panjang karakter pada kata yang dicari maka waktu pencarian akan semakin lama dan penggunaan memori juga akan semakin besar. Semakin banyak simpul yang tersedia untuk melakukan pancarian maka waktu pencarian akan semakin cepat, dimana persentase peningkatan kecepatan pencarian dengan penambahan simpul sebanyak 100% dapat meningkat hingga 21,99% dibandingkan dengan tidak ada penambahan simpul.

Kata kunci : Pencarian Jalur, Teka-teki pencarian kata, *Simplified Memory-Bounded A* (SMA*)*

1. PENDAHULUAN

Permainan *word search puzzle* atau permainan pencarian kata adalah permainan berbasis *puzzle* untuk mencari kata-kata yang disusun dalam bentuk matriks. Kata-kata tersebut dapat disusun secara horizontal, vertikal maupun tersusun dengan lebih dari satu ruas garis yang terhubung secara horizontal dan vertikal. Penyelesaian dari permainan *word search puzzle* ini adalah menemukan semua kata yang tersembunyi pada papan permainan yang berbentuk matriks. Permasalahan yang dihadapi adalah bagaimana sistem dapat menemukan semua kata yang tersembunyi di dalam puzzle yang telah tersusun secara random baik secara horizontal, vertikal maupun tersusun dengan lebih dari satu ruas garis yang terhubung secara horizontal dan vertikal.

Sebelumnya telah ada penelitian tentang analisis perbandingan algoritma *Knuth-Morris-Pratt* dengan algoritma *Boyer-Moore* pada permainan *word search puzzle* [1]. Dari penelitian tersebut diketahui bahwa algoritma *Boyer-Moore* efisien untuk pencarian secara horizontal dan vertikal, sedangkan algoritma *Knuth-Morris-Pratt* lebih efisien pada pencarian secara diagonal. Akan tetapi algoritma ini tidak dapat digunakan dalam penyelesaian permainan ini karena aturan pencarian pada kata yang tersusun tidak hanya horizontal, vertikal dan diagonal saja.

Maka dari itu dalam penelitian ini digunakan algoritma *Simplified Memory-Bounded A** untuk melakukan pencarian kata pada permainan *word search puzzle*. Algoritma ini dipilih karena melihat karakteristik permasalahan *word search puzzle* tersebut sama dengan permasalahan pada *pathfinding*. Algoritma ini adalah algoritma yang sering digunakan pada pencarian jalur terpendek dan merupakan algoritma pencarian *heuristic*. Algoritma ini memiliki kelebihan yaitu membutuhkan memori yang lebih kecil dibandingkan dengan algoritma A* [2].

Berdasarkan penjelasan yang telah dipaparkan diatas, maka diharapkan solusi pada permainan *word search puzzle* dapat ditemukan dan diketahui bahwa algoritma yang dipilih merupakan algoritma yang efektif untuk melakukan pencarian kata pada permainan *word search puzzle* tersebut.

Tujuan yang diharapkan akan dicapai dalam penelitian ini adalah:

1. Mengetahui apakah algoritma *Simplified Memory-Bounded A** dapat digunakan untuk mencari solusi pada permainan *word search puzzle*.
2. Mengetahui performansi dalam kecepatan dan penggunaan memori dari algoritma *Simplified*

*Memory-Bounded A** untuk pencarian kata pada permainan *word search puzzle*.

2. ISI PENELITIAN

2.1 Permainan *Word Search Puzzle*

Permainan *word search puzzle* dikenal sebagai *word find game*, yang terkenal karena membantu siswa untuk mengenali kata-kata. Pada permainan ini huruf dari sebuah kata terletak pada grid dan biasanya memiliki bentuk persegi. Untuk memainkan game ini, pemain mencari dan menandai semua kata-kata tersembunyi di dalam grid. Dalam permainan *word search puzzle*, daftar kata tersembunyi disediakan. Biasanya banyak kata yang terkait mudah bagi pemain untuk dicari. Kata terdaftar dapat diatur dalam arah horizontal, vertikal atau diagonal dalam grid. Semakin cepat penyelesaian setiap tingkat, maka skor yang lebih tinggi akan didapatkan. Dalam mencari kata-kata, pengguna membaca dan menghafal kata-kata saat mereka bermain game yang membantu mereka mempelajari kata-kata dan ejaan, huruf demi huruf, dalam teka-teki [5]. Berikut ini adalah gambar dari salah satu permainan *word search puzzle*:



Gambar 1 *Word Search Puzzle* [5]

2.2 Metode Pencarian Heuristik (*Informed Search*)

Kata *heuristic* berasal dari sebuah kata kerja bahasa Yunani, *heuriskein*, yang berarti 'mencari' atau 'menemukan'. Dalam dunia pemrograman, sebagian orang menggunakan kata heuristik sebagai lawan kata dari algoritmik, dimana kata heuristik ini diartikan sebagai 'suatu proses yang mungkin dapat menyelesaikan suatu masalah tetapi tidak ada jaminan bahwa solusi yang dicari selalu dapat ditemukan'. Di dalam mempelajari metode-metode pencarian ini, kata heuristik diartikan sebagai suatu fungsi yang memberikan suatu nilai berupa biaya perkiraan (estimasi dari suatu solusi) [2].

Metode-metode yang termasuk dalam teknik pencarian yang berdasarkan pada fungsi heuristik adalah: *Generate and Test, Hill Climbing (Simple Hill Climbing dan Steepest-Ascent Hill Climbing), Simulated Annealing, Best First Search(Greedy Best First Search* dan A* dengan berbagai varisinya, seperti *Simplified Memory-Bounded A**).

2.3 Algoritma *Simplified Memory-Bounded A* (SMA*)*

Untuk masalah tertentu, di mana memori komputer terbatas, algoritma A* mungkin tidak mampu menemukan solusi karena sudah tidak tersedia memori untuk menyimpan simpul-simpul yang dibangkitkan. Algoritma IDA* dapat digunakan untuk kondisi seperti ini karena IDA* hanya membutuhkan sedikit memori. Tetapi, satu kelemahan IDA* adalah bahwa pencarian yang dilakukan secara iteratif akan membutuhkan waktu yang lama karena harus membangkitkan simpul berulang kali [2].

Berlawanan dengan IDA* yang hanya mengingat satu *f-limit*, algoritma SMA* mengingat *f-Cost* dari setiap iterasi sampai sejumlah simpul yang ada di dalam memori. Karena batasan memori dalam jumlah tertentu, maka dapat membatasi pencarian hanya sampai pada simpul-simpul yang dapat dicapai dari root sepanjang suatu jalur yang memorinya masih mencukupi. Kemudian mengembalikan suatu rute terbaik diantara rute-rute yang ada dalam batasan jumlah simpul tersebut. Jika memori komputer hanya mampu menyimpan 100 simpul, maka bisa membatasi proses pencarian sampai level 99.

Pada algoritma SMA* terdapat sebuah senarai yang digunakan untuk memanipulasi antrian simpul yang terurut berdasarkan *f-cost*. Di sini yang dimaksud *f-cost* adalah gabungan biaya sebenarnya dan biaya perkiraan, yang secara matematika dinyatakan seperti pada persamaan 2.1 berikut:

$$f(n) = g(n) + h(n) \qquad (2.1)$$

Dengan:

n = simpul saat ini
g(n) = biaya (*cost*) dari simpul awal ke simpul *n* sepanjang jalur pencarian
h(n) = perkiraan cost dari simpul *n* ke simpul tujuan (nilai *heuristic*)
f(n) = total cost dari simpul *n* ke simpul tujuan

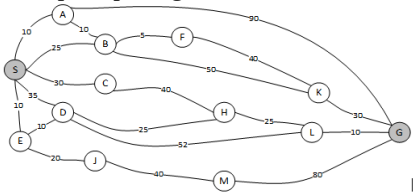
Berikut ini adalah algoritma *Simplified Memory-Bounded A** [2]:

function SMA*(masalah) returns solusi
inputs : masalah, sebuah masalah
local variables : Queue, antrian nodes yang terurut berdasarkan f-cost
Queue ← MAKE-QUEUE({MAKE-SIMPUL(INITIAL-STATE[masalah])})
loop do
    if Queue kosong **then** return gagal
    n ← simpul di Queue yang memiliki f-cost terkecil dan level terdalam
    if GOAL-TEST(n) **then** return sukses
    suk ← NEXT-SUCCESSOR(n)

if suk bukan goal dan levelnya sudah maksimum **then**
f(suk) ← INFINITE
else
f(suk) ← MAX(f(n), g(n) + h(n))
end
if semua suksesor dari n sudah dibangkitkan **then**
Ganti f-cost pada n dengan nilai f(suk) yang terkecil. Gantikan nilai f(suk) terkecil ini ke semua ancestors dari n(ayah, kakek, dan seterusnya keatas) kecuali ancestors yang memiliki f-cost lebih kecil daripada f(suk)terkecil itu.
if semua SUCCESSOR(n) sudah di memori **then**
Keluarkan n dari Queue (tetapi tidak dihapus secara fisik di memori)
if memori penuh **then**
if suk = Goal and f(suk) = f(start) **then** return sukses dan exit
Hapus simpul terburuk di dalam Queue yang memiliki f-cost terbesar dan level terdangkal.
Keluarkan simpul terburuk tersebut dari daftar suksesor parent-nya.
Masukkan parent dari simpul terburuk tersebut ke Queue jika parent tersebut tidak ada di Queue.
end
insert suk in Queue

Gambar 2 Algoritma *Simplified Memory-Bounded A**

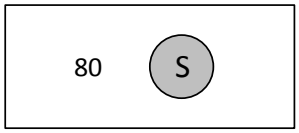
Misalnya terpadat masalah pencarian rute terpendek seperti pada gambar berikut:



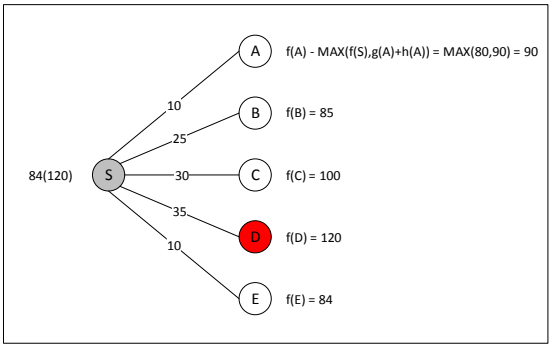
n	S	A	B	C	D	E	F	G	H	J	K	L	M
h(n)	80	80	60	70	85	74	70	0	40	100	30	20	70

Gambar 3 Masalah Pencarian Rute

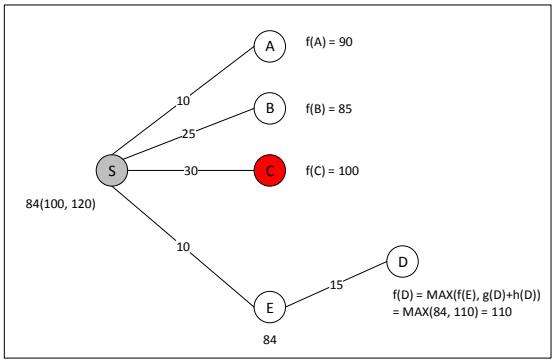
Berikut ini adalah langkah-langkah penyelesaian menggunakan algoritma *Simplified Memory-Bounded A**:



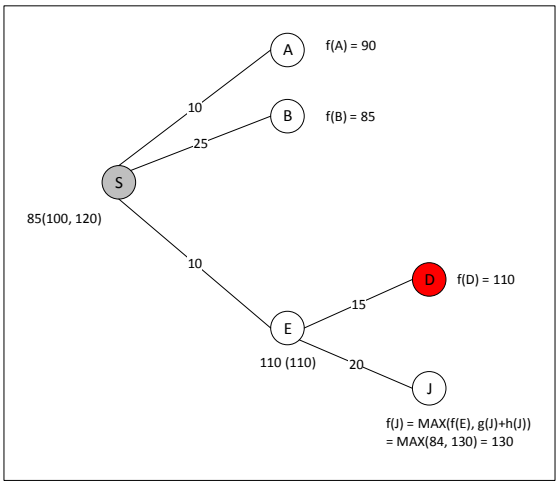
Gambar 4 Langkah 1 Pencarian Rute



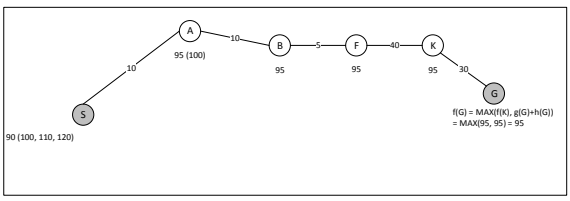
Gambar 5 Langkah 2 Pencarian Rute



Gambar 6 Langkah 3a Pencarian Rute



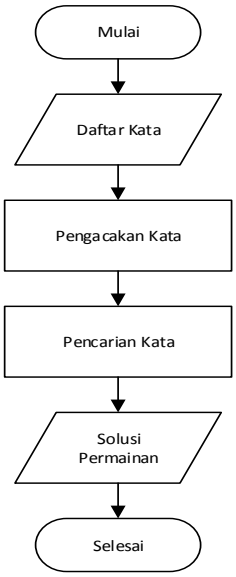
Gambar 7 Langkah 3b Pencarian Rute



Gambar 8 Langkah 8 Pencarian Rute (hasil akhir pencarian)

2.4 Analisis Sistem

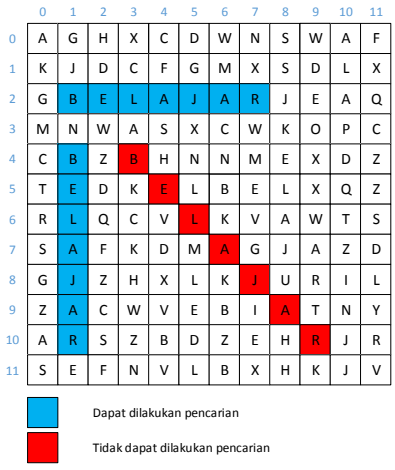
Permainan *word search puzzle* terdiri dari 2 proses yaitu pengacakan kata dan pencarian kata. Proses pengacakan kata yaitu mengisi papan permainan dengan kata yang tersusun secara acak dan mengisi huruf acak pada matriks yang kosong. Proses pencarian kata yaitu melakukan pencarian kata pada papan permainan yang telah terisi kata yang tersusun secara acak. Berikut ini adalah *flowchart* dari permainan *word search puzzle*:



Gambar 9 *Flowchart Word Search Puzzle*

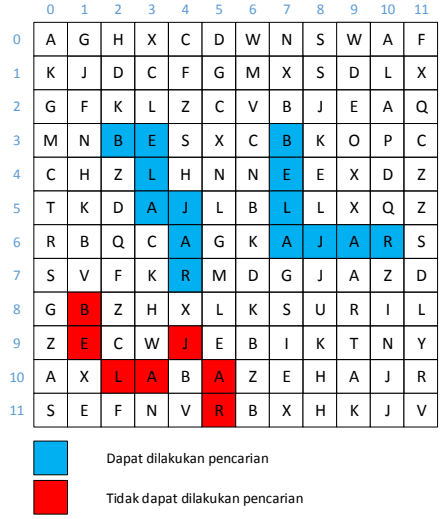
Pada permainan *word search puzzle* terdapat beberapa aturan yang berbeda antara satu permainan dengan permainan lainnya. Untuk aturan yang ada pada permainan ini antara lain sebagai berikut:

a. Pencarian kata dapat dilakukan secara horizontal dan vertikal, dan tidak bisa secara diagonal.



Gambar 10 Aturan Pencarian Kata

b. Pencarian kata juga dapat dilakukan dengan berbelok-belok ke atas, ke bawah, ke kiri dan ke kanan, tetapi antara karakter satu dengan yang lainnya harus terhubung secara horizontal atau vertikal.



Gambar 11 Aturan Pencarian Kata(2)

c. Pencarian dapat dimulai dari manapun antara koordinat (0,0) hingga koordinat (19,19) pada matriks.
d. Panjang karakter pada kata yang dicari antara 5 hingga 30 karakter.

2.5 Analisis Masalah

Permasalahan yang muncul pada permainan *word search puzzle* adalah bagaimana sistem dapat menemukan semua kata yang tersembunyi di dalam matriks. Pertama dilakukan pencarian huruf pertama pada kata yang akan dicari, kemudian dicari huruf kedua dengan mencari ke atas, bawah, kiri dan kanan yang mengelilingi huruf pertama tersebut. Metode tersebut diulang hingga kata yang dicari ditemukan.

Untuk melakukan pencarian kata pada permainan *word search puzzle* dapat menggunakan algoritma *path finding*. Terdapat beberapa algoritma *path finding* yang bisa digunakan. Algoritma yang digunakan adalah algoritma *Simplified Memory-Bounded A**. Algoritma *Simplified Memory-Bounded A* (SMA*)* merupakan pengembangan dari algoritma A* yang mengatasi masalah untuk melakukan pencarian kata pada permainan *word search puzzle*. Algoritma ini mengatasi kelemahan yang dimiliki oleh algoritma A* yaitu penggunaan memori yang besar.

Oleh karena itu , yang akan diteliti untuk mengukur efektifitas penggunaan algoritma SMA* adalah sebagai berikut:

1. Jumlah simpul yang dibangkitkan dan dihapus selama proses pencarian.
2. Waktu eksekusi untuk pencarian solusi.

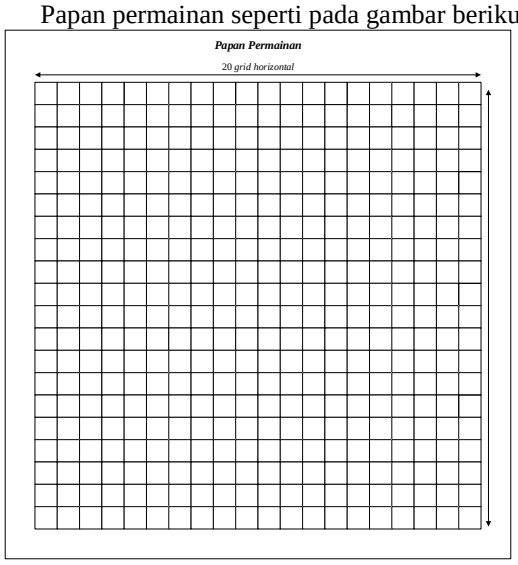
2.6 Analisis Game *Word Search Puzzle*

Analisis pada *game word search puzzle* dilakukan untuk melihat keefektifan dalam hal waktu eksekusi dan jumlah simpul yang dibangkitkan dan dihapus. Dalam mengimplementasikan algoritma SMA* akan dibangun sebuah *prototype* dengan gambaran game *word search puzzle* adalah sebagai berikut:

1. Game terdiri dari matriks dengan ukuran minimal yaitu 3 baris x 3 kolom dan maksimal 20 baris x 20 kolom.
2. Penerapan algoritma yang dipilih yaitu algoritma SMA*.
3. Nilai masukan parameter yaitu jumlah simpul maksimum.
4. Kata yang dicari diletakkan pada matriks secara *random*.
5. Pencarian hanya bisa dilakukan secara horizontal, vertikal, dan gabungan antara horizontal dan vertikal yang terhubung dengan lebih dari satu ruas garis.

Papan permainan pada *word search puzzle* dibuat dengan skala *pixel* yang berukuran panjang 500 *pixel* dan lebar 500 *pixel*. Dalam papan permainan tersebut disimpan beberapa *grid* ukuran panjang 25 *pixel* dan lebar 25 *pixel*, didapatkan 20 *grid* horizontal dan 20 *grid* vertikal. Jika dijumlahkan keseluruhan *grid*-nya menjadi 400 *grid*.

Papan permainan seperti pada gambar berikut:



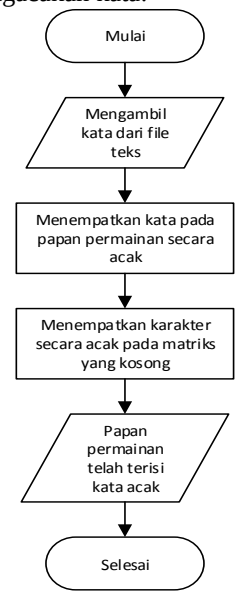
Gambar 12 Papan Permainan

Pada permainan ini terdapat 2 proses besar yang dilakukan antara lain:

1. Pengacakan Kata
2. Pencarian Kata

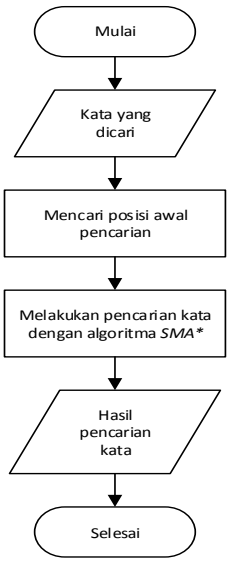
Proses pengacakan kata dilakukan dengan mengisi matriks dengan kata-kata yang dicari dengan aturan sesuai dengan yang telah dijelaskan diatas, kemudian mengisi matriks yang kosong dengan huruf acak. Proses pencarian kata dilakukan dengan melakukan pencarian kata sesuai dengan aturan yang telah dijelaskan diatas, setelah pencarian ditemukan maka disimpan koordinat dari kata yang dicari.

Untuk lebih jelasnya berikut ini adalah *flowchart* dari proses pengacakan kata:



Gambar 13 *Flowchart* Pengacakan Kata

Pada penelitian ini proses yang akan dibahas lebih dalam adalah proses pencarian kata menggunakan algoritma *Simplified Memory-Bounded A**. Pada proses pencarian kata terdapat beberapa langkah, antara lain seperti pada *flowchart* berikut:



Gambar 14 *Flowchart* Pencarian Kata

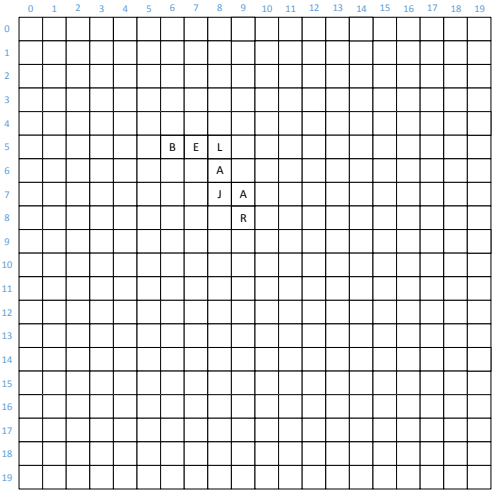
2.7 Analisis Masukan

Analisis data masukan yang dibutuhkan pada algoritma SMA* yaitu posisi huruf pertama dari kata yang dicari (*initial state*), kata yang dicari (*goal state*). Posisi huruf pertama dicari secara sekuensial dari kolom pertama dan baris pertama hingga kolom terakhir dan baris terakhir atau sampai posisi

ditemukan. Selain itu, dalam pencarian algoritma SMA*, dibutuhkan pencarian untuk menghitung nilai *f(n)*.

Pencarian kata dilakukan pada papan permainan yang berukuran 20x20. Papan permainan telah terisi kata yang dicari yang disusun secara acak. Papan permainan terisi dengan kata dengan panjang maksimal 30 karakter. Kata yang akan dimasukkan ke dalam matriks tersebut disimpan di dalam file teks, dan diambil secara acak untuk dimasukkan ke dalam matriks.

Setelah proses penempatan kata pada papan permainan, maka papan permainan telah terisi kata yang tersusun secara acak. Papan permainan yang telah terisi kata seperti pada gambar berikut:



Gambar 15 Papan Permainan Yang Terisi Kata

Kemudian setelah proses penempatan kata, maka proses selanjutnya yaitu mengisi matriks yang kosong dengan huruf acak. Setelah matriks yang kosong terisi huruf acak, maka hasilnya seperti pada gambar berikut:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	A	G	H	X	C	D	W	N	S	W	A	F	G	P	O	I	T	F	V	S
1	K	J	D	C	F	G	M	X	S	D	L	X	K	Q	Y	T	J	E	R	R
2	G	H	J	D	R	E	D	H	J	E	A	Q	D	F	R	E	F	J	J	M
3	M	N	W	A	S	X	C	W	K	O	P	C	V	F	G	X	G	H	G	N
4	C	V	Z	N	H	N	N	M	E	X	D	Z	F	Q	I	U	Q	T	D	Y
5	T	Y	D	K	D	L	B	E	L	X	Q	Z	F	R	Y	K	Z	X	D	S
6	R	W	Q	C	V	N	K	V	A	W	T	S	D	K	D	L	O	P	E	Q
7	S	J	F	K	D	M	K	G	J	A	Z	D	A	X	M	X	N	C	B	V
8	G	N	Z	H	X	L	K	S	U	R	I	L	U	L	D	Y	H	P	Y	R
9	Z	Q	C	W	V	E	B	I	R	T	N	Y	M	U	I	T	O	P	S	A
10	A	W	S	Z	B	D	Z	E	H	H	J	R	L	J	B	N	J	X	K	L
11	S	E	F	N	V	L	B	X	H	K	J	V	X	B	F	T	V	N	F	D
12	D	G	S	Z	C	N	M	R	X	X	J	V	B	B	K	N	M	X	Z	J
13	D	V	X	B	C	R	C	C	X	Z	Z	C	V	N	B	T	Y	L	M	U
14	V	M	B	B	N	Z	A	N	Z	Z	D	T	T	M	V	N	K	C	X	G
15	F	N	Q	R	X	W	R	E	V	R	T	T	F	U	D	S	I	J	O	A
16	C	X	V	C	B	B	Z	X	N	W	T	W	T	M	T	X	E	Z	R	H
17	H	N	V	X	Z	C	L	L	B	B	M	N	L	K	M	L	L	J	C	P
18	A	B	Z	N	D	H	X	M	J	I	C	O	O	L	P	V	X	N	C	M
19	Z	A	B	S	N	D	M	Z	F	N	Q	G	W	E	H	R	T	J	Y	K

Gambar 16 Papan Permainan Yang Terisi Huruf Acak

2.8 Analisis Algoritma

Analisis algoritma yang dilakukan dalam penelitian ini adalah untuk menganalisis cara kerja algoritma SMA* (*Simplified Memory-Bounded A**) terhadap masalah *word search puzzle* untuk melakukan pencarian kata. Pada bagian ini akan dijelaskan proses secara manual untuk menghasilkan *output*.

Pada algoritma SMA* (*Simplified Memory-Bounded A**) untuk melakukan pencarian maka dilakukan perhitungan *f-cost* untuk setiap simpul yang dibangkitkan. Untuk menghitung *f-cost* maka secara matematika dinyatakan pada persamaan 2.1 berikut:

$$f(n) = g(n) + h(n)$$

Dengan:

n = simpul saat ini

g(n) = biaya (*cost*) dari simpul awal ke simpul *n* sepanjang jalur pencarian

h(n) = perkiraan cost dari simpul *n* ke simpul tujuan (nilai *heuristic*)

f(n) = total cost dari simpul *n* ke simpul tujuan

Untuk menghitung nilai *heuristic*, maka dicari mengunakan total huruf pada kata yang dicari dikurangi total huruf yang benar dari pencarian. Perhitungan nilai *heuristic* adalah seperti berikut:

h(n) = Total huruf kata dicari – total huruf benar

Adapun langkah-langkah manual pencarian *SMA** adalah seperti berikut:

a. Pilih simpul dengan cost paling kecil dan level terdalam
b. Bangkitkan suksesor selanjutnya dari simpul *n*

suk = NEXT-SUCCESOR(n)

c. Hitung *f-cost* dari suksesor yang telah dibangkitkan

f(suk) = MAX(f(n), g(n) + h(n))

d. Jika semua suksesor simpul terpilih sudah dibangkitkan, maka ganti *f-cost* pada *n* dengan nilai *f(suk)* yang terkecil.
e. Jika semua suksesor *n* sudah di memori, maka keluarkan n dari *Queue* (tetapi tidak dihapus secara fisik di memori).
f. Jika memori penuh, maka hapus simpul terburuk di dalam *Queue* yang memiliki *f-cost* terbesar dan level terdangkal. Keluarkan simpul terburuk tersebut dari daftar suksesor parent-nya Masukkan parent dari simpul terburuk tersebut ke *Queue* jika parent tersebut tidak ada di *Queue*.
g. Masukkan simpul suksesor ke *Queue*

Dengan:

n = simpul saat ini

suk = simpul suksesor dari *n*

f(suk) = *f-cost* simpul suksesor *n*

MAX = nilai maksimum

NEXT-SUCCESSOR(n) = simpul suksesor selanjutnya dari *n*

Misalnya pada kasus ini akan dicari kata "BELAJAR" pada matriks yang telah tersusun secara random. Proses pertama yang dilakukan adalah mencari posisi dari huruf pertama dari kata yang akan dicari. Untuk melakukan pencarian tersebut maka harus dilakukan pencarian secara sekuensial dari kolom ke-0 dan baris ke-0. Disini dilakukan pencarian karakter **B** pada matriks tersebut. Kemungkinan huruf **B** pada matriks tersebut akan lebih dari satu, sehingga akan diambil huruf yang pertama kali ditemukan. Apabila posisi sudah ditemukan, maka akan dilakukan pencarian dengan posisi awal simpul adalah posisi yang telah ditemukan tersebut. Misalnya pada kasus tersebut jumlah simpul maksimum adalah 8 simpul.

Berikut ini adalah contoh matriks dengan karakter pertama pada kata yang telah ditemukan:

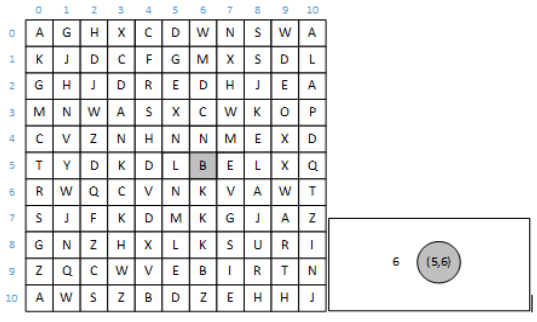
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	A	G	H	X	C	D	W	N	S	W	A	F	G	P	O	I	T	F	V	S
1	K	J	D	C	F	G	M	X	S	D	L	X	K	Q	Y	T	J	E	R	R
2	G	H	J	D	R	E	D	H	J	E	A	Q	D	F	R	E	F	J	J	M
3	M	N	W	A	S	X	C	W	K	O	P	C	V	F	G	X	G	H	G	N
4	C	V	Z	N	H	N	N	M	E	X	D	Z	F	Q	I	U	Q	T	D	Y
5	T	Y	D	K	D	L	B	E	L	X	Q	Z	F	R	Y	K	Z	X	D	S
6	R	W	Q	C	V	N	K	V	A	W	T	S	D	K	D	L	O	P	E	Q
7	S	J	F	K	D	M	K	G	J	A	Z	D	A	X	M	X	N	C	B	V
8	G	N	Z	H	X	L	K	S	U	R	I	L	U	L	D	Y	H	P	Y	R
9	Z	Q	C	W	V	E	B	I	R	T	N	Y	M	U	I	T	O	P	S	A
10	A	W	S	Z	B	D	Z	E	H	H	J	R	L	J	B	N	J	X	K	L
11	S	E	F	N	V	L	B	X	H	K	J	V	X	B	F	T	V	N	F	D
12	D	G	S	Z	C	N	M	R	X	X	J	V	B	B	K	N	M	X	Z	J
13	D	V	X	B	C	R	C	C	X	Z	Z	C	V	N	B	T	Y	L	M	U
14	V	M	B	B	N	Z	A	N	Z	Z	D	T	T	M	V	N	K	C	X	G
15	F	N	Q	R	X	W	R	E	V	R	T	T	F	U	D	S	I	J	O	A
16	C	X	V	C	B	B	Z	X	N	W	T	W	T	M	T	X	E	Z	R	H
17	H	N	V	X	Z	C	L	L	B	B	M	N	L	K	M	L	L	J	C	P
18	A	B	Z	N	D	H	X	M	J	I	C	O	O	L	P	V	X	N	C	M
19	Z	A	B	S	N	D	M	Z	F	N	Q	G	W	E	H	R	T	J	Y	K

Keterangan :

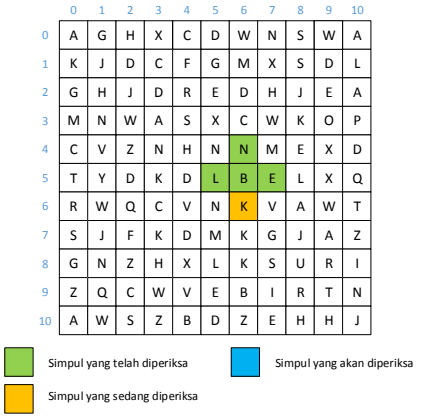
B Simpul Awal

Gambar 17 Kondisi Awal Pencarian Kata

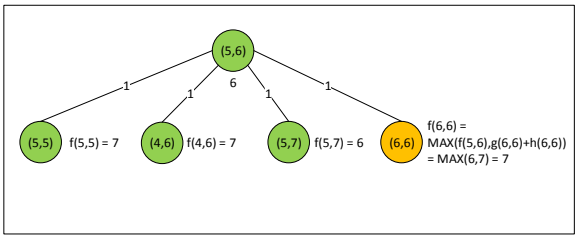
Setelah huruf pertama ditemukan, maka dilakukan pencarian menggunakan algoritma SMA* dengan langkah-langkah seperti berikut:



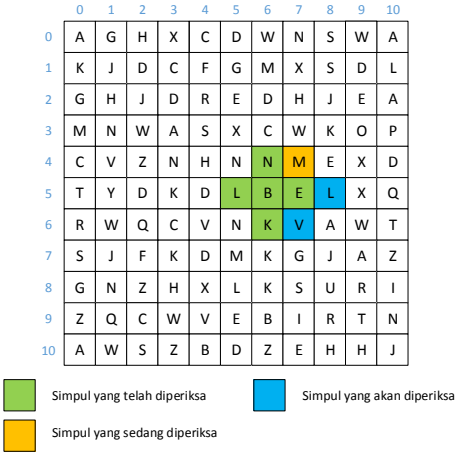
Gambar 18 Langkah 1 Pencarian Kata



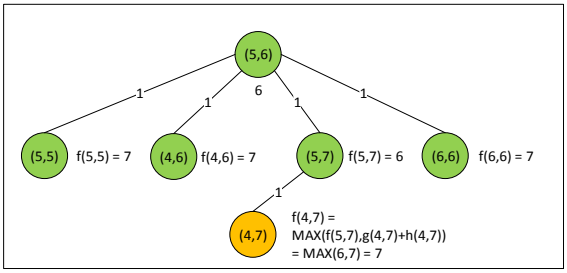
Gambar 19 Matriks Langkah 2 Pencarian Kata



Gambar 20 Langkah 2 Pencarian Kata



Gambar 21 Matriks Langkah 3a Pencarian Kata

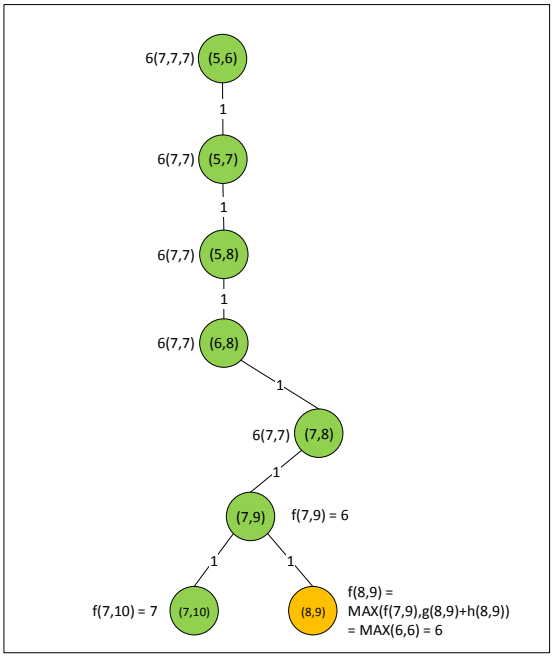


Gambar 22 Langkah 3a Pencarian Kata

	0	1	2	3	4	5	6	7	8	9	10
0	A	G	H	X	C	D	W	N	S	W	A
1	K	J	D	C	F	G	M	X	S	D	L
2	G	H	J	D	R	E	D	H	J	E	A
3	M	N	W	A	S	X	C	W	K	O	P
4	C	V	Z	N	H	N	N	M	E	X	D
5	T	Y	D	K	D	L	B	E	L	X	Q
6	R	W	Q	C	V	N	K	V	A	W	T
7	S	J	F	K	D	M	K	G	J	A	Z
8	G	N	Z	H	X	L	K	S	U	R	I
9	Z	Q	C	W	V	E	B	I	R	T	N
10	A	W	S	Z	B	D	Z	E	H	H	J

Simpul yang telah diperiksa
Simpul yang akan diperiksa
Simpul yang sedang diperiksa

Gambar 23 Matriks Langkah 7b Pencarian Kata



Gambar 24 Langkah 7b Pencarian Kata (Solusi)

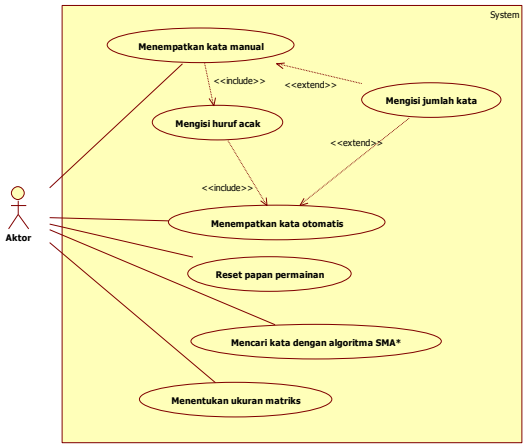
Pada gambar diatas menunjukkan pencarian telah selesai, dengan jalur yang ditemukan adalah **(5,6) – (5,7) – (5,8) – (6,8) – (7,8) – (7,9) – (8,9)**. Sehingga apabila simpul tersebut digabungkan maka akan membentuk kata "BELAJAR".

2.9 Analisis Kebutuhan Fungsional

Analisis kebutuhan fungsional bertujuan untuk menganalisis proses yang akan diterapkan dalam sistem dan menjelaskan kebutuhan yang diperlukan agar aplikasi dapat berjalan dengan baik. Analisis kebutuhan fungsional meliputi kebutuhan user, pemodelan menggunakan *Unified Modelling Language* (UML), perancangan antarmuka, dan perancangan method.

2.9.1. Use Case Diagram

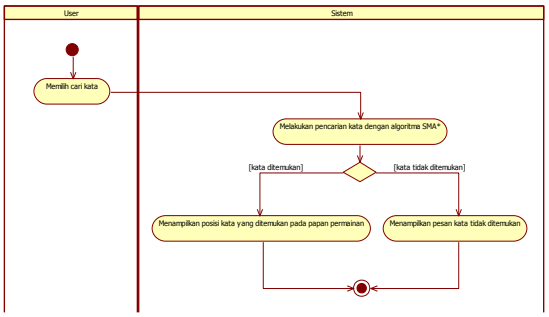
Diagram *use case* menggambarkan fungsi sebuah sistem dari perspektif pengguna, sehingga memudahkan dalam merumuskan spesifikasi kebutuhan sistem yang akan dibangun maupun dikembangkan. Dari hasil analisis permainan yang dibangun, maka *use case diagram* seperti pada gambar berikut:



Gambar 25 *Use Case Diagram*

2.9.2. Activity Diagram

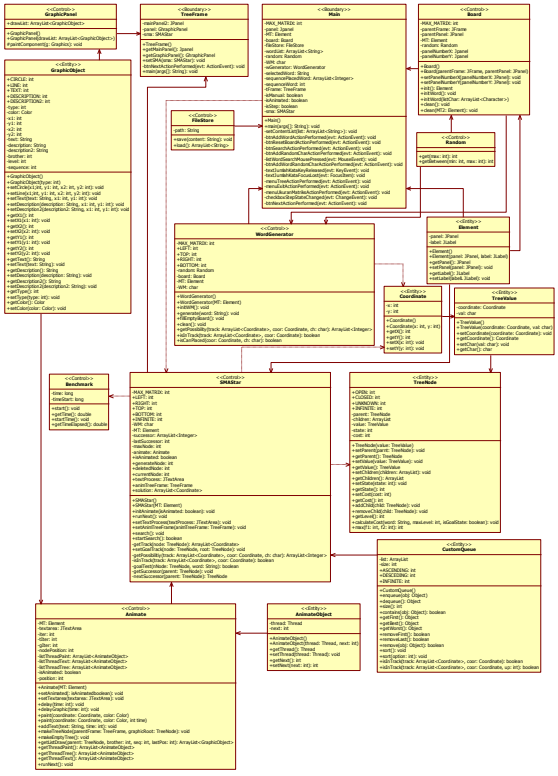
Activity diagram menggambarkan berbagai alir aktifitas dalam sistem yang dirancang. *Activity diagram* juga dapat menggambarkan proses pararel yang mungkin terjadi pada beberapa eksekusi. Berikut ini adalah *activity diagram* mencari kata dengan algoritma SMA*:



Gambar 26 *Activity Diagram Mencari Kata Dengan Algoritma SMA**

2.9.3. Class Diagram

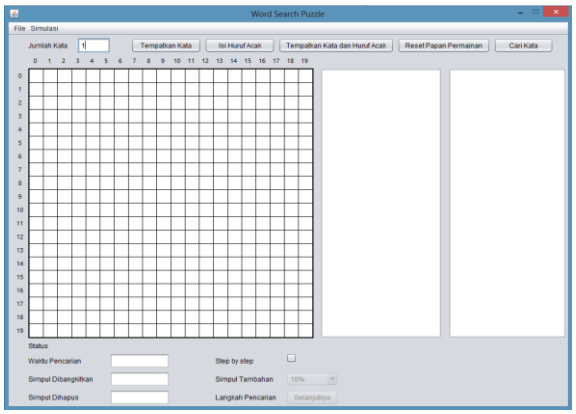
Class diagram merupakan diagram yang menggambarkan struktur sistem dari aspek pendefinisian kelas-kelas yang dibuat pada sistem. *Class diagram* pada permainan adalah seperti pada gambar berikut:



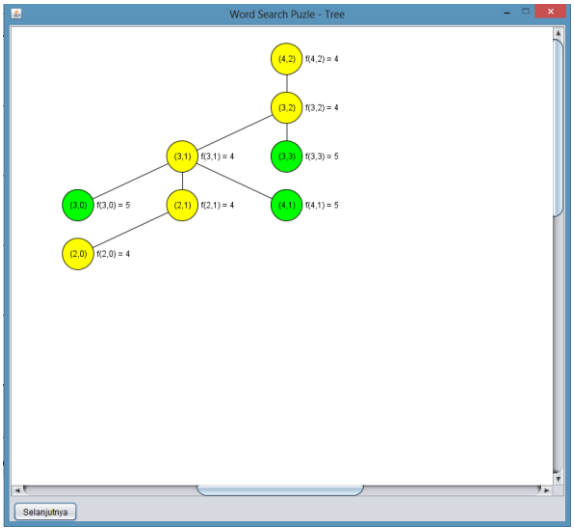
Gambar 27 *Class Diagram*

2.10 Implementasi Antarmuka

Antarmuka utama merupakan tampilan awal dimana merupakan tampilan pengujian ketika user menentukan posisi kata yang akan dicari oleh algoritma *Simplified Memory-Bounded A**. Pada antarmuka ini terdiri dari papan permainan, daftar kata, proses pencarian, dan *button* lainnya. Tampilan dari antarmuka utama dapat dilihat pada gambar berikut:



Gambar 28 Antarmuka Utama



Gambar 29 Antarmuka Penelusuran *Tree*

3 PENUTUP

3.1 Kesimpulan

Berdasarkan penelitian mengenai implementasi algoritma *Simplified Memory-Bounded A** untuk pencarian kata pada permainan *word search puzzle* maka dapat dibuat kesimpulan:

1. Algoritma *Simplified Memory-Bounded A** dapat digunakan untuk melakukan pencarian kata pada permainan *word search puzzle*.
2. Berdasarkan hasil pengujian bahwa semakin panjang karakter pada kata yang dicari maka waktu pencarian akan semakin lama dan penggunaan memori juga akan semakin besar. Semakin banyak simpul yang tersedia untuk melakukan pencarian maka waktu pencarian akan semakin cepat, dimana persentase peningkatan kecepatan pencarian dengan penambahan simpul sebanyak 100% dapat meningkat hingga 21,99% dibandingkan dengan tidak ada penambahan simpul.

3.2 Saran

Adapun saran yang dapat diberikan untuk pengembangan selanjutnya antara lain adalah sebagai berikut:

1. Menentukan persentase penambahan simpul yang optimal untuk setiap kasus pencarian.
2. Penambahan daftar kata dapat dilakukan secara otomatis oleh sistem dengan mengambil kata dari kamus Bahasa Indonesia.

DAFTAR PUSTAKA

[1] A. Rojali, "Analisis Perbandingan Algoritma Knuthmorris-Pratt dengan Algoritma Boyer-Moore Pada Permainana Word Search Puzzle," *Skripsi Teknik Informatika, Universitas Komputer Indonesia*, 2014.

[2] Suyanto, Artificial Intelligence Searching, Reasoning, Planning dan Learning, Bandung: Informatika Bandung, 2014.

[3] M. Nazir, Metodologi Penelitian, Bogor: Ghalia Indonesia, 2005.

[4] R. S. Pressman, Rekayasa Perangkat Lunak Pendekatan Praktis, Yogyakarta: Andi, 2012.

[5] A. Sukstrienwong and P. Vongsumedh, "Software Development of Word Search Game on Smart Phones in English Vocabulary Learning," *International Conference on Education and Modern Educational Technologies*, 2013.

[6] S. Russel and P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall International, Inc, 1995.

[7] B. Hariyanto, Esensi-esensi Bahasa Pemrograman Java, Bandung: Informatika, 2014.

[8] A. Nugroho, Rational Rose Untuk Pemodelan Berorientasi Objek, Bandung: Informatika, 2005.

[9] Munawar, Pemodelan Visual dengan UML, Yogyakarta: Graha Ilmu, 2005.

[10] K. Hamilton and R. Miles, Learning UML 2.0, Sebastopol: O'Reilly Media, Inc., 2006.

[11] Ayuliana, "Teknik Pengujian Perangkat Lunak," *Jurnal Teknik Informatika, Universitas Gunadarma*, 2009.