Pathfinding of 2D & 3D Game Real-Time Strategy with Depth Direction A*Algorithm for Multi-Layer

Khammapun Khantanapoka and Krisana Chinnasarn

Abstract— **The pathfinding analysis has importance for various working such as logistics, transportation, operation management, system analysis and design, project management, network and production line. Especially, game programming technology has effect to economic and dominates culture, increasingly. The shortest path analysis is artificial intelligent which developed capability about think cause and effect, learning and thinking like human. Heuristic technique is method for solving a problem which gets result or not. It may return unexpected value which depends on each problem. 3D game real-time strategy uses shortest path analysis control path for movement of character, wheeled vehicle, animals, and fantasy buildings. It uses path for movement ever time, both in sea, terra and sky (three layers). The position of 3D object use refers 2D coordinate. Generally, the most of 3D game strategy both on LAN and online will play on terrain more than inside building or the room. This research proposes two essences. It proposes comparison rapidity, intelligence and efficiency about seven path finding algorithms which is principle algorithm are familiar. These algorithms used in game computer extensively as follows [1] Depth First Search [2] Iterative Deepening [3] Breadth First Search [4] Dijkstra's Algorithm [5] Best First Search [6] A-Star Algorithm (A*) [7] Iterative Deepening A*. Besides, this research proposes Depth Direction A* algorithm method which new method which use linear graph theory cooperate with A* Algorithm calculate increase efficiency of avoid barrier object on maps and search for shortest path of multi-layer. It movement are natural characteristic more other method. The cost node grows less than A* algorithm in clear area and plan about expand node before processed. It guarantee about expand node less than A* algorithm node certainly. It takes time movement and avoid barrier object less than uses other algorithm. This work able decrease expand node 22.12-70.67% depend on property of area between pathway by define 3D rendering rate are stable.**

I. Introduction

A Specific problem of real time strategy for games is pathfinding, and its related problems of tactical and terrain analysis. Pathfinding is an important element of real time strategy games. Because the game involves so many units doing so many things at one time, the player simply does not have the time to be concerned with getting every individual unit where it needs to go at every moment. It is up to the games built in artificial intelligence to determine how a unit should proceed to a given destination. Generally, these algorithms try searching for best of the pathway and most efficient for search a path to the goal. This algorithm is based on a heuristic function. The pathfinding analysis used provide appropriate pathway for movement from starting position to destination position in various games, both player section and artificial intelligent section. It can solve a problem movement pass avoid barrier object cleverly with flexible method. Each algorithm will call stack and queue which are difference each genres. The popular genres of computer games are real time strategy. It is games have been around for quite a few years, and have grown and evolved with technology, as well as work in the field of artificial intelligence. The programming process use stack and queue for storage data of node and expand node which effect to rapidity and efficiency of algorithm very much. The shortest path analysis objective search for pathway shortest in moves to destination position. The shortest path analysis use cost value and amount of node measure efficiency which depend on capability about calculate cost value, decrease expand node and method for sort node inside stack and queue of each algorithm. The goal of shortest path analysis is optimal way.

Fig. 1. Example of 3D Game Real-Time Strategy

Beside, in game Real-Time Strategy has movement in Multilayer such as some object can move in the sea, on the ground and flying in the sky that each layer has cost value and barrier object which are difference. The 3D game has rendering process. The experiment must define rendering time are stable because will measure efficiency of pathfinding algorithm by independency.

Manuscript received August 8, 2009.

Khammapun Khantanapoka, Department of Computer Science, Faculty of Science, Burapha University 169 Long-Hard Bangsaen Road, Tambon Saensook, Muang, Chonburi, 20131, Thailand (e-mail: doramon86@hotmail.com)

Krisana Chinnasarn , Department of Computer Science, Faculty of Science, Burapha University 169 Long-Hard Bangsaen Road, Tambon Saensook, Muang, Chonburi, 20131, Thailand.

II. Algorithm detial

A. Analysis of Depth First Search algorithm

Depth first search is useful for find a path from one vertex to another Whether or not graph is connected computing a spanning tree of a connected graph. Depth first search uses the backtracking technique. The computing a spanning forest of graph are computing a path between two vertices of graph or equivalently reporting that no such path exists. Computing a cycle in graph or equivalently reporting that no such cycle exists.It must protect expend node to duplicate.

Depth First Search Pseudo Code

```
Create Start Node with Current Position
Add Start Node to Stack
    While Stack Not Empty
        Get Last Node From Stack call Node "N"
        If N is Goal Then Found and Exit Loop
        If N Cost > Max Depth Then Exit Loop
    Else
        Mark N Node as Visited
        Expand a reachable Node from N call Node "Next N"
        DFS with Next N (recursive)
Loop
```

B. Analysis of Iterative Deepening algorithm

Iterate Deepening algorithm working on the search depth, Iterative deepening has been invented as the basic time management strategy in Depth-First searches. Search to depth 1, then 2, then 3, until resources run out. The advantage is that result gets the deepest possible search depth given the resource constraints

Iterative Deepening Pseudo Code

```
Create Start Node with Current Position
Add Start Node to Stack
    While Stack Not Empty
        Get Last Node From Stack call Node "N"
        If N is Goal Then Found and Exit Loop
        If N Cost > Max Depth Then Exit Loop
            Else
        Mark N Node as Visited
        Expand a reachable Node from N call Node "Next N"
        DFS with Next N (recursive)
    Loop
If Not Found Then DFS with Max Depth = Max Depth + 1
```

C. Analysis of Breadth First Search algorithm

Breadth First Search algorithm used in Prim's MST algorithm, Dijkstra's single source shortest path algorithm. Like depth first search, Breadth First Search algorithm traverses a connected component of a given graph and defines a spanning tree. Breadth First Search algorithm starts at a given vertex, which is at level 0. In the first stage, system visits all vertices at level 1. In the second stage, we visit all vertices at second level. These new vertices are adjacent to level 1 vertex, and so on. The Breadth First Search algorithm traversal terminates when every vertex has been visited. The BFS uses queue kind First in First out This algorithm assure return shortest path certainly which take time very much.

Breadth First Search Pseudo Code

```
Create Start Node with Current Position
Add Start Node to Queue
        While Queue Not Empty
        Get Best Node in Same Depth From Queue call Node "N"
                If N is Goal Then Found and Exit Loop
                Else
                Mark N Node as Visited
        Expand each reachable Node from N call Node "Next N"
Loop
```

D. Analysis of Dijkstra's Algorithm

Dijkstra's algorithm works by visiting vertices in the graph starting with the object's starting point. It then repeatedly examines the closest not-yet-examined vertex, adding its vertices to the set of vertices to be examined. it expands outwards from the starting point until it reaches the goal. Dijkstra's algorithm is guaranteed to find a shortest path from the starting point to the goal, as long as none of the edges have a negative cost, because there are often multiple equivalently short paths. Dijkstra's algorithm works harder but is guaranteed to find a shortest path.

Dijkstra's Algorithm Pseudo Code

```
Create Start Node with Current Position
Add Start Node to Queue
        While Queue Not Empty
        Get Best Node in Same Depth From Queue call Node "N"
                If N is Goal Then Found and Exit Loop
                Else
                Mark N Node as Visited
Expand each reachable Node from N call Node "Next N"
If "Next N" is Diagonal Direction Then Cost = Cost + 0.4
Loop
```

E. Analysis of Best First Search Algorithm

The Best-First-Search algorithm works in a similar way, except that it has some estimate or called a heuristic of how far from the goal any vertex is. Instead of selecting the vertex closest to the starting point, it selects the vertex closest to the goal.

Best First Search Pseudo Code

```
Create Start Node with Current Position
Add Start Node to Queue
        While Queue Not Empty
        Sort Node Queue by Cost Value in Ascending
        Get First Node From Queue call Node "N"
                If N is Goal Then Found and Exit Loop
                Else
                Mark N Node as Visited
Expand each reachable Node from N call Node "Next N"
Loop
```

F. Analysis of A-Star(A*) Algorithm

A* is graph search algorithm that finds the least-cost path from a given initial node to one goal node (out of one or more possible goals). It uses a distance-plus-cost heuristic function (usually denoted f(x)) to determine the order in which the search visits nodes in the tree. The distanceplus-cost heuristic is a sum of two functions: the path-cost function (usually denoted g(x), which may or may not be a heuristic) and an admissible heuristic estimate of the distance to the goal .The path-cost function g(x) is the cost from the starting node to the current node.

A-Star Algorithm Pseudo Code

```
Create Start Node with Current Position
Add Start Node to Queue
        While Queue Not Empty
        Sort Node Queue by f(N) Value in Ascending
        Get First Node From Queue call Node "N"
        If N is Goal Then Found and Exit Loop
        Else
        Mark N Node as Visited
Expand each reachable Node from N call Node "Next N"
f(Next N) = g(Next N) + h(Next N)
Loop
```

G. Analysis of Iterative Deepening A Algorithm*

Iterative Deepening A-Star Pseudo Code

```
DF-Aux(Node, Cutoff)
        if f(Node) > Cutoff then return f(Node)
        if  solution(Node) then return Node
        Min = Temp
   for each Child of Node do
        Temp = DF-Aux(Child, Cutoff)
        if solution(Temp) then return Temp
        if Temp < Min then Min = Temp
        return Min
IDA*(Root)
        Cutoff = h(Root)
        loop
        Temp = DF-Aux(Root, Cutoff)
        if solution(Temp) then return Temp
        if Temp =  then return failure
        Cutoff = Temp
```

III. EXPERIMENT AND RESULT

This research proposes Depth Direction A* Algorithm method which new method which use linear graph theory cooperate with A* Algorithm calculate increase efficiency of avoid barrier object on maps and search for shortest path of multi-layer.

A. Depth Direction A(This research proposes)*

Depth Direction A-Star Algorithm Pseudo Code

```
Create Start Node with Current Position
Create A Linear Function by m = (Start Y – Goal Y) / (Start X –
Goal X)
Add Start Node to Queue
        While Queue Not Empty
        Sort Node Queue by f(N) Value in Ascending
Get First Node From Queue call Node "N"
    If N is Goal Then Found and Exit Loop
    Else
    Mark N Node as Visited
Expand a Node by Linear Function call Node "Next N"
    If Found Obstructions Then
    f(Next N) = g(Next N) + h(Next N)
    adjust Algorithm Style to A-Star Algorithm
    If Around Current Position <> 1 Then call Node "Next N"
Loop
```

B. Two dimension of Pathfinding

From experiment, the first of step, there are compare between seven type of pathfinding algorithm sure get algorithm which expand node and take time are least when working alone. This experiment use multilayer terrain by one layer, three layers and nine layers. It shows result of working clearly.

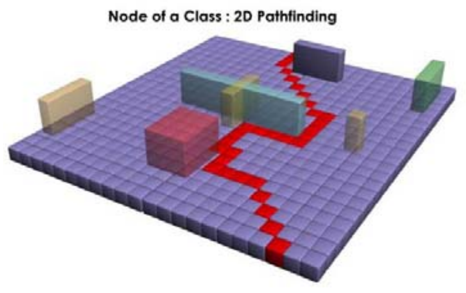


Fig. 2. Experiment of 2D Game Real-Time Strategy

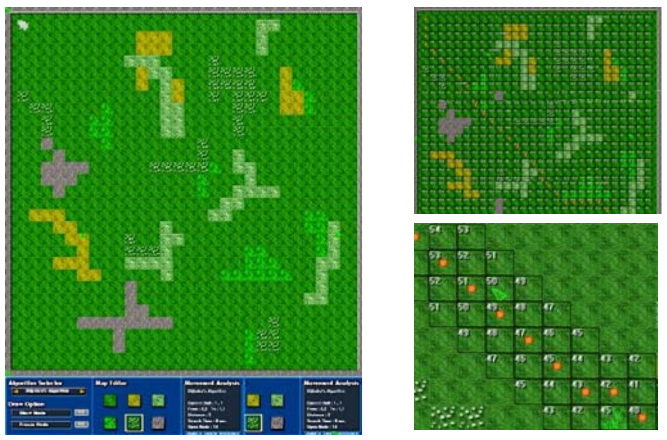
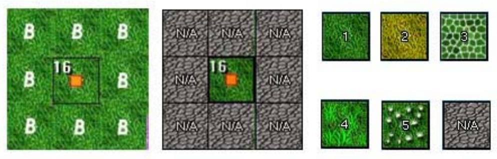


Fig. 3. Experiment with application on directX Technology.

The A* Algorithm has pathway as non-smooth. This paper solve a problem using directed graph theory(DD) cooperate with A-Star (A*) Algorithm. It use A* Algorithm for some situation when avoid barrier object, it use directed graph theory for outdoor situation which Increase rapidly, decrease expand node and memory process at the same time.

$$m_{layer} = \frac{y1 - y2}{x1 - x2} \tag{1}$$

where: m as slope of straight line.



(a) *(b)* *(c)*

Fig. 4. *From Experiment (a)* Around of current node have not barrier every side.(AND), *(b)* Around of current node at least one side close barrier which weight extremely and can not pass certainly (OR), *(c)* the kind of barrier which are weight value different.

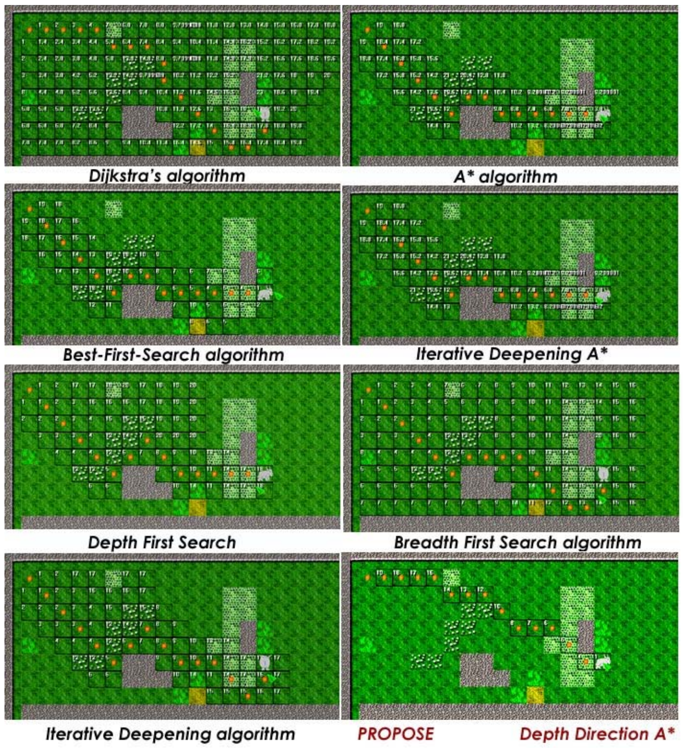


Fig. 5. *Example of experiment: Pathway of each algorithm which difference about rapidity, expand node, intelligence*.

If current node is not close any barrier every side use calculates directed graph promptly and when current node has at least one side close any barrier switch use A* algorithm. When current node is not close any barrier every side switch use calculates directed graph again. It applies to multi-layer of pathfinding. It switch are cycle until to destination position.

C. Three dimension of Pathfinding

There are expanding to work with three dimensions or multi-layer. In this experiment define change layer point has a position of each layer. The class has not beginning point or destination point will manage by AI. Each class fix sub-begin point which use position for change class (layer)in Y-axle (CLY) and fix sub-destination of that class. In each class have CLY from other class and CLY to the next class. The class has beginning point will manage beginning point to CLY of that class. Also the class has destination point will manage CLY of that class to destination point. Other class will manage by AI which use Depth Direction switch A* algorithm.

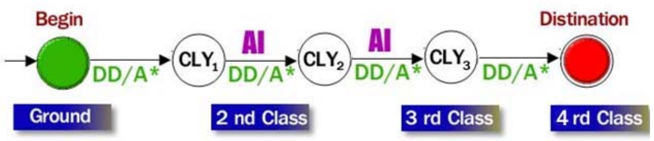


Fig. 6. State transition operation of four layers.

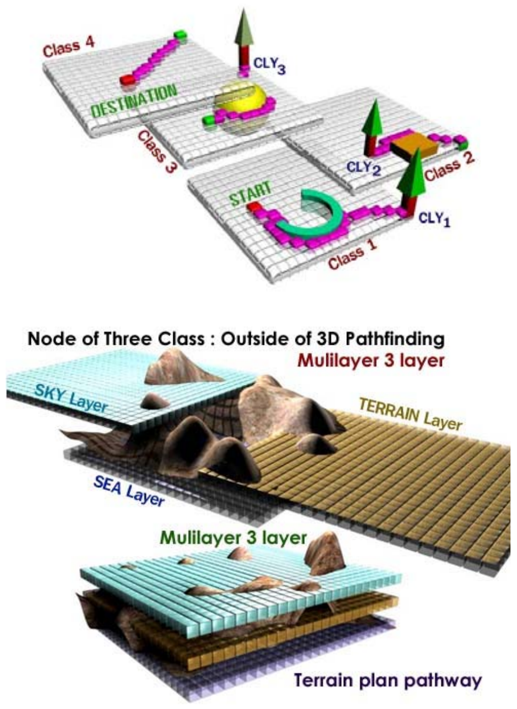
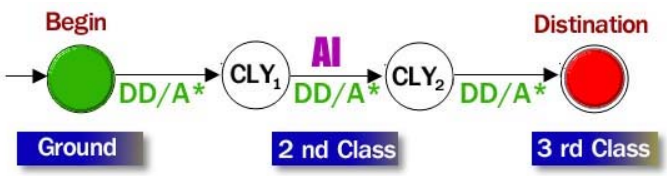


Fig. 7.Node of terrain multilayer(Three layers) which can change layer in Y-axle (Y-layer$_1$,Y-layer$_2$,Y-layer$_3$) with X-axle and Z-axle of position in lower layer or higher layer which must are not barrier.



Case 1: The ladder flow same Y-axle (change layer Y-axle)calculates with equation as follow:

$$SV_{node} = \sum_{j=1}^{layer}\sum_{i=1}^{n}\left(\frac{Y1-Y2}{X1-X2}\right)_i + (g(n+h(n))_i \qquad (2)$$

$$TSV_{node} = \mathrm{SV}_{\mathrm{node}} + \mathrm{CLY}_{\mathrm{n}} \qquad (3)$$

where

SV	as total of node every class (All layer).
*CLY*i	as node of change layer in each class.
n	as amount of layer.

Case 2: The ladder is not same Y-axle (change layer Y-axle) calculate with equation as follow:

$$TSV_{node} = \sum_{j=1}^{layer}\sum_{i=1}^{n}\left(\frac{Y1-Y2}{X1-X2}\right)_i + (g(n+h(n))_i + CLY_i \qquad (4)$$

where

SV	as total of node every class (All layer).
*CLY*i	as node of change layer in each class.
n	as amount of layer.

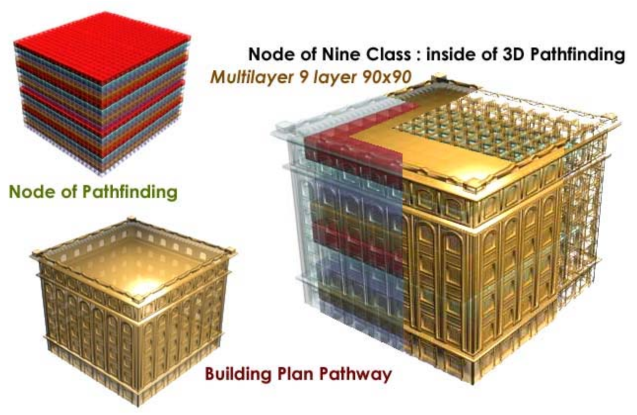


Fig. 8.Node of building multilayer(nine layers) which can change layer specifically area of change layer such as ladder, case do not use lift by define value change of layer are stable(Change Layer in Y-axle: CLY_n=1)

When define area for change level from a level to close level such as ladder in building. Example, begin point is ground floor and destination point is in 3rd class. First of path, begin point on ground to ladder#1.Second are path from ladder#1 to ladder#2 on the 2nd class. Third of path, define from ladder#2 on second class to destination point in 3rd class. This paper propose separate pathway with area for change level. In this experiment define weight of area for change level is stable (Change Layer in Y-axle: (CLY_n=1).

D. Result of Experiment

TABLE I
ONE LAYER OF GROUND :OUTDOOR

Algorithm	**Map 1# have not barrier.**		**Map 2#**	
	Take time(ms.)	**Expand Node**	**Take time(ms.)**	**Expand Node**
Dijkstra	109	783	109	643
BFS	46	133	31	123
DFS	63	397	47	217
IDA	15	133	16	113
A*	32	133	32	126
IDA*	31	133	31	126
Breadth FS	125	783	94	643
DepthD A*	<0	27	16	90
Algorithm	**Map 3#**		**Map 4#**	
	Take time(ms.)	**Expand Node**	**Take time(ms.)**	**Expand Node**
Dijkstra	94	557	109	582
BFS	31	115	31	328
DFS	47	220	63	271
IDA	32	282	16	128
A*	32	108	31	129
IDA*	16	113	31	127
Breadth FS	94	555	94	582
DepthD A*	15	88	15	91

[Dijkstra] Dijkstra's algorithm, [BFS] Best First Search, [DFS] Depth First Search,[IDA] Iterative Deepening algorithm, [A*] A-Star Algorithm (A*),[IDA*] Iterative Deepening A* algorithm, [Breadth FS] Breadth First Search [***DepthD A****] Depth Direction A*

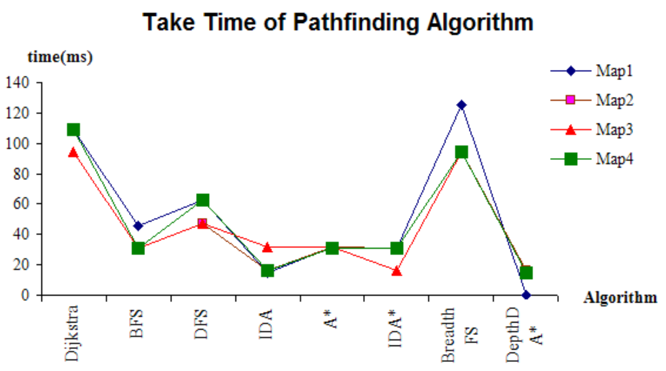


Fig. 8.Take Time of Pathfinding Algorithm

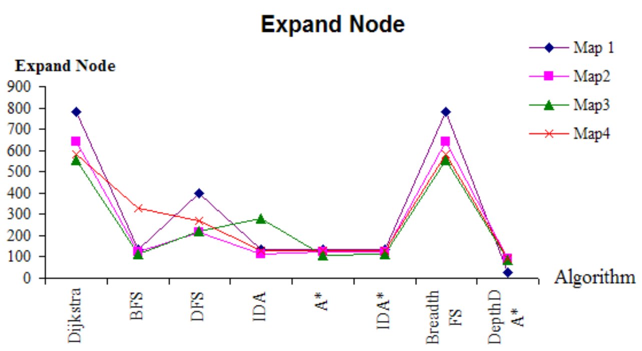


Fig. 9.Expand Node of Pathfinding Algorithm

IV. CONCLUSION

The Depth First Search may not found destination node in some case which node is depth than Maximum depth and is not shortest path. It has not good process for expand node. time complexity = O(b+n) and space complexity = O(b*n). The Iterative Deepening assure to destination node certainly by time complexity = O(b^n) and space complexity = O(1).The Breadth First Search need a lot of memory for Queue by time complexity = O(b^d) and space complexity = O(b^d). Dijkstra's Algorithm can found shotest path by time complexity = O(b^d) space complexity = O(b^d). Best First Search has amount of a few node but has rapidly when time complexity = O(b^d) and space complexity = O(b^d). A-Star algorithm return pathway is not smooth which time complexity = O(b*d) and space complexity = O(b*d).Iterative Deepening A-Star use least memory which time complexity = O(b*d) and space complexity = O(d).

For Depth Direction A*Algorithm has very smooth pathway and expand node more than A* algorithm. It can calculate about barrier before processed. It guarantee expend node less than A* algorithm. time complexity = O(b*d) and space complexity = O(d).

REFERENCES

[1] WATT, Alan & POLICARPO, Fabio. "*3D Games,Vol. 2: Animation and Advanced Real-Time Rendering*". Harlow, England: Addison-Wesley,2002.
[2] WOODCOCK, Steve. "*Game AI: The State of the Industry 2001-2002*". Game Developer Magazine,Vol. 9, No. 7, pp. 26-31, July 2002.