International Journal of Mechatronics, Electrical and Computer Technology

Vol. 4(12), Jul, 2014, pp. 1072-1094, **ISSN**: 2305-0543

Available online at: http://www.aeuso.org

Efficient Grid-Based Path Finding Techniques For Android Games Environments

Mehri Fayyazi[1*], Faranak Fotouhi Ghazvini [2] and Ali Ramzi[3]

[1]MSc graduate, Department of Information Technology

[2]Assistant Professor, Department of Information Technology

[3] Instructor, Department of Computer Science

Qom University, Qom, Iran

*Corresponding Author's E-mail: *Fayazi01.m@gmail.com*

Abstract

Path finding is one of the main challenges in the artificial intelligence system of game. Path finding is an expensive operation with large memory consumption; hence, it is not efficient in games that play on devices with memory constraints, like smart phones. This study aims at comparing memory consummation and execution time among A*, IDA* and our proposed algorithms on Android operating systems. The paper begins with an almost complete review of the current state of path finding algorithms for the different types of game environments. It then analyzes the search behavior of the aforementioned algorithms. The article presents novel works in the form of search algorithms for use in grid-based game maps: the SEA*, SEIDA*, DEA* and DEIDA* algorithms, which have significant improvements within testbeds and require less use of resources.

Keywords: A* Search; IDA* Search; Android Games; Artificial Intelligence (AI); Path Finding; Algorithm Performance.

1. Introduction

Path finding is one of the most popular problems encountered in artificial intelligence. It has many applications in operation research, computer science, telecommunications and

Vol. 4(12), Jul, 2014, pp. 1072-1094, ISSN: 2305-0543
Available online at: http://www.aeuso.org

multimedia systems. One of these applications is the path finding of characters in the game environment. Path finding is an expensive operation that uses up a considerable amount of system resources, such as CPU and memory. It thus becomes an important problem in games. Today's high quality game graphics and simulation of physics are very consuming, by nature, and therefore very few resources are available for AI and path finding systems. Subsequently, few resources for path finding are even considered [1].

Game developers are looking for solutions that are efficient and which can solve the path finding problem associated especially in environments with limited memory[2][3]. Due to the success of the A* algorithm in search problems, many researchers have focused on the optimization of A* for game environments.

On the other hand, smart phones play an important role - with applications like games which entertain and attract many users. Game development, in smart phones, is different to consoles or PCs due to their differences in CPU speed and memory size [4]. All in all, the use of more efficient path finding algorithms is now an essential factor in the evolution of smart phone games.

This study aims to present an innovative work whereby search algorithms are employed for grid-based games and devices with limited resources like smart phones. This article focuses on sub-goal selection and explores faster techniques that aim to reduce the size of the search space while preserving optimality.

The following sub-objectives will be discussed and investigated in this article:

1) A complete review of the current state of pathfinding in games with regards to the graph search algorithms.

2) A representation and classification of game worlds into searchable graphs.

3) Several potential optimizations of the A* algorithm.

4) Investigation into the performance criteria in evaluating path finding algorithms in games.

Vol. 4(12), Jul, 2014, pp. 1072-1094, ISSN: 2305-0543
Available online at: http://www.aeuso.org

The required terms and concepts are briefly described in Section 2. Section 3 lays out the framework of research methodology, in detail. In section 4 we review other recent works that have been carried out in this field. Section 5 discusses representations of Game Environments and the sixth section provides description of the A* algorithm and it's optimizations. In Section 7, a description of the system is provided. Section 8 describes the novel contribution of this paper. In Section 9, we give empirical evaluation of algorithms in testbed and evaluate novel algorithms in the domain of pathfinding. Finally in sections 10 and 11, the conclusion and suggestions will be presented.

2.Terms and Concepts

2.1.Navigational Maps

The complex 3D game environments (and their native storage formats) are highly unsuitable for searching; game developers are thus required to generate separate simplified navigational maps out of these complex environments. These navigational maps contain only the essential navigational data required to store the navigational data in a readily searchable format [5].

2.2. Static Or Dynamic Game Environment

With the game environment, if the obstacles' location have fixed the environment, the latter will be deemed as static, and if the environment is changed, it will be deemed dynamic. Therefore our study will focus on both static and dynamic environments [6].

2.3.Game Characters

Game characters are anything that can be viewed as perceiving its environment through input data and acting upon the environment through output data. Path finding systems are defined for characters that are not controlled by humans, namely none player characters (NPCs) [7].

International Journal of Mechatronics, Electrical and Computer Technology

Vol. 4(12), Jul, 2014, pp. 1072-1094, **ISSN**: 2305-0543
Available online at: http://www.aeuso.org

3.The framework of research methodology

According to the purposes of this study, the research process starts from the search and study of the key concepts and ends with the evaluation results of the proposed methods on the smart phone. Figure 1 has shown the framework of the research. The following steps were carried out:

1. The Key Concepts: the concepts of pathfinding algorithms in the game were determined, and the game AI and method for evaluating the algorithms' performance were explored.

2. The Literature review: was carried out by the adaptive of the agent's path finding system in the game development.

3. The path finding algorithms: those algorithms that applied in the game environment were studied and selected accordingly.

4. Select OS and Device: the appropriate device and the operating system were chosen for the game development.

5. The Idea(s): a new algorithm was designed to improve the character's path finding whilst taking into consideration the limitations of the smart phone resources.

6. User Interface Creation: a test bed was designed for the evaluation algorithms.

7. Programming: the designed algorithms were implemented in Android language.

8. Testing: the code was tested in the game environment.

9. Another Idea: the algorithm was improved after code testing.

10. Creation of a final project to android platform.

11. Test on phone: the final code was tested on the device to identify bugs.

12. Collect result on device: the amount of resource consumption was collected and saved in the TXT file of the SD card.

13. The Evaluating Result: the proposed algorithms - by the time and memory consumption - were evaluated.

International Journal of Mechatronics, Electrical and Computer Technology

Vol. 4(12), Jul, 2014, pp. 1072-1094, **ISSN**: 2305-0543

Available online at: http://www.aeuso.org

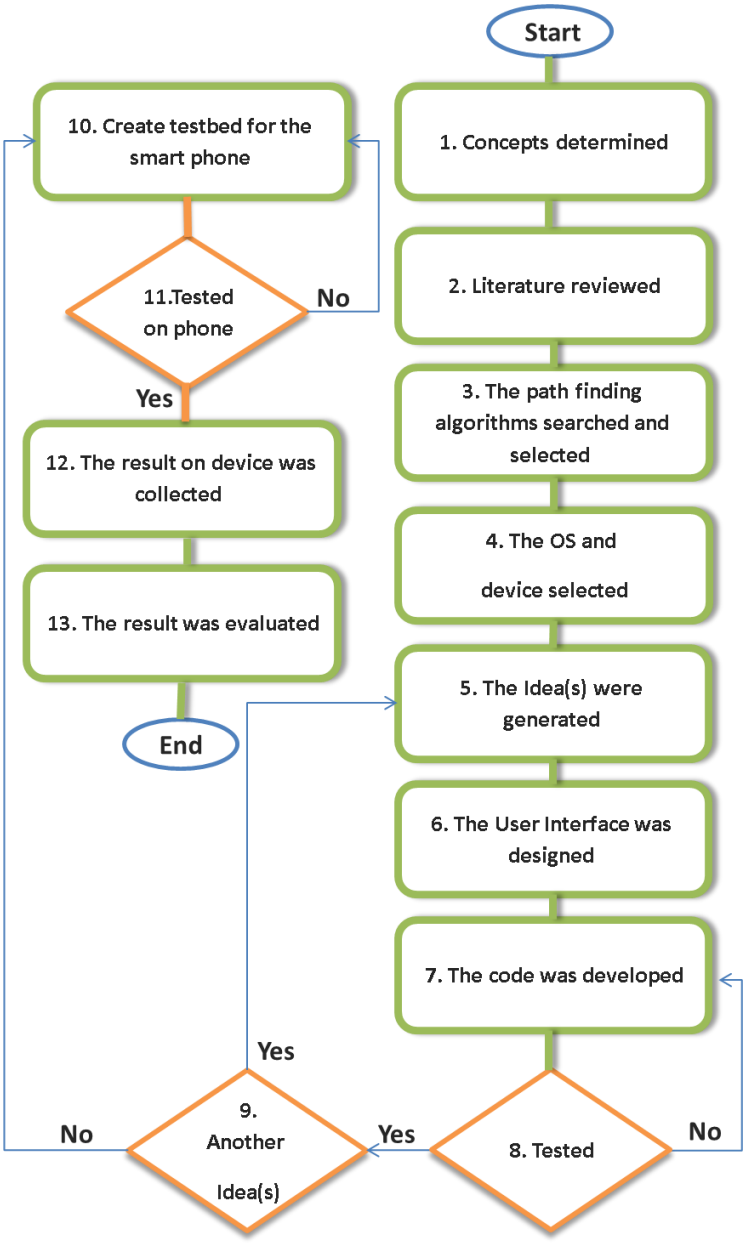


Figure 1: **The framework of the research methodology.**

International Journal of Mechatronics, Electrical and Computer Technology

Vol. 4(12), Jul, 2014, pp. 1072-1094, **ISSN**: 2305-0543
Available online at: http://www.aeuso.org

4. Literature Reviews

In this section, the current path finding methods in digital games are reviewed. It should be noted that a few of these methods are focused on reducing memory and time consumption. However in the field of robotics, many studies have been carried out on path finding or path planning. However these methods cannot be used directly in the game environment because the game environment 's features is different with robots structure and surrounding environment. For example, robots each have their own processing hardware and memories dedicated to their AI systems. Each robot‘s AI system is only responsible for that single robot and therefore most robotic pathfinding search algorithms are agent-centric [8]. An agent-centric algorithm maintains data relevant to that specific agent and cannot be used for any other agent.

In this section, we have reviewed basic techniques for implementing an efficient path finding in games (presented in Table 1). These techniques have three key factors, namely (1) representation of environment in game map, (2) efficient search algorithm and (3) appropriate heuristic function.

Recent literature concerns are development of heuristic function in larger environments, replacing simple maps with more complex maps, using hierarchical structures, considering concept of symmetry to prune search space.

Table 1: **A review on the previous studies findings in the area of game path finding**

Order	Writer(s)	Year	Research Findings
1	Goldberg and Harrelson	2005	This research has described an admissible heuristic as part of their ALT search method by considering a given node l, called a landmark, on the graph. A pre-computation step has provided a look-up table with true distances d (n, l) from every node n to l. they applied their ideas to graphs such as roadmaps and grids with the non-uniform costs [9].
2	Bj ornsson et al.	2005	In this paper, "Fringe Search" is introduced, a new algorithm that spans the space/time trade-off between A* and IDA* and experimental results evaluating A*, memory enhanced IDA*, and Fringe Search. The new algorithm is generalized into a

International Journal of Mechatronics, Electrical and Computer Technology

Vol. 4(12), Jul, 2014, pp. 1072-1094, **ISSN**: 2305-0543

Available online at: http://www.aeuso.org

			search framework that encompasses many of the important single-agent search algorithms [10].
3	Bj ornsson and Halld'orsson	2006	This article has focused on reduction of the state-space exploration while still making it possible to account for dynamic obstacles. They have presented the two effective heuristics. The former, the dead-end heuristic, eliminates from the search map areas that are provably irrelevant for the current query, whereas the second heuristic used the so-called gateways to improve its estimates [11].
4	Pochter et al.	2009	This paper have proposed a method that relied on identifying areas that tend to be searched needlessly (these areas were called swamps), and exploited this knowledge to improve search. The method required relatively little memory, and has reduced the search cost drastically, while still finding optimal paths [12].
5	Sun et al.	2009	This paper have reduced the number of priority queue operations for A* and its extensions. They have performed experiments in randomly generated four connected torus-shaped grids of size 200×200. Experimental results have shown that their optimizations speed up the "Generalized Adaptive A*" if its priority queue is implemented as a binary heap [13].
6	Goldenberg et al.	2010	This work has introduced a portal-based heuristic. In this approach, the domain is partitioned into regions and the portals between them. The two most important advantages of this work are the more accurate heuristic values and the ability to sub-divide the path-finding problem into a number of smaller problems by applying the Portal-Based Search algorithm [14].
7	Bulitko et al.	2010	This study has demonstrated a simpler and more memory-efficient way of pre-computing sub goals thereby eliminating the main obstacle to applying state-of-the-art real-time search methods in the video games. The new algorithm solves a number of randomly chosen problems off-line, compresses the solutions into a series of sub goals and stores them in a database [15].
8	Harabor and Botea	2010	In this paper a new faster technique was explored that aimed to reduce the size of the search space while preserving optimality. This work was focused on eliminating symmetric path segments from 4-connected grid maps, which allowed straight but not the diagonal movement. They have evaluated technique on a range of different grid maps including a well-known set from the popular video game Baldur's Gate II [16].
9	Koenig and et al	2010	This article has introduced two any-angle path-planning algorithms (1) Theta* and (2) Angle-Propagation Theta* (AP Theta*) to generate smooth paths. They have extended the algorithms to grids that contain unblocked cells with non-uniform traversal costs and provide shorter paths than both A* with post-smoothed paths and Field D* [17].
10	Harabor and Grastien	2011	In this paper they the path symmetries has been dealt with by developing a macro operator that selectively expands certain nodes from the grid, which is called jump points. To evaluate jump points they use a generic implementation of A* which

International Journal of Mechatronics, Electrical and Computer Technology

Vol. 4(12), Jul, 2014, pp. 1072-1094, **ISSN**: 2305-0543

Available online at: http://www.aeuso.org

			adapts to facilitate the online neighbor pruning and the jump point identification [18].
11	Botea and Harabor	2013	In this work, at the preprocessing step, all-pairs shortest paths are recorded in the database and the agent movement is based on the paths that are stored in the database. This approach is useful for the pathfinding in game environment as that can preprocess all paths. The goal is increase the speed and the efficiency of the storage. For reduction the size of the compressed path database, the run-length encoding and the sliding-window compression could be used. This idea has been tested on the game grid maps and the roadmap of Australia [19].

4. Representations Of The Game Environments

The game environment needs to be searchable; the pathfinding search algorithm firstly needs to be able to understand the environment and secondly, the environment needs to be able to be stored in a simple, easy to search format. In most cases, these complex game environments are stored in the specialized spatial trees used to facilitate collision detection and rendering optimizations [20] [21]. However the complex game environments are highly unsuitable for searching, so game developers are required to generate separate simplified navigational maps from these complex environments. The three most common techniques for the game environments abstraction are introduced. The first technique presented, is the waypoint graph, which creates the graph by connecting manually placed waypoints [22]. The second approach uses polygonal meshes to represent the floor of the game environment. These polygonal meshes represent the nodes in the graph and are connected to one another based on the common edges [23]. The final technique simply overlays a fixed size grid on the game environment and calculates which grid cells are traversable and which are obstructed. These grid cells are connected using the connection geometry to create the final grid-based graph [24]. Table 2 represents these techniques. In this table, environment abstraction techniques are compared in regards to the size of the resulting graph as well as the suitability of an abstraction technique for application within the dynamic or static game environments. The advantages and disadvantages are also listed.

International Journal of Mechatronics, Electrical and Computer Technology

Vol. 4(12), Jul, 2014, pp. 1072-1094, **ISSN**: 2305-0543

Available online at: http://www.aeuso.org

Table 2: **Comparison of game environment representations**

Representations Of Game Environments	Advantages	Shortcomings
Way Point Based Graphs	• Traditional method of abstraction for creating a navigational graph from a game environment. • Offering the smallest search spaces of the another technique	• unsuitable to use within dynamic environments[22] • To create require human involvement
Mesh Based Graphs	• Generated automatically, with high quality, and small size. • The numbers of graph nodes necessary to represent an environment were minimized and branching factor was reduced. • Near perfect coverage of the traversable environment was guaranteed rather than the Way Point technique. • Represent the smallest search space was represented rather than the Way Point technique.	• Expensive operation required to update graph when environmental changes occur • High costs associated with the creation and updating the mesh-based graphs • Limitation when using in dynamic environments [23] • Subdivision of the game floor comes at a very high computational cost
Grid Based Graphs	• An extremely cheap graph updates was offered. • Highly suitable for dynamic environments. • Information about both the traversable areas as well as the obstructed areas was stored, unlike the waypoint and the mesh graphs that only contain a representation of the traversable area • Used in RTS games, especially those featuring the dynamic environments.	• In the case of hex and octile grid representations, the graph will also have a high maximum branching factor [24].

5.The A* Algorithm

A* algorithm has become the benchmark for all point-to-point graph search algorithms, especially in the pathfinding field [16]. This algorithm often used in the path finding of characters in the real-time strategy game development. The standard terminologies used for

International Journal of Mechatronics, Electrical and Computer Technology

Vol. 4(12), Jul, 2014, pp. 1072-1094, **ISSN**: 2305-0543
Available online at: http://www.aeuso.org

this algorithm are, (1) g (n) which represents the exact cost of the path from the starting point to any vertex n, (2) h (n) which represents the heuristic estimated cost from vertex n to the goal, and (3) f (n) is the total cost of the results of the addition of g (n) with h (n).

There are several optimizations for A* from the four different perspectives, (1) for the search space optimization, (2) the heuristic function optimization, (3) the data structure optimization and (4) the memory optimization.

5.1.The Search Space Optimization

Node creation for every location in the game environments causes wasting resources. For example, in a large graph or platform with the memory constraints, large number of nodes is important and needs more consideration [25]. The proposed solution is used for the array of pointers instead of array of records. During the running algorithm, when a node is created, pointer to recode of node is created. This approach is more appropriate than the way the predetermined memory was allocated.

5.2.The Heuristic Function Optimization

The A* algorithm is version of the Dijkstra's algorithm with a change to the manner in which nodes are selected from the open list. A heuristic value (H) is created for each node and is stored alongside the other per-node data. The heuristic value is calculated by a heuristic function that, given two nodes, returns a numeric measure of how close the nodes are together. A simplistic way of describing the heuristic value is to term it the "estimated remaining cost".

The appropriate heuristic function increases speed of the algorithm. For example, if the heuristic estimate is slightly higher than the actual cost to destination, search algorithm finds the answer faster than the usual [26].

International Journal of Mechatronics, Electrical and Computer Technology

Vol. 4(12), Jul, 2014, pp. 1072-1094, **ISSN**: 2305-0543
Available online at: http://www.aeuso.org

5.3.The Data Structure Optimization

The characters pathfinding system requires using data that is stored in the data structures. The A* algorithm needs to retrieve data such as the G and H costs. These data should be stored in the structures that are not time consuming to insert and retrieve data from them. For example, a priority queue can reduce the search time if it is implemented as a binary heap data structure [27].

5.4.The Memory Optimization

Although the A* is the most popular search algorithm in the game, this point should be kept in mind that it causes the high memory and the processing cost if it runs on a large and complex environments. Reducing the memory consumption of this algorithm is one of the challenges researchers face.

One proposed solution is to consider and allocate minimum memory for the path finding operations before the implementation. At the run time, a buffer is allocated if more memory is needed and its size is changeable. The initial memory allocation depends on the complexity of the environment.

Another solution that has been proposed is to reduce the memory. This method creates partial routes that are based on the IDA* algorithm. IDA* algorithm is the simplest way to reduce memory requirements for the A* to adapt the idea of iterative A*. This algorithm has a cutoff value that is the smallest f-cost of any node, at any given iteration, that exceeds the cutoff on the previous iteration [28]. IDA* algorithm improves the memory consumption [29].

The main disadvantage of the A* algorithm is to compute memory because all the nodes produced by the algorithm is kept in the memory. Due to the fact that the memory consummation is more important than time execution, the A* it is not practical on many major issues.

International Journal of Mechatronics, Electrical and Computer Technology

Vol. 4(12), Jul, 2014, pp. 1072-1094, **ISSN**: 2305-0543
Available online at: http://www.aeuso.org

6.The System Description

In this research, the system operation starts by selecting the environment and the algorithm type, then, the selected point on the smart phone is considered as a destination point and the system displays the result path (Figure2).

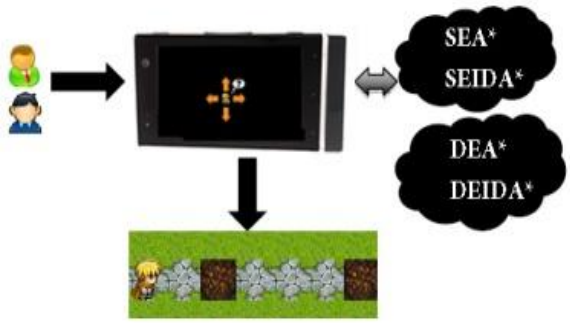


Figure 2: The system description.

7.The Proposed Algorithms

This section discusses a few novel approaches for performing the pathfinding searches within the grid based game maps; firstly in the static environments with memory constraint, namely called SEA* and SEIDA*, and secondly in the dynamic environments, namely DEA* and DEIDA*. These algorithms are presented as a means for performing subdivision and sub goal selection on the grid-based maps with the memory and time reduction. In these algorithms, unlike the other methods, there is no need for a general search. Sections creation and connection between them is basis of these algorithms. The advantages of these methods is that the search space of A* algorithm is deducted by the creation the sub problems. These algorithms are partially similar to hierarchical algorithms but this main difference is that the hierarchical methods have higher execution speed and consumption memory. In the proposed algorithms memory and the processing costs has been improved as compared to the A* and IDA* algorithms.

International Journal of Mechatronics, Electrical and Computer Technology

Vol. 4(12), Jul, 2014, pp. 1072-1094, **ISSN**: 2305-0543
Available online at: http://www.aeuso.org

7.1.The Development process of the proposed algorithms

In this study, we initially implemented the A* algorithm on the required test environment. Then, to reduce the memory consumption, we proposed a new algorithm i.e. Static Environment A*(SEA*) which is the reason why this path finding algorithm is developed to run on the static environments. This algorithm is composed of the A* and the Bresenham line algorithms. Since most of the games have the dynamic environments, another hybrid algorithm is presented for the path finding for the dynamic environments i.e. Dynamic Environment A* (DEA*). In this algorithm, if an obstacle is created in the path on which a character is moving, it becomes dynamically detected and so does not pass.

According to the characteristics of the mobile games, the memory consumption was more important in the proposed algorithms. For this reason, in this study, after SEA* and DEA* algorithms development, IDA* algorithm was used instead of A* algorithm and the new algorithms are proposed. These novel algorithms are named SEIDA* and DEIDA*. The reason for this selection was the simple implementation of the IDA* algorithm in comparison to another algorithm that is used for the memory reduction (e.g.: SMA* algorithm).

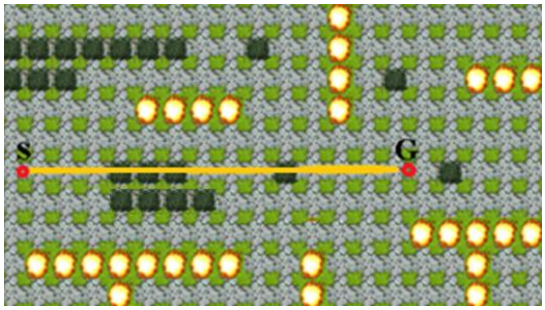
7.2.The sub goal selection phase in static environments

In our algorithms, firstly a direct line is drawn between the start and end locations. For this purpose, one of the most popular algorithms is the "Bresenham" line algorithm. If there is not any obstacle on the line, this line is the same optimal path between the two locations. But if there is an obstacle between two locations, the line is divided into the sub lines. Figure3 a, b and c represent how the algorithms progress.

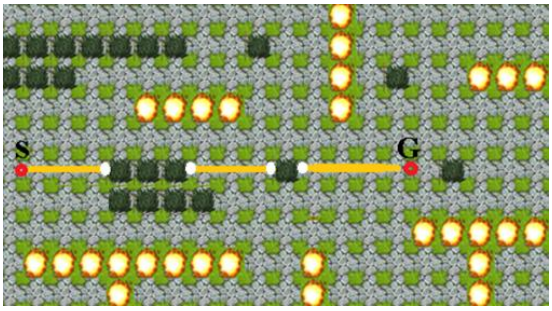
International Journal of Mechatronics, Electrical and Computer Technology

Vol. 4(12), Jul, 2014, pp. 1072-1094, **ISSN**: 2305-0543

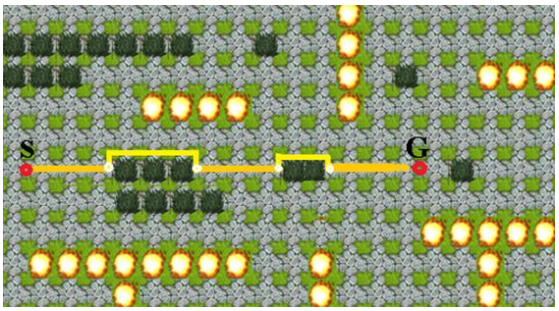
Available online at: http://www.aeuso.org



a) The Bresenham line algorithm between the start and the goal.



b) The direct line division into sub lines (line segments) and the determined sub starts and sub goals.



c) The line segments connection using the A*/IDA* algorithm.

Figure 3: The proposed technique for static environments

In Fig3, the sub start and the sub goal points are determined based on the obstacles location. In this case, in the direct path between the two nodes, after every block, the first place that is not blocked is selected as a sub goal. There is a sub start for each sub goal. After

Vol. 4(12), Jul, 2014, pp. 1072-1094, ISSN: 2305-0543
Available online at: http://www.aeuso.org

determining the sub starts and the sub goals between each pair of the start and the goal location, we store all these points (plus the start and the goal cell) in an array and run A* or IDA* algorithm between each pair of the sub start and the sub goal. The resulting path is added to the line segments. Finally, all the sections are joined together and a path is created between the original start and the goal node. The algorithm ends.

SEA* and SEIDA* are tried to reduce the unnecessary search space exploration by replacing large straight sections within the A*/IDA* search with line segments and only performing A*/IDA* searches around obstacles.

The sub goal selection phase in dynamic environments

The DEA* and DEIDA* algorithms are used for dynamic sub-goaling in the dynamic environments. In dynamic sub-goaling, first the direct path is drawn towards the goal node and the character is moved on it until an obstacle has been encountered; after the encountered obstacle, the sub start and the sub goal are determined with consideration of the obstacle location. The sub start and the sub goal are given the A*/IDA* algorithm to run. The resulting path is given the character to run it. During the movement, if the obstacle location is changed in the remaining path, the character becomes aware of it.

8.The Empirical Evaluation

In order to evaluate the efficiency of the proposed algorithms, compared with the A* and IDA* algorithm, algorithms have been tested on Android OS. The testbed is created by the researchers and is written for Android OS version 4.1. We have used the grid base representation and 4- connected grid maps. After installing the testbed APK file on a smart phone, a folder is created in an SD card. The content of this folder is the text files that are stored in execution time, and the memory consumption of the algorithms. In the testbed, we have evaluated six algorithms A*, IDA*, SEA*, SEIDA*, DEA* and DEIDA*. We used the test environment to empirically evaluate

International Journal of Mechatronics, Electrical and Computer Technology

Vol. 4(12), Jul, 2014, pp. 1072-1094, **ISSN**: 2305-0543

Available online at: http://www.aeuso.org

aforementioned algorithms using different grid criteria that is provided in the next section.

8.1.The performance criteria in evaluating game path finding algorithms in games

There are a set of elements that you need to know to implement and tune an algorithm, one of these elements is the analyses of the algorithms performance [27].

There are two major differences between an Android device, in a smart phone, and a traditional computer:

1. The amount of physical memory
2. The ability to do virtual memory swapping

Due to the differences mentioned that influence on the efficiency of the program, attention to the performance is more important in smart phone games [30]. The performance criteria proposed in recent years, are collected in Table 3.

Table 3: Performance criteria in game path finding algorithms that proposed in recent years

Order	Title Of Articles And Thesis	Year	Memory	Runtime	Path Length	Number of expanded nodes	Total cost of path	Path quality	re-visiting nodes
1	Efficient Path Finding For 2d Games [1]	2004	✓	✓					
2	Fringe Search Beating A* At Pathfinding On Game Maps [10]	2005		✓	✓		✓		✓
3	Fast Exact Multi Constraint Shortest Path Algorithms [31]	2007		✓					
4	Memory-Efficient Abstractions For Pathfinding [32].	2007	✓	✓					
5	Simple Optimization Techniques For A*-Based Search [13]	2009		✓		✓			
6	Online Graph Pruning For Pathfinding	2011		✓					

International Journal of Mechatronics, Electrical and Computer Technology

Vol. 4(12), Jul, 2014, pp. 1072-1094, **ISSN**: 2305-0543

Available online at: http://www.aeuso.org

Order	Title Of Articles And Thesis	Year	Memory	Runtime	Path Length	Number of expanded nodes	Total cost of path	Path quality	re-visiting nodes
	On Grid Maps [18]								
7	Efficient Path Finding For Tile-Based 2d Games [6].	2012	✓	✓					
8	Grid-Based Path Planning Competition Results [33].	2012	✓	✓	✓				
9	Automated Planning for Path finding in Real-Time Strategy Games [34].	2012	✓		✓				
10	Path Planning With Compressed All-Pairs Shortest Paths Data [19]	2013	✓	✓					
11	Sub goal Graphs For Optimal Pathfinding In Eight-Neighbor Grids [35].	2013	✓	✓					
12	Path Finding In Game [36].	2013	✓	✓				✓	

According to Table 3, the memory consumption and the speed of the algorithm (time) are the two main factors for comparing performance of algorithms. On a par with the aim of these articles, memory reduction is deemed more important than time reduction. Applications spend a significant portion of their time dealing with the data in memory. While many developers are aware of the need to try to use as little memory as possible on devices like smart phones, not all realize the impact of memory usage on the performance.

8.2.The Empirical Comparison

We have evaluated the efficiency of the new algorithms by running them on the testbed by aforementioned criteria. Figure 5 represents the result of our pathfinding experiments with static and dynamic obstacle in path.

International Journal of Mechatronics, Electrical and Computer Technology

Vol. 4(12), Jul, 2014, pp. 1072-1094, **ISSN**: 2305-0543

Available online at: http://www.aeuso.org

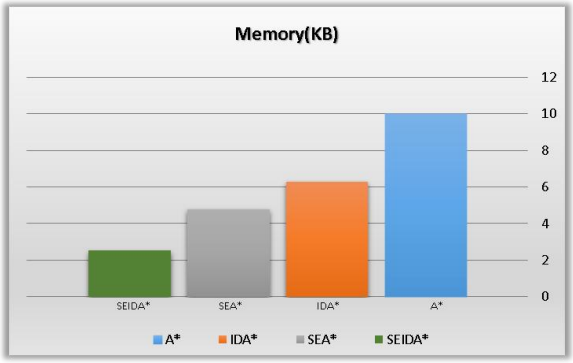


Figure 4: Comparing memory consumption of algorithms in paths with static obstacles (KB)

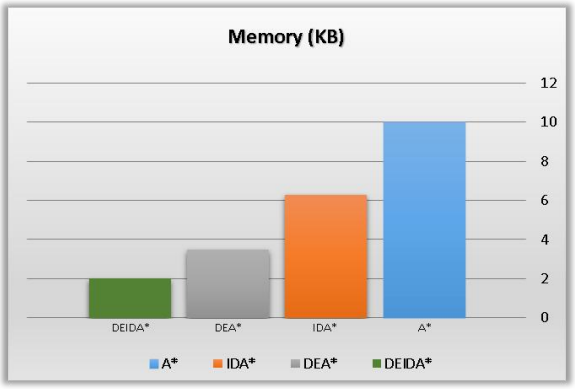


Figure 5: Comparing memory consumption of algorithms in paths with dynamic obstacles (KB)

As you can see in Figures 4 and 5, memory consumption of the proposed algorithms have significantly improved, compared to the A* and IDA* algorithms after removing parts of the search space.

International Journal of Mechatronics, Electrical and Computer Technology

Vol. 4(12), Jul, 2014, pp. 1072-1094, **ISSN**: 2305-0543

Available online at: http://www.aeuso.org

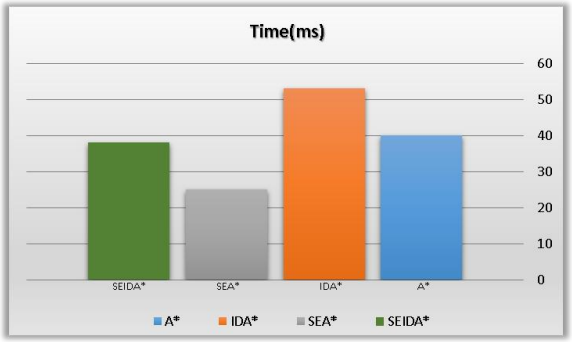


Figure 6: Comparing time consumption of algorithms in paths with static obstacles (MS)

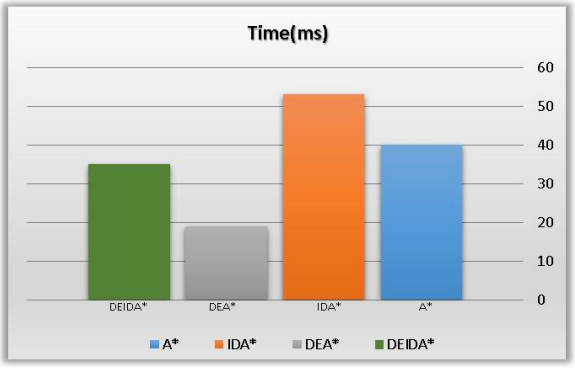


Figure 7: Comparing time consumption of algorithms in paths with dynamic obstacles (MS)

Figures 6 and 7 present time performances in each proposed algorithm compared with A* and IDA* algorithms. According to this chart, the SEA*and the DEA* algorithms have improved comparing to the A* algorithm. Therefore the SEIDA* and DEIDA* are faster, compared to the IDA* algorithm.

International Journal of Mechatronics, Electrical and Computer Technology

Vol. 4(12), Jul, 2014, pp. 1072-1094, **ISSN**: 2305-0543
Available online at: http://www.aeuso.org

Conclusions

In this paper we have provided the new algorithms for the characters path finding in Android games. The experiments and the results presented are relevant to many aspects in game industry. In this article we have presented some insight into the possibilities of path finding techniques that could be applied in game environments.

The combination of A*/IDA* algorithm and Bresenham line technique provided a flexible approach. Our proposed algorithms significantly improved the memory and the time consumption of A*/IDA* algorithms by sub dividing the path finding problem into a number of smaller problems and then applying the sub goal selection. The results represent an interesting difference between the A*/IDA* and the proposed algorithms. The solutions are human-readable, and thus easy to analyze.

In short, it could be concluded that there is a gap between academic researchers and game developers. It is hoped that results of our study help the game industry.

Future Work

Future work could improve these results by extending research into several interesting directions:

(1) In this paper we have only experimented with grid based graphs. In order to better demonstrate the advantages and limitations of algorithms it should be also implemented on other map representations, such as mesh.

(2) We have shown that the memory and the time optimizations could be applied to improve the performance of path finding algorithms like A* and IDA*. This line of research could be continued for other performance criteria.

(3) The performance of the proposed algorithms could be compared with other path finding algorithms that are not mentioned in our research. (i.e. except A* and IDA*)

(4) Further research in this field could be focused on improving the heuristic function of A* algorithm when used in SEA* or DEA* algorithms and comparing it

Vol. 4(12), Jul, 2014, pp. 1072-1094, **ISSN**: 2305-0543
Available online at: http://www.aeuso.org

with the result of this research.

(5) Our proposed algorithm could be extended for 8-connected grid maps and diagonal movements.

(6) A potential optimization was presented as a means of improving proposed algorithms performance within maps that feature disjoint regions as well as an extremely low connectivity between regions. If a line segment is created within a disjoint region of the grid map, this will cause the proposed algorithm to fail even though a valid path does in fact exist.

References

[1] Pawan Kumar, Len Bottaci, Quasim Mehdi, Norman Gough, Stephane Natkin." EFFICIENT PATH FINDING FOR 2D GAMES", (2004).

[2] David Bourg, M. Seeman, "AI for Game Developers", O'Reilly, (June 2004), Chapter 7.

[3] A Botea, M Müller, J Schaeffer, "Near Optimal Hierarchical Path-Finding", Journal of Game Development, vol.1, 1 (2004), pp.7-28.

[4] Thepparit Sinthamrongruk, Krisada Mahakitpaisarn, and Wapee Manopiniwes, "A Performance Comparison between A* Pathfinding and Waypoint Navigator Algorithm on Android and IOS Operating System," International Journal of Engineering and Technology vol. 5, no. 4 (2013), pp. 498-501.

[5] Syrus Mesdaghi, "Artificial Intelligence: Pathfinding Overview." [ed.] Steve Rabin. Introduction to Game Development. s.l.: Charles River Media, vol.5, 4 (2010), pp. 559-576.

[6] Masilo Mapaila, Efficient Path Finding For Tile-Based 2d Games ,University of Cape Town Rondebosch Cape Town, South Africa, (2012).

[7] Bobby Anguelov, "Video Game Pathfinding and Improvements to Discrete Search on Grid-based Maps", Master‘s dissertation, University of Pretoria, Department of Computer Science, Pretoria, South Africa, (2011).

[8] S. Koenig, "Agent-centered search." AI Magazine, Vol. 22, 4 (2001), pp. 109-131.

[9] A. Goldberg, and C. Harrelson, "Computing the shortest path: A* search meets graph theory". In SODA, (2005), pp.156-165.

[10] Yngvi Björnsson, et al., "Fringe Search: Beating A* at Pathfinding on Game Maps." Colchester, Essex: IEEE. IEEE Symposium on Computational Intelligence in Games, (2005), pp. 125-132.

[11] Y. Björnsson and K Halld´orsson, "Improved heuristics for optimal path-finding on game maps". In AIIDE, (2006), pp. 9-14.

Vol. 4(12), Jul, 2014, pp. 1072-1094, **ISSN**: 2305-0543
Available online at: http://www.aeuso.org

[12] N. Pochter; A. Zohar; and J. S. Rosenschein. Using swamps to improve optimal pathfinding. In AAMAS, (2009), pp. 1163-1164.

[13] X. Sun; W. Yeoh; P.-A. Chen; and S. Koenig. Simple optimization techniques for a*-based search. In AAMAS (2), (2009), pp. 931-936.

[14] M. Goldenberg, Ar. Felner, N. Sturtevant, and J. Schaeffer. "Portal-based true-distance heuristics for path finding". In Proceedings of the Symposium on Combinatorial Search SoCS-10, (2010), pp.39-45.

[15] Vadim Bulitko, Yngvi Björnsson, Ramon Lawrence, "Case-Based Subgoaling in Real-Time Heuristic Search for Video Game Pathfinding", Journal of Artificial Intelligence Research, 39, (2010), pp.269-300.

[16] D. Harabor and A. Botea, "Breaking path symmetries in 4-connected grid maps". In AIIDE, (2010), pp. 33-38.

[17] S. Koenig, and A. Felner," Theta*: Any-angle path planning on grids". JAIR 39, (2010), pp. 533-579.

[18] Daniel Harabor and Alban Grastien, "Online Graph Pruning for Pathfinding on Grid Maps", Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, (2011).

[19] Adi Botea, "Path Planning with Compressed All-Pairs Shortest Paths Data", Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling, Daniel Harabor, (2013).

[20] Jason Gregory, "Game Engine Architecture. Wellesley" MA : A K Peters Ltd, (2009).

[21] Tomas Akenine-Mölle, Eric Haines and Naty Hoffman, Real-Time Rendering. 3rd. s.l.: AK Peters, (2008).

[22] Mark DeLoura, "A* Speed Optimizations." [ed.]. Best of Game Programming Gems. Boston, MA: Course Technology, Cengage Learning, vol. 3, 3 (2008), pp. 255-270.

[23] David Hamm, "Navigation Mesh Generation: An Empirical Approach." [ed.] Steve Rabin. AI Game Programming Gems 4. Boston, MA: Course Technology, Cengage Learning, (2008), pp. 113-124.

[24] Peter Yap, "Grid-Based Path-Finding." s.l: Springer, Advances in Artificial Intelligence: 15th Conference of the Canadian Society for Computational Studies of Intelligence, (2002), pp. 44-55.

[25] B. Stout, "The basics of A* for path planning," in Game Programming GEMS, Charles River Meida, America, (2000), pp.254-262.

[26] S. Rabin, "A* speed optimizations," Game Programming GEMS, Charles River Media, (2000), pp. 264-271.

[27] Ian Millington, "Artificial Intelligence for Games", Elsevier Inc, (2006).

[28] Stuart Russel, and Peter. Norvig, "Artificial Intelligence: A Modern Approach." [book auth.] Peter Norvig Stuart Russel. Artificial Intelligence: A Modern Approach. s.l. : Prentice Hall, vol.3, 3 (2002), pp. 59-94.

[29] Dentson Cockburn and Ziad Kobti, "Memory-Bounded D* Lite", Proceedings of the Twenty-First International FLAIRS Conference, (2008).

[30] Guihot Hervé, "Pro Android Apps Performance Optimization", (2012).

[31] Li Yuxi; J. Harms; Robert Holte, "Fast Exact MultiConstraint Shortest Path Algorithms", Communications, ICC '07. IEEE International Conference, (2007), pp. 123 - 130

International Journal of Mechatronics, Electrical and Computer Technology

Vol. 4(12), Jul, 2014, pp. 1072-1094, **ISSN**: 2305-0543

Available online at: http://www.aeuso.org

[32] R. Nathan Sturtevant "Memory-Efficient Abstractions for Pathfinding" In AIIDE, (2007), pp. 31-36.

[33] Nathan Sturtevant, Grid-Based Path Planning Competition Results, university of DENVER, (2012).

[34] Naveed Munir Hussain, "Automated Planning for Pathfinding in Real-Time Strategy Games". Doctoral thesis, University of Huddersfield, (2012).

[35] Tansel Uras, Sven Koenig, Carlos Hern´andez,. Subgoal Graphs for Optimal Pathfinding in Eight-Neighbor Grids, Association for the Advancement of Artificial Intelligence, (www.aaai.org), (2013)

[36] Adi Botea, Bruno Bouzy, Michael Buro, Christian Bauckhage and Dana Nau. "path finding in games", licensed under Creative Commons License CC-BY. Artificial and Computational Intelligence in Games. Dagstuhl Follow-Ups, Vol.6, ISBN 978-3-939897-62-0 (2013)