Disadvantages of algorithm.

- Lack of uniform distribution of agents in the study area.

E. Fireflies algorithm

This algorithm was proposed by X. Sh.Yang in 2007.

All fireflies attract each other. Attractiveness of firefly is proportional to its brightness. Less attractive fireflies move to more attractive fireflies. Brightness of firefly for other glowworm decreases with increasing distance between them. If firefly do not see more bright firefly than it, then it move in random direction.

Advantages of algorithm.

1. Equivalence and interchangeability of agents.
2. There is no need to track position of colony mass center, as in case of SWARM algorithm.
3. Simple scalable.

Disadvantages of algorithm.

1. Lack of leader in colony leads to difficulty of managing move direction of swarm.

PATHFINDING ALGORITHMS EFFICIENCY ESTIMATING IN DISCRETE LABYRINTH BASED ON SOFTWARE SIMULATION

Krasnov E.S., Bagaev D.V.
Instrument engineering dept.
Kovrov State Technological Academy named after V.A. Degtyarev
Kovrov, Russia
krasnoves42@gmail.com, dmitrybag@gmail.com

Abstract – This article continues the research, started in the first article – «Strategy of analyzing most common algorithms for path finding in discrete labyrinth using software statistic data collector» [1]. It is dedicated to experiment's overview and summarizing its results. The common structure of the experiment, its stages, collecting data and methods of its processing are described. The main conclusions are made at the end of this article.

Keywords – algorithms, maze solving, data analyzing, software simulation

I. INTRODUCTION

Statistics was collected via special simulation software, previously described in the first article. It was improved in ways of usability, results' displaying, but not in way of changing calculation methods, described in the first article [1]. The detailed description of the software will be given below.

II. SOFTWARE DESCRIPTION

The software, used in the experiment, is meant to be run on Microsoft Windows • platform. It is a sort of «sandbox» for creating two-dimensional discrete labyrinths

(also maps or mazes; for more detailed description see article [1]) and manipulating with them. Also, it can perform different pathfinding algorithms on created map (see [1] for the list of used algorithms), collecting the required stat data. Here the main features of the software:

- creation of two-dimensional discrete labyrinths (via built-in simple graphics editor, similar to Microsoft Paintbrush, allowing to draw obstacles) and saving them to/loading them from file;
- running selected algorithm on the drawn map with displaying the result route, obtained via this algorithm;
- sequential running of all built-in algorithms with collecting stat data (see [1] for the list of collecting values) and its saving for further analysis;
- viewing and analysis of stat data and exporting for further processing in the third party software;

The figure below represents the modified and improved interface of the simulation software, that is displaying the middle-sized map (filled for 42% with obstacles) and a result route, built with A-Star (A*) algorithm (Fig. 1).

Let's introduce a notion: test (session) – it is single run of all of the built-in pathfinding algorithms on the current map (labyrinth), with measuring all the required values and saving collected data. The test (session) could be run by pressing the «Run all algorithms» button.

To obtain a reliable time value of algorithm run, software needs to repeat the run of every algorithm multiple times (from 1 to 500 times; it is a user-defined value). It enables to obtain the average value of the time value.

Running of all algorithms was initially programmed to take the special order, for avoiding influence of processor's cache memory mechanism (it could give a erroneous values). Such a thing could happen if the algorithms would be run in the next order: 1, 1, 1, 1, ... 2, 2, 2, 2, ... 3, 3, 3, 3, 3... and so on. To avoid this, the next order was used: 1-2-3-4..., 1-2-3-4... and so on, where 1-9 are the numbers of the algorithms. But the experiment showed, that it doesn't actually happen, and moreover, the selected order makes impossible to measure the time of some algorithms (due to their extremely high speed). For example, the measuring of single run of the classic wave algorithm always resulted in zero milliseconds. So, the order of algorithms' running was restored, and the full time of full repeat cycle for every algorithm was measured. Later it was divided at by the number of repeats.

After finishing the session, the collected data is being saved for further analysis (single algorithms runs are ignore, see explanation below). The viewing of collected data and its partial analysis can be done via stat form window, displayed on the figure below (Fig. 2).

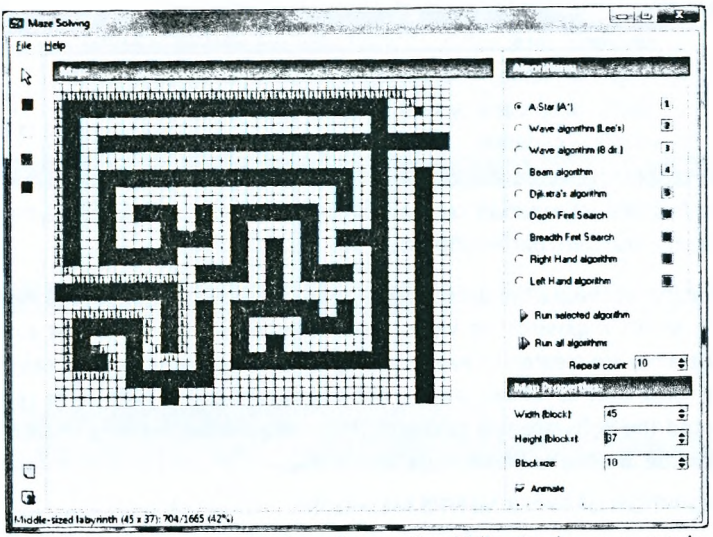

Figure 1 - Software graphic interface with middle-sized map example.

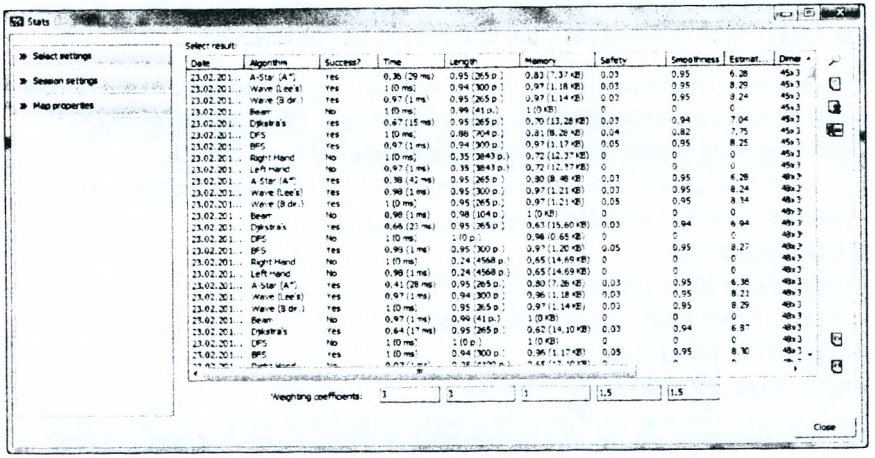

Figure 2 - Stats viewing window.

The main features of the stats module:

- collecting and keeping of the stat data;
- calculation of the normalized and non-normalized characteristic values (see [1]) and the F value (estimation of the algorithm, also described in [1]), calculated with user-specified weighting coefficients k_I, k_J, k_M, k_H, k_A;

- selection of the partial data by the following criterions:
    - session's date;
    - map's filename;
    - width, height and type of the map (by size);
    - map's occupancy;
- data export (selected test or the whole amount) to the HTML-file (for more comfortable information representation, its further editing/formatting and transfer into third party software).

The selection of the partial data by described criterions is designed to filter the whole test, so it's impossible to view results only for A* algorithm, for example. These constraints are created to avoid the improper characteristic values calculation (they have sense only as relative values, not as absolute ones). With some test data being deleted the software will calculate the wrong values, what is unacceptable and tests can be deleted or filtered as a whole only.

III. DESCRIPTION OF EXPERIMENT'S METHODS

The only improvement of the calculations' methods is as follows: the points of the map, close to map's edge are considered as hazardous during calculating an *H* value.

For collecting all required data, the special method was invented.

The main map's characteristics are its size and occupancy by obstacles. As you can see in article [1], there's a following classification of maps made (three occupancy-types):

- poorly-filled (less than 10% of coverage by obstacles);
- medium-filled (more than 10% and less than 40% of coverage by obstacles);
- strong-filled (more than 50% coverage by obstacles).

and four types by size:

- small (about 10x10 blocks);
- medium (from 40x40 up to 60x60 blocks);
- large (about 100x100 blocks);
- huge (about 200x200 blocks or more).

So, its necessary to run at least 12 groups of tests (3 x 4), which enables to make a conclusion about algorithms' efficiency in every group. The number of tests (sessions) in each group is about one hundred. Every test is run on the map with a little modified size, occupancy and a structure.

After all the tests run, the average value of *F* (see [1]) is calculated (for every group), which allows to make a number-supported conclusion about each algorithm's efficiency.

TABLE I - RESULT OF THE PRIMARY SERIOES OF TESTS

	Poorly-filled		Medium-filled		Strong-filled	
Small	A*	6,82	A*	6,41	A*	6,05
	Wave	8,16	Wave	8,01	Wave	7,53
	Beam	1,53	Beam	0,97	Beam	0,53
	D	6,54	D	6,46	D	6,44
	DFS	4,32	DFS	4,16	DFS	3,05
	BFS	**8,17**	**BFS**	**8,02**	**BFS**	**7,57**
	R	5,62	R	3,71	R	7,2
	L	5,78	L	3,72	L	7,24
Medium	A*	6,88	A*	6,71	A*	6,5
			Wave	**7,71**	Wave	7,67
	Wave8	**8,59**				
	Beam	2,34	Beam	0,32	Beam	0
	D	7,35	D	5,99	D	6,05
	DFS	8,12	DFS	0,92	DFS	0,56
	BFS	**8,41**	**BFS**	**7,7**	**BFS**	**7,68**
	R	6,2	R	5,69	R	5,55
	L	6,44	L	5,81	L	5,49
Large	A*	6,22	A*	6,18	A*	6,06
					Wave	**7,98**
	Wave8	**8,13**				
	Beam	3,65	Beam	0,11	Beam	0
	D	6,69	D	6,58	D	6,33
	DFS	1,23	DFS	0,42	DFS	0
	BFS	**8,04**	**BFS**	**8,02**	**BFS**	**7,97**
	R	5,65	R	5,8	R	5,78
	L	5,74	L	5,76	L	5,81
Huge	A*	6,51	A*	6,31	A*	6,2
	Wave	8,04	Wave	8,02	Wave	7,98
	Beam	1,03	Beam	0,05	Beam	0
	D	7,04	D	6,58	D	6,48
	DFS	0	DFS	0	DFS	0
	BFS	**8,31**	**BFS**	**8,11**	**BFS**	**8,09**
	R	5,41	R	5,43	R	5,45
	L	5,50	L	5,49	L	5,7

Note: the following designations were made:

- A* – A-Star algorithm;

- Wave – Wave algorithm (Lee's);
- Wave8 – Wave algorithm (8 directional);
- Beam – Beam algorithm;
- D – Djikstra's algorithm;
- DFS – Depth First Search;
- BFS – Breadth First Search;
- R – right-hand algorithm;
- L – left-hand algorithm.

TABLE II - RESULTS OF THE SERIES OF TESTS

	Poorly-filled		Medium-filled		Strong-filled	
Small	R	7,26	R	7,81	R	7,31
	L	7,84	L	7,83	L	7,26
Medium	R	7,35	R	7,69	R	7,64
	L	7,22	L	7,81	L	7,42
Large	R	7,46	R	7,62	R	7,88
	L	7,71	L	7,56	L	7,73
Huge	R	7,33	R	7,4	R	7,68
	L	7,44	L	7,32	L	7,59

The F value can be calculated by the following equation:

$$F_{alg} = k_T \cdot T_N + k_L \cdot L_N + k_M \cdot M_N + k_H \cdot H_N + k_A \cdot A_N, \quad (1)$$

which is described in details in [1].

The following weighting coefficients were used during experiment:

$$k_T = 3;\ k_L = 3;\ k_M = 1;\ k_H = 1.5;\ k_A = 1.5;\ ,$$

which are limiting the *F* value to range: from 0 to 10 (see [1] for more). Time and route-length characteristics values T_N and L_N are the most important, while the M_N value have the lowest weighting coefficient, since it is the least significant.

Also, the additional batch of tests was run to answer a question, asked in [1] – which algorithm is better – right-hand or the left-hand one? Since both algorithms can find a route only if the end-point is close to the wall, the test were run with such a condition, taken into the account (unlike to the main series of tests, in which it wasn't mentioned for avoiding the improper high *F* values for these algorithms).

Experiment's results. In this section the main results of the stat data analyzing are given (without listing all the processed data due to its huge size and unobviousness). All calculations were made in Micrisoft Excel ®.

The results are given as a table, which has 12 sections (one for every group of tests), divided into a smaller blocks, that represent average F values for each algorithm. The «best» algorithms are highlighted. Here it goes:

The results are given as a table, which has 12 sections (one for every group of tests), divided into a smaller blocks, that represent average F values for each algorithm. The «best» algorithms are highlighted. Here it goes (Table I).

IV. CONCLUSIONS

According to experiments results, the following alignment of forces has taken its place:

1. A* and Djikstra's algorithms have the lowest performance and require more memory than other algorithms. But route, they provided is often the shortest.
2. Classic Wave algorithm (Lee's) is faster than the 8-directional one (about 30-60% faster), but considering their great speed it doesn't make any real difference. At the other hand, the route, provided by the 8-directional Wave algorithm is much smoother with a low number of turns. Also it is as short as A* and Djikstra's ones most of the times.
3. Beam algorithm has extremely high fail rate, so it can be used only for very liitle situations and studying examples, with the simple maps only. Still it has the the great performance rate. Using it with real problems is mostly inappropriate.
4. Depth first search can't reach the end-point in large and huge labyrinths, due to overflowing of the stack in recursion. It faster even than the both Wave algorithms, but can be used for large maps.
5. Breadth first search is comparable to the 8-directional Wave algorithm, but provide a little longer routes. In common it's a little better, than the classic Wave algorithm.
6. The right/left hand algorithms are the fastest (in case of succesing), but they have about 50% of failing rate. The secondary series of tests cleared that these algorithms are equal in common.

So the 8-directional Wave algorithm seems to be the best one. It is fast enough, consumes little memory and provides a short and smooth route, which is medium-safe.

Thus the main questions, asked in [1], have been answered. The results are statistically reliable and there's only one «winner».

REFERENCES

[1] Krasnov E.S., Strategy of analyzing most common algorithms for path finding in discrete labyrinth using software statistic data collector// « East-West Design & Test Symposium (EWDTS 2012)», p. 34-41.